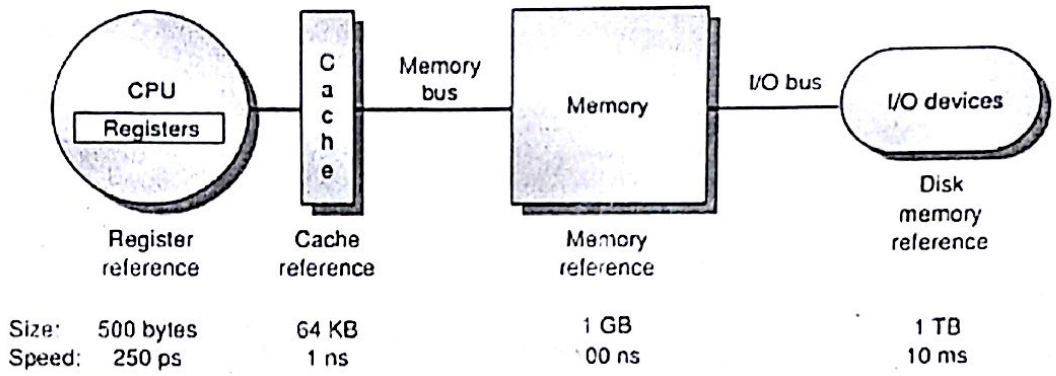
7c(B)

MEMORY HIERARCHY DESIGN

PROGRAMMERS WOULD WANT UNLIMITED AMOUNT OF FAST MEMORY

- Computer pioneers correctly predicted that Programmers would want unlimited amount of fast memory. An economical solution to that desire is a memory hierarchy, which takes advantage of locality and cost-performance of memory technologies. The goal is to provide a memory system with cost almost as low as the cheapest level of memory and speed almost as fast as the fastest level.

THE LEVELS IN A TYPICAL MEMORY HIERARCHY



CACHE

- Cache is the name given to the first level of the memory hierarchy encountered once the address leaves the CPU.

- Cache Hit- when CPU finds a requested data item in the cache, it is called a cache hit.

- Cache Miss- when the CPU does not find a data item it needs in the cache, a cache miss occurs.

How cache work

- A fixed size collection of data containing the requested word, called a block, is retrieved from the main memory and placed into the cache. Temporal locality tells us that we are likely to need this word again in the near future, so it is useful to place it in the cache where it can be accessed quickly. Because of spatial locality, there is a high probability that the other data in the block will be needed soon. ,

Cache Performance

- Memory Stall Cycle- the number of cycles during which the CPU is stalled waiting for a memory access is called the memory stall cycle.
- The performance is then the product of the clock cycle time and the sum of CPU cycles and the memory stall cycles:

CPU execution time = (CPU clock cycles + Memory stall cycles) × Clock Cycle time

This equation assumes that the CPU clock cycle include the time to handle a cache hit, and that the CPU is stalled during a cache miss.

The number of memory stall cycle depends on both the number of misses and the cost per miss, which is called the miss penalty.

Memory stall cycle = Number of misses × Miss penalty

$$= IC \times \frac{\text{Memory accesses}}{\text{instruction}} \times \text{Miss rate} \times \text{Miss penalty}$$

CACHE PERFORMANCE (MATH)

Example: assume we have a computer where the clocks per instruction (CPI) is 1.0 when all memory accesses hit in the cache. The only data accesses are loads and stores, and these total 50% of the instructions. If the miss penalty is 25 clock cycles and the miss rate is 2%, how much faster would the computer be if all instruction were cache hits?

Answer: first compute the performance for the computer that always hits:

$$\begin{aligned}\text{CPU execution time} &= (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock Cycle time} \\ &= (\text{IC} \times \text{CPI} + 0) \times \text{Clock Cycle time} \\ &= \text{IC} \times 1.0 \times \text{Clock Cycle time}\end{aligned}$$

Now for the computer with the real cache, first we compute memory stall cycles:

$$\begin{aligned}\text{Memory stall cycle} &= \text{IC} \times \frac{\textit{Memory accesses}}{\textit{instruction}} \times \text{Miss rate} \times \text{Miss penalty} \\ &= \text{IC} \times (1 + 0.5) \times 0.02 \times 25 \\ &= \text{IC} \times 0.75\end{aligned}$$

(here (1 + 0.5) represents 1 instruction access and 0.5 data access per instruction)

CACHE PERFORMANCE (MATH CONT.)

Thus total performance,

$$\begin{aligned}\text{CPU execution time}_{\text{cache}} &= (\text{IC} \times 1.0 + \text{IC} \times 0.75) \times \text{Clock Cycle time} \\ &= \text{IC} \times 1.75 \times \text{Clock Cycle time}\end{aligned}$$

Now performance ratio is the inverse of the execution time,

$$\frac{\text{CPU execution time}_{\text{cache}}}{\text{CPU execution time}} = \frac{\text{IC} \times 1.75 \times \text{Clock Cycle time}}{\text{IC} \times 1.0 \times \text{Clock Cycle time}}$$

$$= 1.75$$

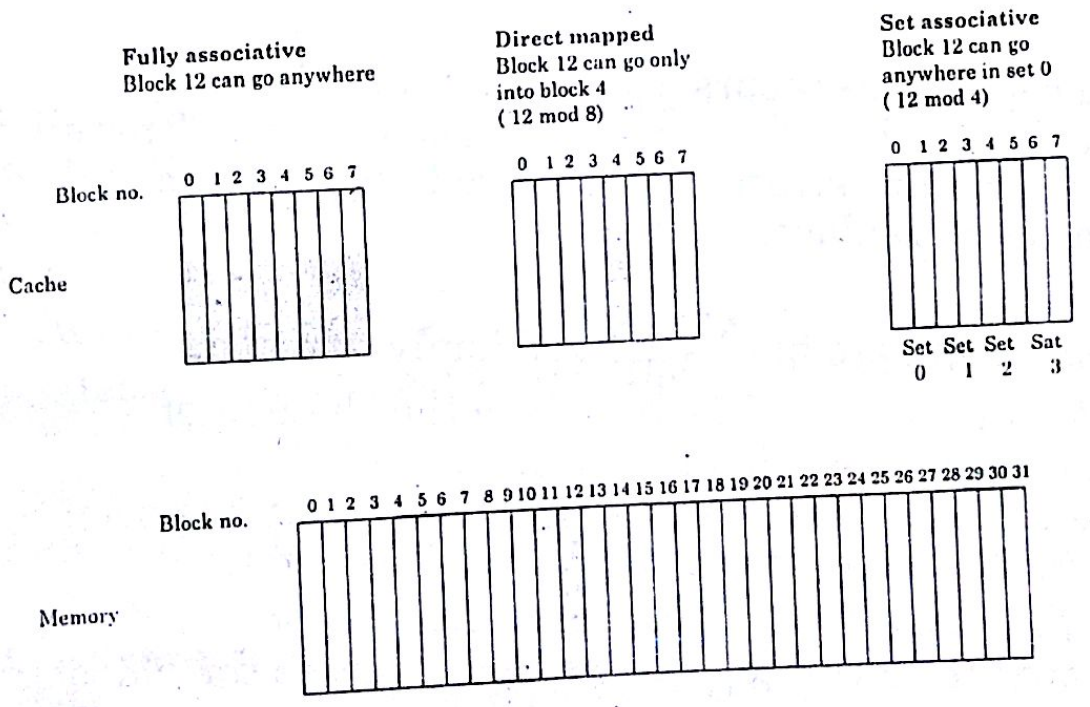
The computer with no cache misses is 1.75 times faster.

WHERE CAN A BLOCK BE PLACED IN A CACHE?

There are three categories of cache organization based on the restriction on where a block is placed.

1. *Fully associative* – if a block can be places anywhere in the cache, the cache is said to be fully associative.
2. *Direct mapped* – if each block has only one place it can appear in the cache, the cache is said to be direct mapped. The mapping is usually
   (*Block address*) MOD (*Number of blocks in the cache*)
3. *Set associative* – if a block can be placed in a restricted set of places in the cache, the cache is called set associative. A set is a group of blocks in the cache. Block can be placed anywhere within the set. Set is usually chosen by bit selection, that is,
   (*Block address*) MOD (*Number of sets in the cache*)

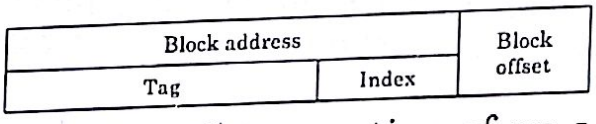
WHERE CAN A BLOCK BE PLACED IN A CACHE?(CONT.)



This example cache has eight block frames and memory has 32 blocks

How is a block found if it is in the cache?

Caches have an address tag on each block frame that gives the block address. The tag of every cache block that might contain the desired information is checked to see if it matches the block address from CPU.



the figure shows the three portion of an address in a set associative or direct-mapped cache. The tag is used to check all the blocks in the set and the index is used to select the set. The block offset is the address of desired data within the block. Fully associative cache have no index field.

Which block should be replaced on a cache miss?

When a cache miss occurs, the cache controller must select a block to be replace with the desired data. There are three primary strategies employed for selecting which block to replace:

Random- candidate blocks are randomly selected.

Least Recent Used (LRU) – to reduce the chances of throwing out information that will be needed soon, accesses to blocks are recorded. The block that has been unused to the longest time will be replaced first.

Firs in, First out (FIFO) – because LRU can be complicated to calculate, this method replaces the oldest block first.

WHAT HAPPENS ON A WRITE?

There are two basic options when writing to the cache:

- *Write through*- the information is written on both the block in the cache and the block in the lower level memory.
- *Write back* – the information is written only to the block in the cache. The modified cache is written to main memory only when it is replaced.

Dirty Bit – to reduce the frequency of writing back blocks on replacement, a feature called dirty bit is commonly used. This status bit indicates whether the block is dirty (modified in the cache) or clean (not modified). If it is clean, the block is not written back on a miss, since identical information to the cache is found in lower levels.

COMPARISON BETWEEN WRITE BACK AND WRITE THROUGH?

Both write back and write through have their advantages.

Write back –

- Write occurs at the speed of cache memory.
- Multiple write within a block requires only one write to the lower level memory.
- Uses less memory bandwidth.
- Saves power.

Write through-

- Is easier to implement than write back.
- Simplified data coherency.

SIX BASIC CACHE OPTIMIZATION

1. Larger block size to reduce miss rate.
2. Bigger cache to reduce miss rate.
3. Higher associativity to reduce miss rate.
4. Multilevel cache to reduce miss penalty.
5. Giving priority to read miss over write miss to reduce miss penalty.
6. Avoiding address translation during indexing of the cache to reduce hit time.