Objectives

After studying this unit, you will be able to:

- Explain memory hierarchy
- Describe main memory
- Describe cache memory
- Define virtual memory

Introduction

“Ideally one would desire an indefinitely large memory capacity such that any particular word would be immediately available.

We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.”

A. W. Burks, H. H. Goldstine, and J. Von Neumann

Computer architects rightly predicted that programmers would want unlimited amounts of memory. A cost-effective way to provide large amounts of memory is by using a memory hierarchy, which takes advantage of locality and performance of memory technologies (such as SRAM, DRAM, and so on). The principle of locality states that most programs do not refer all code or data in a uniform manner. Locality occurs in time and in space. The belief that smaller hardware can be made faster, contributed to the development of memory hierarchies based on different speeds and sizes.

Notes

The significance of memory hierarchy has improved with advances in performance of processors. As the performance of the processor increases, the memory should also support this performance. Therefore, computer architects are working towards minimizing the processor-memory gap.

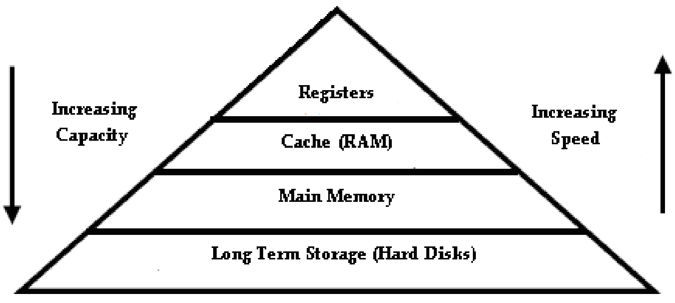
Memory Hierarchy

You may be aware that the data required to operate a computer is stored in the hard drive. However, other storage methods are also required. This requirement is for a number of reasons, but mostly it is because the hard drive is slow and executing programs using it could be impractical. When the processor requires data or functions, it first fetches the required data from the hard drive and then loads them into the main memory (RAM). This increases the operation speed and programs are executed faster.

Main memory and the hard drive form two levels of the computer's ***memory hierarchy***. In memory hierarchy, the storage devices are arranged in such a way that they take advantage of the characteristics of different storage technologies in order to improve the overall performance of a computer system.

Figure 12.1 shows the standard memory hierarchy of a computer.

Figure 12.1: Representation of Standard Memory Hierarchy



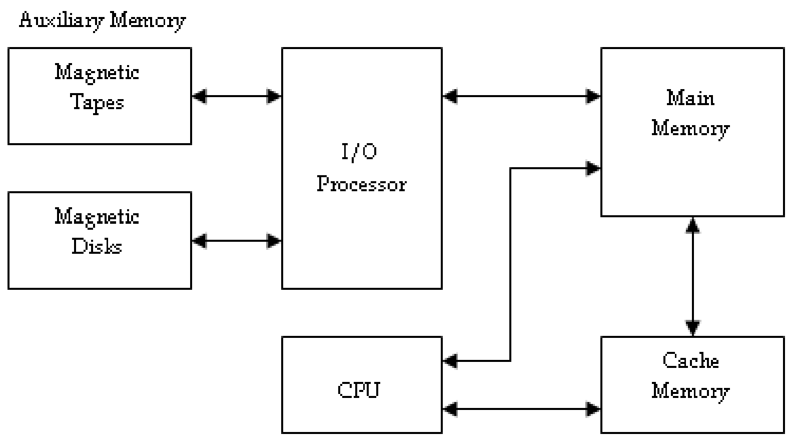
The memory unit is an essential component of any digital computer, because all the programs and data are stored here. A very small computer with a limited purpose may be able to accomplish its intended task without the need for additional storage capacity. Nearly all conventional computers would run more efficiently if they were provided with additional storage space, excluding the capacity of the main memory. It is not possible for one memory unit to accommodate all the programs used in a conventional computer due to lack of storage space. Additionally, almost all computer users collect and continue to amass large amount of data processing software. Not all information that is gathered is needed by the processor at the same time. Thus, it is more appropriate to use low-cost storage devices to serve as a backup for storing the information that is not currently used by the CPU.

The memory unit that exchanges information directly with the CPU is called the main memory. Devices that support backup storage are referred to as auxiliary memory. The most common devices that support storage in typical computer systems are magnetic disks and magnetic tapes. They are mainly used to store system programs, large data files, and other backup information. Only programs and data that are currently required by the processor reside in the main memory. All other information is stored in auxiliary/secondary memory and moved to the main memory when needed.

The overall memory capacity of a computer can be pictured as being a hierarchy of components. The memory hierarchy system includes all storage devices used in a computer system. They range from the slow but large capacity auxiliary memory to a comparatively faster main memory, to an even smaller but faster cache memory accessible to the high-speed processing logic.

Figure 12.2 shows the components in a typical memory hierarchy.

Figure 12.2: Memory Hierarchy in a Computer System



Placed at the bottom of the hierarchy is the comparatively slow magnetic tape used to store removable files. Next is the magnetic disk that is used as backup storage.

The main memory takes up a central position because of its ability to communicate directly with the CPU and with auxiliary memory devices, through an Input/Output (I/O) processor. When the CPU needs programs that are not present in the main memory, they are brought in from the auxiliary memory. Programs that are not currently required in the main memory are moved into the auxiliary memory to provide space for currently used programs and data.

To increase the speed of processing, a very-high-speed memory, known as cache, is used. It helps in making the current data and programs available to the CPU at high speed. The cache memory is used in computer systems to compensate for the difference between the main memory access time and processor logic speed. CPU logic is quicker than main memory access time. Yet, the processing speed of the CPU is limited by the main memory's speed. The difference in these operating speeds can be balanced by using an extremely fast, small cache between the CPU and main memory. The cache access time is nearly equal to the processor logic clock cycle time. The cache is used to store the following:

1. Fragments of program that is presently being executed in the CPU
2. Temporary data that is repeatedly needed in the current operation

When programs and data are available at a high rate, it is possible to increase the performance rate of the computer.

Although the I/O processor manages data transfers between main memory and auxiliary memory, the cache organization deals with the transfer of information between the main memory and the CPU. Thus, the cache and the CPU interact at different levels in the memory hierarchy system. The main reason for having two or three levels of memory hierarchy is to achieve cost- effectiveness. As the storage capacity of the memory increases, the cost-per-bit of storing binary information decreases and the memory access time increases. When compared to main memory, the auxiliary memory has a large storage capacity and is relatively inexpensive. However, the auxiliary memory has low access speed. The cache memory is very small, quite expensive, and has very high access speed. Consequently, as the memory access speed increases, so does its relative cost. The main purpose of employing a memory hierarchy is to achieve the optimum average access speed while minimizing the total cost of the entire memory system.

Did u know? Many operating systems are designed in such a way that they allow the CPU to process a number of independent programs concurrently. This concept, called multiprogramming, refers to the presence of two or more programs in different parts of the memory hierarchy at the same time. In this way, it is possible to make use of the computer by processing several programs in sequence.

Example: Assume that a program is being executed in the CPU and an I/O operation is required. The CPU instructs the I/O processor to start processing the transfer. At this point, the CPU is free to execute another program. In a multiprogramming system, when one program is waiting for an input or output process to take place, there is another program ready to use the CPU for execution.

Main Memory

The main memory is the fundamental storage unit in a computer system. It is a relatively large and fast memory, and stores programs and data during computer operations. The technology that makes the main memory work is based on semiconductor integrated circuits.

As mentioned earlier, RAM is the main memory. Integrated circuit Random Access Memory (RAM) chips are available in two possible operating modes. They are:

1. ***Static***: It basically consists of internal flip-flops, which store the binary information. The stored information remains valid as long as power is supplied to the unit. The static RAM is simple to use and has shorter read and write cycles.
2. ***Dynamic***: It stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are made available inside the chip by Metal Oxide Semiconductor (MOS) transistors. The stored charge on the capacitors tends to discharge with time and thus, the capacitors must be regularly recharged by refreshing the dynamic memory. Refreshing is done by progressively supplying the capacitor with words every few milliseconds to restore the decaying charge. The dynamic RAM uses minimum power and provides ample storage capacity in a single memory chip.

Most of the main memory in a computer is typically made up of RAM integrated circuit chips, but a part of

Notes Metal Oxide Semiconductor (MOS) transistor is the basic building block of almost all modern digital memories, processors and logic chips. MOS transistors find many uses and can function as a switch, an amplifier or a resistor.

the memory may be built with Read Only Memory (ROM) chips. Formerly, RAM was used to refer to a random-access memory. But now it is used to address a read/write memory to distinguish it from a read-only memory, although ROM is also random access. RAM is used for storing the volumes of programs and data that are subject to change. ROM is used for storing programs that permanently reside in the computer. It is also used for storing tables of constants whose value does not change once the computer is constructed.

Notes The ROM part of main memory is used for storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to initiate the computer software when power is turned on. As RAM is volatile, its contents are lost when power is turned off. The contents of ROM are not lost even when power is turned off.

RAM and ROM chips are available in a variety of sizes. If the memory requirement for the computer is larger than the capacity of a single chip, a number of chips can be combined to form the required memory size.

Random Access Memory

The term Random Access Memory or RAM is typically used to refer to memory that is easily read from and written to by the microprocessor. In reality, it is not right to use this term. For a memory to be called random access, it should be possible to access any address at any time. This differentiates RAM from storage devices such as tapes or hard drives where the data is accessed sequentially.

Practically, RAM is the main memory of a computer. Its objective is to store data and applications that are currently in use. The operating system controls the usage of this memory. It gives instructions like when the items are to be loaded into RAM, where they are to be located in RAM, and when they need to be removed from RAM. RAM is intended to be very fast both for reading and writing data. RAM also tends to be volatile, that is, all the data is lost as soon as power is cut off.

Read Only Memory

In every computer system, there must be a segment of memory that is stable and unaffected by power loss. This kind of memory is called ***Read Only Memory or ROM***. Once again, this is not the right term. If it was not possible to write to this type of memory, it would not have been possible to store the code or data that is to be contained in it. It simply indicates that without special mechanisms in place, a processor may not be able to write to this type of memory.

Did u know? ROM also stores the computer's BIOS (Basic Input/Output System). BIOS is the code that guides the processor to access its resources when power is turned on, it must be present even when the computer is powered down. The other function of BIOS is storing the code for embedded systems.

Example: It is important for the code in your car's computer to persist even if the battery is disconnected.

There are some categories of ROM that the microprocessor can write to. However, the time taken to write to them, or the programming requirements needed to do so, makes it difficult to write to them regularly. This is why these memories are still considered read only.

There are few situations where the processor cannot write to a ROM under any conditions.

Example: There is no need to modify the code in your car's computer. This ROM is programmed before installing. In order to install a new program in the car's computer, the old ROM is removed and a new ROM is installed in its place.

SRAM

RAMs that are made up of circuits and can preserve the information as long as power is supplied are referred to as Static Random Access Memories (SRAM). Flip-flops form the basic memory elements in a SRAM device. A SRAM consists of an array of flip-flops, one for each bit.

Since SRAM consists of an array of flip-flops, a large number of flip-flops are needed to provide higher capacity memory. Because of this, simpler flip-flop circuits, BJT and MOS transistors are used for SRAM. This helps to save chip area and provides memory integrated circuits at relatively reduced cost, increased speed and reduces the power dissipation as well. SRAMs have very short access times typically less than 10 ns. SRAMs with battery backup are commonly used to provide stability to data during power loss.

DRAM

SRAMs are faster but their cost is high, because their cells require many transistors. RAMs can be obtained at a lower cost if simpler cells are used. A MOS storage cell based on capacitors can be used to replace the SRAM cells. Such a storage cell cannot preserve the charge (that is, data) indefinitely and must be recharged periodically. Therefore, these cells are called as dynamic storage cells. RAMs using these cells are referred to as Dynamic RAMs or simply DRAMs.

Cache

Even with improvements in hard drive performance, it is still not practical to execute programs or access data directly from the mechanical devices like hard disk and magnetic tapes, because they are very slow. Therefore, when the processor needs to access data, it is first loaded from the hard drive into the main memory where the higher performance RAM allows fast access to the data. When the processor does not require the data anymore, it can either be discarded or used to update the hard drive.

The cost makes the capacity of a computer's main memory inadequate when compared to its hard drive. However, this would not have a major impact as all of the data on a hard drive need not be accessed all the time by the processor. Only the currently active data or programs need to be in RAM. Additional performance improvements can be achieved by taking this concept to another level.

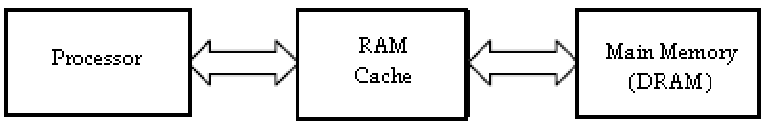
As discussed earlier, there are two main classifications of RAM: Static RAM (SRAM) and Dynamic RAM (DRAM). SRAM is faster, but that speed comes at a high cost - it has a lower density and it is more expensive. Since main memory needs to be relatively large and inexpensive, it is implemented with DRAM.

Main memory improves the performance of the system by loading only the data that is currently required or in use from the hard drive. However, the system performance can be improved considerably by using a small, fast SRAM to store data and code that is in immediate use. The code and data that is not currently needed can be stored in the main memory.

You must be aware that in a programming language, instructions that are executed often tend to be clustered together. This is mainly due to the basic constructs of programming such as loops and subroutines. Therefore, when one instruction is executed, the chances of it or its surrounding instructions being executed again in the near future are high. Over a short interval, a cluster of instructions may execute over and over again. This is referred to as the **principle of locality**. Data also behaves as per this principle as related data is often stored in consecutive locations.

To benefit from this principle, a small, fast SRAM is placed between the processor/CPU and main memory to hold the most recently used instructions/programs and data under the belief that they will most likely be used again soon. This small, fast SRAM is called a **RAM cache** or just a **cache**. The location of a cache in a memory hierarchy is shown in Figure 12.3.

Figure 12.3: Cache Placement between Main Memory and Processor

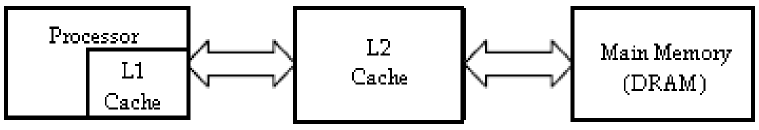


The SRAM of the cache needs to be small, the reason being that the larger address decoder circuits are slower than small address decoder circuits. As the memory increases, the complexity of the address decoder circuit also increases. As the complexity of the address decoder circuit increases, the time taken to select a memory location based on the address it received also increases. For this reason, making a memory smaller makes it faster.

The concept of reducing the size of memory can be optimized by placing an even smaller SRAM between the cache and the processor, thereby creating two levels of cache. This new cache is usually contained inside of the processor. As the new cache is put inside the processor, the wires connecting the two become very short, and the interface circuitry becomes more closely integrated with that of the processor. These two conditions together with the smaller decoder circuit facilitate faster data access. When two caches are present, the cache within the processor is referred to as a level 1 or L1 cache. The cache between the **L1 cache** and memory is referred to as a level 2 or **L2 cache**.

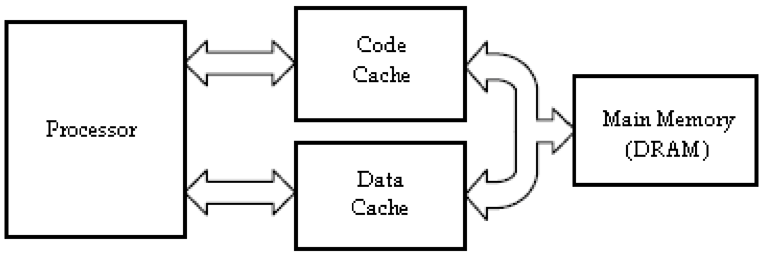
Figure 12.4 shows the placement of L1 and L2 cache in memory.

Figure 12.4: L1 and L2 Cache Placement



The **split cache,** another cache organization, is shown in figure 12.5. Split cache requires two caches. In this case, a processor uses one cache to store code/instructions and a second cache to store data. This cache organization is typically used to support an advanced type of processor architecture such as pipelining. Here, the mechanisms used by the processor to handle the code are so distinct from those used for data that it does not make sense to put both types of information into the same cache.

Figure 12.5: Split Cache Organization



The success of caches depends upon the principle of locality. The principle proposes that when one data item is loaded into a cache, the items close to it in memory should be loaded too.

If a program enters a loop, most of the instructions that are part of that loop are executed multiple times. Therefore, when the first instruction of a loop is being loaded into the cache, it loads its bordering instructions simultaneously to save time. In this way, the processor does not have to wait for the main memory to provide subsequent instructions. As a result of this, caches are organized in such a way that when one piece of data or code is loaded, the block of neighboring items are loaded too. Each block loaded into the cache is identified with a number known as a tag. This tag can be used to find the original addresses of the data in the main memory. Therefore, when the processor is in search of a piece of data or code (hereafter referred to as a word), it only needs to check the tags to see if the word is contained in the cache.

Each block of words and its corresponding tag is combined in the cache to form a **line**. The lines are structured into a table. When the main memory needs a word from within a block, the whole block is moved into one of the lines of the cache along with its tag, which is used to identify the address of the block.

Cache Operation

The processor first checks the cache when it needs a word from the memory. If the word is not in the cache, a condition known as miss occurs. In order to ensure that no time is lost due to a miss, the process for accessing the same word from main memory is simultaneously activated. If the word is present in the cache, then the processor uses the cache's word and disregards the results from the main memory. This mechanism is referred to as a hit.

In case of a miss, the entire block containing the word is loaded into a line of the cache, and the word is sent to the processor. Depending on the design of the cache/processor interface, the word is either loaded into the cache first and then given to the processor or it is loaded into the cache and sent to the processor simultaneously. In the first case, the cache is controlled by the memory interface and lies between memory and the processor. In the second case, the cache acts like an extra memory on the same bus with the main memory.

Suppose the main memory is divided into ***n*** blocks and the cache has space to accommodate exactly ***m*** blocks. By virtue of the nature of the cache, ***m*** is much smaller than ***n***. If we divide ***m*** into ***n*** , get an integer which illustrates the number of times that the main memory could fill the cache with different blocks from its contents.

Example: Suppose main memory is 128 M (2^{27}) and a block size is four words (2^2), then main memory contains $n = 2^{27-2} = 2^{25}$ blocks. If the cache for this system can hold 256 K (2^{18}) words, then $m = 2^{18-2} = 2^{16}$ blocks. Therefore, the main memory could fill the cache $n/m = 2^{25}/2^{16} = 2^{25-16} = 2^9 = 512$ times.

The main memory is much larger than a cache, so each line in the cache is responsible for storing one of many blocks from main memory. In our above example, each line of the cache is responsible for storing one of 512 different blocks from main memory at a specific time.

The process of transferring data from main memory to cache memory is called as mapping. There are three methods used to map a line in the cache to an address in memory so that the processor can quickly find a word. They are:

1. Direct Mapping
2. Associative Mapping
3. Set Associative Mapping

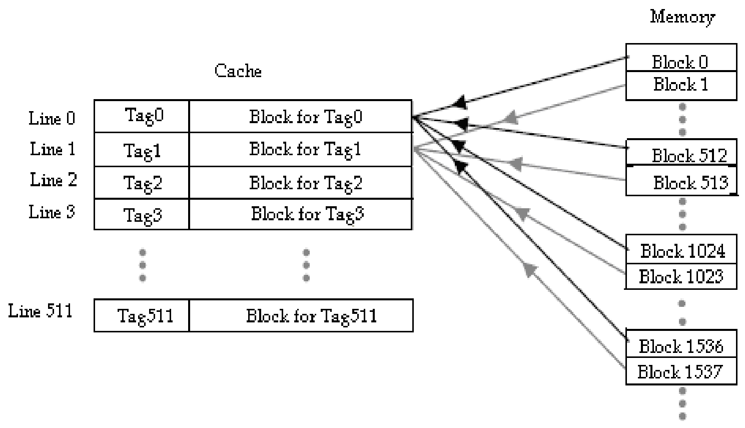
Caution The only one mistake that can be made in computer design is not having enough address bits for memory addressing and memory management. It is difficult to correct this.

Direct Mapping

Direct mapping is a procedure used to assign each memory block in main memory to a specific line in the cache. If a line is already filled with a memory block and a new block needs to be loaded, then the old block is discarded from the cache.

Figure 12.6 shows how multiple blocks from the above example are mapped to each line in the cache.

Figure 12.6: Direct Mapping of Main Memory to Cache

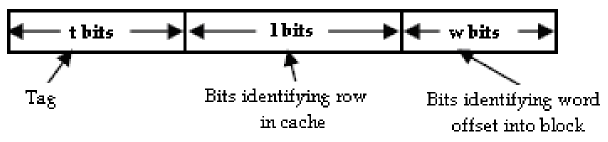


Just like locating a word within a block, bits are taken from the main memory address to uniquely describe the line in the cache where a block can be stored.

Example: Consider a cache with 2^9 = 512 lines, then a line would need 9 bits to be uniquely identified.

Therefore, the 9 bits of the address immediately to the left of the word identification bits would recognize the line in the cache where the block is to be stored. The bits of the address that were not used for the word offset or the cache line would be used for the tag. Figure 12.7 represents this partitioning of the bits.

Figure 12.7: Direct Mapping Partitioning of Memory Address


t bits
l bits
w bits
Tag
Bits identifying row in cache
Bits identifying word offset into block

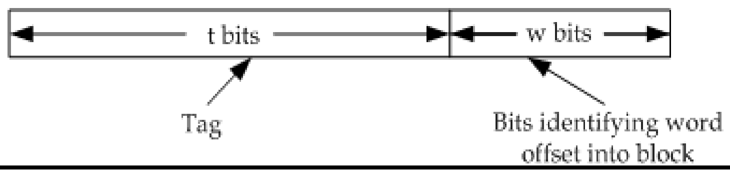
As soon as the block is stored in the line of the cache, the tag is copied to the tag location of the line. From the cache line number, the tag, and the word position within the block, the original address of the word can be reproduced.

In short, direct mapping divides an address into three parts: **t** tag bits, **l** line bits, and **w** word bits. The word bits are the least significant bits that identify the specific word within a block of memory. The line bits are the next least significant bits that identify the line of the cache in which the block is stored. The remaining bits are stored along with the block as the tag which locates the block's position in the main memory.

12.3.2 Associative Mapping

Associative mapping or fully associative mapping does not make use of line numbers. It breaks the main memory address into two parts - the word ID and a tag as shown in Figure 12.8. In order to check for a block

Figure 12.8: Associative Partitioning of Memory Address



stored in the memory, the tag is pulled from the memory address and a search is performed through all of the lines of the cache to see if the block is present.

This method of searching for a block within a cache appears like it might be a slow process, but it is not. Each line of the cache has its own compare circuitry, which can quickly analyze whether or not the block is contained at that line. With all of the lines performing this comparison process in parallel, the correct line is identified quickly.

This mapping technique is designed to solve a problem that exists with direct mapping where two active blocks of memory could map to the same line of the cache. When this happens, neither block of memory is allowed to stay in the cache as it is replaced quickly by the competing block. This leads to a condition that is referred to as **thrashing.** ***In*** thrashing, a line in the cache goes back and forth between two or more blocks, usually replacing a block even before the processor goes through it. Thrashing can be avoided by allowing a block of memory to map to any line of the cache.

However, this advantage comes with a price. When an associative cache is full and the processor needs to load a new block from memory, a decision has to be made regarding which of the existing blocks should be discarded. The selection method, known as a replacement algorithm, should aim to replace the block least likely to be needed by the processor in the near future.

There are many replacement algorithms, none of which has any precedence over the others. In an attempt to realize the fastest operation, each of these algorithms is implemented in hardware.

1. ***Least Recently Used (LRU):*** This method replaces the block that has not been read by the processor in the longest period of time.
2. ***First In First Out (FIFO):*** This method replaces the block that has been in cache the longest.
3. ***Least Frequently Used (LFU):*** This method replaces the block which has had fewest hits since being loaded into the cache.
4. ***Random:*** This method randomly selects a block to be replaced. Its performance is slightly lower than LRU, FIFO, or LFU.

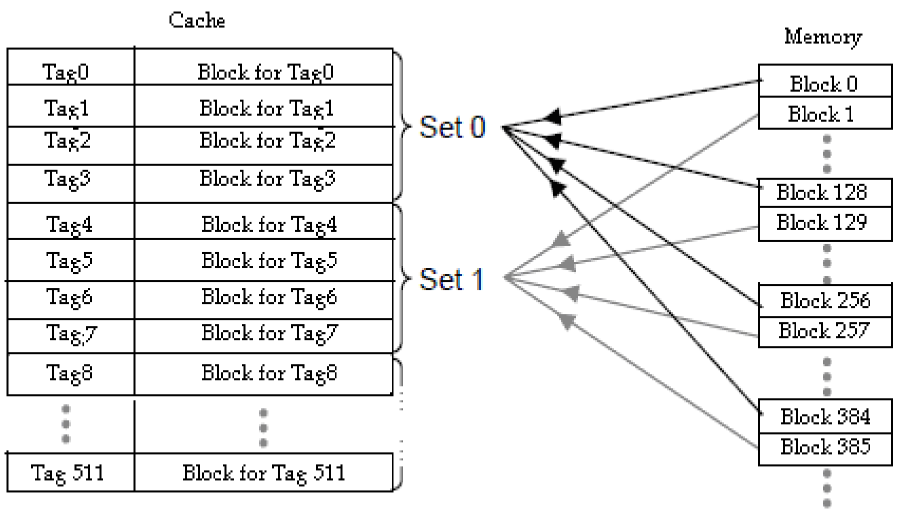
The objective of a replacement algorithm is to try to remove the page least likely to be referenced in the immediate future.

12.3.3 Set Associative Mapping

Set associative mapping merges direct mapping with fully associative mapping by grouping together lines of a cache into sets. The sets are determined using a direct mapping scheme. However, the lines within each set are considered as tiny fully associative cache where any block that is to be stored in the set can be stored to any line within the set.

Figure 12.9 represents this arrangement using a sample cache that uses four lines to a set.

Figure 12.9: Set Associative Mapping of Main Memory to Cache



A set associative cache that contains *k* lines per set is called as a *k* way set associative cache. Since the mapping technique uses the memory address just like direct mapping does, the number of lines contained in a set must be equal to an integer power of two, for example, two, four, eight, sixteen, and so on.

Example: **Description of set associative mapping**.

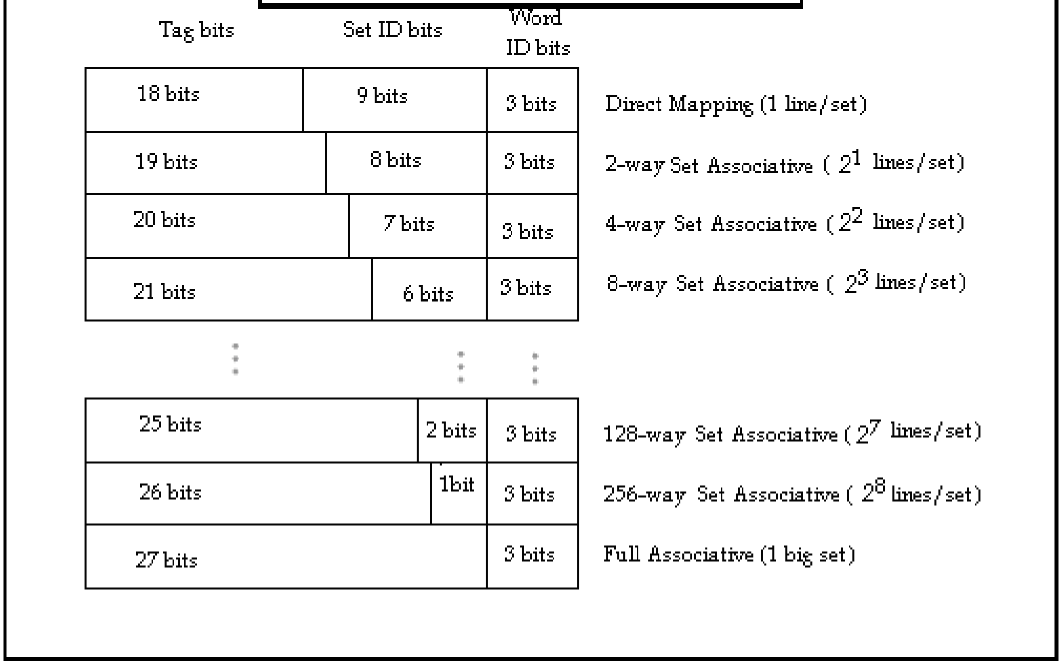
Consider a cache with $2^9 = 512$ lines, a block of memory contains $2^3 = 8$ words, and the full memory space contains $2^{30} = 1G$ words. In a direct mapping scheme, this would leave $30 – 9 – 3 = 18$ bits for the tag.

Caution Note that the direct mapping method is the same as set associative method where the set size is equal to one line.

By shifting from direct mapping to set associative with a set size of two lines per set, the number of sets obtained equals to half the number of lines. In the instance of the cache having 512 lines, we would obtain 256 sets of two lines each, which would need eight bits from the memory address to identify the set. This would leave $30 – 8 – 3 = 19$ bits for the tag. By shifting to four lines per set, the number of sets is reduced to 128 sets requiring 7 bits to identify the set and twenty bits for the tag.

Every time the number of lines per set in the example is doubled, the number of bits used to identify the set is reduced by one, thus increasing the number of tag bits by one. This is shown in the Figure 12.10.

Figure 12.10: Effect of Cache Set Size on Address Partitioning


Tag bits
Set ID bits
Word ID bits
18 bits
9 bits
3 bits
Direct Mapping (1 line/set)
19 bits
8 bits
3 bits
2-way Set Associative (2^1 lines/set)
20 bits
7 bits
3 bits
4-way Set Associative (2^2 lines/set)
21 bits
6 bits
3 bits
8-way Set Associative (2^3 lines/set)
25 bits
2 bits
3 bits
128-way Set Associative (2^7 lines/set)
26 bits
1bit
3 bits
256-way Set Associative (2^8 lines/set)
27 bits
3 bits
Full Associative (1 big set)

Whenever a block from memory has to be stored in a set already filled with other blocks, one of the replacement algorithms described for fully associative mapping is used.

Virtual Memory

In typical computer systems, data and programs are initially stored in auxiliary memory. Fragments of data or programs are brought into main memory as and when the CPU requires them. Virtual memory is a method or approach used in some large computer systems. It allows the user to construct programs as though a large memory space was available, which is equal to the whole of auxiliary memory. Every address that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in the main memory. Virtual memory is made use of to give programmers the impression that they have a very large memory at their disposition, even though the computer actually has a relatively small main memory. A virtual memory system implements a mechanism that translates program-generated addresses into correct main memory locations. This translation happens dynamically even as programs are being executed in the CPU. The translation or mapping is automatically handled by the hardware by means of a mapping table.

Address Space and Memory Space

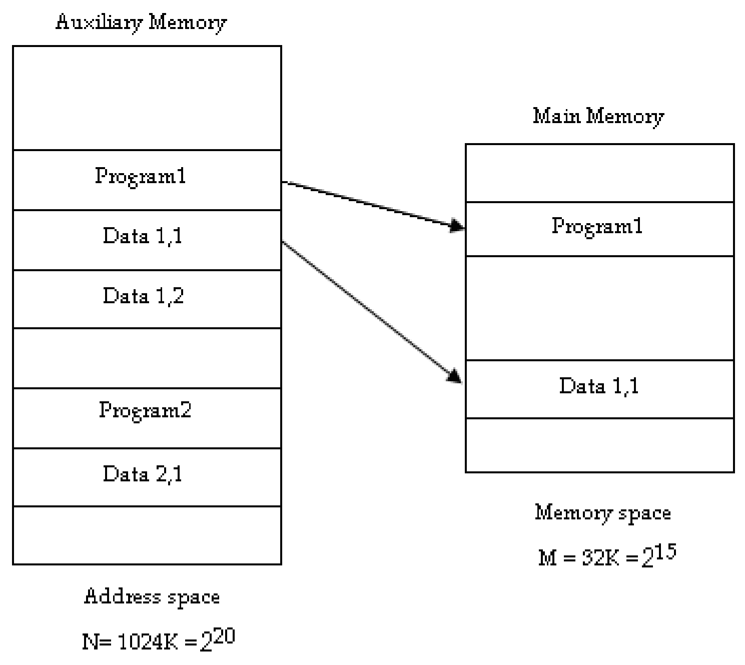
Addresses that are used by programmers are called virtual addresses, and the set of such addresses is called the address space. The space or spot where the address is stored in the main memory is called a location or physical address and the set of such locations is called the memory space. Therefore, the address space is the set of addresses generated by programs as they reference instructions and data. The memory space holds the actual main memory locations that are directly addressable for processing. In most computers, the address and memory spaces are the same.

Did u know? The address space is permitted to be bigger than the memory space in computers with virtual memory.

Example: Consider, main-memory having capacity of 32K words (K = 1024). 15 bits are required to specify a physical address in memory since 32K = 2^{15}. Assuming that the computer has available auxiliary memory for storing 2^{20} = 1024K words. Thus auxiliary memory has a storage capacity equivalent to the capacity of 32 main memories. If the address space is denoted by N and the memory space by M, we then have for this example N = 1024K and M = 32K.

In a multiprogramming computer system, programs and data are transferred to and from auxiliary memory and main memory when required by the CPU. Suppose Program1 is currently being executed in the CPU, Program1 and a section of its associated data are transferred from auxiliary memory into main memory as shown in figure 12.11. The associated programs and data need not be in adjacent locations in the memory, since information is being moved in and out, and empty spaces may be scattered in the memory.

Figure 12.11: Relation between Address and Memory Space in a Virtual Memory System



In a virtual memory system, programmers are made to believe that they have the total address space for their use. Additionally, the address field of the instruction code has a sufficient number of bits to specify all virtual addresses. Suppose, the address field of an instruction code consists of 20 bits, but physical memory addresses can only be specified with 15 bits. As a result, CPU will reference instructions and data with a 20-bit address, but the information at this address must be taken from physical memory because access to auxiliary storage for individual words would be extremely long.

Thus, a table is needed to map a virtual address of say 20 bits to a physical address of say 15 bits. Mapping is a dynamic process, which means that every address is translated instantly as a word is referenced by CPU.

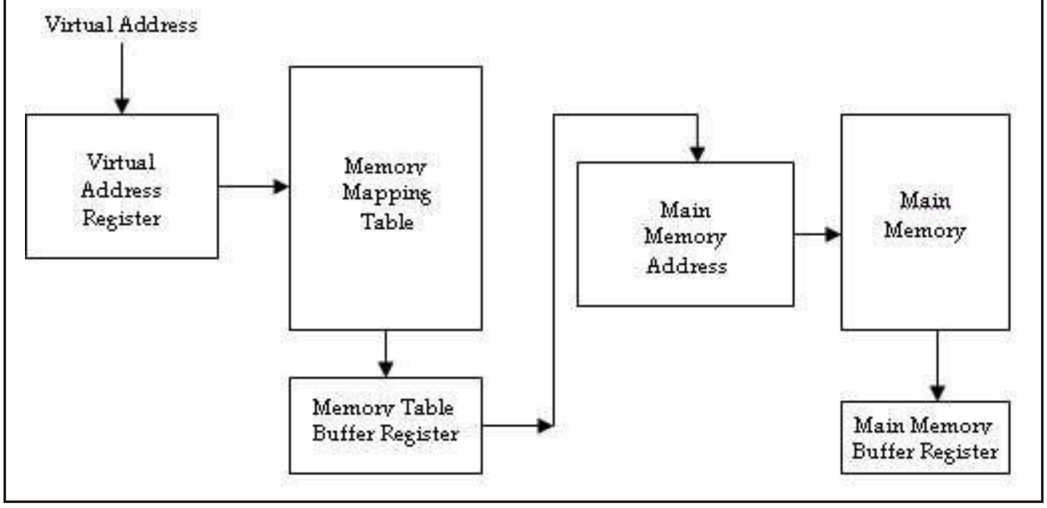
A separate memory or main memory may be used to store the mapping as shown in figure 12.12.

1. In the first case, an additional memory unit is needed along with one extra memory access time.

2. In the second case, the table takes space from main memory and two accesses to memory are required with the program running at half speed.

3. In the third case, an associative memory can be used. The associative mapping technique is discussed later.

Figure 12.12: Memory Table for Mapping a Virtual Address



In figure 12.12, a virtual address of 20 bits is mapped and listed in the memory mapping table. It is then mapped to a 15 bit physical address which refers to a location in the main memory.

Address Mapping Using Pages

Presenting the address mapping in table form is simplified, if the information in the address space and the memory space are each divided into groups of fixed size. The physical memory is divided into clusters of equal size called ***blocks,*** which may range from 64 to 4096 words each. The term page refers to clusters of address space of the same size.

Example: Suppose a page or block consists of 1K words, then address space can be divided into 1024 pages and main memory can be divided into 32 blocks.

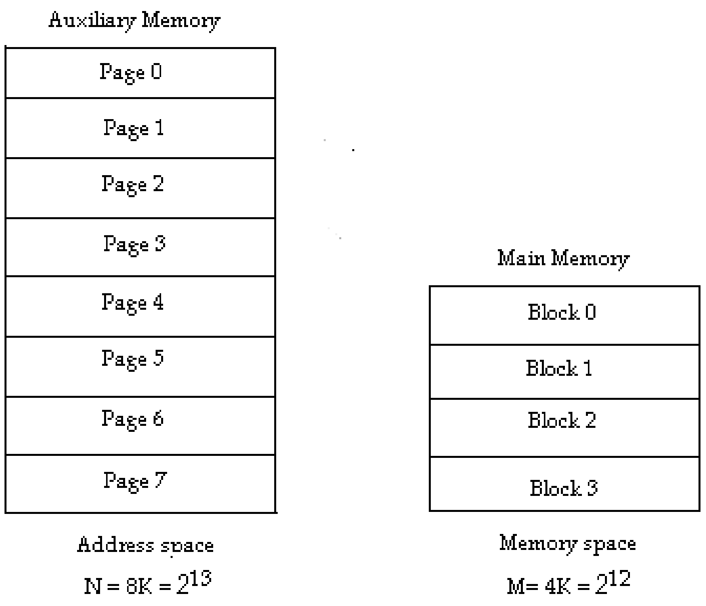
Even though both a page and a block are split into groups of 1K words, a page refers to the cluster of address space, while a block refers to the cluster of memory space. The programs are also split into pages. Segments of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term page frame is at times used to identify a block.

Example: Suppose a computer has an address space of 8K and a memory space of 4K. If they are each split into groups of 1K words, we get eight pages and four blocks as shown in figure 12.13. At any given time, up to four pages of address space may be available in main memory in any one of the four blocks.

The mapping from address space to memory space is made possible if each virtual address is considered to be represented by two numbers - both a page number address and a line within the page. In a computer with 2^p words per page, p bits are used to specify a line address and the remaining high-order bits of the virtual address specify the page number.

Example: The figure 12.13 shows a virtual address having 13 bits. As each page contains 2^{10} = 1024 words, the higher-order three bits of a virtual address will specify one of the eight pages and the lower-order 10 bits specify the line address within the page.

Figure 12.13: Address Space and Memory Space Split into Groups of 1K Words

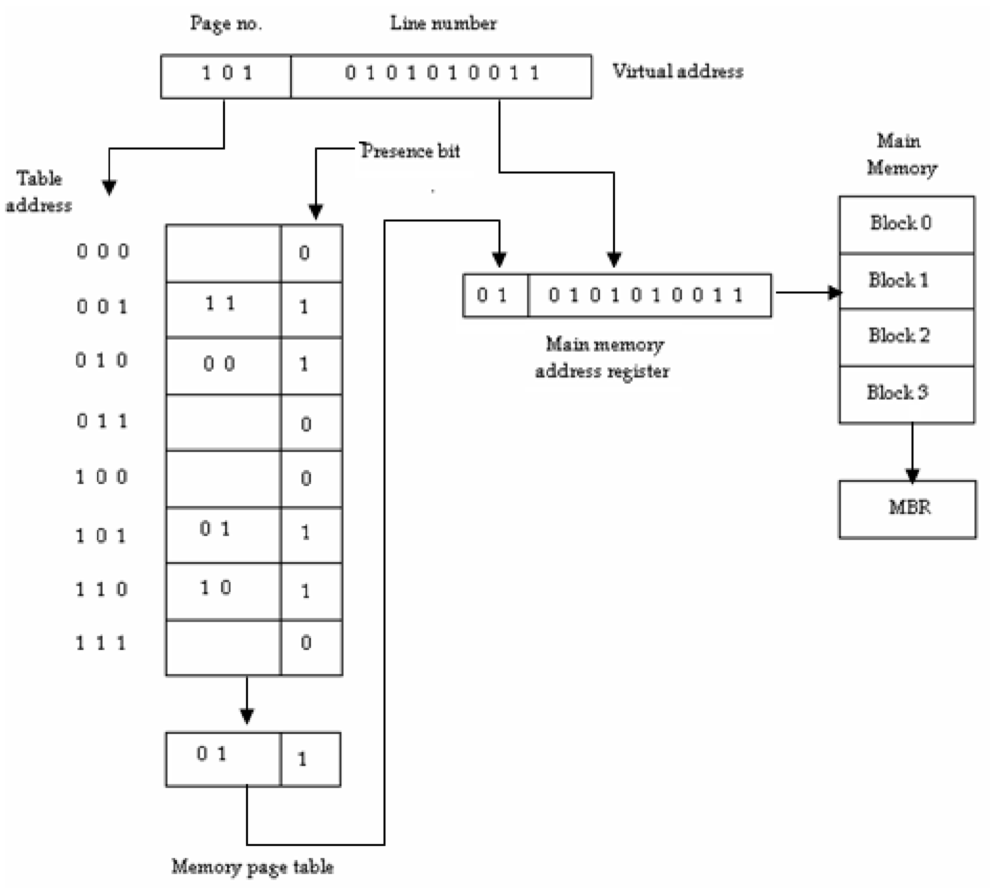


Notes The line address is the same in address space as well as memory space; the only mapping needed is from a page number to a block number.

The structure of the memory mapping table in a paged system is shown in Figure 12.14. The memory-page table comprises eight words, one for each page. The address in the page table denotes the page number and the content of the word gives the block number where that page is stored in main memory. The table shows that pages 1, 2, 5, and 6 are now present in the main memory in blocks 3, 0, 1, and 2, respectively. A presence bit in each location signifies whether the page has been moved from auxiliary memory into main memory. Zero in the presence bit signifies that this page is not available in main memory. The CPU points to a word in memory with a virtual address of 13 bits. The three high-order bits of the virtual address specify a page number and also an address for the memory-page table.

Figure 12.14 shows memory table in a paged system.

Figure 12.14: Memory Table in a Paged System



The content of the word in the memory page table at the page number address is copied into the memory table buffer register. If the presence bit is 1, the block number thus copied is transferred to the two high-order bits of the main memory address register. The line number from the virtual address is moved into the 10 lower-order bits of the memory address register. A read signal to main memory moves the content of the word to the main memory buffer register that is ready to be used by the CPU. If the presence bit of the word copied from the page table is 0, it indicates that the content of the word referenced by the virtual address is not present in the main memory. A request to the operating system is then generated to get the required page from auxiliary memory and place it into main memory before resuming computation.

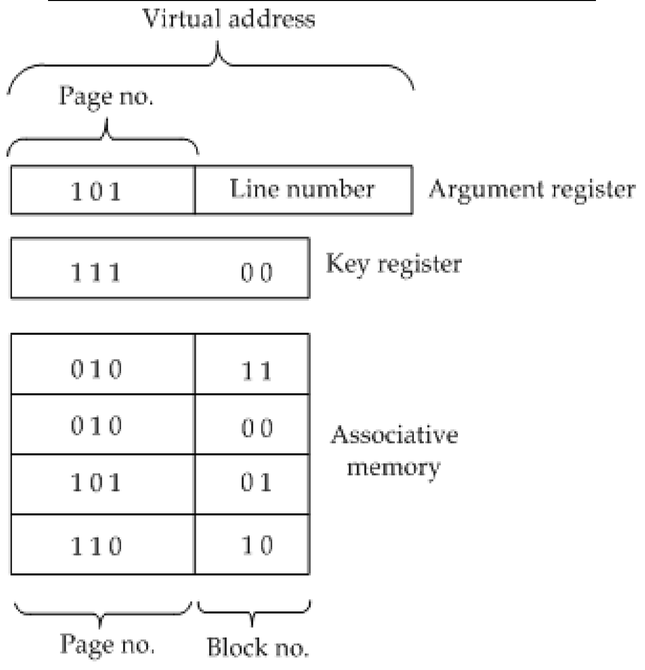
Associative Memory Page Table

A random-access memory page table is not appropriate when it comes to storage utilization. In the illustration in figure 12.14, we noticed that eight words of memory are needed, one for each page. But at least four words are always marked empty because the main memory cannot accommodate more than four blocks. Normally, a system with *m* blocks and *n* pages would require a memory-page table of *n* locations of which, up to *m* blocks is marked with block numbers and all others are empty.

Example: Consider a computer with an address space of 1024K words and memory space of 32K words. If each page or block is composed of 1K words, the number of pages will be 1024 and the number of blocks will be 32. The capacity of the memory-page table will be 1024 words and there will be only 32 locations having a presence bit equal to 1. At any given time, at least 992 locations will be free.

A better approach towards organizing the page table would be to construct it with a number of words equal to the number of blocks in the main memory. In this way, the memory size is reduced and each location is fully utilized. This method can be implemented using an associative memory where each word in memory includes a page number together with its related block number. Each word's page field is compared with the page number present in the virtual address and if a match is found, the word is read from memory and its corresponding block number is determined.

Figure 12. 15: An Associative Memory Page Table



Let us consider the same case of eight pages and four blocks shown in the example of figure 12.14. If we replace the random access memory-page table with an associative memory of four words as shown in figure 12.15, then each entry in the associative memory array would consist of two fields. By observing the figure, you would find that the first three bits specify a field for storing the page number and the last two bits make up a field for storing the block number. The virtual address is stored in the argument register. The page number bits in the page field of the associative memory and the page numbers in the argument register are matched against each other. If a match is found, the 5-bit word is read out from memory and the related block number is transferred to the main memory address register. If no match occurs, a request to the operating system is generated to fetch the required page from auxiliary memory.

12.4.4 Page Replacement

A virtual memory organization is a combination of hardware and software systems. To make efficient utilization of memory space all the software operations are handled by the memory management software. The memory management software must decide on the following three things,

1. Which page in main memory must to be removed to create space for a new page?
2. When a new page is to be moved from auxiliary memory to main memory?
3. Where the page is to be located in main memory?

The hardware mapping system and the memory management software together form the architecture of a virtual memory.

When the program execution starts, one or more pages are moved into main memory and the page table is set to indicate their location. The program is executed from main memory until a reference is made for a page that is not in memory. This event is termed as page fault. When page fault occurs, the program that is currently in execution is stopped until the required page is moved into main memory. Since the act of loading a page from auxiliary memory to main memory is basically an I/O operation, the operating system assigns this task to the I/O processor. In this interval, control is transferred to the next program in main memory that is waiting to be processed in the CPU. Soon after the memory block is assigned and then moved, the suspended program can resume execution.

If main memory is full, a new page cannot be moved in. Therefore, it would be necessary to remove a page from a memory block to accommodate the new page. The decision of removing specific pages from memory is determined by the replacement algorithm. The different replacement algorithms have been discussed briefly in the previous sections.

Summary

- Memory hierarchy is essential in computers as it provides an optimized low-cost memory system.
- Main memory and the hard drive form two levels of the computer's memory hierarchy.
- The main memory plays an important role as it can communicate directly with the CPU and auxiliary memory.
- Most of the main memory in a computer is typically made up of RAM integrated circuit chips, but a part of the memory may be built with ROM chips also.
- SRAM can be used to construct high capacity memory. SRAMs have very short access times.
- Cache memory is built using SRAMs.
- Cache is used to store temporary data that will be used in the immediate future.
- In order to have data that would be needed frequently, a mapping system is employed.
- There are three important methods used to map a line in the cache to an address in memory. They are direct mapping, associative mapping, and set associative mapping.
- Virtual memory is a memory management system that will ensure that the CPU is not left idle at any given time.

Keywords

Access Time: The time taken to locate the address and perform the transfer.

Argument Register: It is the small amount of storage available on CPU to store arguments.

Memory Access Time: It is the time interval between a memory operation request (read or write) and the time the memory operation completes.

Processor Cycle Time: It is the time taken to perform a basic operation performed by a CPU.

Transfer Rate: Rate at which data is transferred to/from the memory device.