Lab 11 - Memory design

Part 2

This laboratory work presents the second part of the design methodology of memory modules for microprocessor-based systems.

1. Designing dynamic memories

Basically, the design process is almost the same as for static memories, with the following amendments:

- A periodic refresh mechanism must be added
- The addresses must be multiplexed (the row and column addresses are generated sequentially on the same address signals)
- Validation signals for row (RAS – Row Address Signal) and column address (CAS Column Address Signal)
- Circuit selection is done by the RAS and CAS signals

For dynamic RAMs, multiplexing the addresses is necessary in order to reduce the number of pins of the memory circuit and, by default, its size. Note that dynamic circuits have a relatively large size, which imposes a large number of addresses for selection. The internal organization of a dynamic memory is basically a matrix, with columns and rows; selecting a memory location is done by specifying its row and column address.

A periodic refresh of the memory is necessary because the data is stored for a limited time after a read or write operation (due to the capacitor which discharges in time). The control of the refresh process can be done locally, at memory module level, or centralized, system-wide level. Regardless of the method chosen, a designer must make sure that they don't overlap with the regular read or write cycles. The refresh operation is done simultaneously for each line of the memory matrix. The figures below illustrate the time diagrams of a read, write and refresh cycle.

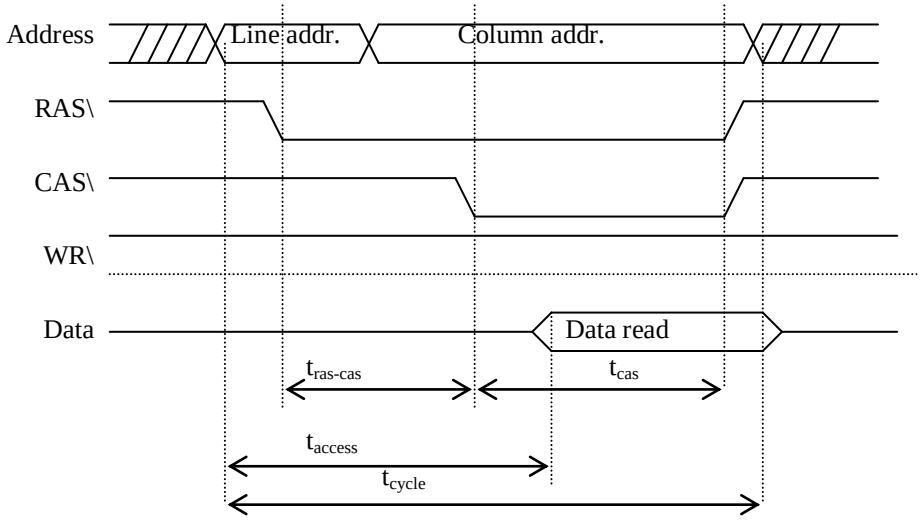


Figure 1. Read cycle

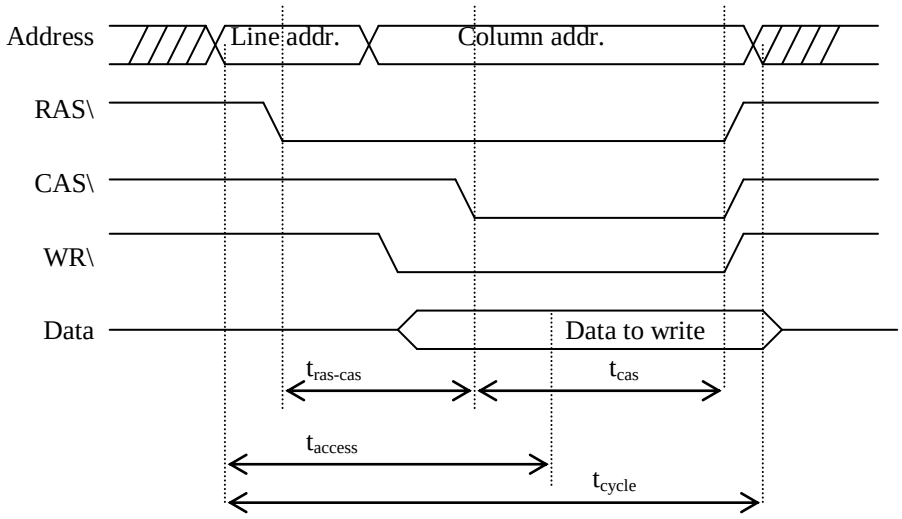


Figure 2. Write cycle

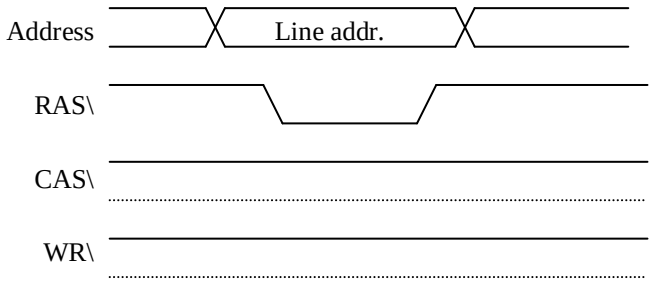


Figure 3. Refresh cycle

Design example:

- Memory size: 16 MB
- Structure: word (16 bits), can address the first or last 8 bits of a word
- Bus characteristics
  - 28 address lines
  - 16 data lines
  - Command signals: RD\, WR\, Refresh\
- Start address: D0.0000 H
- Memory circuits of 1M * 8 bits

Main block diagram

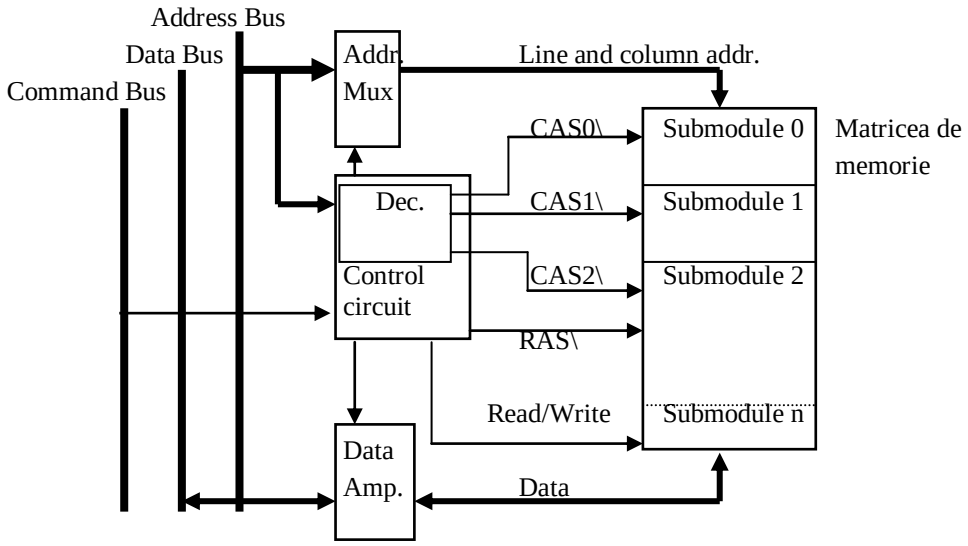
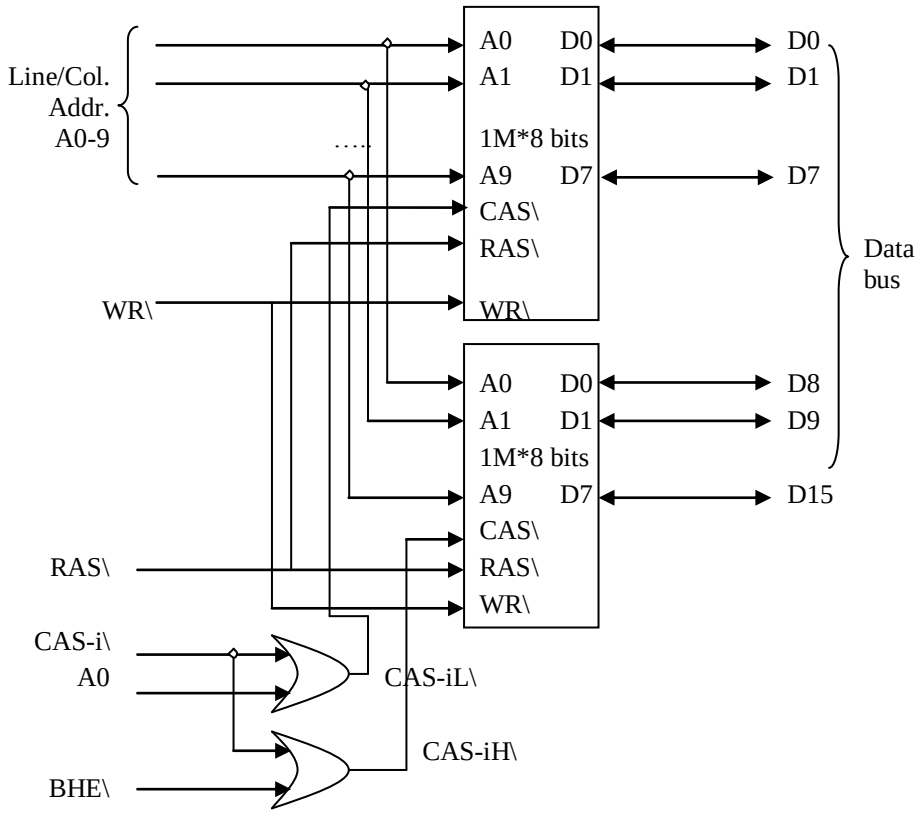


Figure 4. Main block diagram of a memory module

Submodule block diagram



Submodule *i* (1M*16 bits = 2MB)

Figure 5. Block diagram of a submodule

Memory matrix

This block diagram is similar with the one from the static memory, with the following changes:

- Instead of *Sel-i*\ signals, *CASi*\ signals are used
- Instead of addresses A1-A16, A0-A9 are used
- The *RAS*\ signal is connected to each submodule

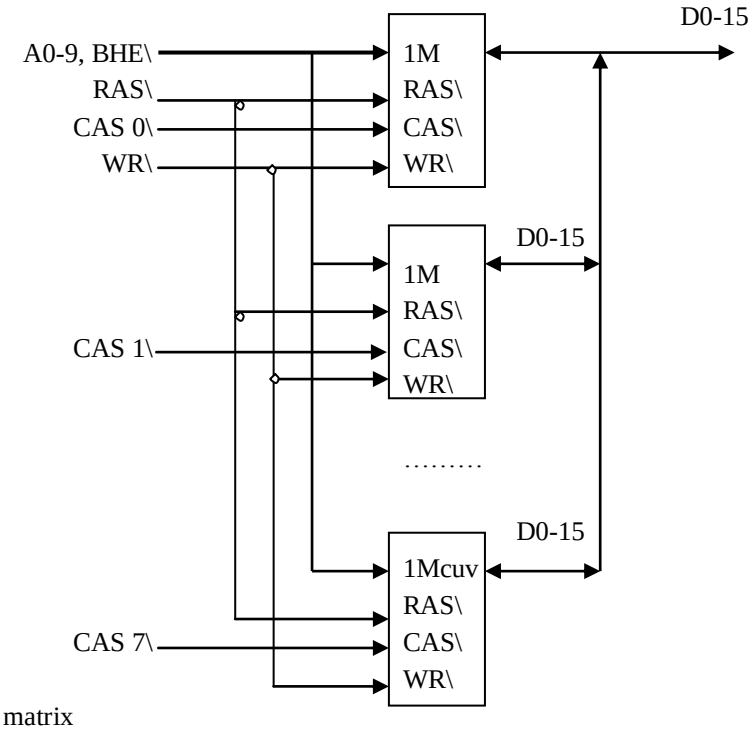


Figure 6. Memory matrix

Decoder and command circuit

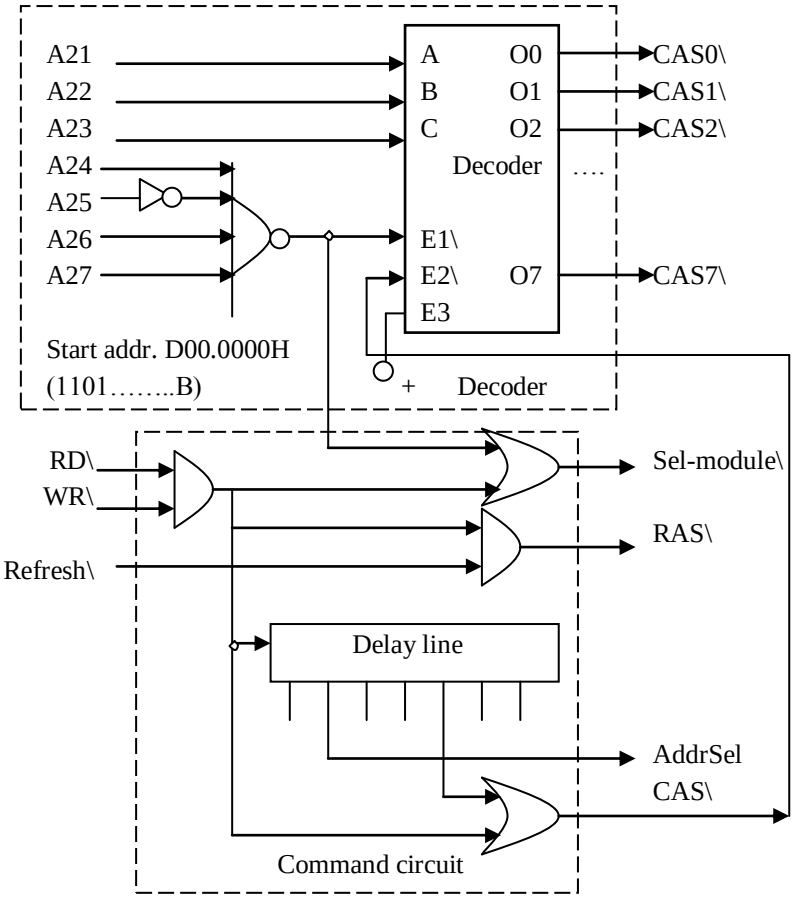


Figure 7. Block diagram of decoder and command circuit

Amplification and multiplexing module

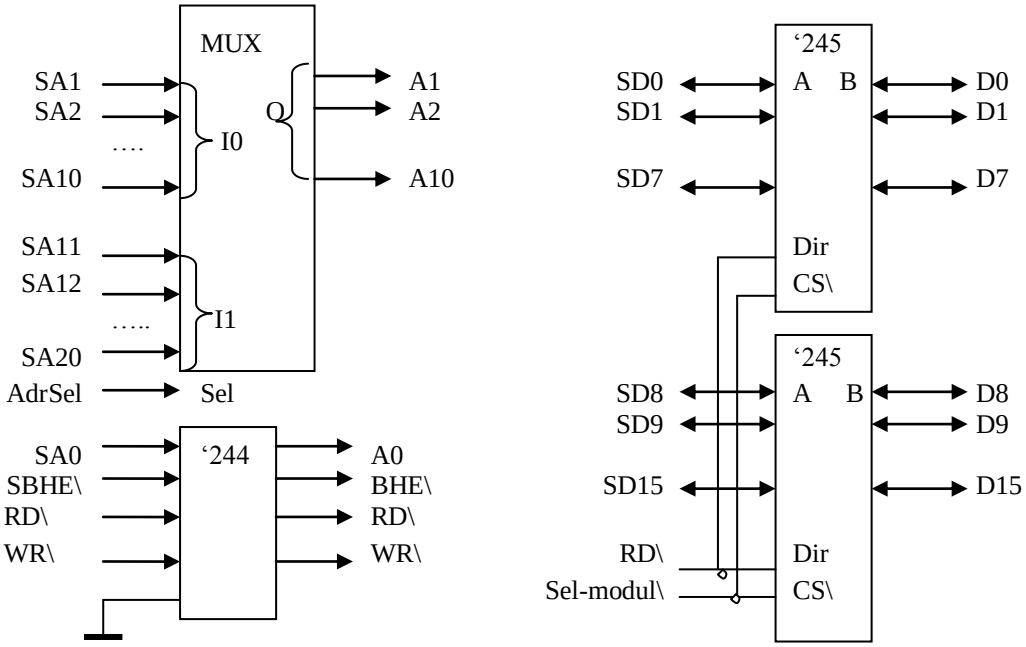


Figure 8. Amplification and multiplexing module

2. Designing cache memories

The cache memory is the fastest memory in a computer system and is used to store temporarily a portion of data and instructions for immediate use. Because the cache memory is very fast, it is also very expensive, thus the size is limited and depends from architecture to architecture.

The placement of the cache memory is between the main memory and CPU. In order to improve performance, modern processors have multiple interacting levels of cache memories on the same chip. The cache memory contains copies of some blocks from the main memory. If a word is requested by the CPU, the cache memory is searched first. In case the word is found (i.e. cache hit), the data is sent back to the CPU; otherwise (i.e. cache miss), the word is retrieved from the main memory and then sent back to the CPU.

The memory words are grouped in pages, either blocks or lines. Each block is marked with an address, referred to as a *tag*. The collection of tag addresses assigned to the cache memory is stored in a *tag memory* (Figure 9).

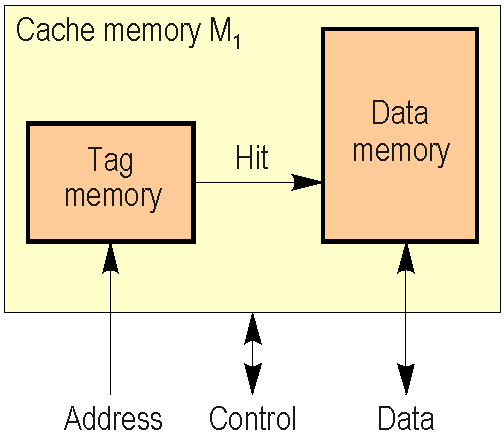


Figure 9. Basic structure of a cache memory

Consider that the CPU generates a *read* request. As explained above, when this request reaches the cache memory, the data is searched within the cache. If the word is not found, the requested data is supplied by the main memory. But if the word is found in the cache memory, there is no need to access the main memory. Figure 10 below illustrates an execution of a read operation.

For a *write* request generated by the CPU, the cache memory operation is similar to a *read* request. If the word is found in the cache, the write operation is performed. If the word is not found in the cache, a copy of the word is loaded from the main memory into the cache memory (which is followed by the write operation). Figure 11 below illustrates an execution of a write operation.

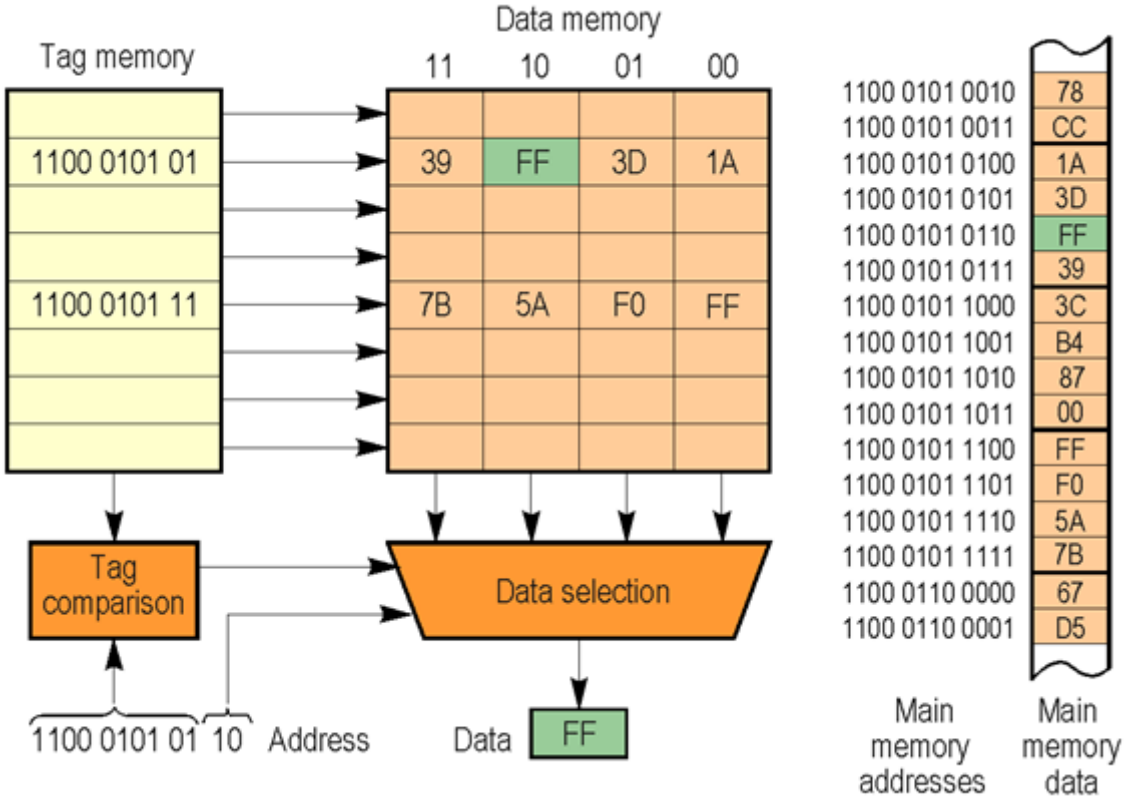


Figure 10. Execution of a read operation

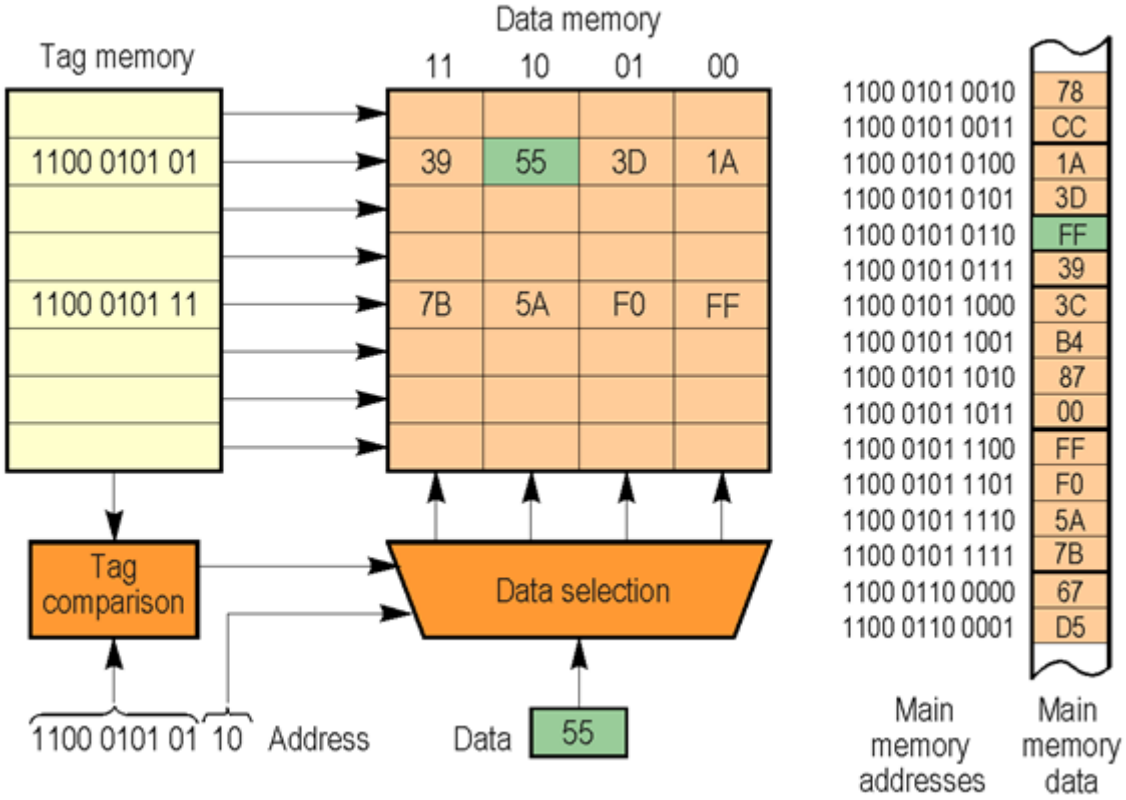


Figure 11. Execution of a write operation

2.1 Associative Mapping

In a fully associative cache memory, the address and the contents are stored as separate words in the cache memory (i.e. any location can be used). The organization is a combination of an associative memory and RAM, thus, only the addresses of the words are stored in the associative part (Figure 12).

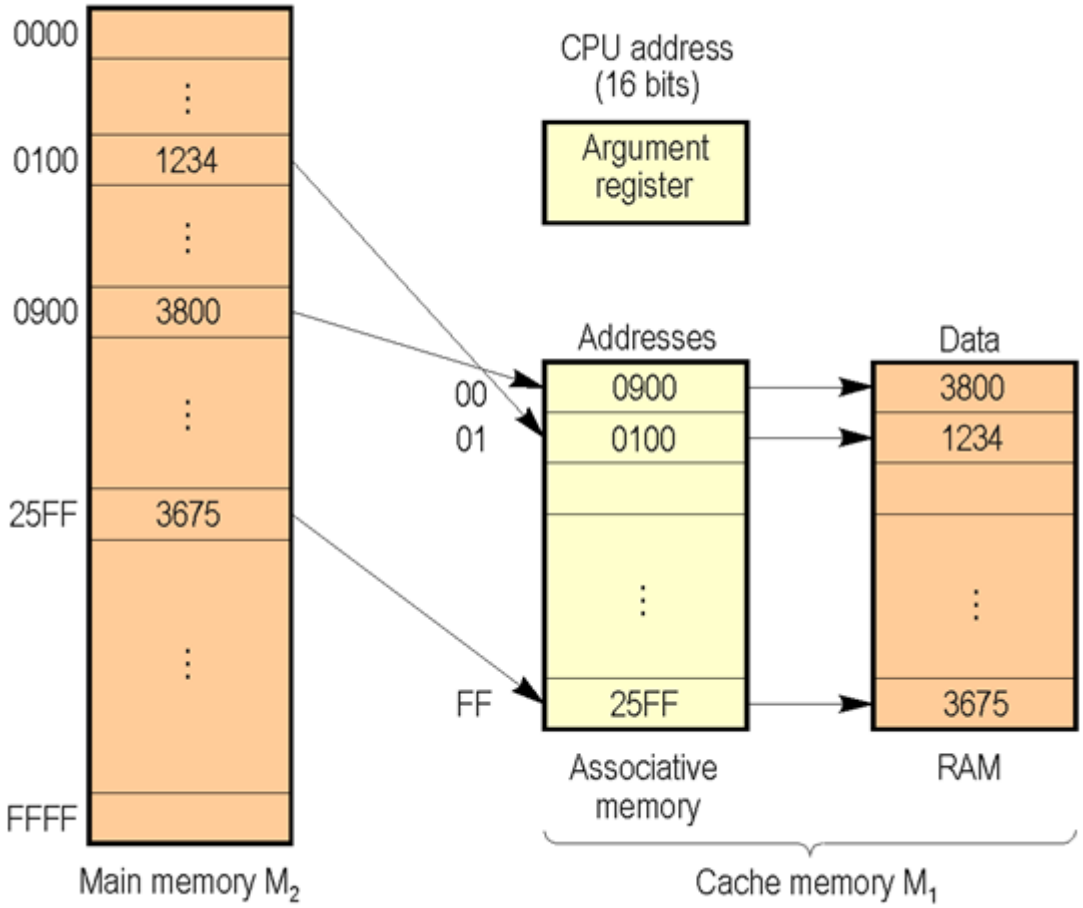


Figure 12. Associative-mapping cache memory

2.2 Direct Mapping

If RAM memories are used instead of associative memories, the cost of the cache memory can be reduces. Consider that the cache memory M_1 is divided into $b=2^s$ regions $M_1(0), M_1(1),..,M_1(b-1)$ called *sets;* and the main memory M_2 divided into *blocks*. Each block $M_2(i)$ in M_2 is mapped into a set $M_1(j)$ in M_1. The set address j is $j=i \bmod b$.

Now, consider that a cache memory set contains a single word and the memory addresses are divided into two parts:

- **Index** – the low-order *s* bits of the main memory address-> identifies the cache set that can store the memory block
- **Tag -** the remaining high-order *t* bits of the main memory address

The address specified by the index points out:

- A tag, which is stored in the tag memory
- A memory block, which is stored in the data memory

The tag memory can be a normal RAM and is addressed by the *s*-bit index. If there are 2^d words per set, the low-order *d* bits of the address form the *displacement* of the word within the set. In figure 13 and 14, the architecture of a direct-mapping cache memory is described.

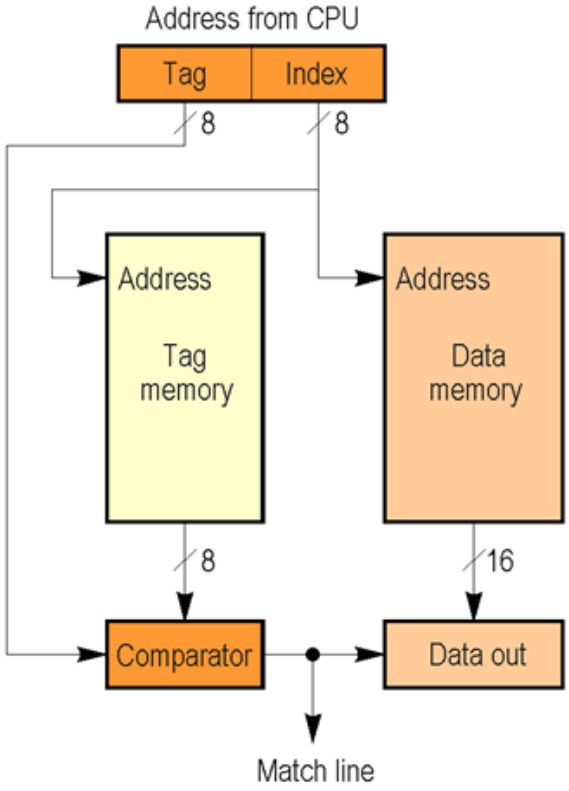


Figure 13. Direct-mapping cache memory block diagram

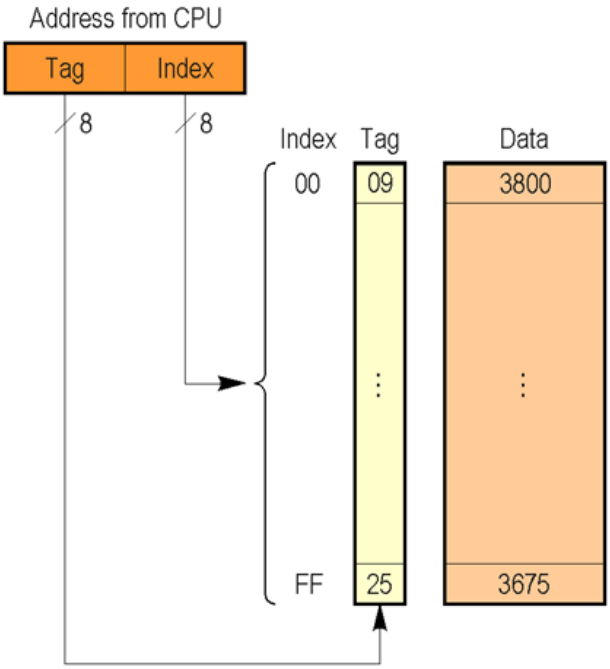


Figure 14. Direct-mapping cache memory address structure

Direct-mapping cache memories have the advantage of requiring less number of bits for each word in the cache memory. Also, this type of architecture requires no associative memory. But the performance can decrease if two or more words (with the same index, but different tags) are accessed frequently.

Design example of a direct-mapped cache memory

Consider a processor connected to a byte-addressable external memory. The address bus is 32-bit wide and the data bus is 64-bit wide. The capacity of the cache memory is 256 KB and the size of a set (block) is 32 B.

Dividing the entire capacity to the size of a set will point out the number of sets.

No. of sets: 256 KB / 32 B = 8 K = 2^{13} (13 bits for selecting the sets)

Thus, the capacity of the tag memory is *8 K * t bits*.

The capacity of the data memory must be 256 KB. Dividing it by 8 B, we find the total capacity for words of 64 bits.

Capacity of data memory: 256 KB / 8 B = 32 K words = 2^{15} *words of 64 bits* (15 bits address for the data memory)

In order to address a byte within a set, 5 displacement bits must be used, because there are 32 B in a set (*d=5*).

To select the sets in the data memory, 13 bits are needed (i.e. set address *s=13*). When calculating the number of sets, the exponent reveals the number of bits needed.

Because the address bus is 32-bit wide, the tag size is:

t = 32 - (13 + 5) = 14 bits

The address for the data memory has 15 bits (exponent revealed when finding out how many 64-bit words are in the whole memory).

Figure 15 below illustrates the block design of the direct-mapped cache memory example.

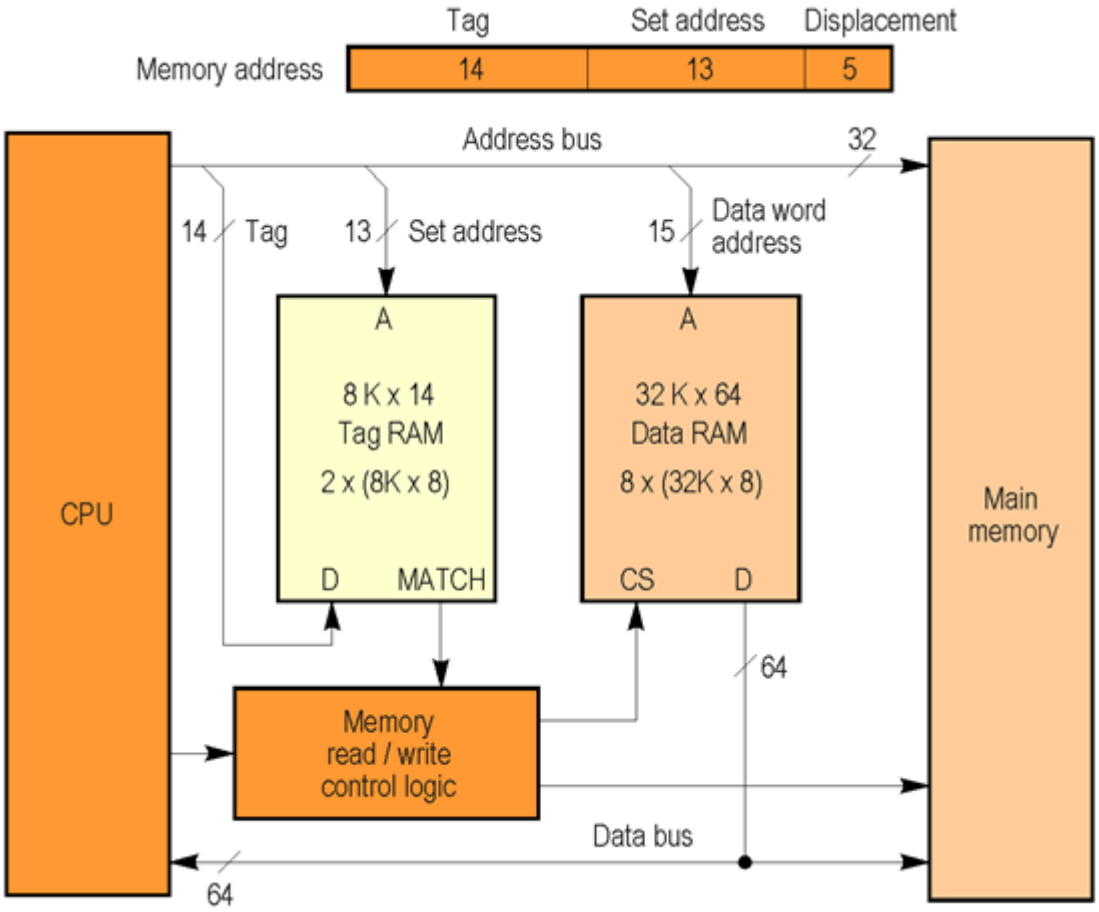


Figure 15. Design example of a direct-mapped cache memory

2.3 Set-Associative Mapping

This type of cache memory allows the storage of more blocks with the same index. The cache memory M^1 is divided into $b=2^s$ sets. Each set can store $k = 2^m$ blocks and each set $M_1(k)$ is an associative memory.

Both associative mapping and direct mapping are special cases of set-associative mapping

- When $k=1$: direct mapping
- When $b=1$: fully associative mapping

In practice, only small values of k are used. This allows using RAMs to store the tags. If k memory blocks, the cache memory is k-way set-associative (Figure 16). Note that each memory block has the same structure as a direct-mapping cache memory.

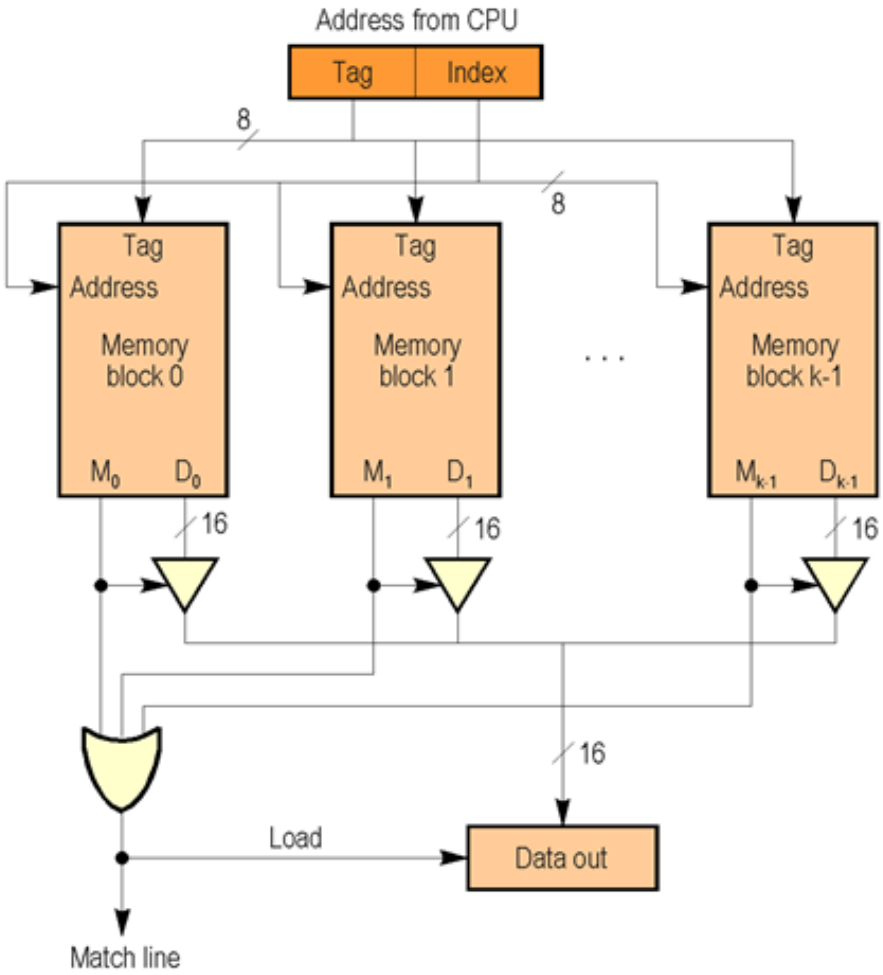


Figure 16. K-way set-associative cache memory

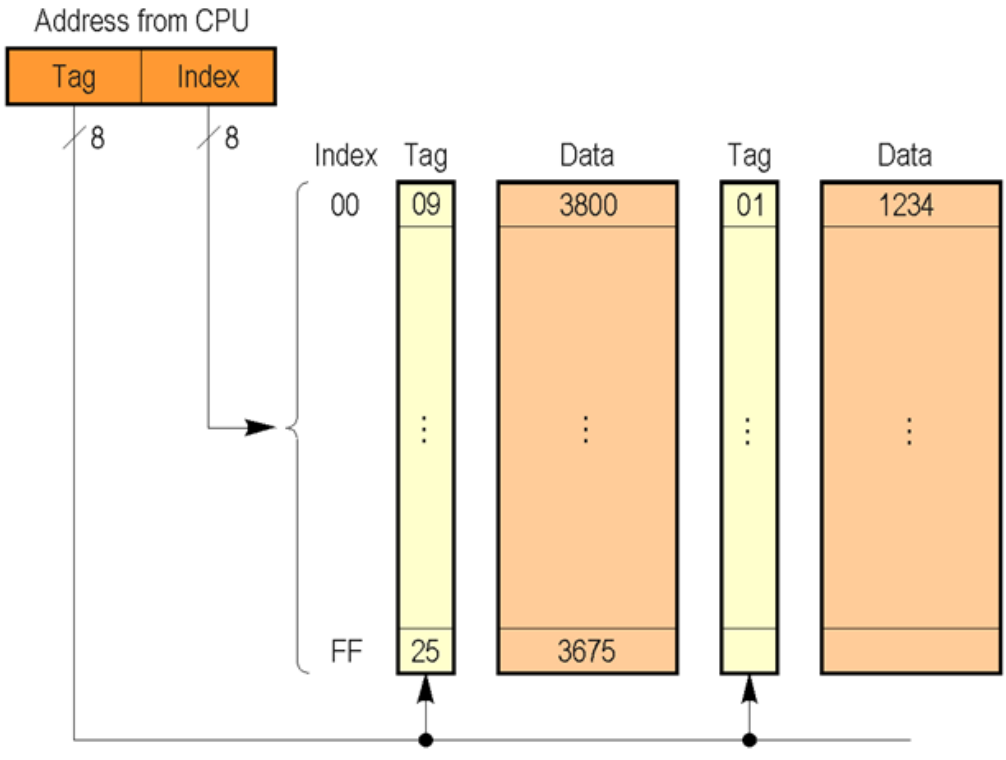


Figure 17. 2-way set-associative mapping cache memory

Design example of a 2-way set-associative cache memory

Consider the capacity of a cache memory of 8 KB. Each set (block) has the size of 8 B. The address bus is 32-bit wide, while the data bus is 64-bit wide. The cache memory is 2-way set-associative, thus $k=2$.

No. of sets: 8 KB / (2(8 B)) = 512 =* 2^9

To select the sets in the data memory (set addresses), 9 bits are needed $s=9$ (the exponent revealed above).

To select a byte within a set, 3 bits are needed $d=3$.

Tag size: $t = 32 - (9 + 3) = 20$ *bits*

Tag memory: 2 * (512 * 20 bits)

No. of words: 8 KB / (2*(8 B)) = 512

Data memory: 2 * (512 * 64 bits)

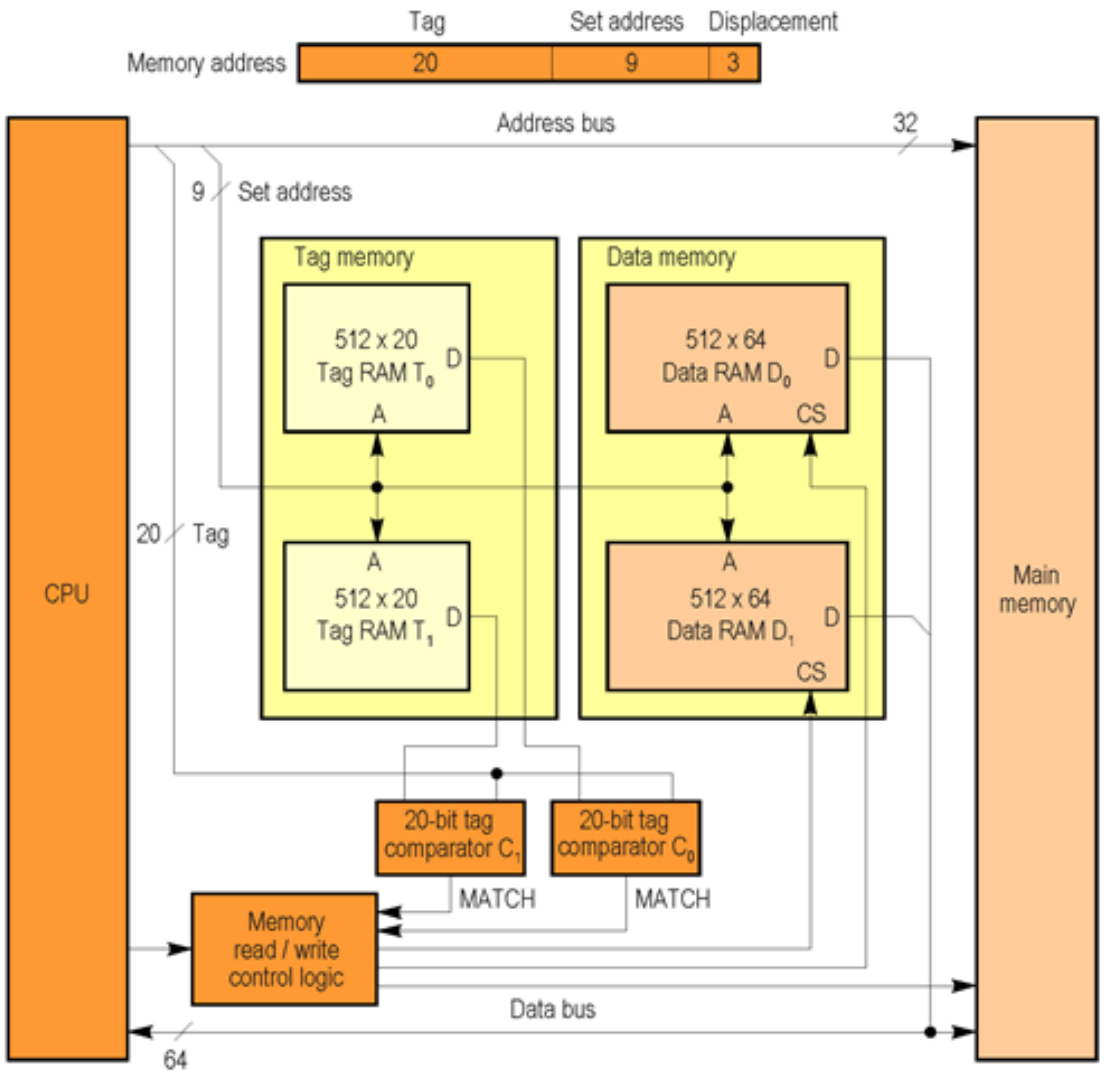


Figure 18. Block diagram of a 2-way set-associative cache memory example

3. Applications

3.1 Design in VHDL a dynamic memory using the structural model. Follow the design parameters mentioned above, as well as the block diagrams. Simulate the design and run it on a FPGA board.

3.2 Design in VHDL a cache memory (direct-mapping type or set-associative type). Use the design parameters from the examples above, as well as the block diagrams. Simulate and run the design of a FPGA board.