Computer Organization

Lab13 Cache Implemention and Test

2021 Spring term

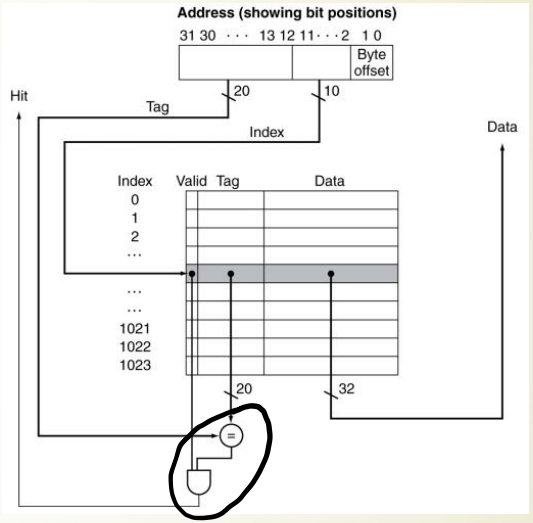
wangw6@sustech.edu.cn

Topics

- Cache
  - Direct Mapped Cache
  - Implement the Direct Mapped Cache in Verilog
  - Test the Performance of Cache by Simulation

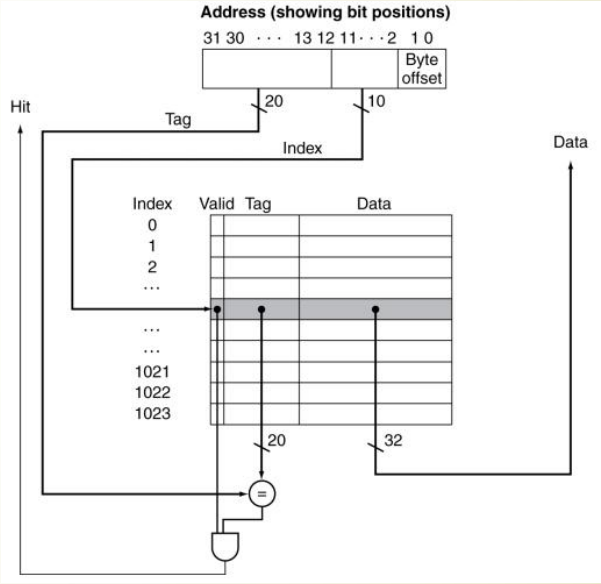
Direct Mapped Cache

- Direct mapped cache：
  - One data in memory is mapped to only one location in cache.
  - Location determined by the address in cache.
  - The lower bits define the address 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).



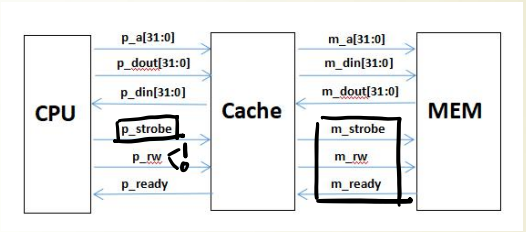
Direct Mapped Cache continued

- **Componets in Cache**
  - **Valid**
    - Initial value is **0**
    - Turns to **1** while the cache unit has been written which means there has been data in the cache item
  - **Tag**: Higer bits of address
  - **Data**: Store the data which has been cached
- **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)**



Implement Direct Map Cache continued

```
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
       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
```



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

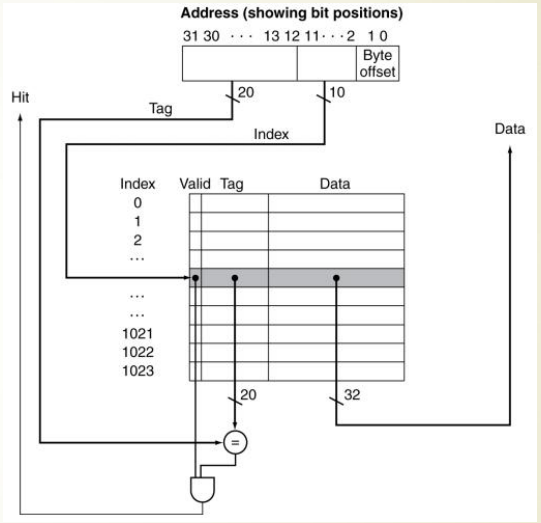
Implement Direct Map Cache continued

A_WIDTH is the width of 'Address', here its value is 32.
C_INDEX is the width of 'Index', here its value is 10.
T_WIDTH is used as the width of 'Tag'.
The width of 'Byte offset' is 2, complete the following statement:

```
localparam T_WIDTH = ? ;

// d_valie is a piece of memory stored the valid info for every block
reg d_valid [ 0 : (1<<C_INDEX)-1];

//d_tags is a piece of memory stored the tag info for every block
reg [ ? : 0] d_tags [0 : (1<<C_INDEX)-1];

//d_data is a piece of memory stored the data for every block
reg [D_WIDTH-1:0] d_data [0 : (1<<C_INDEX)-1];
```

A_width-2-C_index.

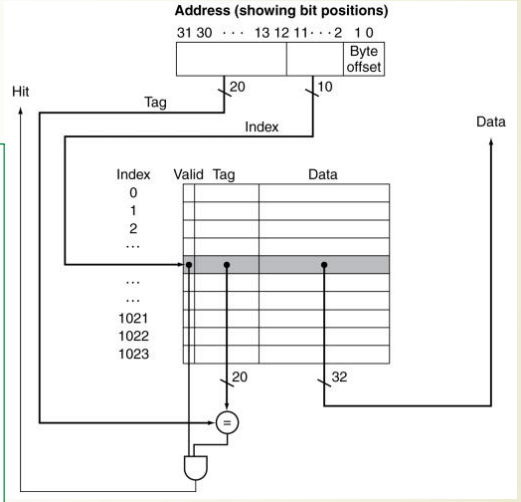
$2^{C\_index}-1$



Implement Direct Map Cache continued

A_WIDTH is the width of 'Address', here its value is 32.
C_INDEX is the width of 'Index', here its value is 10.
T_WIDTH is used as the width of 'Tag'.
The width of 'Byte offset' is 2, complete the following statement:

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

//read from 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];
```



Implement Direct Map Cache continued

```
//cache control
    wire cache_hit = valid & (tagout == tag);
    wire cache_miss = ~cache_hit;

    assign m_din = p_dout;
    assign m_a = p_a;

    assign m_rw = p_strobe & p_rw;  //write_through
    wire c_write = p_rw | cache_miss & m_ready;

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

// if cpu write, write p_dout to cache, if cpu read, write m_dout to cache
    wire sel_in = p_rw;
    wire [D_WIDTH-1:0] c_din = sel_in ? p_dout : m_dout;

// if cache hit, read c_dout from cache to p_din, else read m_dout to p_in
    wire sel_out = cache_hit;
    assign p_din = sel_out? c_dout:m_dout;
```

: "&"优先级高于"|"

```
//write to cache
    always @ (posedge clk, negedge resetn)
        if(resetn == 1'b0) begin
            integer i;
            for(i=0;i<(1<<C_INDEX);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) begin
            d_tags[index] <= tag;
            d_data[index] <= c_din;
        end
```

Performance

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

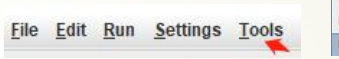
Testing Performance

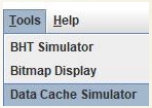
- For the demo on the right hand, how many times of reading data from memory, how many times of writing data to memory?
- Is there any temporal locality here? how does it happen?
- Is there any spatial locality here? where does it happen?

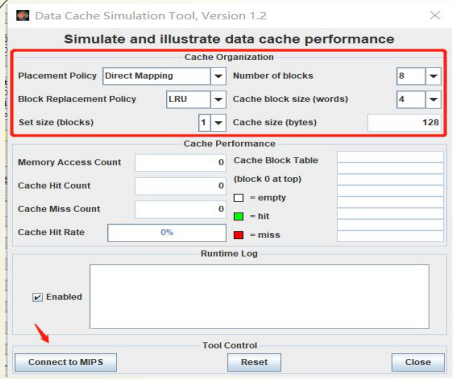
```
.data
        array: .word 1,1,1
        tmp: .word 0 : 100
.text
        la $t0, array
        li $t1, 25
        loop:
                lw $t3, 0($t0)
                lw $t4, 4($t0)
                lw $t5, 8($t0)
                add $t2, $t3, $t4
                add $t2, $t2, $t5
                sw $t2, 12($t0)
                addi $t0, $t0, 16
                addi $t1, $t1, -1
                bgtz $t1, loop
        li $v0, 10
        syscall
```

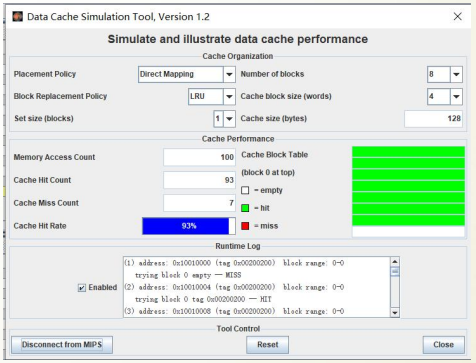
Using 'Data Cache Simulator' of Mars

- **Open** an assembly source file in Mars(a Simulator on MIPS)
- **Assemble** this file.
- Open '**Data Cache Simulator**' of '**Tools**'
- Click '**Connect to MIPS**' in left bottom of '**Data Cache Simulator**'
- **Run** the current program









https://sakai.sustech.edu.cn/portal/site/d50211ea-1586-4344-9d92-a6c42eb7f4e0/tool/b47e4c13-4220-4f61-b881-a211abee2a96?panel=Main

Testing Performance continued

- When the "Address" for memory is 32bits(based on Byte), using a piece of directly mapped cache to improve the performance.

- The size of cache is 128 Byte

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

  - What's the range of 'Address' on memory for the demo on th right hand? list the initial and final values of the 'Address'?
    0x1001_0000, 0x1001_006c
  - How many bits are used for the index of cache block(aka cache unit)? 5
  - How many bits are used to indentify the bytes in a cache block? 4
  - How many bits are left for 'tag' ? 23

- 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 this cache? 76

```
.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
```

Testing 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. Answer the question on Page6, what's the value of T_WIDTH?
- 2. While using a new cache(as following setting a and b) to replace the previous one, what's the 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. If exchange the lw and sw, will the hit rate change?
- 4. Make an asm file to calculate 25 consecutive items in fibonacci sequence by using loop and array, answer the quesion 2 again.

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 page11.