Structure of Large Memories

- Memory chips are connected to form a much larger memory

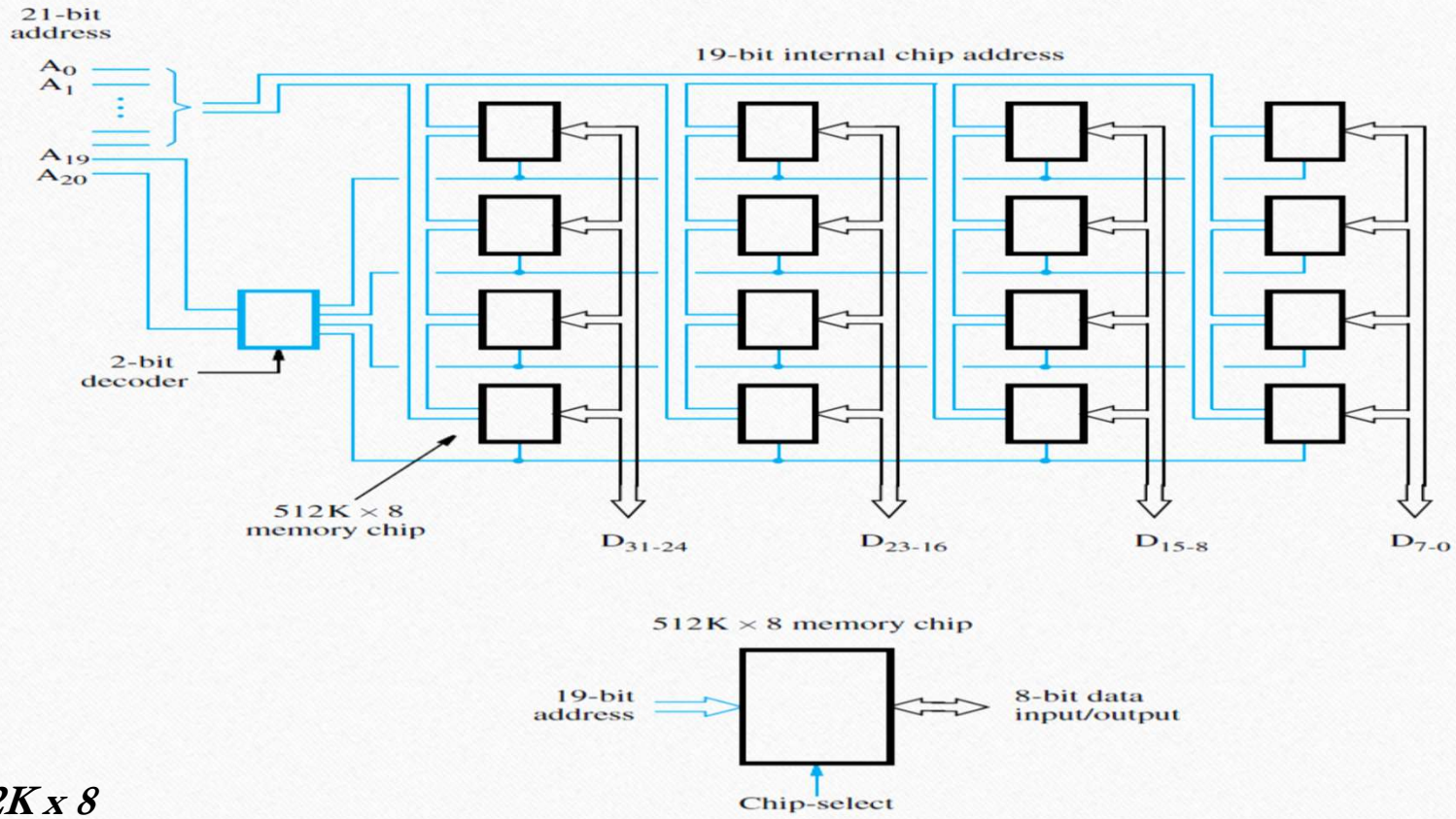


Figure 5.13: Organization of 2Mx32 memory module using 512K x 8 static memory chips

Structure of Large Memories

- Consider a memory consisting of 2M words of 32 bits each. Figure shows how this memory can be implemented using 512K × 8 static memory chips
- Each column in the figure implements one byte position in a word, with four chips providing 2M bytes
- Four columns implement the required 2M × 32 memory
- Each chip has a control input called Chip-select
- When this input is set to 1, it enables the chip to accept data from or to place data on its data lines

Structure of Large Memories

- The data output for each chip is of the tri-state type
- Only the selected chip places data on the data output line, while all other outputs are electrically disconnected from the data lines
- Twenty-one address bits are needed to select a 32-bit word in this memory
- The high-order two bits of the address are decoded to determine which of the four rows should be selected
- The remaining 19 address bits are used to access specific byte locations inside each chip in the selected row
- The R/$\overline{W}$ inputs of all chips are tied together to provide a common Read/$\overline{\text{Write}}$ control line

Direct Memory Access

- Blocks of data are often transferred between the main memory and I/O devices such as disks
- Data are transferred from an I/O device to the memory by first reading them from the I/O device using an instruction such as

  Load R2, DATAIN which loads the data into a processor register
- Then, the data read are stored into a memory location
- The reverse process takes place for transferring data from the memory to an I/O device
- An instruction to transfer input or output data is executed only after the processor determines that the I/O device is ready, either by polling its status register or by waiting for an interrupt request
- In either case, considerable overhead is incurred, because several program instructions must be executed involving many memory accesses for each data word transferred

Direct Memory Access

- An approach is used to transfer blocks of data directly between the main memory and I/O devices, such as disks
- A special control unit is provided to manage the transfer, without continuous intervention by the processor. This approach is called direct memory access, or DMA
- The unit that controls DMA transfers is referred to as a DMA controller
- It may be part of the I/O device interface, or it may be a separate unit shared by a number of I/O devices

Direct Memory Access

- The DMA controller performs the functions that would normally be carried out by the processor when accessing the main memory
- For each word transferred, it provides the memory address and generates all the control signals needed
- It increments the memory address for successive words and keeps track of the number of transfers.

Direct Memory Access

- Although a DMA controller transfers data without intervention by the processor, its operation must be under the control of a program executed by the processor, usually an operating system routine
- To initiate the transfer of a block of words, the processor sends to the DMA controller the starting address, the number of words in the block, and the direction of the transfer
- The DMA controller then proceeds to perform the requested operation
- When the entire block has been transferred, it informs the processor by raising an interrupt.

Direct Memory Access

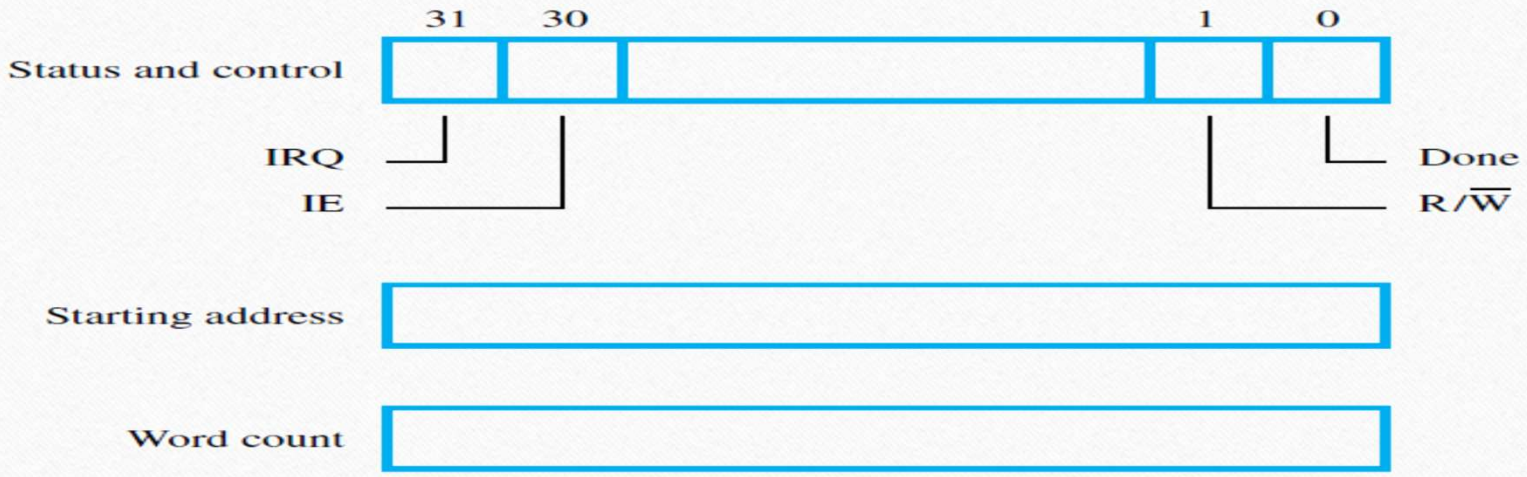


Figure 5.14: Typical Registers in a DMA Controller

Direct Memory Access

- Two registers are used for storing the starting address and the word count
- The third register contains status and control flags
- The R/$\overline{W}$ bit determines the direction of the transfer
- When this bit is set to 1 by a program instruction, the controller performs a Read operation, that is, it transfers data from the memory to the I/O device
- Otherwise, it performs a Write operation
- Additional information is also transferred as may be required by the I/O device
- For example, in the case of a disk, the processor provides the disk controller with information to identify where the data is located on the disk

Direct Memory Access

- When the controller has completed transferring a block of data and is ready to receive another command, it sets the Done flag to 1
- Bit 30 is the Interrupt-enable flag, IE
- When this flag is set to 1, it causes the controller to raise an interrupt after it has completed transferring a block of data
- Finally, the controller sets the IRQ bit to 1 when it has requested an interrupt.

Direct Memory Access

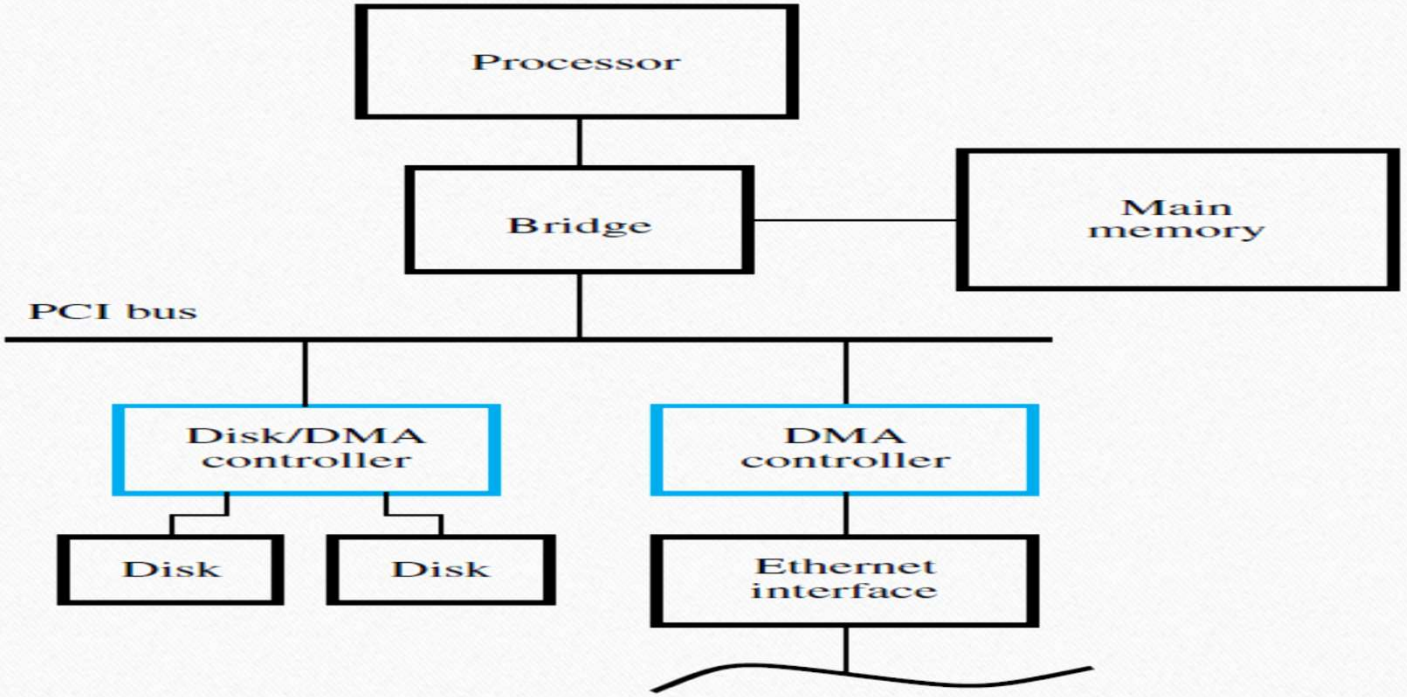


Figure 5.15 Use of DMA controllers in computer system

Direct Memory Access

- Figure 5.15 shows how DMA controllers may be used in a computer system
- One DMA controller connects a high-speed Ethernet to the computer's I/O bus
- The disk controller, which controls two disks, also has DMA capability and provides two DMA channels
- It can perform two independent DMA operations, as if each disk had its own DMA controller
- The registers needed to store the memory address, the word count, and so on, are duplicated, so that one set can be used with each disk.

Direct Memory Access

- To start a DMA transfer of a block of data from the main memory to one of the disks, an OS routine writes the address and word count information into the registers of the disk controller
- The DMA controller proceeds independently to implement the specified operation
- When the transfer is completed, this fact is recorded in the status and control register of the DMA channel by setting the Done bit
- At the same time, if the IE bit is set, the controller sends an interrupt request to the processor and sets the IRQ bit
- The status register may also be used to record other information, such as whether the transfer took place correctly or errors occurred.

Memory Hierarchy

- Programmers want unlimited amount of memory with very low latency.
- Fast memory technology is more expensive per bit than slower memory.
  – SRAM is more expensive than DRAM, DRAM is more expensive than disk.
- Possible solution?
  – Organize the memory system in several levels, called memory hierarchy.
  – Exploit temporal and spatial locality on computer programs.
  – Try to keep the commonly accessed segments of program / data in the faster memories.
  – Results in faster access times on the average.

Memory Hierarchy

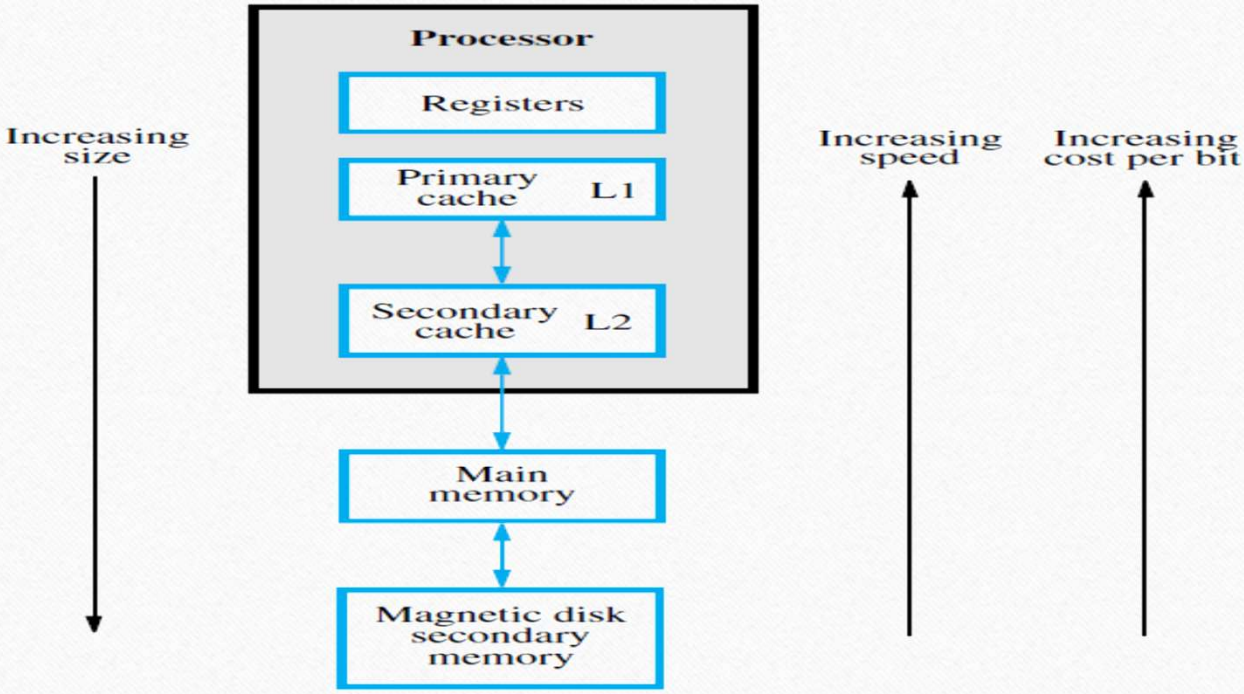


Figure 5.16 Memory Hierarchy

Memory Hierarchy

- Different types of memory units are employed effectively in a computer system
- The entire computer memory can be viewed as the hierarchy depicted in Figure 5.16
- The fastest access is to data held in processor registers. Therefore, if we consider the registers to be part of the memory hierarchy, then the processor registers are at the top in terms of speed of access
- At the next level of the hierarchy is a relatively small amount of memory that can be implemented directly on the processor chip
- This memory, called a processor cache, holds copies of the instructions and data stored in a much larger memory that is provided externally.
- There are often two or more levels of cache. A primary cache is always located on the processor chip
- This cache is small and its access time is comparable to that of processor registers.

Memory Hierarchy

- The primary cache is referred to as the level 1 (L1) cache
- A larger, and hence somewhat slower, secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache .
- Often, the L2 cache is also housed on the processor chip
- Some computers have a *level 3* (L3) cache of even larger size, in addition to the L1 and L2 caches
- An L3 cache, also implemented in SRAM technology, may or may not be on the same chip with the processor and the L1 and L2 caches
- The next level in the hierarchy is the *main memory*
- This is a large memory implemented using dynamic memory components, typically assembled in memory modules.
- The main memory is much larger but significantly slower than cache memories

Memory Hierarchy

- In a computer with a processor clock of 2 GHz or higher, the access time for the main memory can be as much as 100 times longer than the access time for the L1 cache
- Disk devices provide a very large amount of inexpensive memory, and they are widely used as secondary storage in computer systems
- They are very slow compared to the main memory. They represent the bottom level in the memory hierarchy.

Cache Memories

- The cache is a small and very fast memory, interposed between the processor and the main memory
- Its purpose is to make the main memory appear to the processor to be much faster than it actually is
- The effectiveness of this approach is based on a property of computer programs called *locality of reference*

Cache Memories

- Analysis of programs shows that most of their execution time is spent in routines in which many instructions are executed repeatedly.
- These instructions may constitute a simple loop, nested loops, or a few procedures that repeatedly call each other
- The actual detailed pattern of instruction sequencing is not important—the point is that many instructions in localized areas of the program are executed repeatedly during some time period.
- This behavior manifests itself in two ways: temporal and spatial

Cache Memories

- The temporal means that a recently executed instruction is likely to be executed again very soon
- The spatial aspect means that instructions close to a recently executed instruction are also likely to be executed soon.

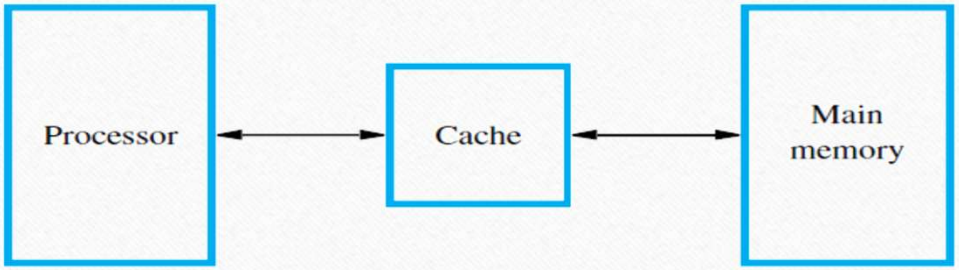


Figure 5.17 Use of a cache memory

Cache Memories

- Conceptually, operation of a cache memory is very simple
- The memory control circuitry is designed to take advantage of the property of locality of reference
- Temporal locality suggests that whenever an information item, instruction or data, is first needed, this item should be brought into the cache, because it is likely to be needed again soon
- Spatial locality suggests that instead of fetching just one item from the main memory to the cache, it is useful to fetch several items that are located at adjacent addresses as well.

Cache Memories

- The term cache block refers to a set of contiguous address locations of some size
- Another term that is often used to refer to a cache block is a cache line
- When the processor issues a Read request, the contents of a block of memory words containing the location specified are transferred into the cache
- Subsequently, when the program references any of the locations in this block, the desired contents are read directly from the cache

Cache Memories

- Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory
- The correspondence between the main memory blocks and those in the cache is specified by a *mapping function*
- When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word
- The collection of rules for making this decision - cache's *replacement algorithm*

Cache Hits

- The processor does not need to know explicitly about the existence of the cache
- It simply issues Read and Write requests using addresses that refer to locations in the memory
- The cache control circuitry determines whether the requested word currently exists in the cache
- If it does, the Read or Write operation is performed on the appropriate cache location.
- In this case, a read or write hit is said to have occurred. The main memory is not involved when there is a cache hit in a Read operation

Cache Hits –Write Operation

Write Through/Store Through Protocol

- Information is written to both the cache block and the main memory block.
- Features:
  - Easier to implement
  - Read misses do not result in writes to the lower level(i.e. MM).
  - The lower level(i.e. MM) has the most updated version of the data – important for I/O operations and multiprocessor systems.

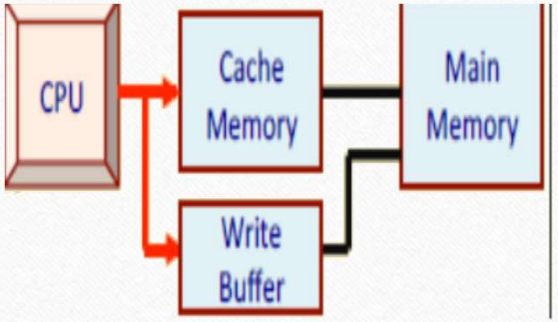
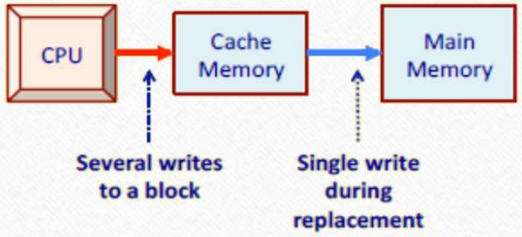


Figure 5.18 Write Through

Cache Hits –Write Operation

Write Back Protocol

- Information is written only to the cache block.
- A modified cache block is written to Main Memory only when it is replaced.
- Features:
  - – Writes occur at the speed of cache memory.
  - – Multiple writes to a cache block requires only one write to Main Memory
  - – Uses less memory bandwidth, makes it attractive to multiprocessors.



- Write-back cache blocks can be *clean* or *dirty*.
  - – A status bit called *dirty bit* or *modified bit* is associated with each cache block, which indicates whether the block was modified in the cache (0: clean, 1: dirty).
  - – If the status is *clean*, the block is not written back to Main Memory while being replaced.

Cache Hits –Write Operation

- The write-through protocol is simpler than the write-back protocol, but it results in unnecessary Write operations in the main memory when a given cache word is updated several times during its cache residency
- The write-back protocol also involves unnecessary Write operations, because all words of the block are eventually written back, even if only a single word has been changed while the block was in the cache
- The write-back protocol is used most often, to take advantage of the high speed with which data blocks can be transferred to memory chips.

Cache Misses

- A Read operation for a word that is not in the cache constitutes a *Read miss*
- It causes the block of words containing the requested word to be copied from the main memory into the cache
- After the entire block is loaded into the cache, the particular word requested is forwarded to the processor
- Alternatively, this word may be sent to the processor as soon as it is read from the main memory.
- The latter approach, which is called *load-through*, or *early restart*, reduces the processor's waiting time somewhat, at the expense of more complex circuitry.

Cache Misses

- When a *Write miss* occurs in a computer that uses the write-through protocol, the information is written directly into the main memory
- For the write-back protocol, the block containing the addressed word is first brought into the cache, and then the desired word in the cache is overwritten with the new information.

Mapping Functions

- Consider a single-level cache, and that part of the memory hierarchy consisting of cache memory and main memory
- Cache memory is logically divided into blocks or lines, where every block (line) typically contains 8 to 256 bytes.
- When the CPU wants to access a word in memory, a special hardware first checks whether it is present in cache memory.
  - – If so (called cache hit), the word is directly accessed from the cache memory.
  - – If not, the block containing the requested word is brought from main memory to cache.
  - – For writes, sometimes the CPU can also directly write to main memory.
- Objective is to keep the commonly used blocks in the cache memory.
  - – Will result in significantly improved performance due to the property of locality of reference.

Mapping Functions

- Where a block can be placed in cache is determined by some mapping algorithms
  - Specifies which main memory blocks can reside in which cache memory blocks.
  - At any given time, only a small subset of the main memory blocks can be held in main memory.
- Three common block mapping techniques are used:

  a) Direct Mapping

  b) Associative Mapping

  c) (N-way) Set Associative Mapping

Example-2 Level Memory Hierarchy

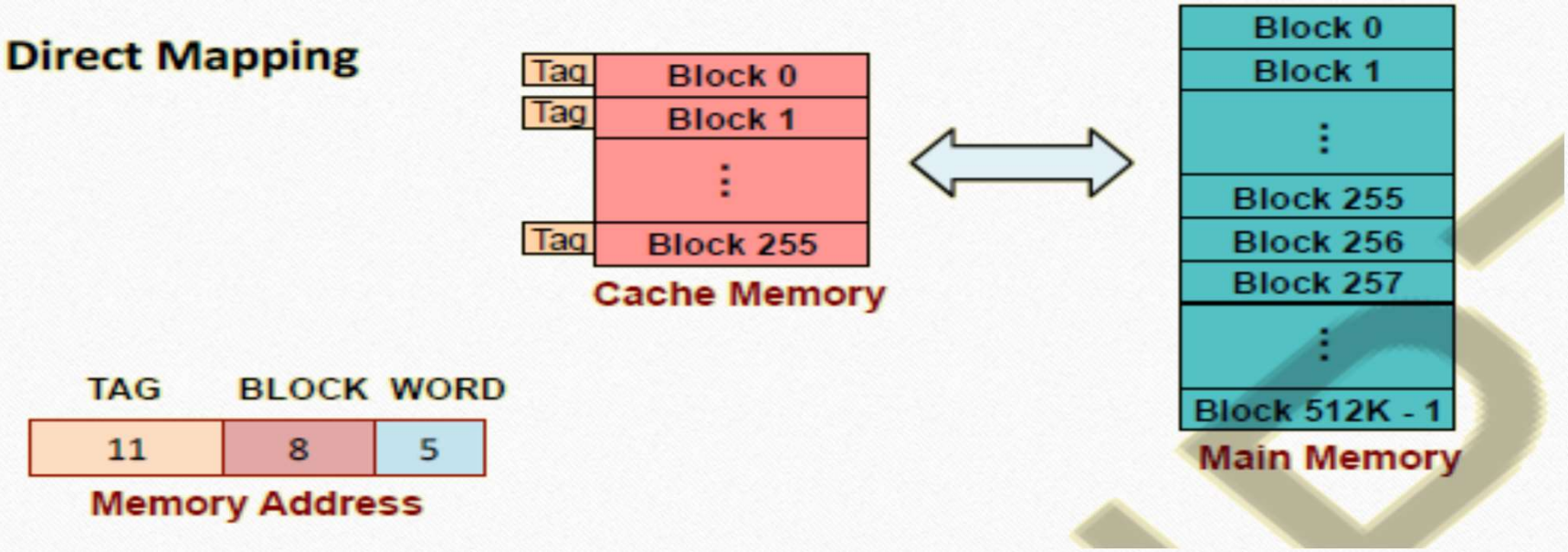
- Consider a 2-level cache memory / main memory hierarchy.
  - The cache memory consists of 256 blocks (lines) of 32 words each.
    - Total cache size is 8192 (8K) words.
- Main memory is addressable by a 24-bit address.
  - Total size of the main memory is 2^{24} = 16 M words.
  - Number of 32-word blocks in main memory = 16 M / 32 = 512K

Direct Mapping

- Each main memory block can be placed in only one block in the cache.
- The mapping function is:

  Cache Block = (Main Memory Block) % (Number of cache blocks)
- For the example,
  - Cache Block = (Main Memory Block) % 256
- Some example mappings:
  - 0 →0, 1 → 1, 255 → 255, 256 → 0, 257 → 1, 512 → 0, 513 → 1, etc.

Direct Mapping



Direct Mapping

- Block replacement algorithm is trivial, as there is no choice.
- More than one MM block is mapped onto the same cache block.
  - May lead to contention even if the cache is not full.
  - New block will replace the old block.
  - May lead to poor performance if both the blocks are frequently used.
- The MM address is divided into three fields: TAG, BLOCK and WORD.
  - When a new block is loaded into the cache, the 8-bit BLOCK field determines the cache block where it is to be stored.
  - The high-order 11 bits are stored in a TAG register associated with the cache block.
  - When accessing a memory word, the corresponding TAG fields are compared.
    - Match implies HIT

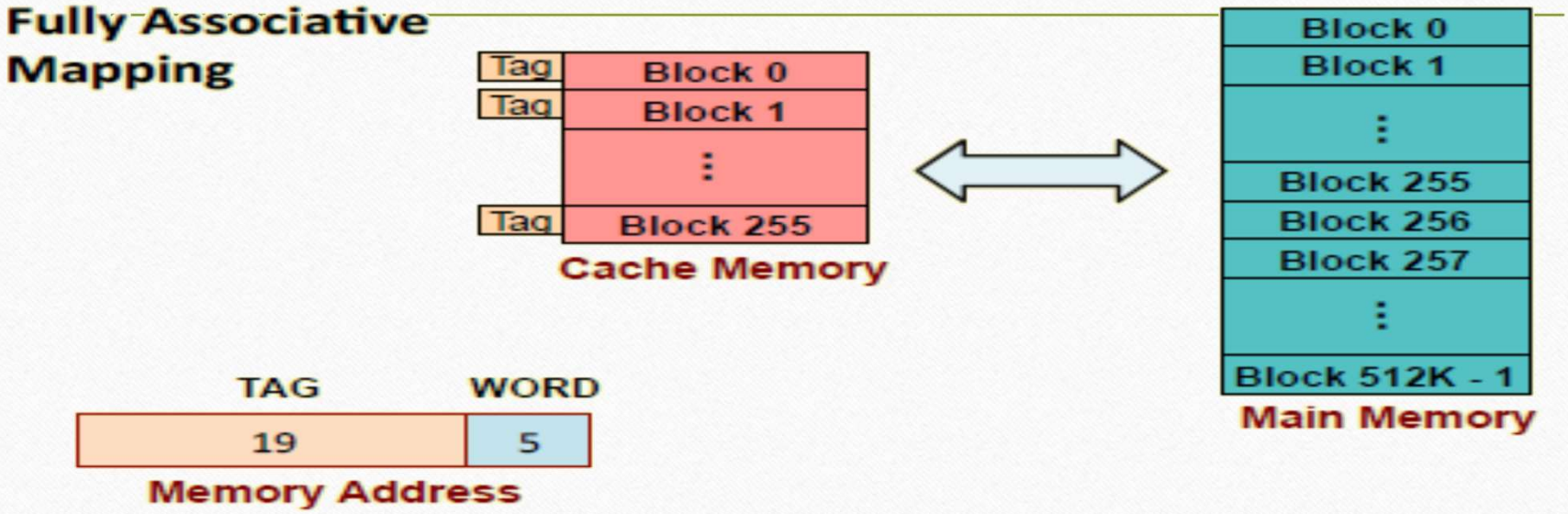
Associative Mapping

- Here, a MM block can potentially reside in any cache block position.
- The memory address is divided into two fields: TAG and WORD.

  – When a block is loaded into the cache from MM, the higher order 19 bits of the address are stored into the TAG register corresponding to the cache block.

  - When accessing memory, the 19-bit TAG field of the address is compared with all the TAG registers corresponding to all the cache blocks.
- Requires associative memory for storing the TAG values
  - – High cost / lack of scalability.
- Because of complete freedom in block positioning, a wide range of replacement algorithms is possible.

Associative Mapping



N-way Set Associative Mapping

- A group of N consecutive blocks in the cache is called a set.
- This algorithm is a balance of direct mapping and associative mapping.
  - Like direct mapping, a MM block is mapped to a set.
    - Set Number = (MM Block Number) % (Number of Sets in Cache)
  - The block can be placed anywhere within the set (there are N choices)
- The value of N is a design parameter:
  - N = 1 :: same as direct mapping.
  - N = number of cache blocks :: same as associative mapping.
- – Typical values of N used in practice are: 2, 4 or 8.

N-way Set Associative Mapping

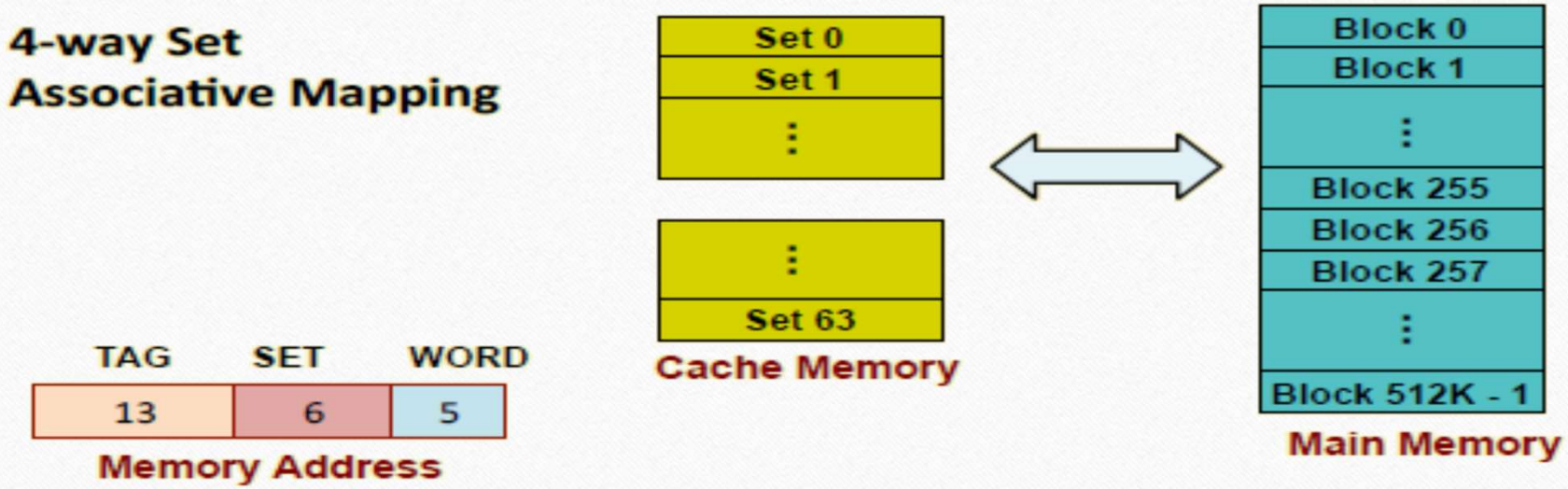


Illustration for N = 4

- Number of sets in cache memory = 64.
- Memory blocks are mapped to a set using modulo-64 operation.
- Example: MM blocks 0, 64, 128, etc. all map to set 0, where they can occupy any of the four available positions.
- MM address is divided into three fields: TAG, SET and WORD.
  - The TAG field of the address must be associatively compared to the TAG fields of the 4 blocks of the selected set.
  - This instead of requiring a single large associative memory, we need a number of very small associative memories only one of which will be used at a time.

How is a block found if present in cache?

- Caches include a TAG associated with each cache block.
  - The TAG of every cache block where the block being requested may be present needs to be compared with the TAG field of the MM address.
  - All the possible tags are compared in parallel, as speed is important.
- Mapping Algorithms?
  - Direct mapping requires a single comparison.
  - Associative mapping requires a full associative search over all the TAGs corresponding to all cache blocks.
  - Set associative mapping requires a limited associated search over the TAGs of only the selected set.

How is a block found if present in cache?

- Use of valid bit:
  - There must be a way to know whether a cache block contains valid or garbage information.
  - A valid bit can be added to the TAG, which indicates whether the block contains valid data.
  - If the valid bit is not set, there is no need to match the corresponding TAG

Cache Replacement

- With fully associative or set associative mapping, there can be several blocks to choose from for replacement when a miss occurs.
- Two primary strategies are used:
  a) **Random**: The candidate block is selected randomly for replacement. This simple strategy tends to spread allocation uniformly.

  b) Least Recently Used (LRU): The block replaced is the one that has not been used for the longest period of time.
- Makes use of a corollary of temporal locality: *"If recently used blocks are likely to be used again, then the best candidate for replacement is the least recently used block"*

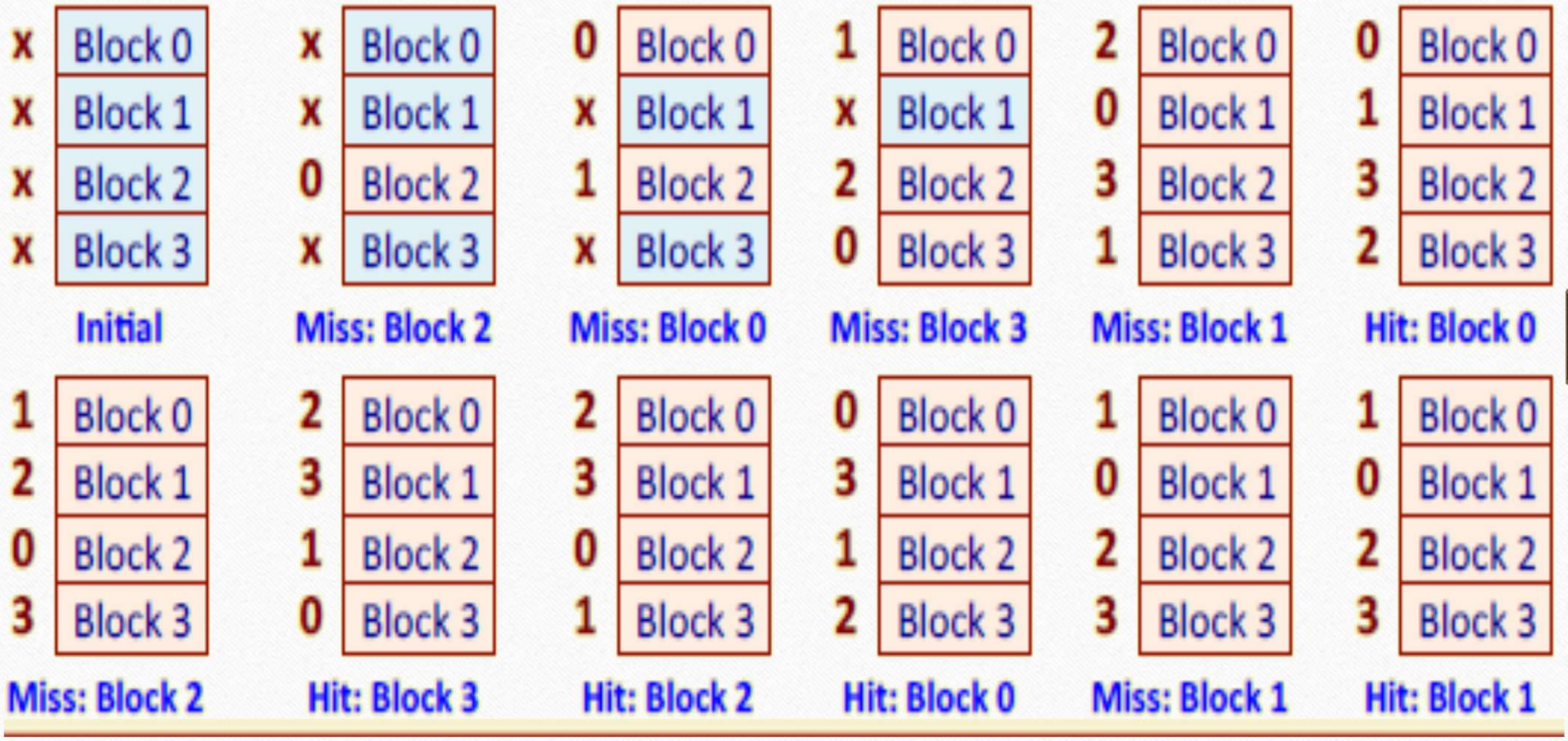
LRU

- To implement the LRU algorithm, the cache controller must track the LRU block as the computation proceeds.
- Example: Consider a 4-way set associative cache.
  - For tracking the LRU block within a set, we use a 2-bit counter with every block.
  - When hit occurs:
    - Counter of the referenced block is reset to 0.
    - Counters with values originally lower than the referenced one are incremented by 1, and all others remain unchanged.
  - When miss occurs:
    - If the set is not full, the counter associated with the new block loaded is set to 0, and all other counters are incremented by 1.

LRU

- If the set is full, the block with counter value 3 is removed, the new block put in its place, and the counter set to 0. The other three counters are incremented by 1.
- It may be verified that the counter values of occupied blocks are all distinct.

Example


x Block 0
x Block 1
x Block 2
x Block 3
Initial
x Block 0
x Block 1
0 Block 2
x Block 3
Miss: Block 2
0 Block 0
x Block 1
1 Block 2
x Block 3
Miss: Block 0
1 Block 0
x Block 1
2 Block 2
0 Block 3
Miss: Block 3
2 Block 0
0 Block 1
3 Block 2
1 Block 3
Miss: Block 1
0 Block 0
1 Block 1
3 Block 2
2 Block 3
Hit: Block 0
1 Block 0
2 Block 1
0 Block 2
3 Block 3
Miss: Block 2
2 Block 0
3 Block 1
1 Block 2
0 Block 3
Hit: Block 3
2 Block 0
3 Block 1
0 Block 2
1 Block 3
Hit: Block 2
0 Block 0
3 Block 1
1 Block 2
2 Block 3
Hit: Block 0
1 Block 0
0 Block 1
2 Block 2
3 Block 3
Miss: Block 1
1 Block 0
0 Block 1
2 Block 2
3 Block 3
Hit: Block 1

Virtual Memory

- In a memory hierarchy system, programs and data are brought into main memory as they are needed by the CPU
- Virtual memory is a concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory
- Each address that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in main memory.

Virtual Memory

- Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory
- A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations
- This is done dynamically, while programs are being executed in the CPU
- The translation or mapping is handled automatically by the hardware by means of a **mapping table.**

Address Space and Memory Space

- An address used by a programmer will be called a **virtual address,** and the set of such addresses the address space
- An address in main memory is called a **location or physical address**
- The set of such locations is called the **memory space**
- Thus the address space is the set of addresses generated by programs as they reference instructions and data; the memory space consists of the actual main memory locations directly addressable for processing
- In most computers the address and memory spaces are identical.