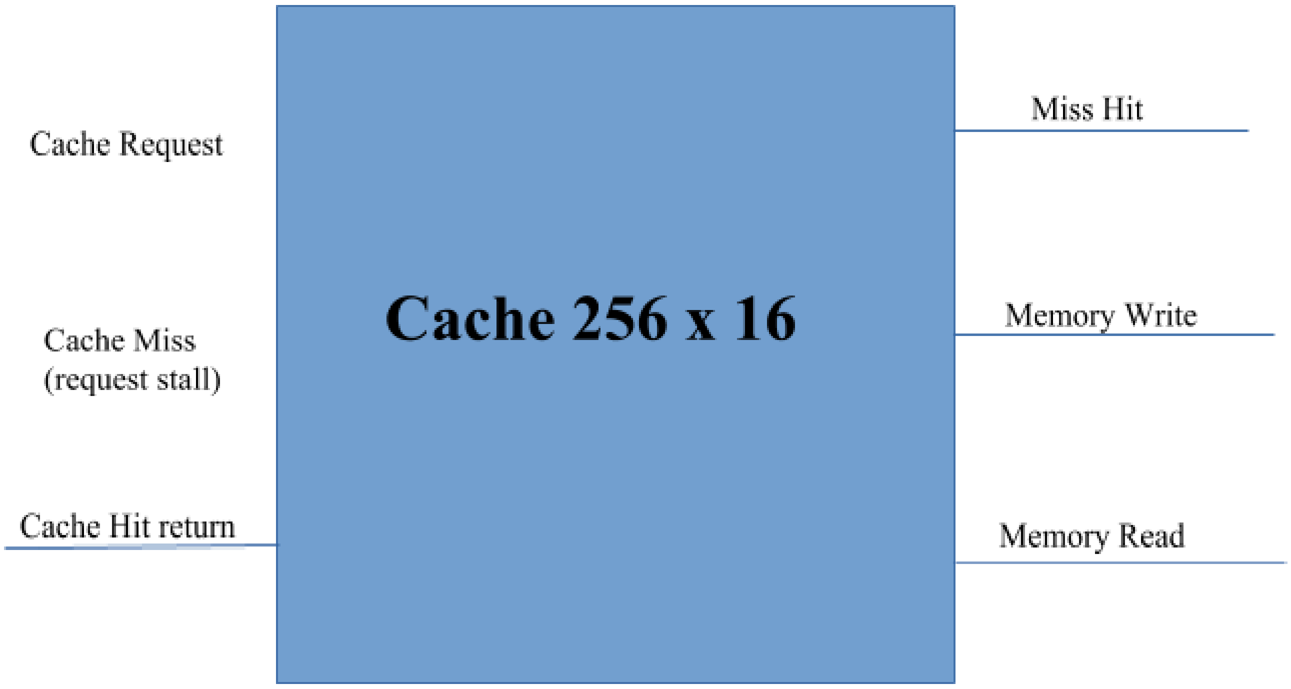
Computer Architecture Project
(Directe Mapped Cache 256 x 16)



Required memory space :

Each of the 256 has 16 x 32 bits. Therefore we have :

Index bits : 8 bits
Tag bits : 18 bits
Byte offset plus Block offset : 6 bits

Here the 2 least significant bits of byte offset are always 00 as each of the address space stores only one byte of data whereas here we have each word constituting 4 bytes of data . So each of the address contains only a byte of the data and we need to concatenate four consecutive bytes of data in order to get a complete word corresponding to that address.

How does the DM Cache work ?

The input for the Cache is the address and it should return the data present in the address as soon as possible. So as soon as the Cache is requested for the data of an address, it splits the address into Index bits, Tag bits and Byte offset bits. The Cache looks into Cache memory and finds the Tag bits associated with the memory(Cache memory) location that is currently holding the data associated with the Index bits of the requested address. It compares the tag bits present in the Cache memory with the tag bits of the address. If both of them happen to match, then it is a Cache hit and the Cache returns the data corresponding to the byte offset bits from the requested address. Otherwise, if the Tag bits do not match, then the Cache has to fetch the data corresponding to this address from the Main memory and store this data in the place of the data corresponding to the set of Index and Tag bits associated with the given address.

Explanation of the Code:

The overall code has been divided into three parts.
1. Cache.v: This module has the code for Cache block implementation.
2. Cache_tb.v: This has the test case on which Cache.v is run.
3. Generate.cpp: This has the code for generating data to be placed in memory.

Coming to the main part, **Cache.v**
Using clock and address as inputs, we first load data into the main memory block implemented as a 2D reg. Then we

take the input address and splice it into Tag, Index and Offset bits which are 18, 8 and 4 respectively.

Now at a positive clock edge, we check if the required address is present in the Cache block which is also a 2D reg. For checking this we first compare the Tag bits of the address with the Tag bits that are present at the location pointed to by the index bits of the address in the Cache block. If it's a match, we take it as a hit and return the data indicates by the Byte Offset. Else, we take it as a miss and fetch the requested data from the Main memory block and also write to the Cache block.