**PRACTICAL 10**

**COMPUTER ORGANISATION AND ARCHITECTURE**

|  |  |
| --- | --- |
| **NAME: VARUN KHADAYATE** | **ROLL NO: A016** |
| **PROGRAM: BTECH SY** | **DIVISION: CSBS** |
| **BATCH: 1** | **DATE OF EXPERIMENT: 27/10/2020** |

**AIM**

**Understanding the behaviour of direct mapped cache from working module and designing a direct mapped cache for given parameters**

**THEORY**

**Design of Direct Mapped Cache:**

Cache memory is a small (in size) and very fast (zero wait state) memory which sits between the CPU and main memory. The notion of cache memory actually rely on the correlation properties observed in sequences of address references generated by CPU while executing a program(principle of locality).When a memory request is generated, the request is first presented to the cache memory, and if the cache cannot respond, the request is then presented to main memory.

* **Hit:** a cache access finds data resident in the cache memory
* **Miss:** a cache access does not find data resident, so it forces to access the main memory.

Cache treats main memory as a set of blocks. As the cache size is much smaller than main memory so the number of cache lines are very less than the number of main memory blocks. So, a procedure is needed for mapping main memory blocks into cache lines. Cache mapping scheme affects cost and performance. There are three methods in block placement-

* **Direct Mapped Cache**
* **Fully Associative Mapped Cache**
* **Set Associative Mapped Cache**

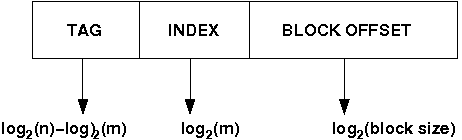
**Direct Mapped Cache**

A given memory block can be mapped into one and only cache line.

**Block identification:** let the main memory contains n blocks (which require log2(n)) and cache contains m blocks, so n/m different blocks of memory can be mapped (at different times) to a cache block. Each cache block has a tag saying which block of memory is currently present in it, each cache block also contains a valid bit to ensure whether a memory block is in the cache block currently.

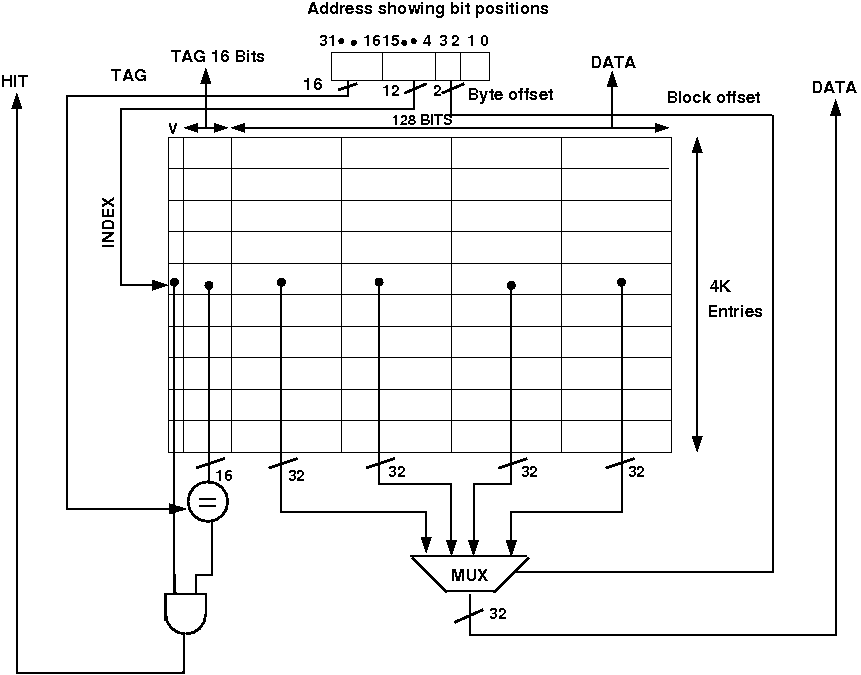
* **Number of bits in the tag: log2(n/m)**
* **Number of sets in the Cache: m**
* **Number of bits to identify the correct set: log2(m)**

The memory address is divided into 3 parts- tag (most MSB), index, block offset (most LSB) in order to do the cache mapping.



* **Select set using index, block from set using tag.**
* **Select location from block using block offset.**
* **tag + index = block address**

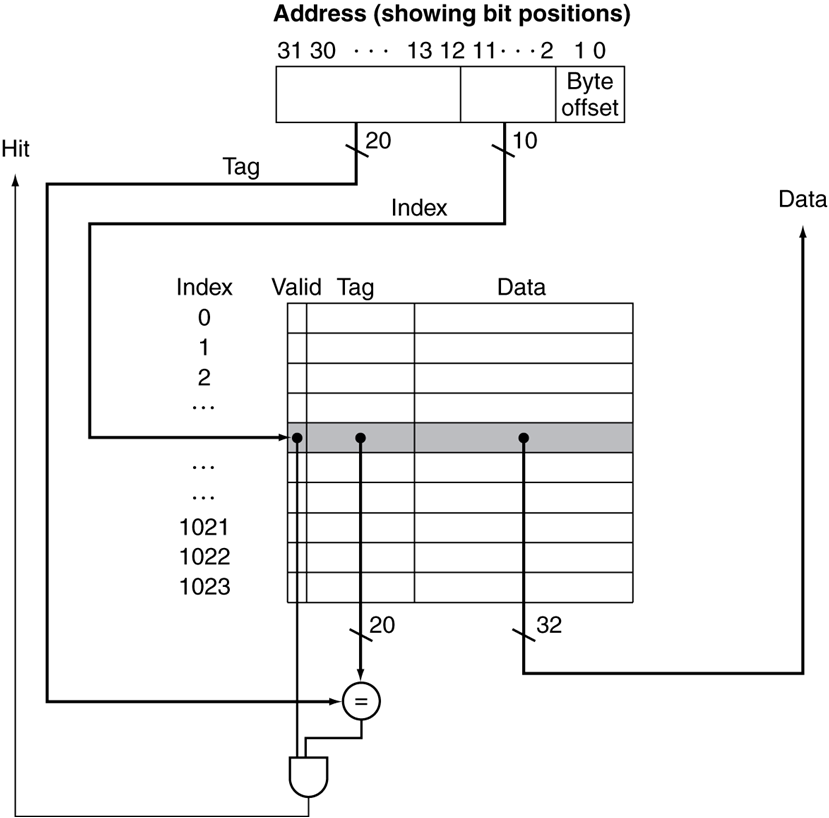
Diagram of a direct mapped cache (here main memory address is of 32 bits and it gives a data chunk of 32 bits at a time):



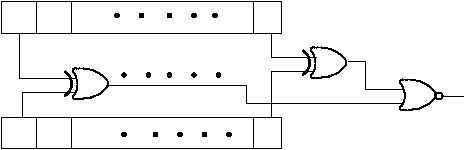
If a miss occur CPU bring the block from the main memory to the cache, if there is no free block in the corresponding set it replaces a block and put the new one. CPU uses different replacement policies to decide which block is to replace. The disadvantage of the direct mapped cache is that it is easy to build but suffer the most from thrashing due to the 'conflict misses' giving more miss penalty.

**Design issues:**

Below is a simple cache which holds 1024 words or 4KB, memory address is 32 bits. The tag from the cache is compared against the most significant bits of the address to determine whether the entry in the cache corresponds to the requested address as the cache has 210 or 1024 words and a block size of one word, 10 bits are used to index the cache, leaving 32-10-2=20 bits to be compared against the tag. If the tag and the most significant 20 bits of the address are equal, and the valid bit is on then the request hits in the cache otherwise miss occurs. No replacement policy has been implemented in the circuit.



The comparator Circuit through which tag is compared with specified bits of address:

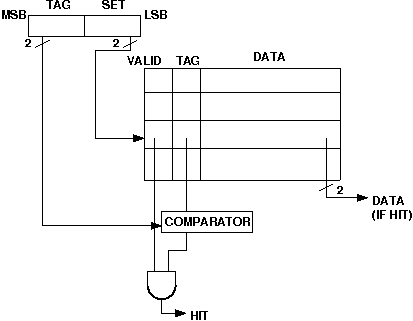


**PROCEDURE**

Design of Direct Mapped Cache:

**Procedure to perform the experiment for Direct mapped cache on the existing component 'Direct Mapped Cache' component in the 'other components' drawer in the simulator. This simulator supports 5-valued logic.**

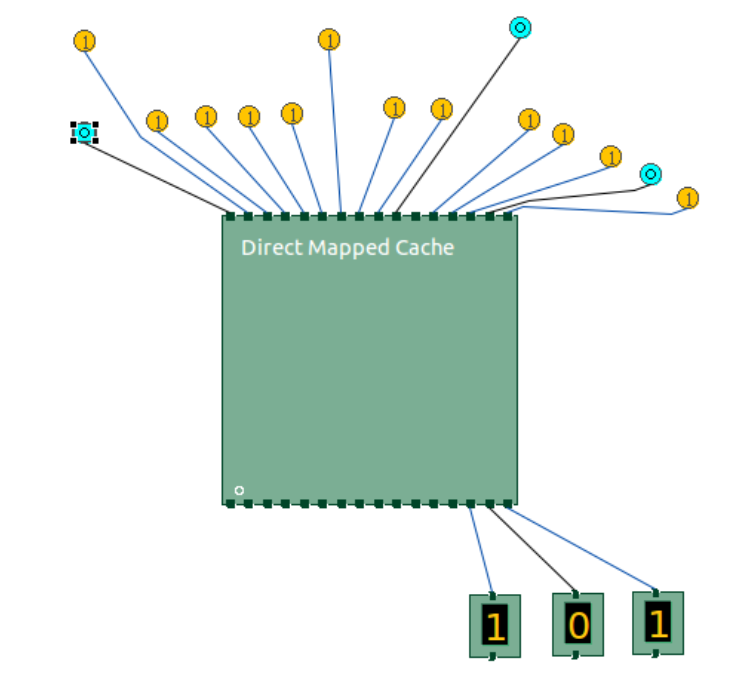
Below is a schematic diagram of the 'Direct Mapped Cache' component in the 'other components' drawer in the simulator:



1. Click on the 'Direct Mapped Cache' component(in the 'other components' drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 15 Bit switches and 3 Bit Displays(from Display and Input drawer of the pallet, if it is not seen scroll down in the drawer)
2. 'Direct Mapped Cache' component in the 'other components' drawer in the simulator supports both writing in the cache and the cache mapping. No replacement policy has been implemented. Initially the cache is empty, user must give inputs. the component contains 4 sets, each set has 5 bits, the left most bit is the valid bit, next 2 bits are tags, next bits are data bits, also it contains a one dimensional array of memory with 4 bit to store the memory address, user has to give this address input also. The cache reads all the data bits at a time, so block offset is not required.
3. The pin configuration of the component can be seen whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner (indicating with the circle) and increases anticlockwise.
4. For a 'Direct Mapped Cache' component pin configuration is:

* pin-32= S (selects whether user wants to perform cache write or cache mapping)
* pin-31= R/W'A (selects whether user wants to input the address or cache mapping)
* pin-30=A3, pin-29=A2, pin-28=A1, pin-27=A0 (these 4 pins are used to give address input). A3 is the most significant bit and A0 is the least significant bit. A3 and A2 will be compared with the tag. A1 and A0 will select the corresponding set.
* pin-26= R/W’D (selects whether user wants to input in the set of cache or cache mapping)
* pin-25= M1, pin-24=M0 (M1 is the most significant bit and M0 is the least significant bit). these two bits are used for cache write purpose, it selects the set of which user wants to give inputs to the valid bit, tag bits and data bits.
* pin-23= Den (this is an enable input which must set for any write purpose in the cache).
* pin-21= valid bit
* pin-20= T1, pin-19=T0 (T1 is the most significant bit and T0 is the least significant bit). These are tag bits.
* pin-18= D1, pin-17=D0 (D1 is the most significant bit and D0 is the least significant bit). These are data bits.
* pin-14= Hit/Miss bit (if it gives 1 then hit otherwise miss)
* pin-15= F1, pin-16=F0 (F1 is the most significant bit and F0 is the least significant bit). These are output data bits and will be given only when there is a hit.

1. **Essential pin configurations for writing in the cache: S=1, R/W'A=0, R/W'D=0, Den= 1**
2. **Essential pin configurations for cache mapping: S=0, R/W'A=1, R/W'D=1, Den= 0**
3. To connect any two components, select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components. After the connection is over click the selection tool in the pallet.
4. See the output, Bit switches are used to give input so that you can toggle its value with a double click and see the outputs with different inputs.



**CONCLUSION**

Hence, we can understand the behaviour of direct mapped cache from working module and designing an associative cache for given parameters.