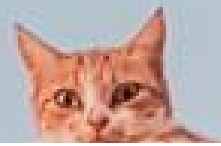


Computer Organization

Lab13 Cache(Direct Mapped) Implemention and Performance

Topics

- Locality
- Cache
  - Direct Mapped Cache
    - Implement the Direct Mapped Cache in Verilog
    - Test the Performance of Direct Mapped Cache

Locality

- Temporal locality: if you used some data recently, you will likely use it again
- Spatial locality: if you used some data recently, you will likely access its neighbors

```
.data
        array: .word 0,1,1
        tmp:   .word 0 : 100
.text
        la $t0, array
        li $t1, 25
        loop:
                lw $t3, 0($t0)
                lw $t4, 4($t0)
                lw $t5, 8($t0)

                add $t2, $t3, $t4
                add $t2, $t2, $t5

                sw $t2, 12($t0)

                addi $t0, $t0, 16
                addi $t1, $t1, -1
                bgtz $t1, loop
        li $v0, 10
        syscall
```

For the demo on the right hand, while it run on the CPU

Q1. Is there any Temporal locality or Spatial locality on **Instruction-Memory**? How does it happen?

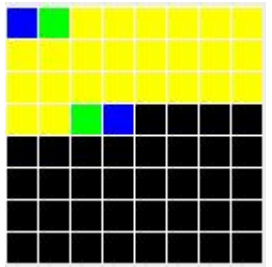
Q2. Is there any Temporal locality or Spatial locality on **Data-Memory**? How does it happen?

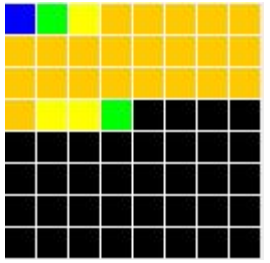
4 Locality continued

```
.data   #demo1
        array: .word 0,1,1
        tmp:   .word 0 : 100
.text
        la $t0, array
        li $t1, 25
        loop:
                lw $t3, 0($t0)
                lw $t4, 4($t0)
                lw $t5, 8($t0)

                add $t2, $t3, $t4
                add $t2, $t2, $t5

                sw $t2, 12($t0)

                addi $t0, $t0, 4
                addi $t1, $t1, -1
                bgtz $t1, loop
        li $v0, 10
        syscall
```

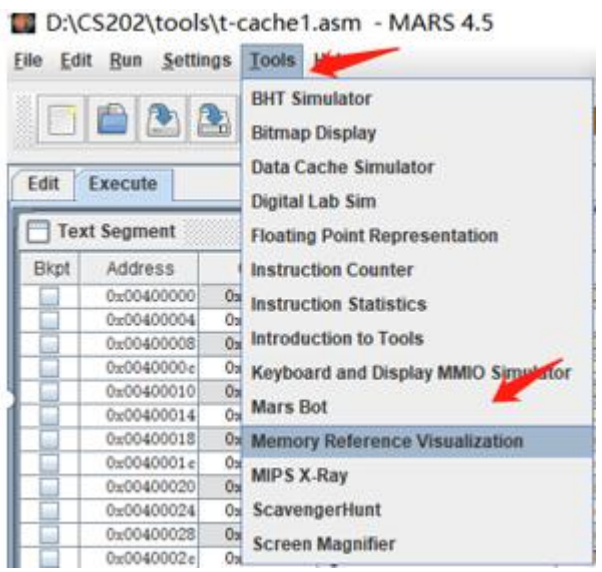
```
.data   #demo2
        array: .word 0,1,1,2
        tmp:   .word 0 : 100
.text
        la $t0, array
        li $t1, 25
        loop:
                lw $t3, 0($t0)
                lw $t4, 4($t0)
                lw $t5, 8($t0)
                lw $t6, 12($t0)

                add $t2, $t3, $t4
                add $t2, $t2, $t5
                add $t2, $t2, $t6

                sw $t2, 12($t0)

                addi $t0, $t0, 4
                addi $t1, $t1, -1
                bgtz $t1, loop
        li $v0, 10
        syscall
```

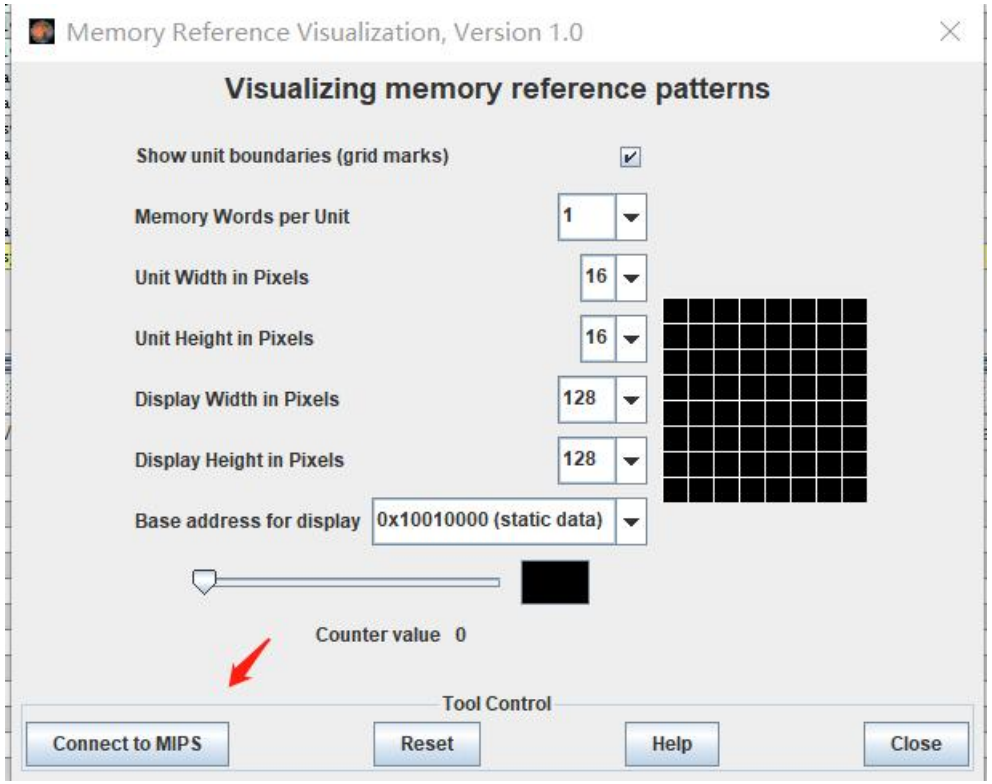
Q. Which demo would lead to more times to access the memory unit's neighbors?





5 ‘Memory Reference Visualization’ of Mars





1. open an MIPS source code in Mars, assemble it

2. open the “Memory Reference Viasualization” tool of Mars

3. Click “Connect to MIPS” botton of the tool

4. Run the current MIPS code

The access time on every memory unit would be present as specified color.

Using “Memory Reference Viasualization” to improve your answer of the question on page 4

The usage of cache

Locality

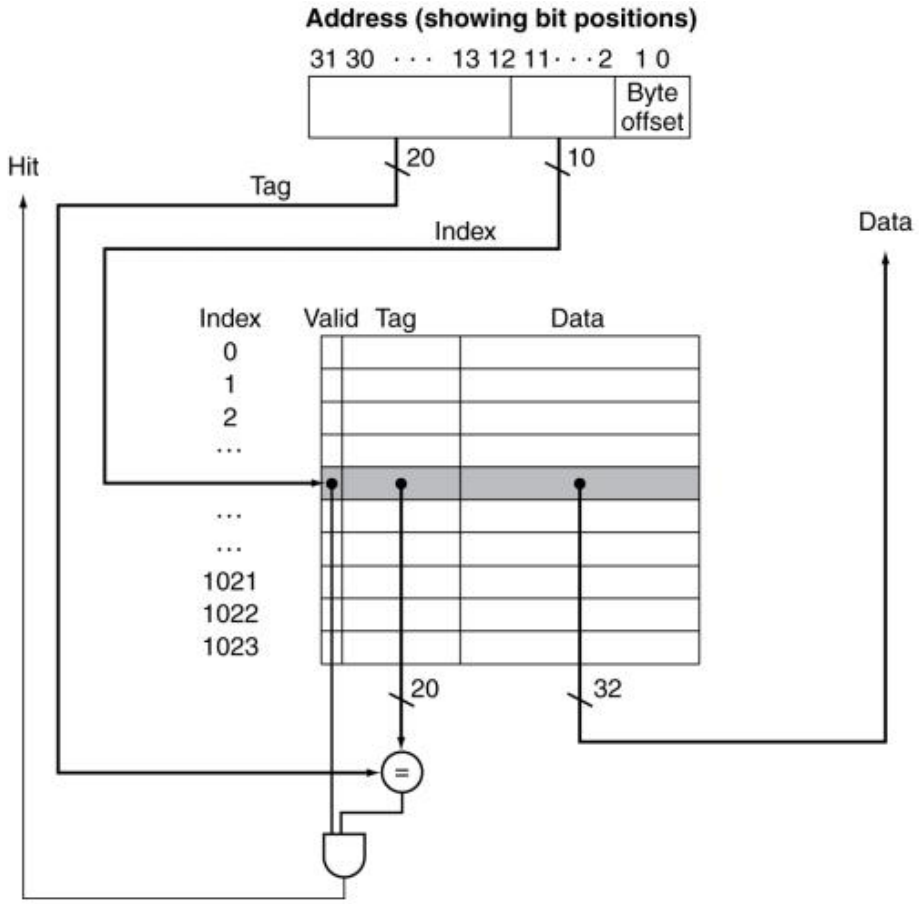
- Why do caches work?
  - Temporal locality: if you used some data recently, you will likely use it again
  - Spatial locality: if you used some data recently, you will likely access its neighbors
- No hierarchy:
  - average access time for data = 300 cycles
- 32KB 1-cycle L1 cache that has a hit rate of 95%:
  - average access time = 0.95 x 1 + 0.05 x (301) = 16 cycles

Direct Mapped Cache

- Direct mapped cache
  - ONE data in MEMORY is mapped to ONLY ONE location in CACHE.
  - Location determined by the ADDRESS:
    - The lower bits of address define the place of the unit(aka block) in the cache.
    - The higher bits are called tag which will be stored in the cache unit(aka cache block).

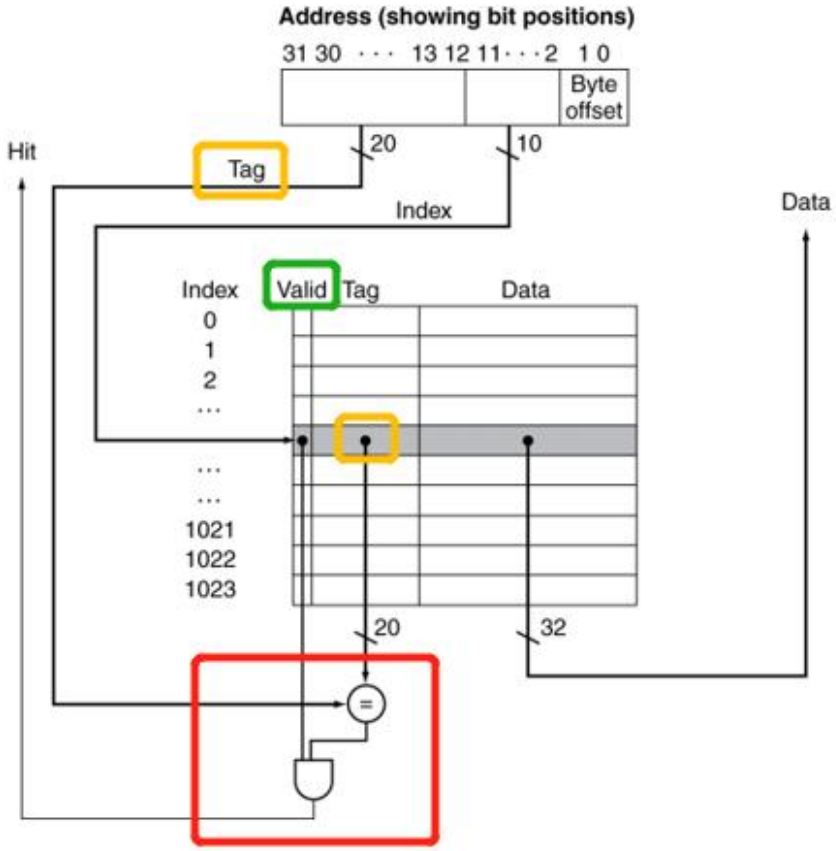
Direct Mapped Cache

- **Componets in Cache**
  - **Valid(1bit)**
    - Initial value is 1’ b0
    - Turns to 1’ b1 while the cache unit has been written which means there has been data in the cache unit
  - **Tag:** Higer bits of address
  - **Data:** Store the data which has been cached



Direct Mapped Cache continued

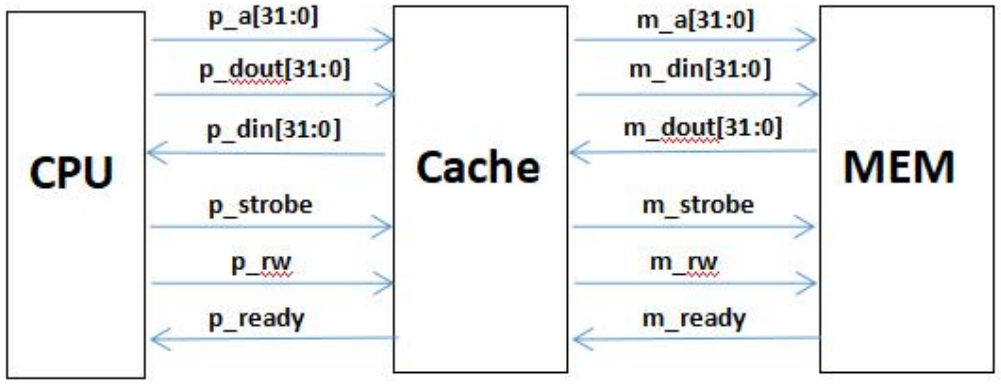
- **Cache Hit** vs **Cache Miss**
  - **Hit**
    - Find the cache unit based on the 'Index' of 'Address'
    - Find the 'valid' and 'Tag' parts from the cache unit
    - If 'valid' is 1 and the 'Tag' is the same as the higer bits of 'Address'.
  - **Miss: If not hit, then miss**
  - **Hit Rate = Hit times / (Hit times + Miss times)**



10 Implement Direct Map Cache(ports)

```
module cache
 #(parameter A_WIDTH=32,  parameter C_INDEX=10, parameter D_WIDTH=32)
(p_a,p_dout,p_din,p_strobe,p_rw,p_ready,clk,resetn,
m_a,m_dout,m_din,m_strobe,m_rw,m_ready);

    input clk, resetn;

    input [A_WIDTH-1:0]   p_a;  //address of memory tobe accessed
    input [D_WIDTH-1:0]   p_dout;          //the data from cpu
    output [D_WIDTH-1:0] p_din;          //the data to cpu
    input   p_strobe;          // 1 means to do the reading or writing
    input   p_rw;              // 0:read, 1:write
    output p_ready;          // tell cpu, outside of cpu is  ready


    output [A_WIDTH-1:0] m_a;     //address of memory tobe accessed
    input [D_WIDTH-1:0]   m_dout;              //the data from memory
    output [D_WIDTH-1:0] m_din;              //the data to memory
    output m_strobe;                  //same as 'p_strobe'
    output m_rw;                      //0:read, 1:write
    input   m_ready;                  //memory is ready
```

Q. What's the relationship between A_WIDTH, **C_INDEX** and D_WIDTH?

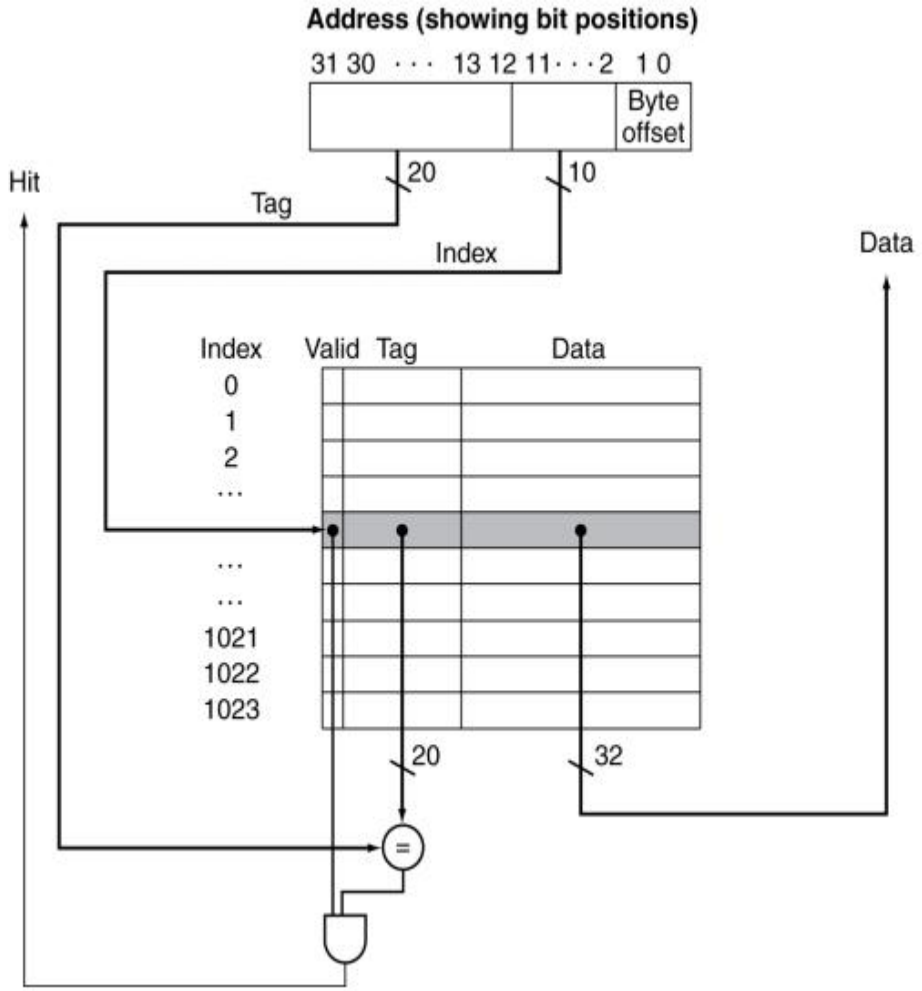


TIPS:
parameter in verilog makes the design more flexiable, which is highly recommended.

Implement Direct Map Cache(componets)

Build the cache, complete the following statement

```
// d_valie is a piece of memory stored the valid info for every block
reg d_valid [ 0 : complete code here  p11-1];

// T_WIDTH is the width of  'Tag'
localparam T_WIDTH = complete code here  p11-2 ;
//d_tags is a piece of memory stored the tag info for every block
reg [T_WIDTH-1: 0] d_tags [0 : complete code here  p11-3];

//d_data is a piece of memory stored the data for every block
reg [D_WIDTH-1:0] d_data [0 : complete code here  p11-4];
```



A_WIDTH : *the width of 'Address', here its value is 32.*

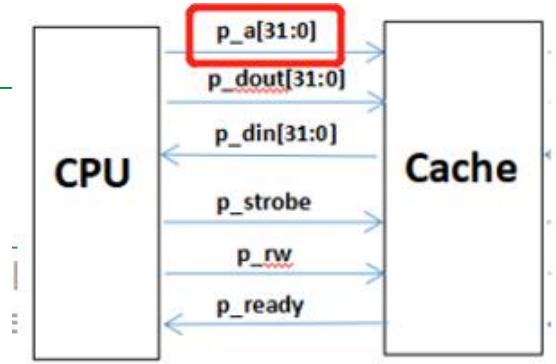
C_INDEX : *the width of 'Index', here its value is 10.*

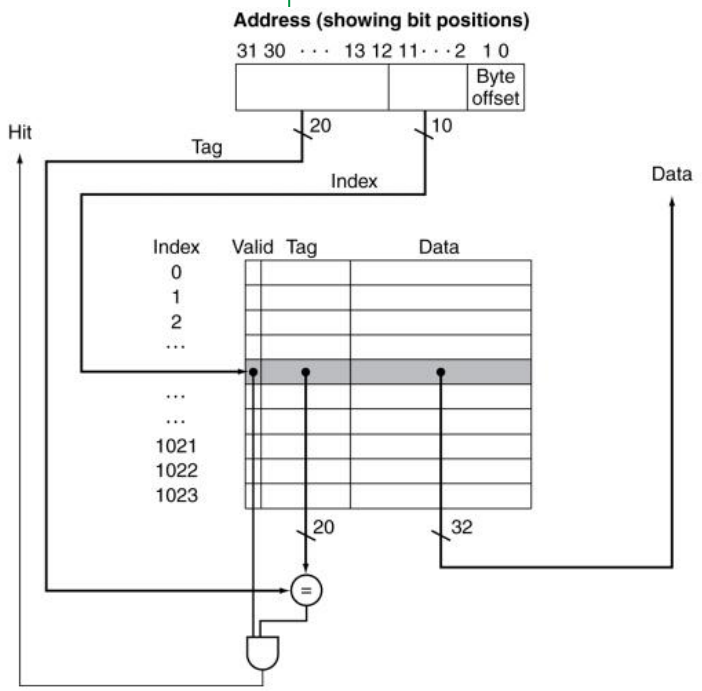
T_WIDTH : *used as the width of 'Tag'.* 'Byte offset' : *2 bit-width*

Implement Direct Map Cache(cache hit vs cache miss)

Complete the following code to implement **cache hit**

```
// d_valie is a piece of memory stored the valid info for every block in cache
// d_tags is a piece of memory stored the tag info for every block in cache
// d_data is a piece of memory stored the data for every block in cache

wire [C_INDEX-1:0] index = p_a[ C_INDEX+1 : 2 ] ;
wire [T_WIDTH-1:0] tag   = p_a[ A_WIDTH-1 :  C_INDEX+2 ] ;

wire valid = d_valid[index];
wire [T_WIDTH-1:0] tagout  = d_tags[index];
wire [D_WIDTH-1:0] c_dout = d_data[index];

//cache control
wire cache_hit =  complete the code here p12
wire cache_miss = ~cache_hit;
```





A_WIDTH : *the width of 'Address', here its value is 32.*

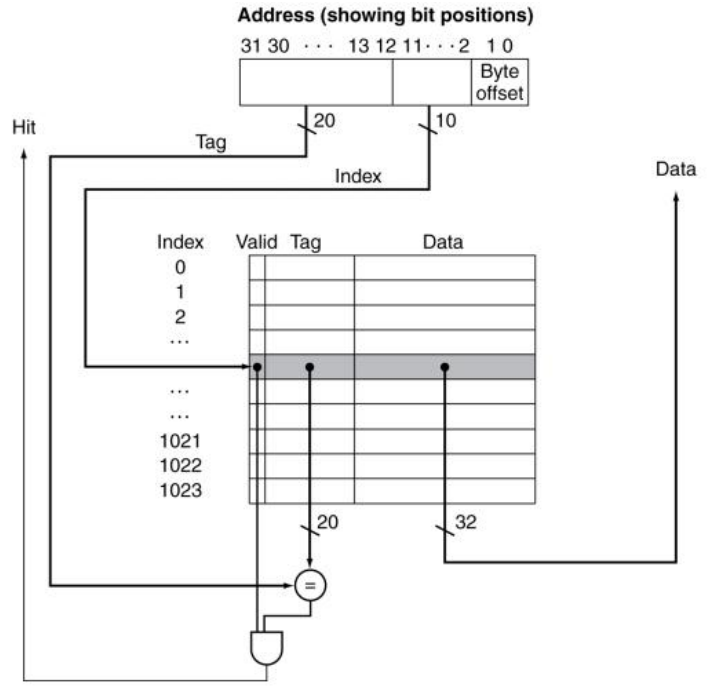
C_INDEX : *the width of 'Index', here its value is 10.*

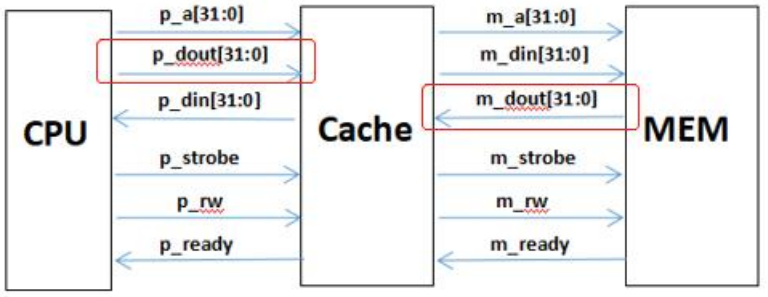
T_WIDTH : *used as the width of 'Tag'.* 'Byte offset' : *2 bit-width*

Implement Direct Map Cache(cache write)

Complete the following code to implement **cache write**

```
wire c_write = p_rw | cache_miss & m_ready ;

always @ (posedge clk, negedge resetn)
        if( resetn ==  complete code here p13-1  ) begin
                integer i;
                for( i=0; i<( complete code here p13-2); i=i+1 )
                        d_valid[i] <=1'b0;
        end
        else if(c_write==1'b1)
                d_valid[index] <=1'b1;

//////////////////////////////////////////////////////////////////////////////////
always @ (posedge clk)
        if(c_write==1'b1)    d_tags[index] <= tag;

//////////////////////////////////////////////////////////////////////////////////
wire sel_in = p_rw ;
wire [D_WIDTH-1:0] c_din =    complete code here p13-3    ;

always @ (posedge clk)
        if(c_write==1'b1)   d_data[index] <= c_din;
```

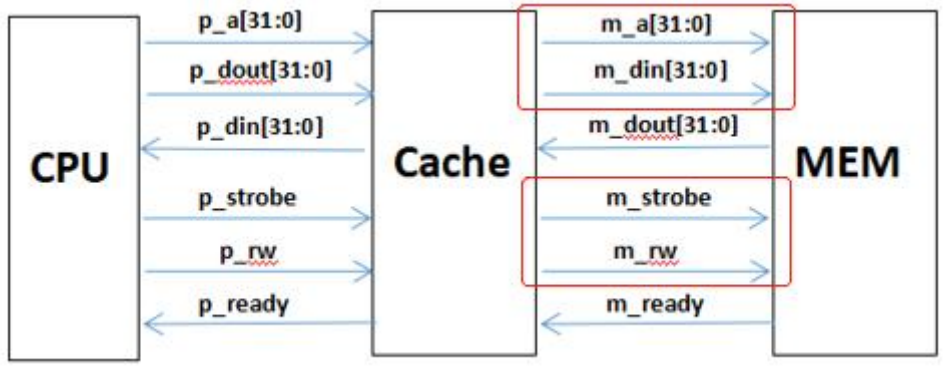




14 Implement Direct Map Cache(memory write)

Complete the following code to implement memory write （ write_through ）

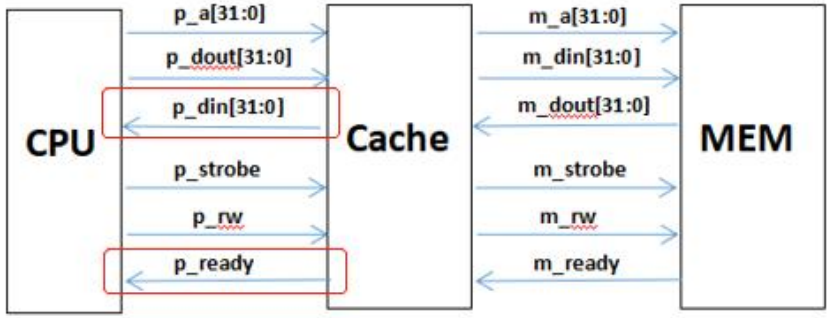
```
// write data (m_din) to the memory unit specified by m_a
        assign m_a = p_a;

        assign m_din = p_dout;

// p_strobe (1 means to do the reading or writing, 0 means else)
// m_strobe (1 means to do the reading or writing, 0 mean else)
// p_rw, m_rw  ( 0:read, 1:write)

        assign m_strobe = p_strobe & (p_rw | cache_miss);

        assign m_rw =  complete code here p14 ;
```



15 Implement Direct Map Cache(CPU read)

Complete the following code to implement CPU read

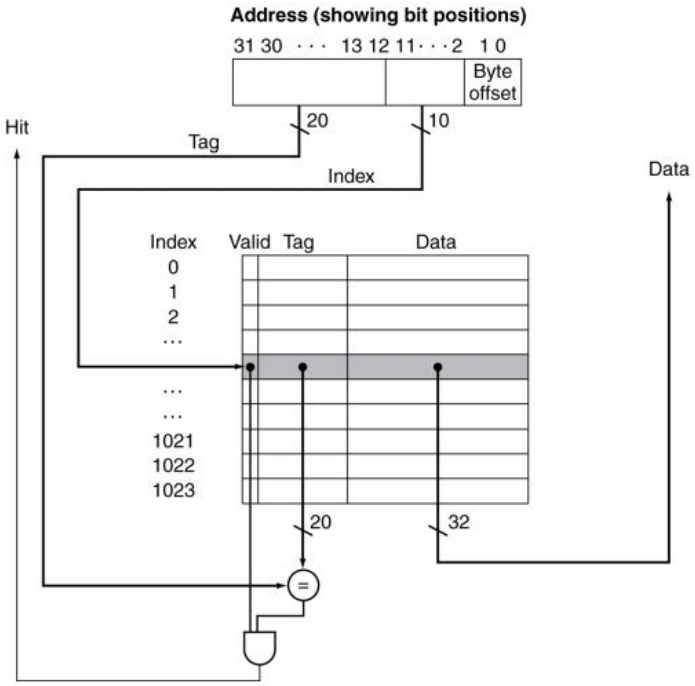
```
//read data to CPU
// if cache hit, read c_dout from cache to p_din, else read m_dout to p_in
// wire [D_WIDTH-1:0] c_dout = d_data[index];
        wire sel_out = cache_hit;

        assign p_din = complete code here p15  ?  c_dout : m_dout;

       assign p_ready = ~p_rw & cache_hit   |   (cache_miss | p_rw)  & m_ready;
```



The Performance

While the " **Address**" for memory is **32bits**(based on Byte)

Using a piece of **Directly Mapped Cache** to improve the performance.



- The size of the **Directly Mapped Cache** is 512 Byte:

512 Byte = 32 Blocks * 4 words/every block * 4 Bytes/every word.

- How many bits are used for the **index** of cache block(aka cache unit)?
- How many bits are used to indentify the **bytes** in a cache block?
- How many bits are left for '**tag**' ?

	Address	Tag	Index	ByteOffset
bit-width	32	?	?	?

The Performance continued

There has been a **Directly Mapped Cache** between the CPU and the Data-memory.

- The " Address" for memory is **32bits(based on Byte)**
- The size of **Directly Mapped Cache** is **512 Byte.**

  512Byte = 32 Blocks * 4 words/every block * 4 Bytes/every word.

The demo on the right hand would be run on the CPU

- What's the **range of 'Address'** on data-memory woudl be accessed for the demo? (List the initial and final values of the 'Address')

- If the **access time that cache hit is 1**, and the **acess time that cache miss is 300+1**, then **what's the average access time for the demo on the right hand after using the Directly Mapped Cache**?

```
.data
        array: .word 1,1,1
        tmp: .word 0 : 100
.text
        la $t0, array
        li $t1, 25
        loop:
                lw $t3, 0($t0)
                lw $t4, 4($t0)
                lw $t5, 8($t0)

                add $t2, $t3, $t4
                add $t2, $t2, $t5

                sw $t2, 12($t0)

                addi $t0, $t0, 16
                addi $t1, $t1, -1
                bgtz $t1, loop

        li $v0, 10
        syscall
```

The Performance continued

- 1st round

  (1) address: **0x10010000,** trying block **0** empty -- **MISS**
  (2) address: **0x10010004,** trying block **0** tag 0x00800800 -- **HIT**
  (3) address: **0x10010008,** trying block **0** tag 0x00800800 -- **HIT**
  (4) address: **0x1001000c** , trying block **0** tag 0x00800800 -- **HIT**

- 2nd round

  (5) address: **0x10010010,** trying block **1** empty -- **MISS**
  (6) address: **0x10010014,** trying block **1** tag 0x00800800 -- **HIT**
  (7) address: **0x10010018,** trying block **1** tag 0x00800800 -- **HIT**
  (8) address: **0x1001001c,** trying block **1** tag 0x00800800 -- **HIT**

- 3rd roud

  (9) address: **0x10010020**, trying block **2** empty -- **MISS**
  (10) address: **0x10010024**, trying block **2** tag 0x00800800 -- **HIT**
  (11) address: **0x10010028**, trying block **2** tag 0x00800800 -- **HIT**
  (12) address: **0x1001002c**, trying block **2** tag 0x00800800 -- **HIT**

- ... There are totally 25 miss and 75 hit in 100 accessing, hit rate is 75%.

```
.data
        array: .word 1,1,1
        tmp: .word 0 : 100
.text
        la $t0, array
        li $t1, 25
        loop:
                lw $t3, 0($t0)
                lw $t4, 4($t0)
                lw $t5, 8($t0)

                add $t2, $t3, $t4
                add $t2, $t2, $t5

                sw $t2, 12($t0)

                addi $t0, $t0, 16
                addi $t1, $t1, -1
                bgtz $t1, loop

        li $v0, 10
        syscall
```

Practices

- 1. Complete the code of the Directly Mapped Cache(page 11-15)

- 2. While using a new cache(as following setting a and b) to replace the previous one(on page16), what's the bit-width of tag? Will the hit rate be higher or lower compared to previous one? Explain why?
  - The size of new cache is 512 Byte
  - a. 512 Byte = 128 Blocks * 1 words/every block * 4 Bytes/every word
  - b. 512 Byte = 1 Block * 128 words/every block * 4 Bytes/every word

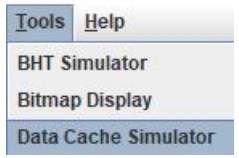
- 3. Make an asm file to calculate 25 consecutive items in fibonacci sequence by using loop and array, answer the quesion 2 again

- 4. While smaller the size of the Directly Mapped Cache to 128 Byte, answer the question 2 again.
  - a. 128 Byte = 32 Blocks * 1 words/every block * 4 Bytes/every word
  - b. 128 Byte = 1 Block * 32 words/every block * 4 Bytes/every word

- Tips: 'Data Cache Simulator' of Mars could help you to check if your analysis and answer is correct or not. More information could be found on page20.

TIPS: 'Data Cache Simulator' of Mars

- 1. Open an assembly source file in Mars(a Simulator on MIPS)
- 2. Assemble this file.
- 3. Open 'Data Cache Simulator' of 'Tools'



- 4. Click 'Connect to MIPS' in left bottom of 'Data Cache Simulator'
- 5. Run the current program