0.4.4 m-way set-associative mapping

In direct mapping of Figure 26, 4K to 1 mapping is considered.

If we realize m-way associative mapping for it, then mapping is $m\times$4K to m.

In Figure 28, a 2-way set-associative mapping is shown.

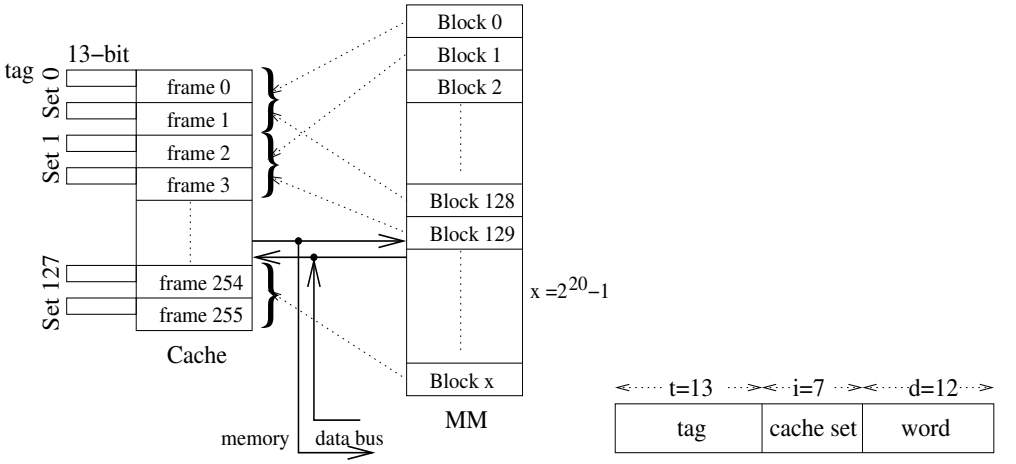


Figure 28: m-way set associative mapping technique

Here, number sets is = 128 = 2^7.

That is,

$\frac{2^{20}}{2^7}$ = 2^{13} MM blocks are competing for a set. This implies tag t = 13 (Figure 28).

The set is also called *index* field.

If m is 4 to 8, its effeciency is almost equivalent as associative mapping.

Direct mapping is an 1-way set associative mapping mapping.

To speed up, in Figure 29, index is compared with two tags of a set.

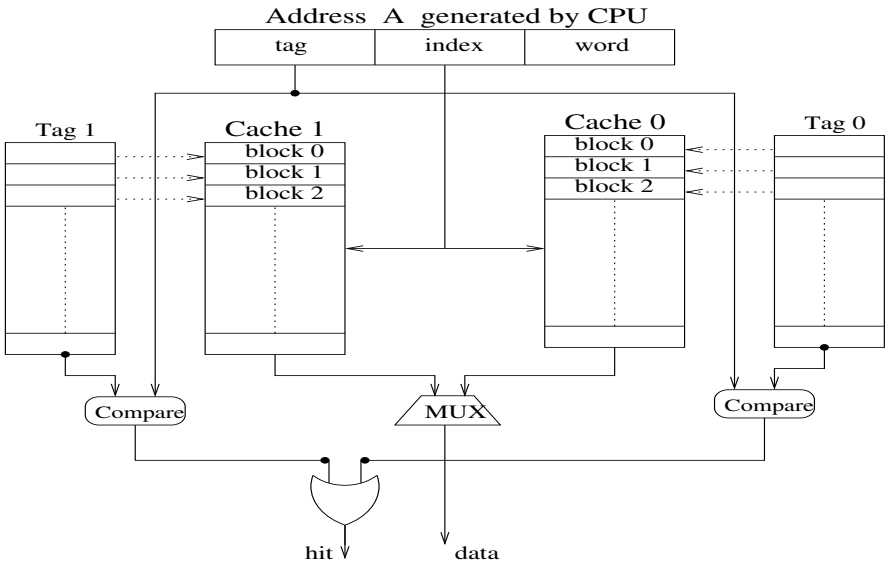


Figure 29: Architecture implementing 2-way set associative mapping scheme

Comparison of mapping techniques is shown in Figure 30.

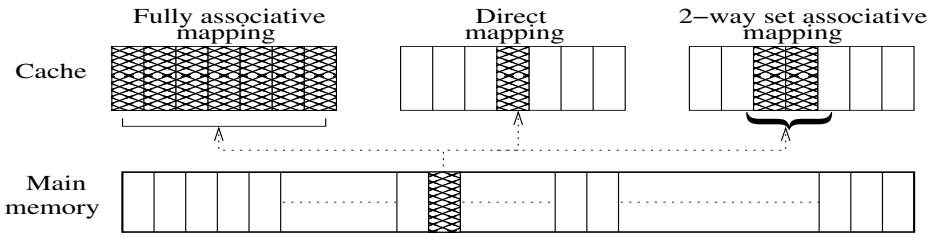


Figure 30: Cmparison of mapping techniques

0.4.5 Replacement algorithms

Cache system implementing direct mapping does not require replacement policy.

There is no other option but replace the block in cache frame selected for placement.

For set associative and fully associative mapping, we need replacement policy.

Three algorithms are considered for such replacement of blocks:

(i) Random replacement,

(ii) FIFO, and

(iii) LRU (least recently used block is replaced) or an approximation.

0.4.6 Cache write policies

It determines when an update to a cache word is forwarded to the main memory.

Two cache write policies - write-back and write-through.

a) *write-back*: Update to block B in cache is written in MM only when replacing B from cache.

It avoids miss penalty for every write operation.

b) *write-through*: While writing B at cache, similar write operation is performed in MM. That is, each write operation in cache is the write miss

The blocks in MM are the latest updated blocks.

Write through is very effective in multi-processor system.

However, successive operations will result in a bottleneck.

To exploit efficiency of write-through, faster implementation of write-through is done with introduction of a buffer in between cache and MM (Figure 31).

Typically, a write buffer is of 4/5 blocks.

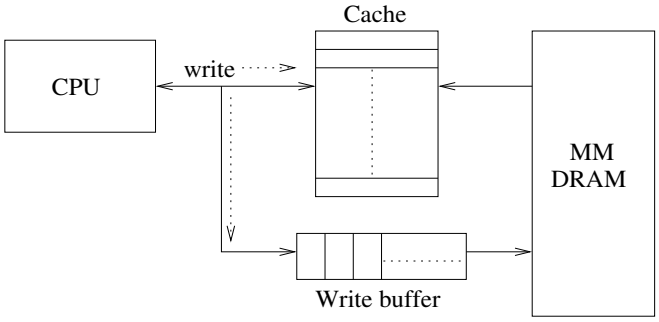


Figure 31: Write buffer

0.5 PRAM

CPU can float addresses much faster than settling time of a memory cell.

For multiple access, CPU generates $addr_1$, $addr_2$, $\cdots$, $addr_i$, $addr_{i+1}$, $\cdots$.

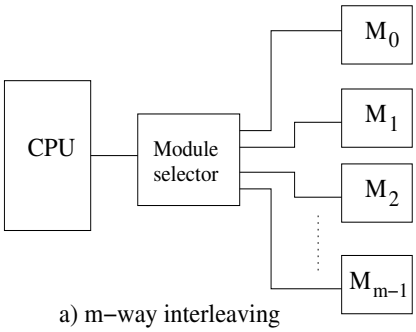
There is single decoding logic - overlapping of requests is difficult to realize.

PRAM (parallel RAM) allows access to more than one locations in a cycle.

Ideal PRAM is not realized. Different organizations of PRAM are proposed.

0.6 Interleaved Memories

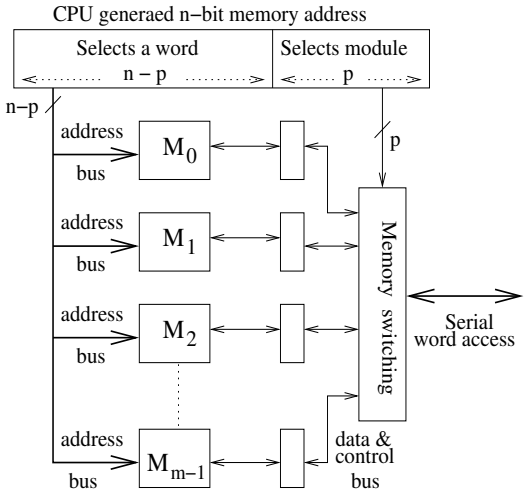
Interleaved memories allow simultaneous access from from main memory.



a) m−way interleaving

Memory address (n)	Module (p)	Word (n−p)
0000.....01 00	0	1
0000.....01 01	1	1
0000.....01 10	2	1
0000.....01 11	3	1
0000.....10 00	0	2
0000.....10 01	1	2

c) Address mapping (m=4)



b) Memory module addressing

Figure 32: Memory interleaving

Memory is partitioned into m separate modules M_0, M_1, $\cdots$, M_{m-1} (Figure 32(a)).

Each module is provided with its own addressing circuitry.

Consecutive memory references A_i and A_{i+1} fall on to different modules (M_j/M_{j+1}).

For m modules, it is m-way interleaving.

0.7 Large Memory Word

Large word size was considered at the early phase of our modern computer.

The recent development is the VLIW (very large instruction word).

In VLIW, in a cycle, a long memory word is fetched.

Performance of Memory

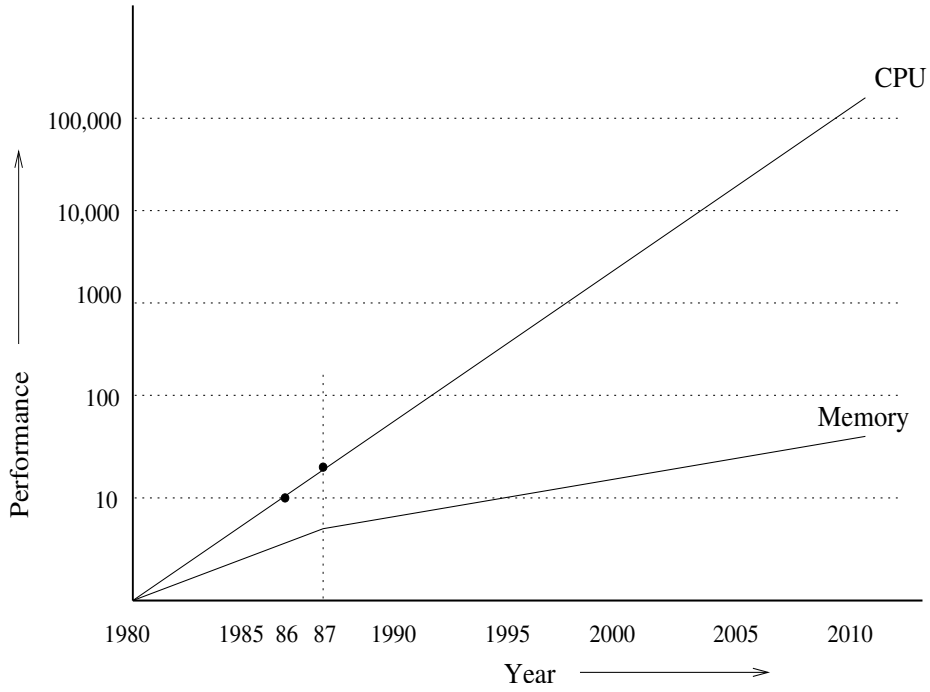


Figure 33: Memory performance

Memory hierarchy

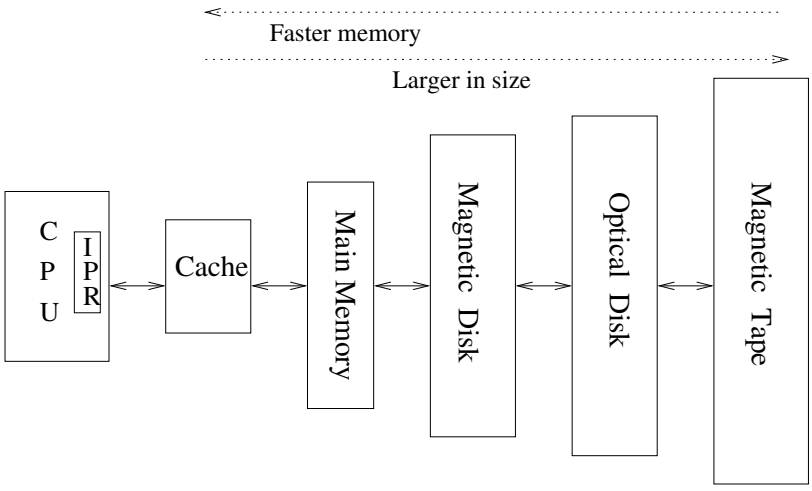


Figure 34: Memory hierarchy

Virtual Memory

Virtial memory (VM) system gives programmer an illusion that he/she has a very large memory at his/her disposal, even though m/c has a relatively small MM.

For example, let a CPU with 32 address lines is having 1MB actual MM. The virtual address space of the CPU is 2^{32}.

A mechanism is needed to map 32-bit VM address to a physical address of 20-bit.