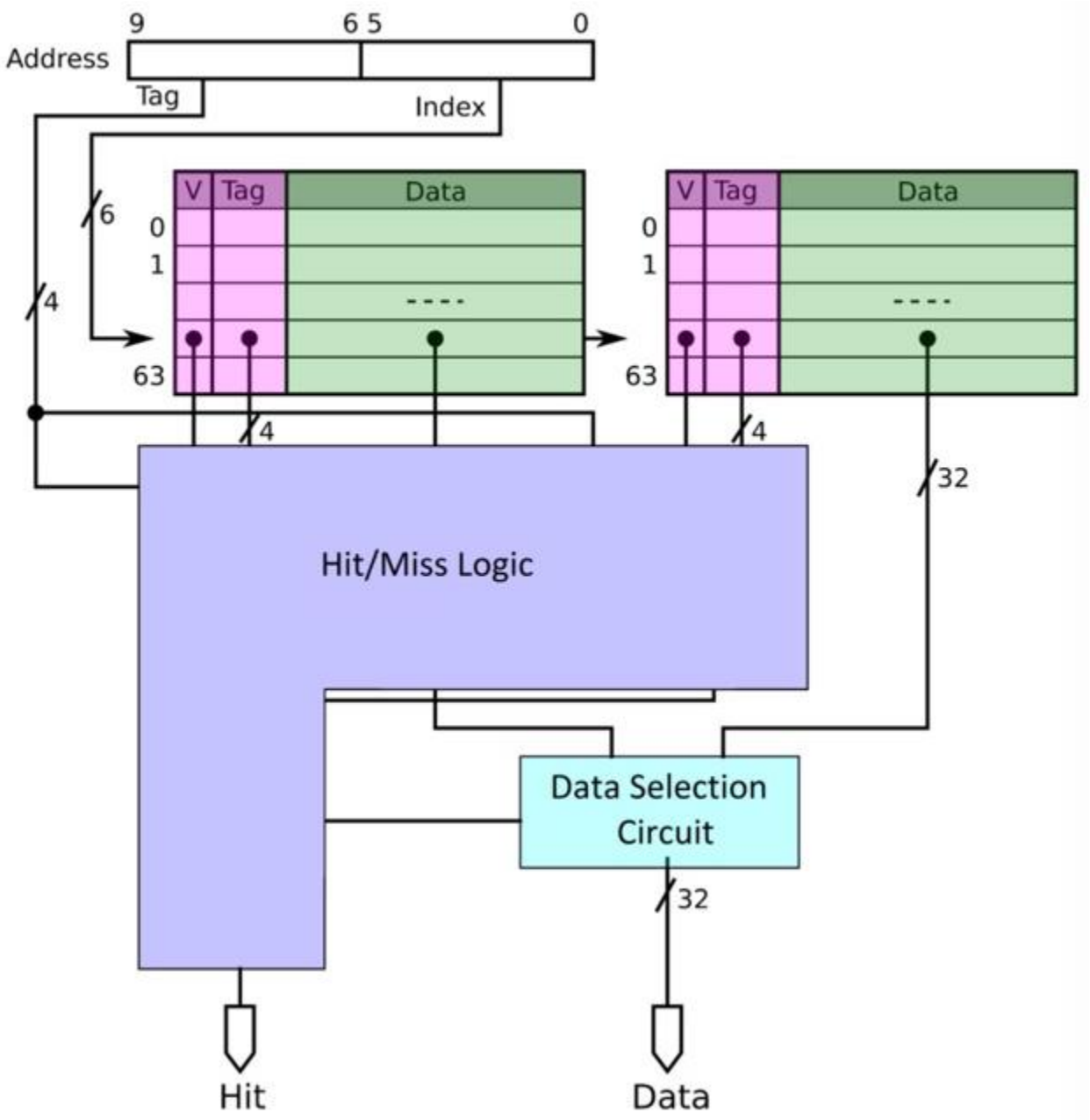
A)Designinig cache



- We don't need offset . (why?):

Cause : The block offset specifies the desired data within the stored data block within the cache row.

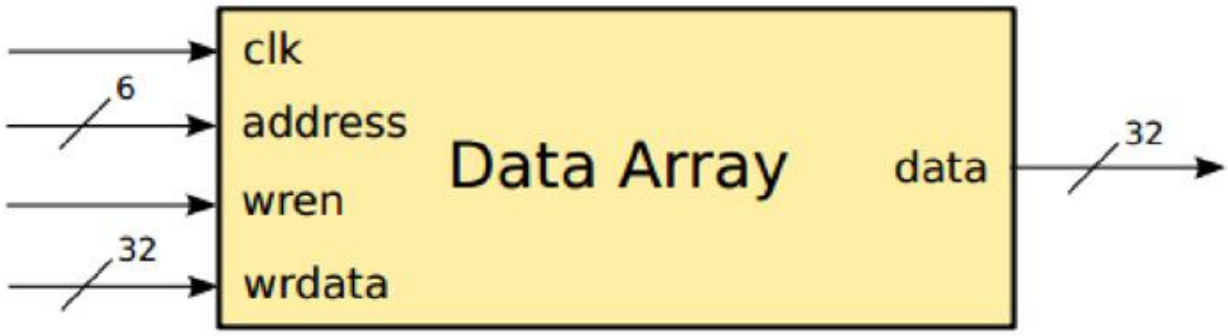
(we have Tag bit to compare , so we don't seem to need this)

What do we do ?

1)use **index** to select a row

2) verify the **tag** of input and compare it with the existing tag of selected row (in 1). If found it , hit . otherwise , miss (you have to go back and search for it in the memory)

2.1 DATA ARRAY



address signal => as : Tag (6 bits)

Read data will be written to output by : **data** signal => (32 bits)

wren signal => if become "1", then input data(**wrdata** signal-32 bits) will be witten in the specific address.

I defined 2 sets like this :

type array_of_data is array (63 downto 0) of std_logic_vector (15 downto 0)
(I think here is a little bit of misunderstanding in the rtl module design , cause each time we just choose a 16 bit data! Not 32 , so I changed it)

Ways for writing in a cache:
1)write through: write both in main mamory +cache
2)write back: just write in cache , and use a flag (valid bit)

2.2 MISS_HIT_LOGIC

Write 0 => Valid & Tag signals of first set .
Write 1 => Valid & Tag signals of second set .

We are after the **Tag** signal .

We must compare the tags . so we define a few signals :

```
signal equal_to_w0: STD_LOGIC_VECTOR(3 downto 0);
signal equal_to__w1: STD_LOGIC_VECTOR(3 downto 0);
signal is_in_w0: STD_LOGIC;
signal is_in_w1: STD_LOGIC;
```

we use gate level design for this part . first we use xnor to compare tag of input , with tag in each set .
0 to 3 in w0 , w1 are tag bits . if all of the conclusions are 1 , (we have to use and) , it means that the tag is found in that set .

For validation , we use “and” function between 5^{th} bit in w0 or w1 , and “ is_in” signal .
And use “or” for initializing “hit”:
hit <= (w0_valid) or (w1_valid)

2.3 TAG_VALID_ARRAY

this module is about to save 4_bit Tag and 1_bit Valid.

wrdata: contains of 4_bit input Tag

invalidate: declare valid alternation of Valid bit in this array.

if (**invalidate** ='1') => valid bit (specified by address signal) <='0'

2.4 CACHE REPLACEMENT POLICY(*MRU*)

Output : a single bit which tells us we should replace way1 or way2

MRU = most recently used

2.5 Cache_Controller

Checks if given address is in the cache or not .

If "yes" => return the address

If"no" => get the address from the memory .

Modules added by myself :

- Memory(ram) : main memory , is going to be used in cache _memory .

  Because I didn't mean to use this project on sayeh , so I wrote a complete memory for it . memory is :

- Cache_Memory :(cache_ram)
- Cache : final approach of all modules
- Mux : will be used in the cache ☺