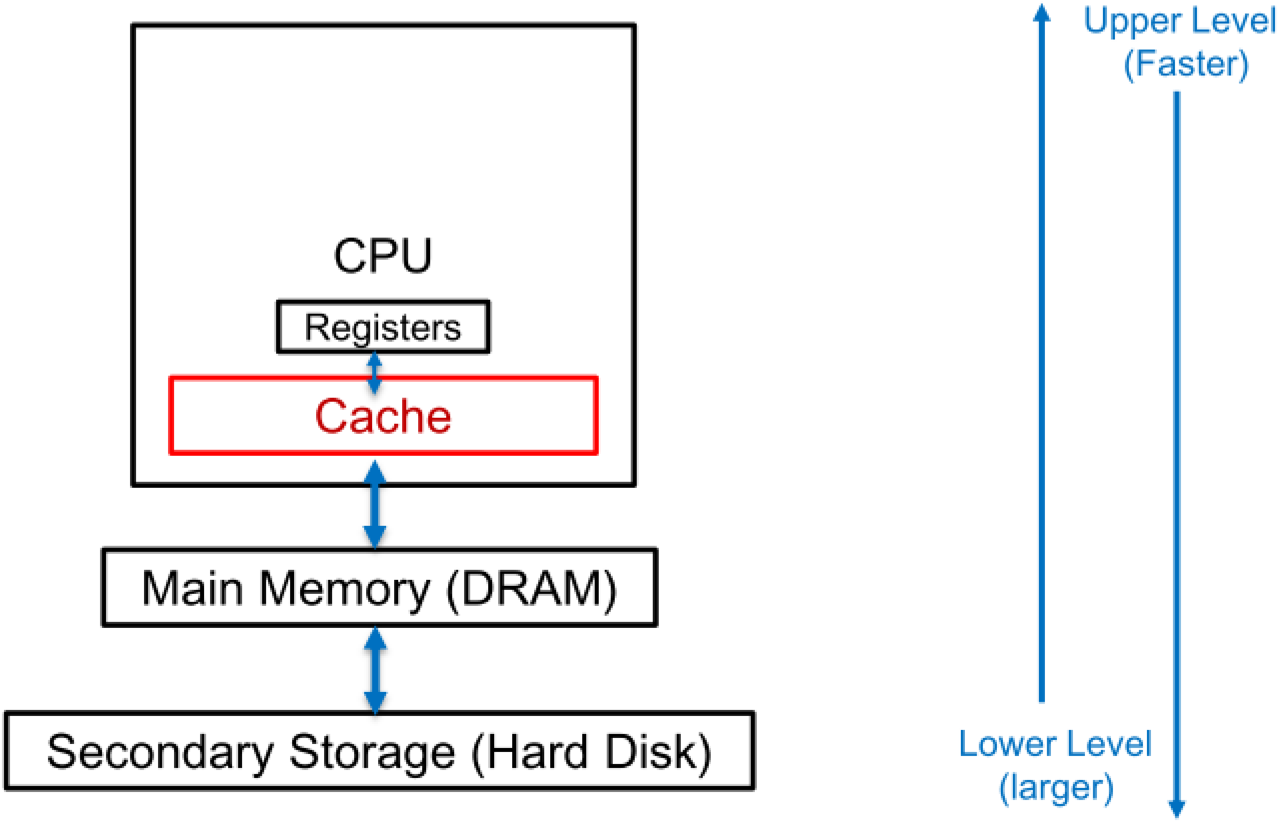
Verilog Implementation Of I-Cache

實驗目的

1. 認識MIPS CPU的Memory Hierarchy
2. 學習Verilog module之應用
3. 了解I-Cache的架構與行為

Levels of Memory Hierarchy



CPU

Registers

Cache

Main Memory (DRAM)

Secondary Storage (Hard Disk)

Upper Level (Faster)

Lower Level (larger)

The Principle of Locality

- Temporal Locality (Locality in Time):

  If an item is referenced, it will tend to be referenced again soon.

  (Example: loop, reuse)

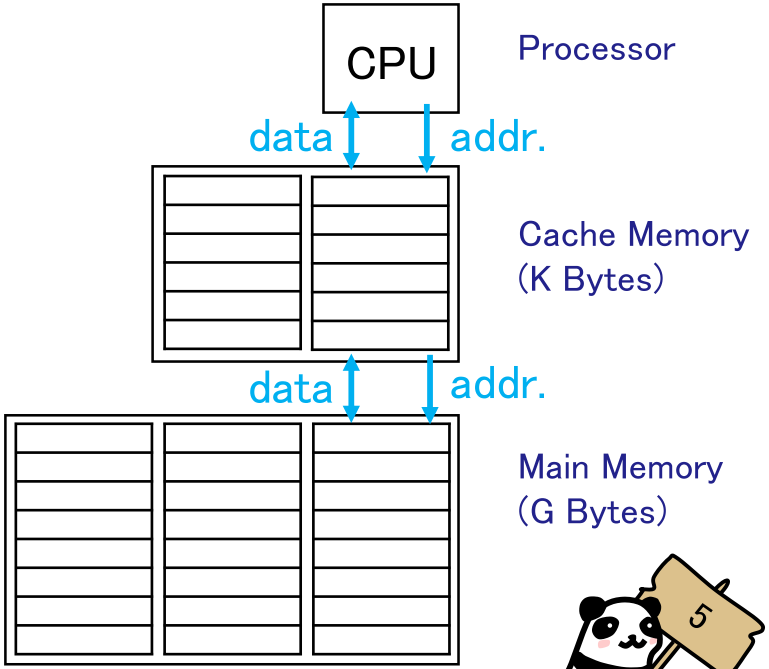
- Spatial Locality (Locality in space):

  If an item is referenced, items whose addresses are close by tend to be referenced soon.

  (Example: array access, straight line code)

Introduction to Cache System

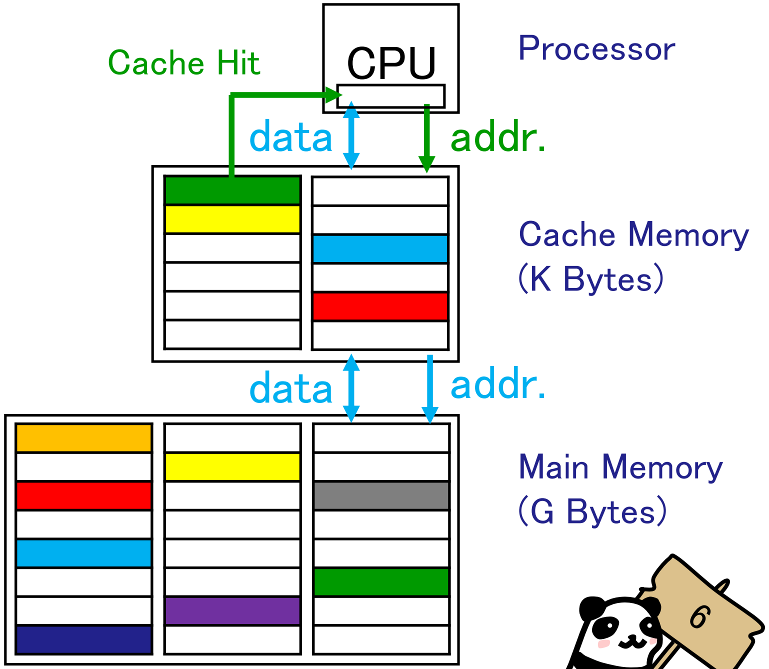
- Insert between CPU and Main Memory
- Implemented with fast Static Ram
- Holds some of a program’s
  - Data (D-Cache)
  - Instructions (I-Cache)



Cache Operation

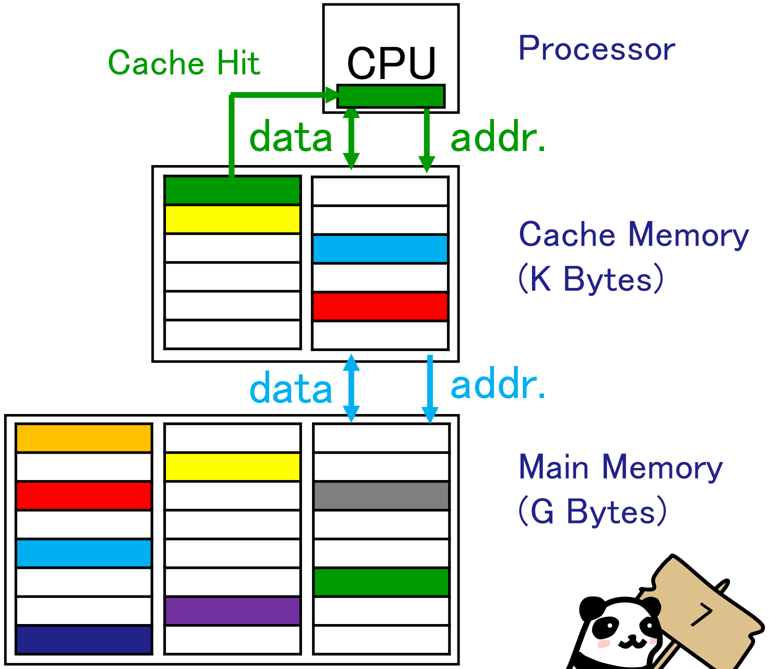
- CPU send an address to Cache
- Hit : Data in Cache (no penalty)
- Miss: Data not in Cache

  (miss penalty)



Cache Operation

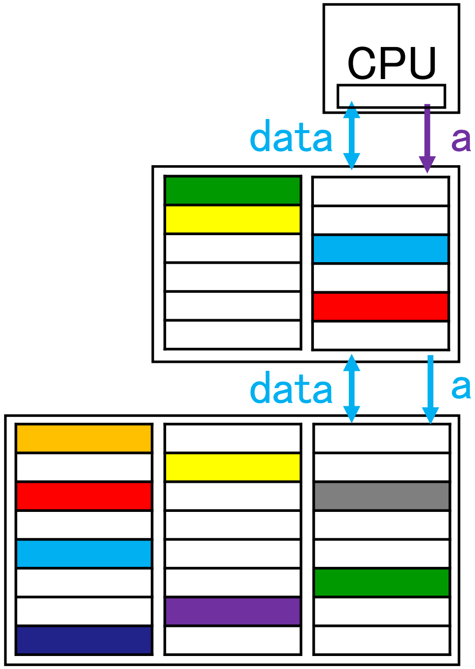
- CPU send an address to Cache
- Hit : Data in Cache (no penalty)
- Miss: Data not in Cache
  (miss penalty)



Cache Operation

- CPU send an address to Cache
- Hit : Data in Cache (no penalty)
- Miss: Data not in Cache

  (miss penalty)



Processor

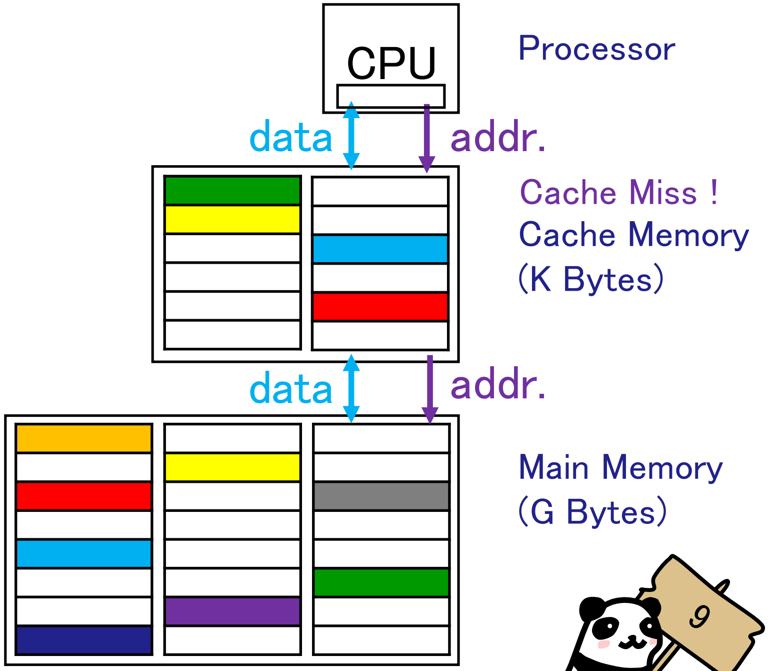
Cache Miss !
Cache Memory
(K Bytes)

Main Memory
(G Bytes)

Cache Operation

- CPU send an address to Cache
- Hit : Data in Cache (no penalty)
- Miss: Data not in Cache

  (miss penalty)



CPU

Processor

data

addr.

Cache Miss !

Cache Memory

(K Bytes)

data

addr.

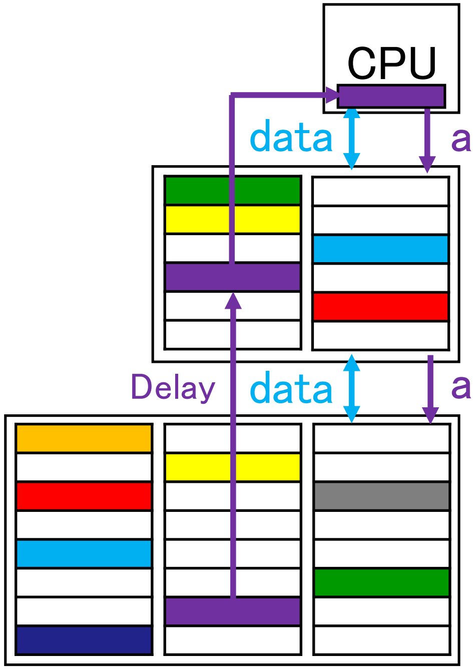
Main Memory

(G Bytes)

Cache Operation

- CPU send an address to Cache
- Hit : Data in Cache (no penalty)
- Miss: Data not in Cache

  (miss penalty)



Processor

Cache Miss !
Cache Memory
(K Bytes)

Main Memory
(G Bytes)

Cache Performance

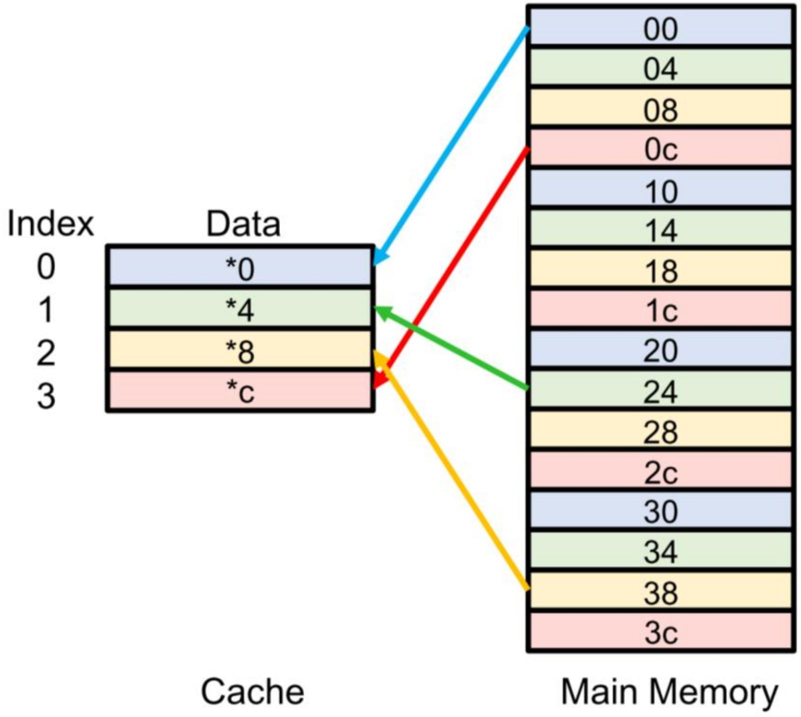
- Hit Rate: Fraction of hit in Cache

  Miss Rate = 1 – (Hit Rate)

- Hit Time: Time to access Cache
- Miss Penalty: Time to replace a block from lower level
- Average memory-access time (AMAT)

  = Hit Time + Miss rate x Miss penalty

Direct Mapped Block Placement



Index	Data
0	*0
1	*4
2	*8
3	*c

Cache

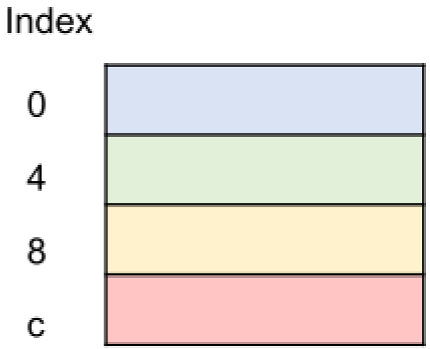
00
04
08
0c
10
14
18
1c
20
24
28
2c
30
34
38
3c

Main Memory

Direct Mapped Block Placement

The sequence of memory access: 00, 04, 08, 0c, 10

Memory Block	Hit/Miss

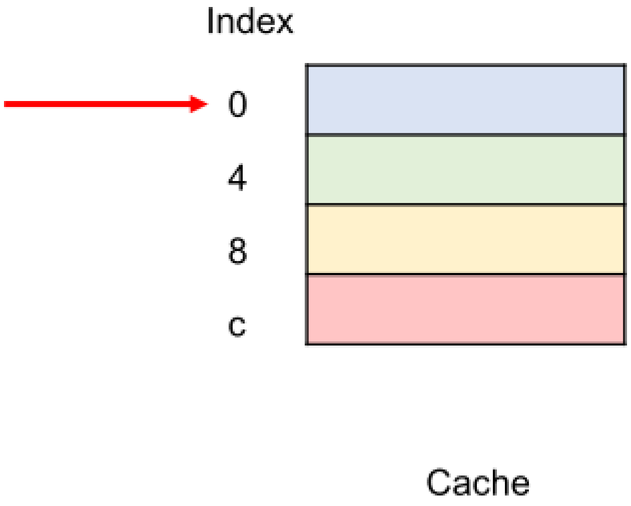


Cache

Direct Mapped Block Placement

The sequence of memory access: 00, 04, 08, 0c, 10

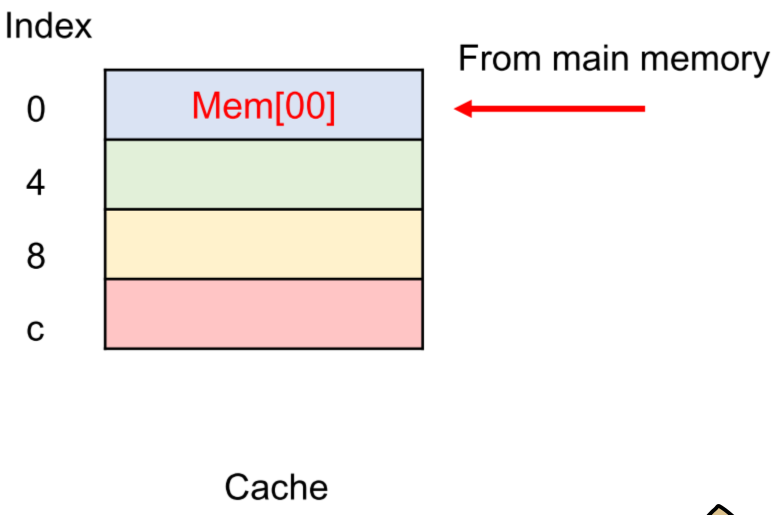
Memory Block	Hit/Miss
00	Miss



Direct Mapped Block Placement

The sequence of memory access: 00, 04, 08, 0c, 10

Memory Block	Hit/Miss
00	Miss



Index	
0	Mem[00]
4	
8	
c	

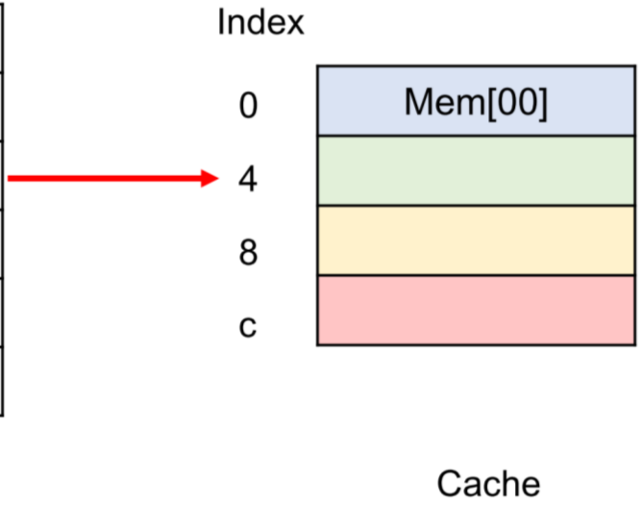
From main memory

Cache

Direct Mapped Block Placement

The sequence of memory access: 00, 04, 08, 0c, 10

Memory Block	Hit/Miss
00	Miss
04	Miss



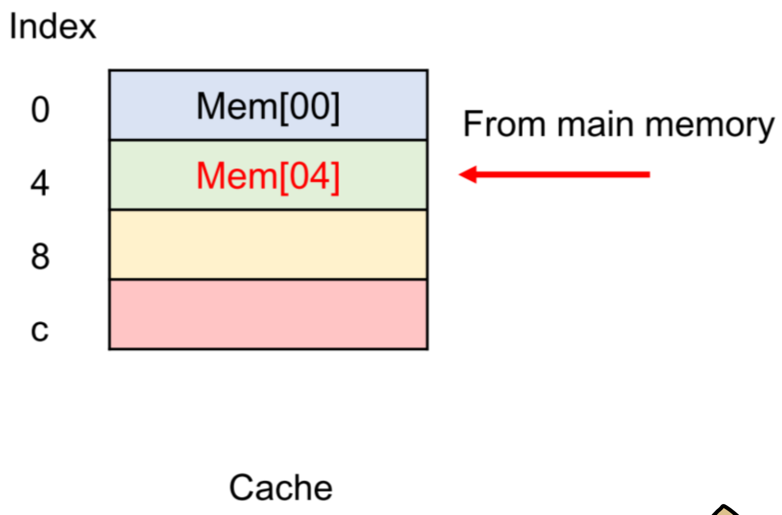
Index	
0	Mem[00]
4	
8	
c	

Cache

Direct Mapped Block Placement

The sequence of memory access: 00, 04, 08, 0c, 10

Memory Block	Hit/Miss
00	Miss
04	Miss



Index	
0	Mem[00]
4	Mem[04]
8	
c	

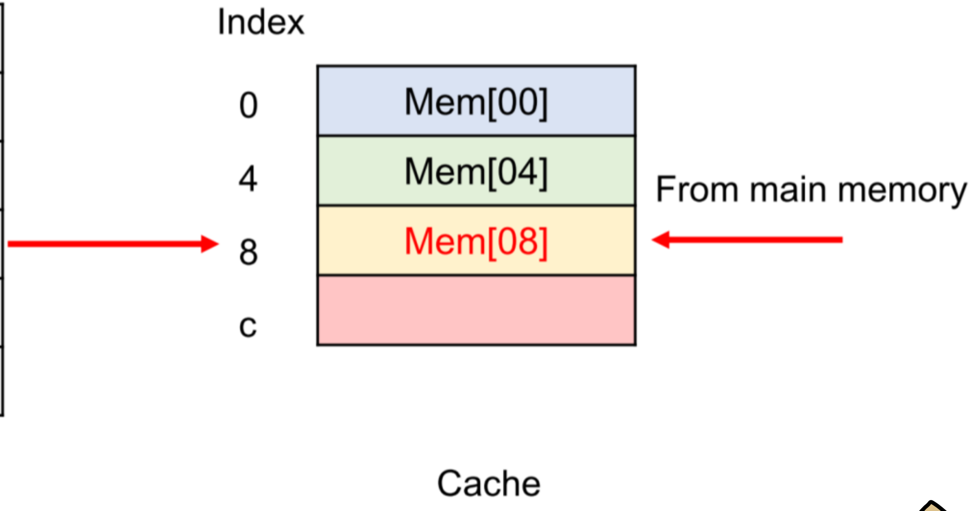
From main memory

Cache

Direct Mapped Block Placement

The sequence of memory access: 00, 04, 08, 0c, 10

Memory Block	Hit/Miss
00	Miss
04	Miss
08	Miss



Index

0	Mem[00]
4	Mem[04]
8	Mem[08]
c	

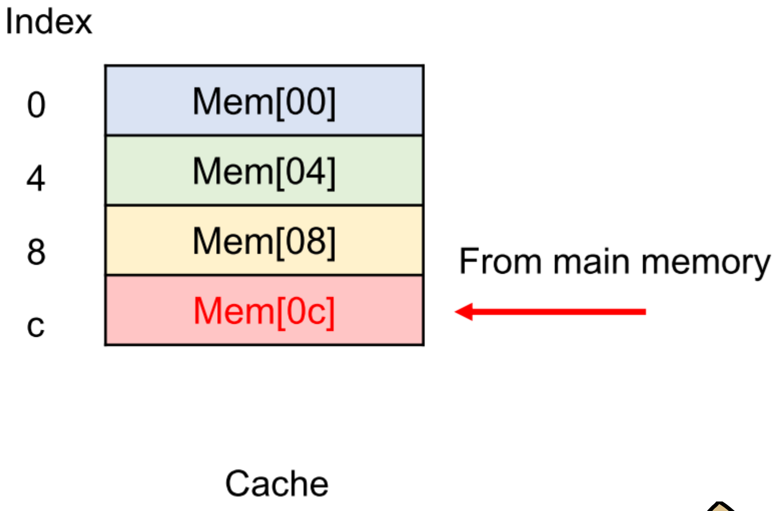
From main memory

Cache

Direct Mapped Block Placement

The sequence of memory access: 00, 04, 08, 0c, 10

Memory Block	Hit/Miss
00	Miss
04	Miss
08	Miss
0c	Miss



Index	
0	Mem[00]
4	Mem[04]
8	Mem[08]
c	Mem[0c]

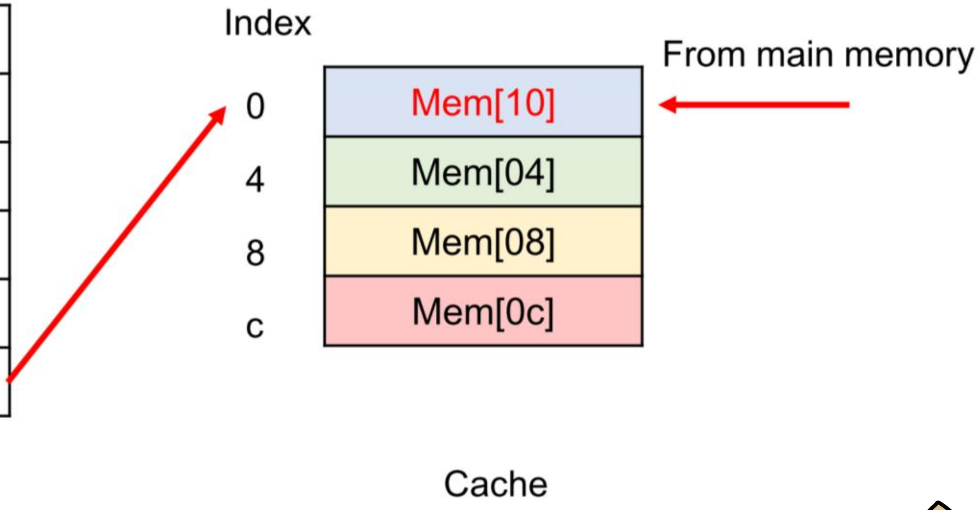
From main memory

Cache

Direct Mapped Block Placement

The sequence of memory access: 00, 04, 08, 0c, 10

Memory Block	Hit/Miss
00	Miss
04	Miss
08	Miss
0c	Miss
10	Miss



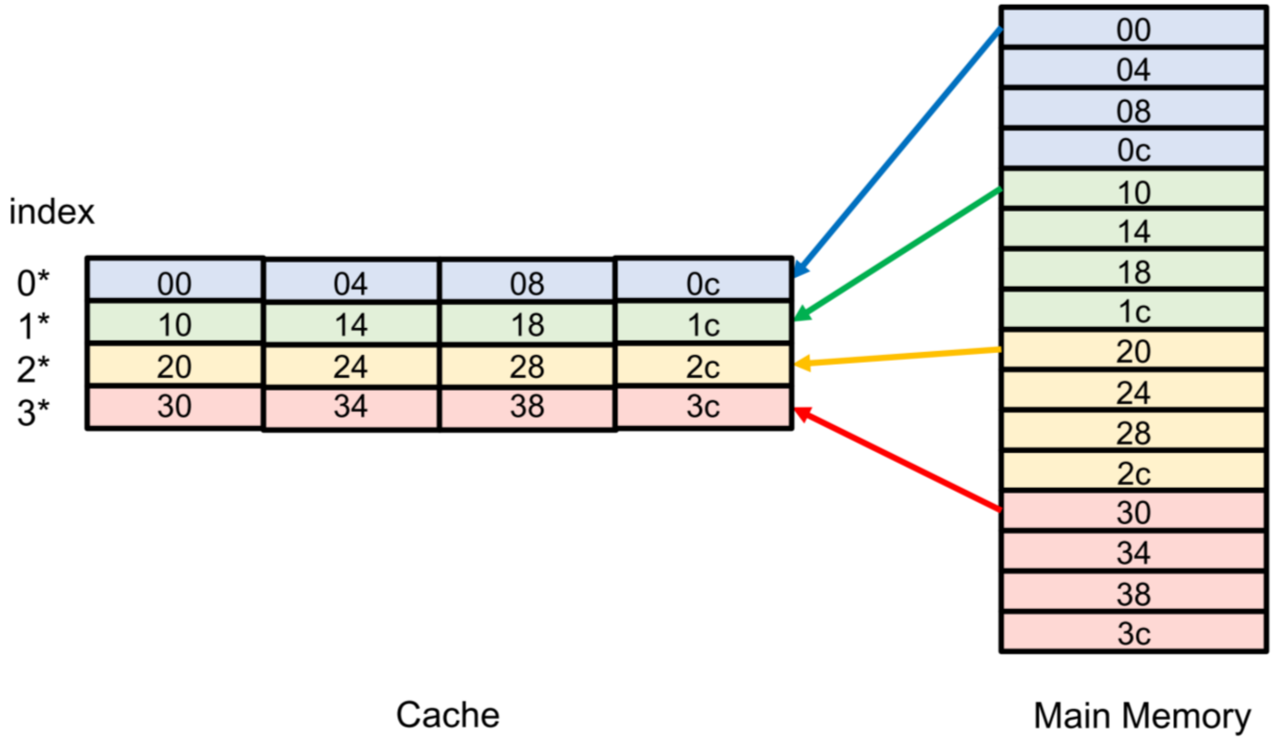
Index	Cache
0	Mem[10]
4	Mem[04]
8	Mem[08]
c	Mem[0c]

From main memory

Cache Line Refill

Load in multi-data when cache miss

Cache Line Refill



index				
0*	00	04	08	0c
1*	10	14	18	1c
2*	20	24	28	2c
3*	30	34	38	3c

Cache

00
04
08
0c
10
14
18
1c
20
24
28
2c
30
34
38
3c

Main Memory

Line Refill Example

The sequence of memory access: 00, 04, 08, 0c, 10

Mem Block	Hit/Miss

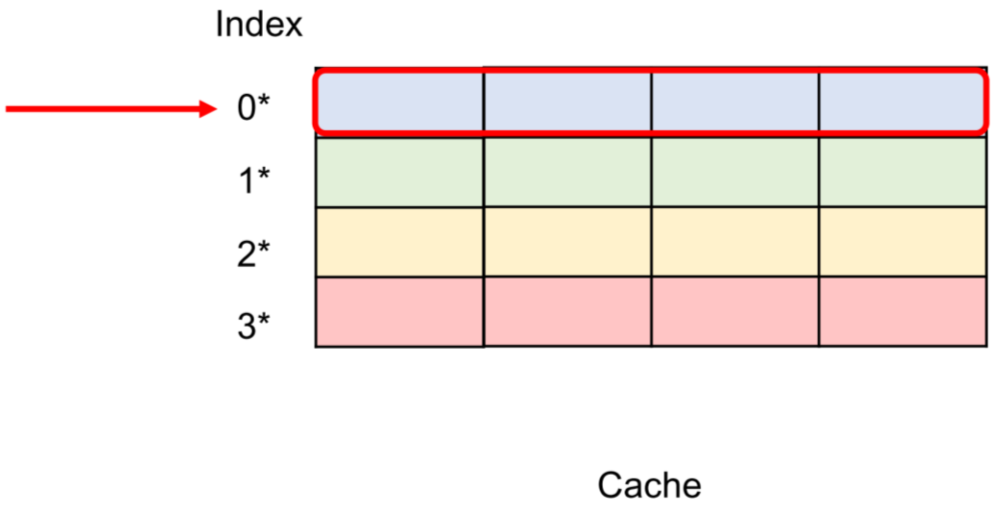
Index				
0*				
1*				
2*				
3*				

Cache

Line Refill Example

The sequence of memory access: 00, 04, 08, 0c, 10

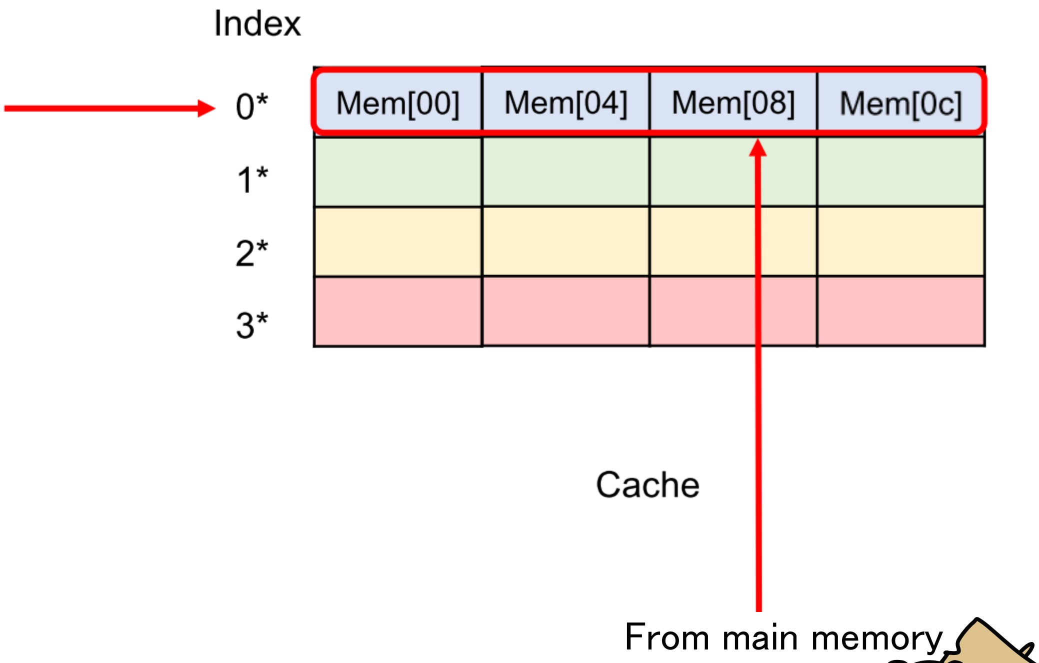
Mem Block	Hit/Miss
00	Miss



Line Refill Example

The sequence of memory access: 00, 04, 08, 0c, 10

Mem Block	Hit/Miss
00	Miss



Index				
0*	Mem[00]	Mem[04]	Mem[08]	Mem[0c]
1*				
2*				
3*				

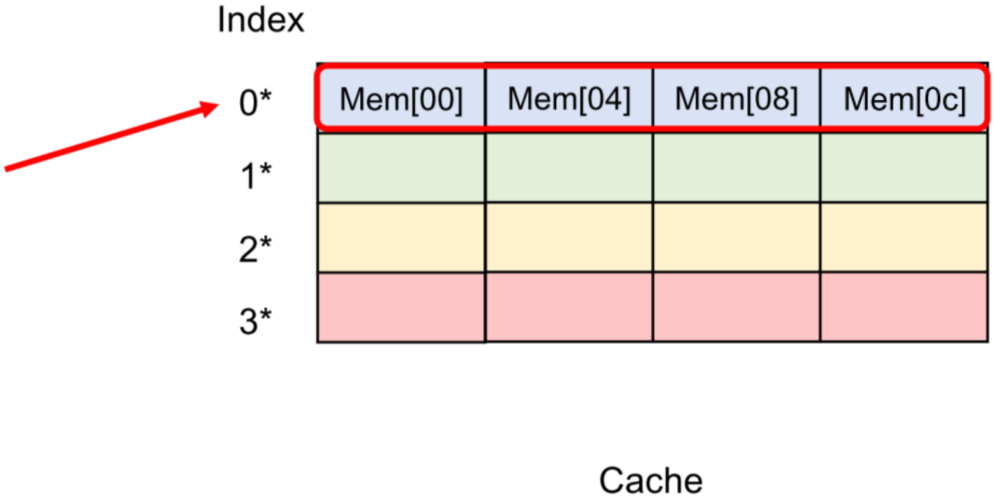
Cache

From main memory

Line Refill Example

The sequence of memory access: 00, 04, 08, 0c, 10

Mem Block	Hit/Miss
00	Miss
04	Hit



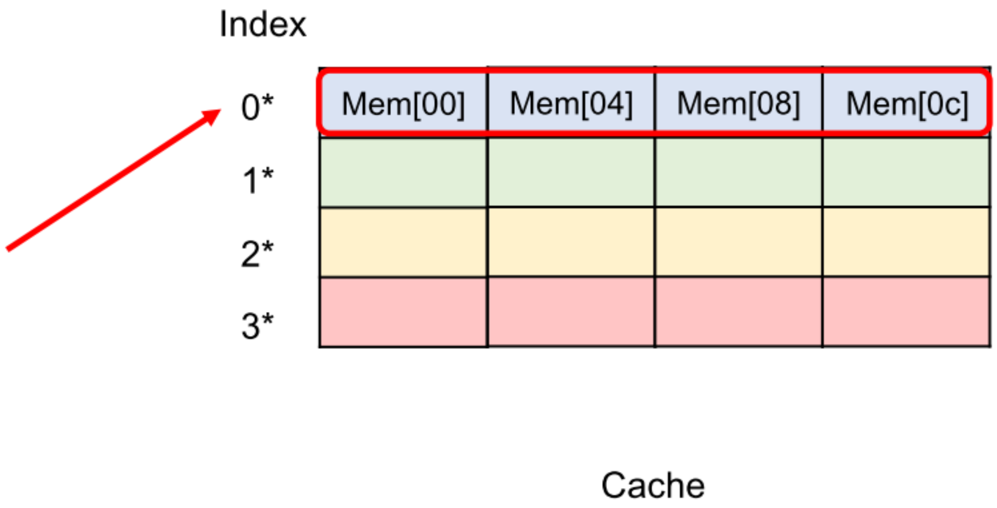
Index				
0*	Mem[00]	Mem[04]	Mem[08]	Mem[0c]
1*				
2*				
3*				

Cache

Line Refill Example

The sequence of memory access: 00, 04, 08, 0c, 10

Mem Block	Hit/Miss
00	Miss
04	Hit
08	Hit

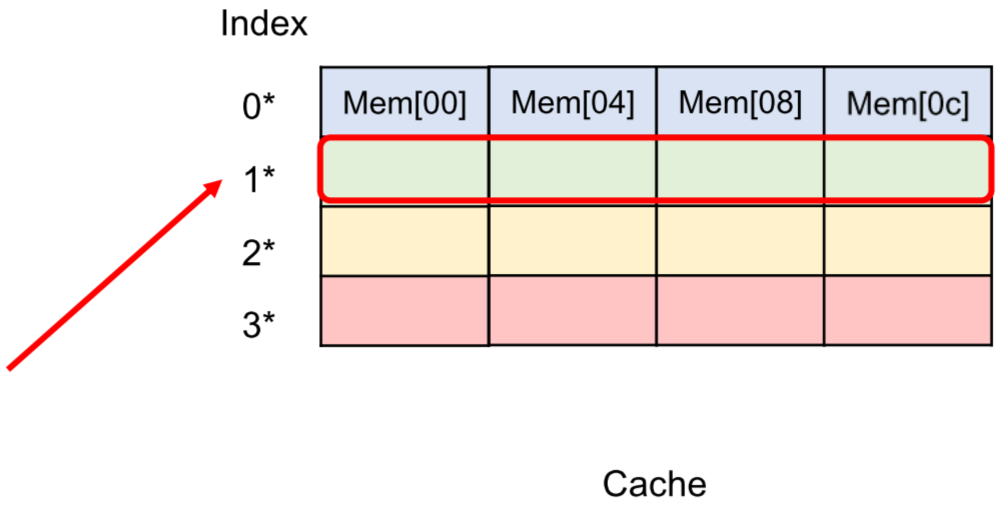


Index				
0*	Mem[00]	Mem[04]	Mem[08]	Mem[0c]
1*				
2*				
3*				

Line Refill Example

The sequence of memory access: 00, 04, 08, 0c, 10

Mem Block	Hit/Miss
00	Miss
04	Hit
08	Hit
0c	Hit
10	Miss



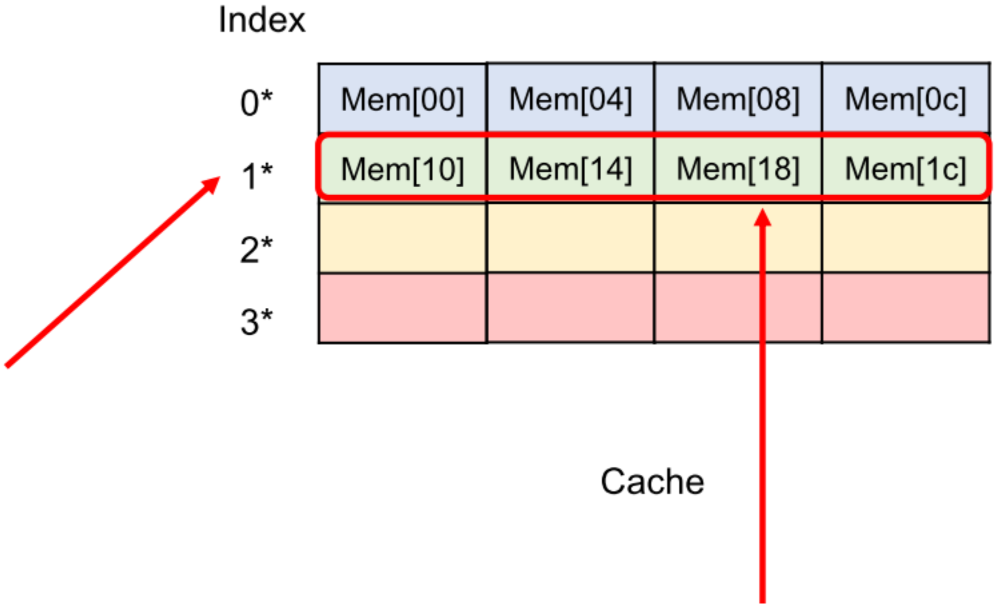
Index				
0*	Mem[00]	Mem[04]	Mem[08]	Mem[0c]
1*				
2*				
3*				

Cache

Line Refill Example

The sequence of memory access: 00, 04, 08, 0c, 10

Mem Block	Hit/Miss
00	Miss
04	Hit
08	Hit
0c	Hit
10	Miss



Index				
0*	Mem[00]	Mem[04]	Mem[08]	Mem[0c]
1*	Mem[10]	Mem[14]	Mem[18]	Mem[1c]
2*				
3*				

Cache

From main memory

Address

Block Address (32-bit)		
Tag	Index	00

Index: Decide which entry of cache should be access

Tag : Check if the data in the cache is hit or not

Block Address (32-bit)			
Tag	Index	line	00

Index : Decide which entry of cache should be access

Tag : Check if the data in the cache is hit or not

line : Decide which data of entry should be output

實作題目演練

Tool used

實驗環境:

1. Modelsim (Run CPU simulator)
    - 看wave來驗證我們的實作是否正確

Modelsim驗證教學

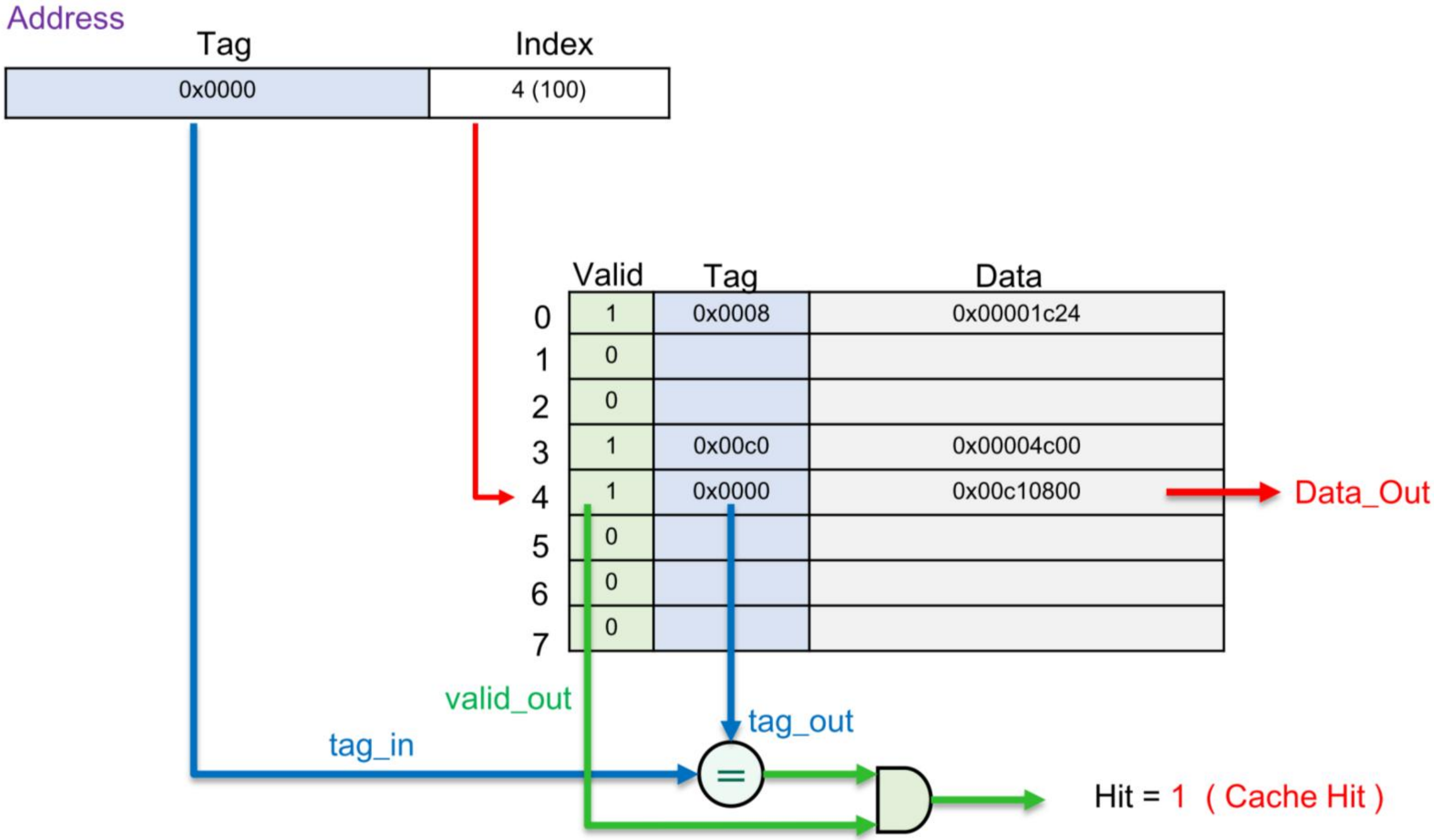
請參考LAB1第33頁投影片

實作一

以RTL code完成這顆MIPS CPU中I_Cache.v空白部分,總共有**4個**部分

使用ModelSim編譯完成,並且完成驗證。

Cache Architecture



Address

Tag	Index
0x0000	4 (100)

	Valid	Tag	Data
0	1	0x0008	0x00001c24
1	0		
2	0		
3	1	0x00c0	0x00004c00
4	1	0x0000	0x00c10800
5	0		
6	0		
7	0		

Data_Out

valid_out

tag_in

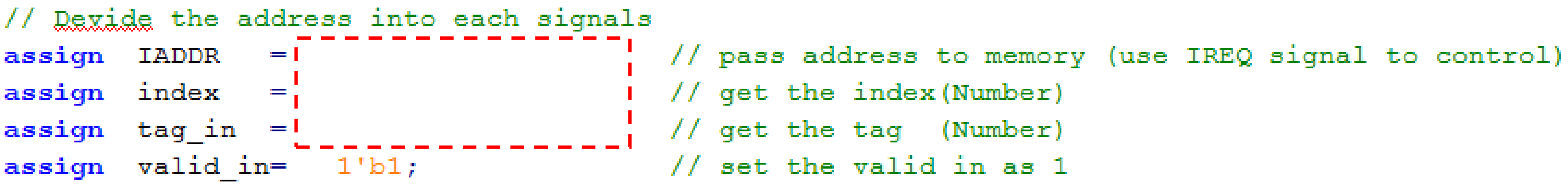
tag_out

=

Hit = 1 (Cache Hit)

(tag_in = tag_out) & valid_out =1

Part 1



```
// Devide the address into each signals
assign  IADDR   =                      // pass address to memory (use IREQ signal to control)
assign  index   =                      // get the index(Number)
assign  tag_in  =                      // get the tag  (Number)
assign  valid_in=   1'b1;              // set the valid in as 1
```

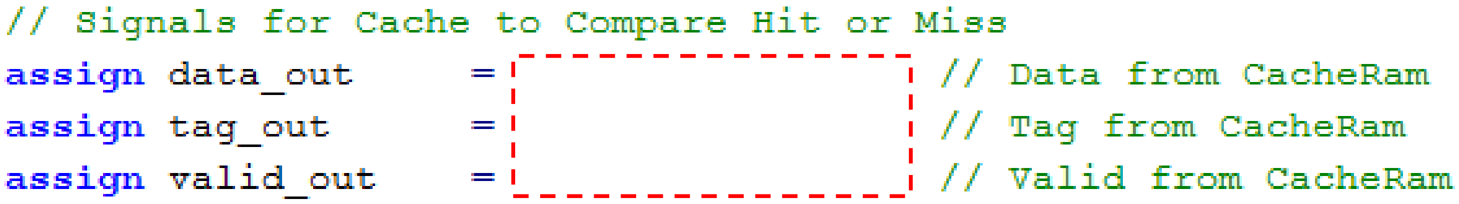
This is a “Direct Mapped I-Cache” with 256 entries each has a 32 bits data.
ADDR [31:0] is a 32-bit signal (address) from CPU to Cache

1. First, you have to pass the ADDR to IADDR. (send addr. to memory if cache miss)
2. The cache has 256 entries which means that your index should be __ bits.
3. Once you know how many bits index and tag_in need, get index & tag_in from ADDR.

31		0
tag_in	index	00

請同學根據前面的內容決定 index, tag_in 分別需要幾個 bit,
並且寫出二個訊號分別來自ADDR當中的哪一段

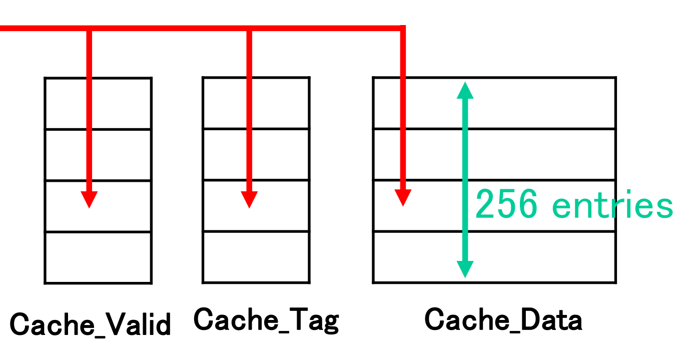
Part 2



```
// Signals for Cache to Compare Hit or Miss
assign data_out     =                    // Data from CacheRam
assign tag_out      =                    // Tag from CacheRam
assign valid_out    =                    // Valid from CacheRam
```

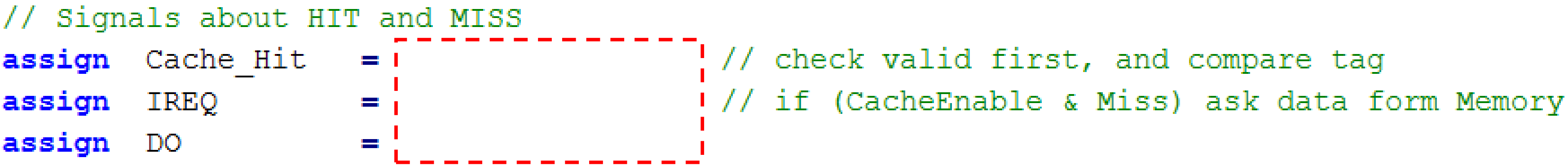
In part2, we have to prepare data_out, tag_out and valid_out signals.
And, we need the signal “index” which we have prepared in part1 to help us.

1. Use index to get data_out from cache memory
2. Use index to get tag_out from cache memory
3. Use index to get valid_out from cache memory



請同學根據前面的提示決定 data_out, tag_out 和 valid_out

Part 3



```
// Signals about HIT and MISS
assign  Cache_Hit   =                          // check valid first, and compare tag
assign  IREQ        =                          // if (CacheEnable & Miss) ask data form Memory
assign  DO          =
```

Now that you have most of the signals you need,
you have to decide if the cache is hit or not.
And send the correct signals to memory and CPU.

1. For Cache_Hit signal, check if valid_out=1, then compare tag_in and tag_out

2. For IREQ signal, make sure that Enable=1 first, and check if Cache_Hit = 1 or 0

   IREQ is a signal to request data from main memory

   Enable is a signal which present if the cache is working or not

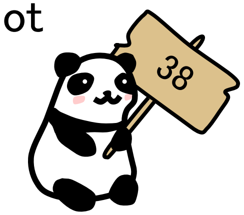
3. For DO signal, use output_enable to decide out data (DO_reg or 32’bz)

   DO (Data Out) is the data form cache to CPU

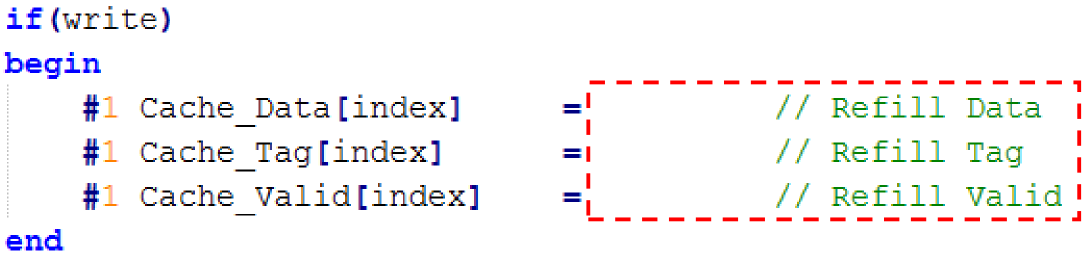
   output_enable is a signal which present if the output data is ready or not

   DO_reg is a register which contains the data for output

請同學根據前面的說明，完成Cache_Hit, IREQ, DO



Part 4



```
if(write)
begin
    #1 Cache_Data[index]      =          // Refill Data
    #1 Cache_Tag[index]       =          // Refill Tag
    #1 Cache_Valid[index]     =          // Refill Valid
end
```

When a cache_miss happened, you have to update the data in I-Cache including Cache_Data, Cache_Tag and Cache_Valid .

1. Put the new data called "Data_in" into Cache_Data

   Data_in is a 32-bit data from main memory

2. Put the new tag called "tag_in" into Cache_Tag

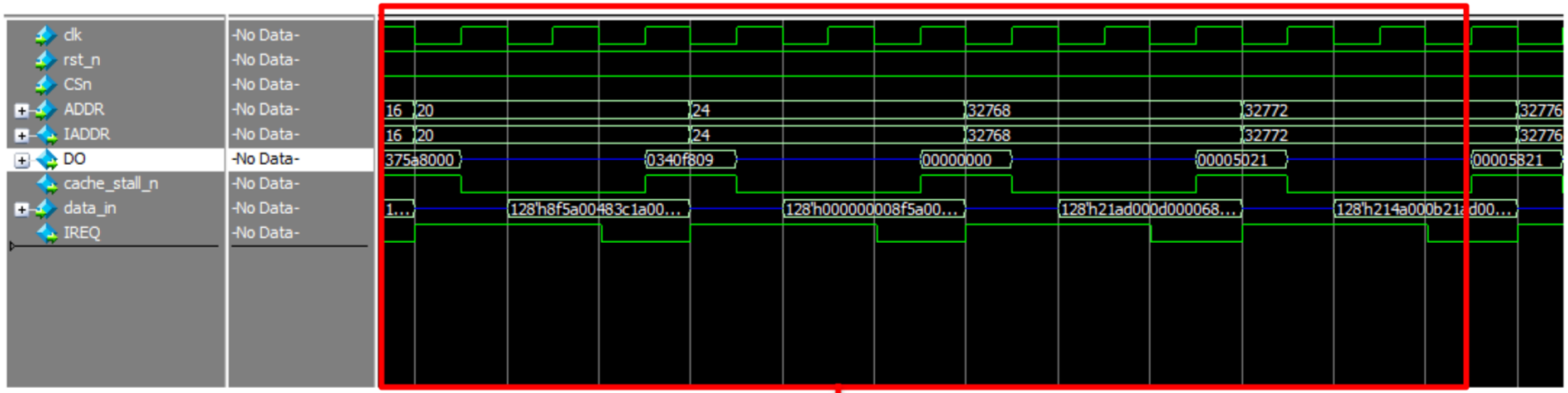
   Tag_in is the new tag which you get from part1

3. Put the new data called "valid_in" into Cache_Valid

   Valid_in is just a signal 1 which indicates that the cache is not empty

請同學根據前面的內容在Cache發生miss的時候更新Cache的內容

Verification



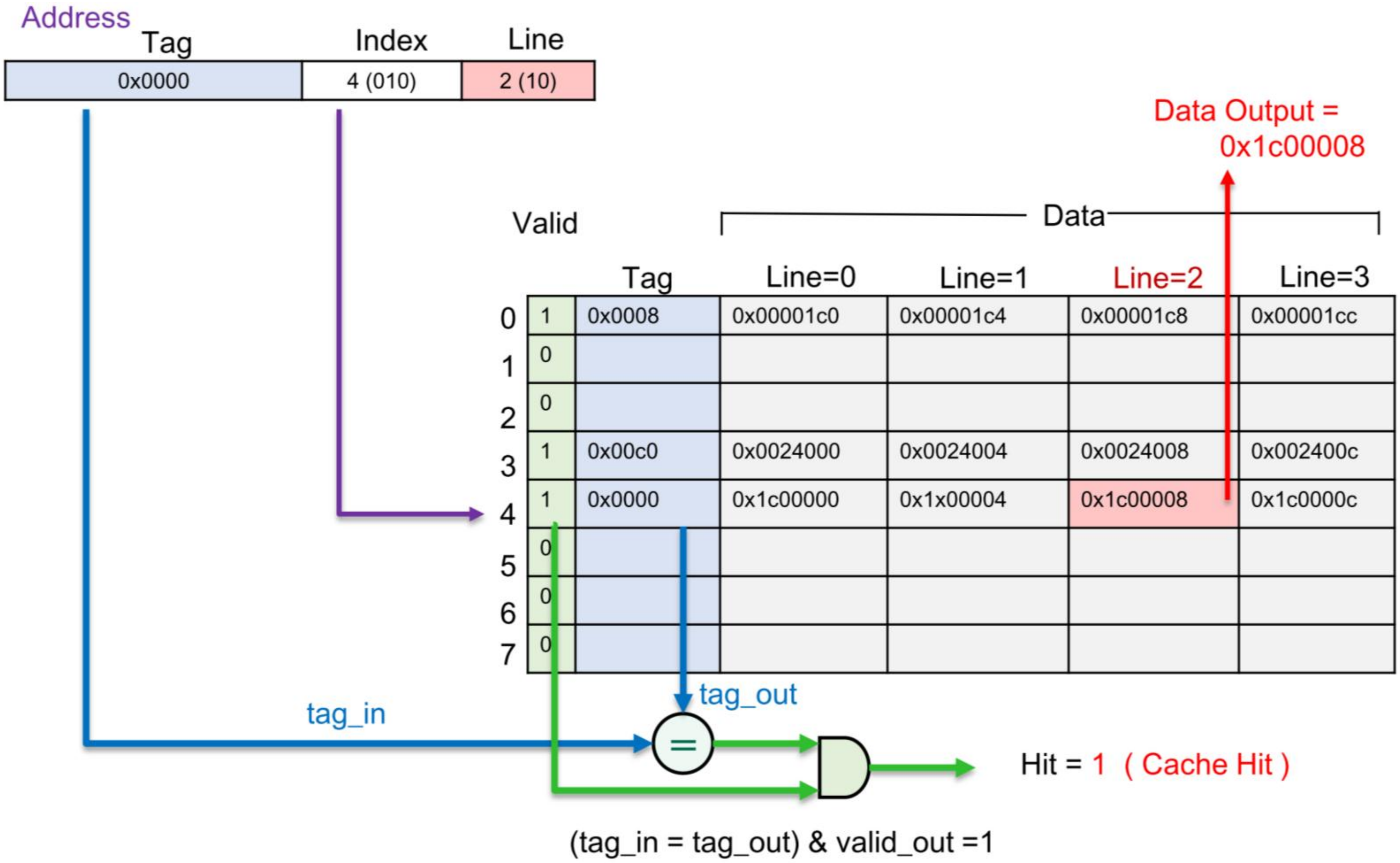
If your RTL code is correct, you will find that the cache will miss whenever the address changes

實作二

以RTL code完成這顆MIPS CPU中I_Cache.v空白部分,總共有**2個**部分

使用ModelSim編譯完成,並且完成驗證。

Cache Architecture



Address

Tag	Index	Line
0x0000	4 (010)	2 (10)

Data Output =
0x1c00008

	Valid	Tag	Data Line=0	 Line=1	 Line=2	 Line=3
0	1	0x0008	0x00001c0	0x00001c4	0x00001c8	0x00001cc
1	0					
2	0					
3	1	0x00c0	0x0024000	0x0024004	0x0024008	0x002400c
4	1	0x0000	0x1c00000	0x1x00004	0x1c00008	0x1c0000c
5	0					
6	0					
7	0					

Hit = 1 (Cache Hit)

(tag_in = tag_out) & valid_out =1

Part 1

```
// Get signals from ADDR
assign  IADDR    =              // pass address to memory (use IREQ signal to control)
assign  word     =              // get the word of the line (Number)
assign  index    =              // get the index(Number)
assign  tag_in   =              // get the tag  (Number)
assign  valid_in=    1'b1;      // set the valid in as 1
```

This is a “Direct Mapped I-Cache” with 256 entries each has a 128 bits data.
ADDR [31:0] is a 32-bit signal (address) from CPU to Cache

1. First, you have to pass the ADDR to IADDR (send addr. to memory if cache miss)
2. The cache has 256 entries which means that your index should be __ bits.
3. The cache has 4 data in each line (entry) which means that your line should be __ bits.
4. Once you know how many bits index, line and tag_in need,
   use ADDR to complete the code

31			0
tag_in	index	line	00

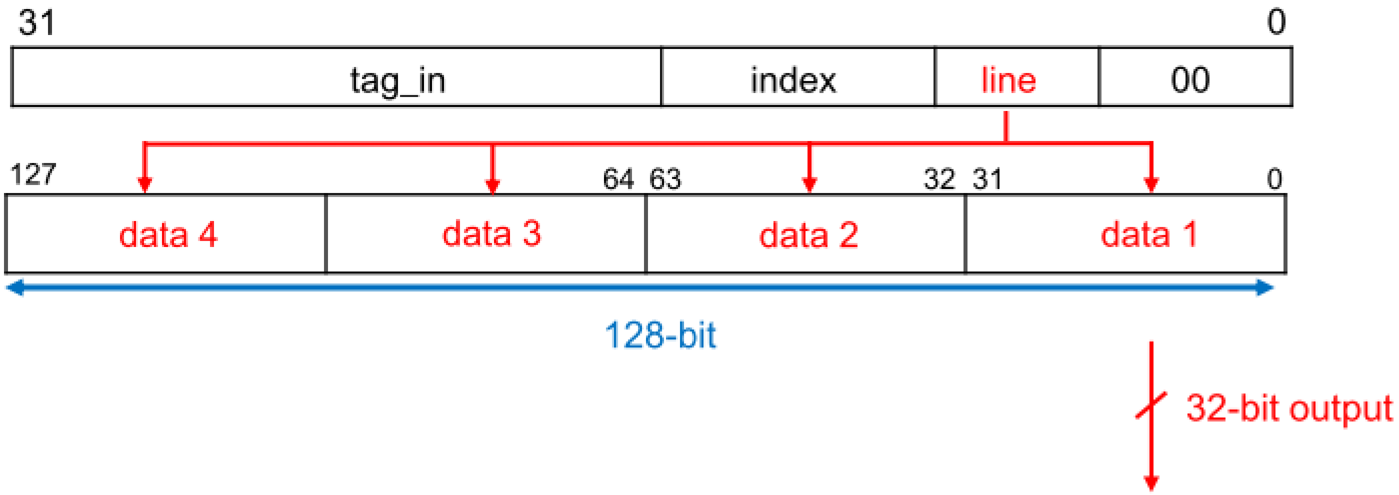
請同學根據前面的內容決定 index, line, tag_in 分別需要幾個 bit,
並且寫出三個訊號分別來自ADDR當中的哪一段

Part 2

We have learned how to use index and tag to check if the cache is hit or miss.

And we also learned how to setup the signal (IREQ) for main memory, and output data (DO) for CPU.

However, there are 4 data in each line (entry). We have to decide which data to output by a 2-bit signal called “line”, before sending data to CPU.



31 0

tag_in | index | line | 00

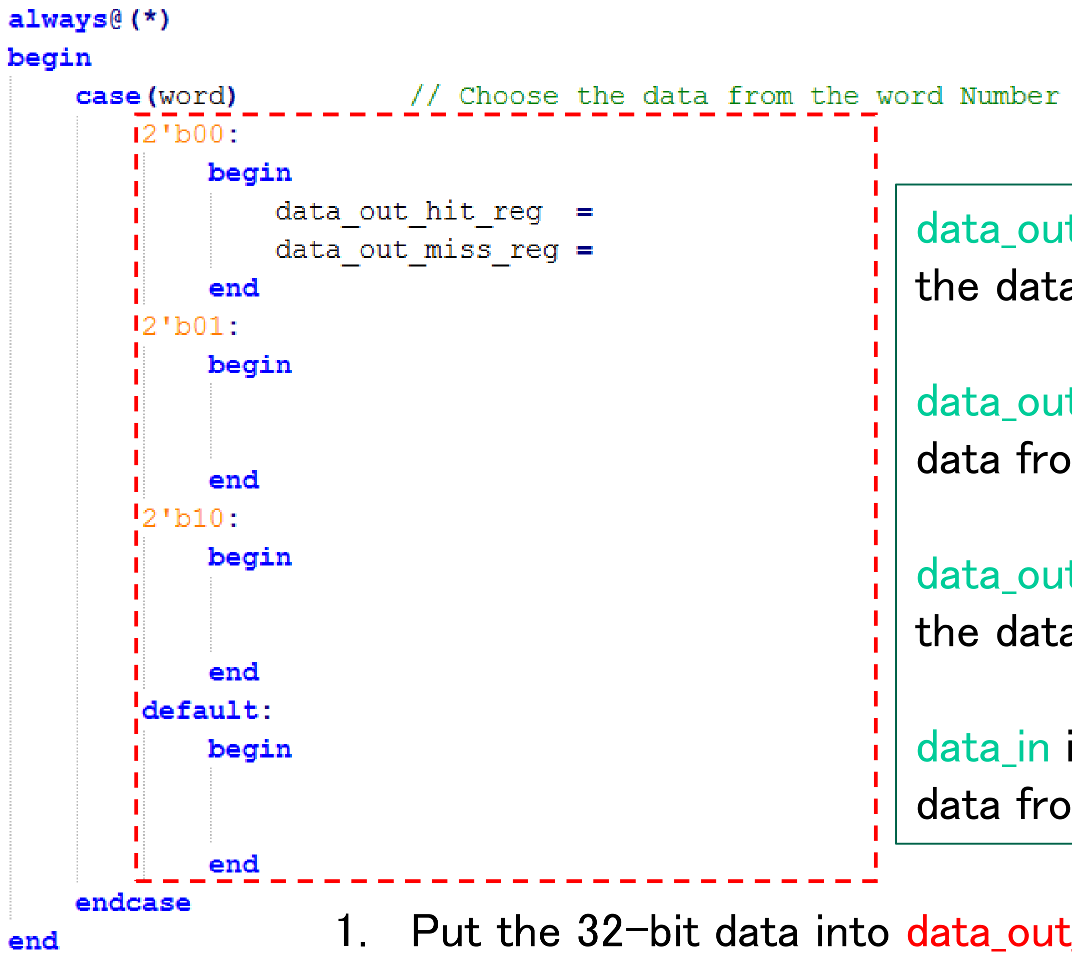
127 64 63 32 31 0

data 4 | data 3 | data 2 | data 1

128-bit

32-bit output

Part 2



```
always@(*)
begin
    case(word)              // Choose the data from the word Number
        2'b00:
            begin
                data_out_hit_reg  =
                data_out_miss_reg =
            end
        2'b01:
            begin


            end
        2'b10:
            begin


            end
        default:
            begin


            end
    endcase
end
```

data_out_hit_reg is a register which contains the data to output when cache hit

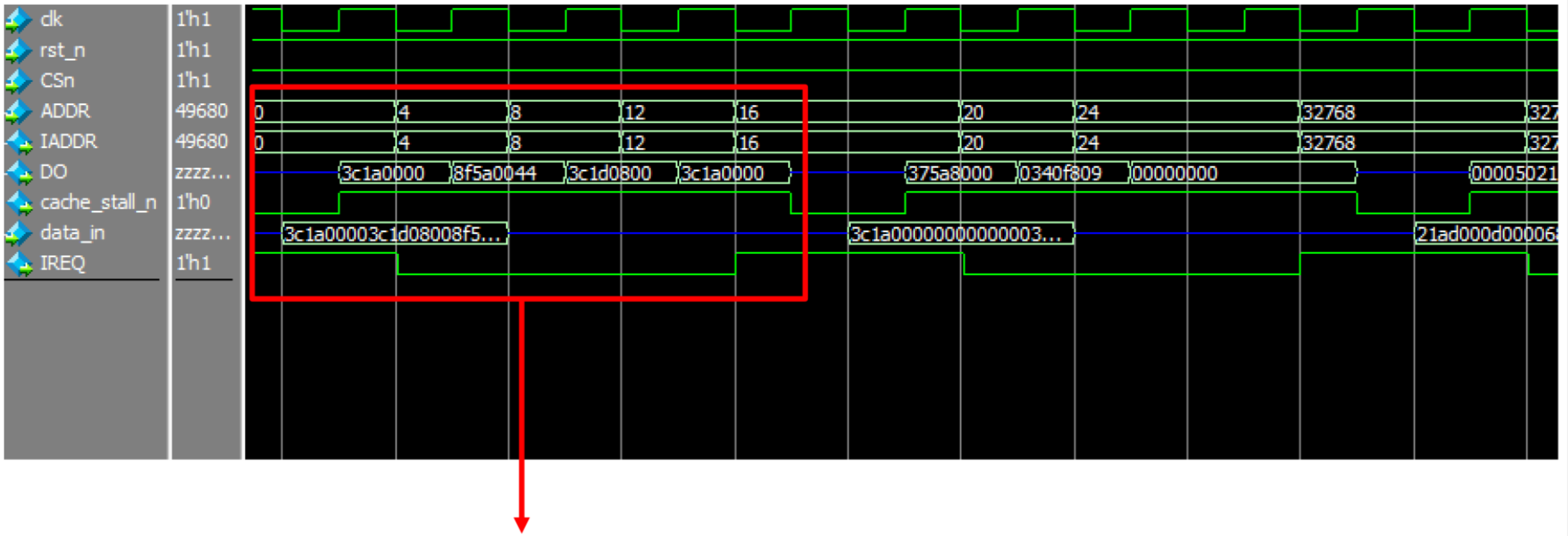
data_out is a 128-bit data which contains the data from cache

data_out_miss_reg is a register which contains the data to output when cache miss

data_in is a 128-bit data which contains the data from memory

1. Put the 32-bit data into data_out_hit_reg from data_out
2. Put the 32-bit data into data_out_miss_reg from data_in

Verification



If your RTL code is correct, you will find that the cache misses every four different address.