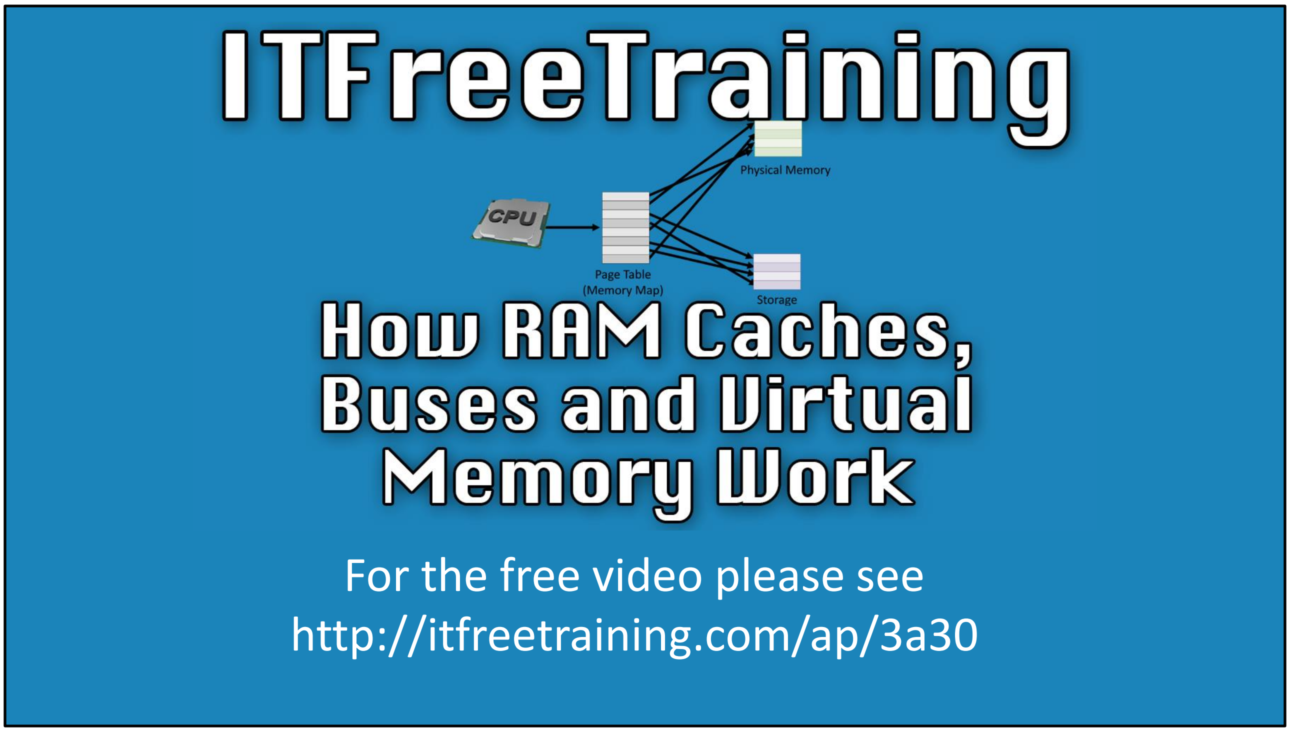
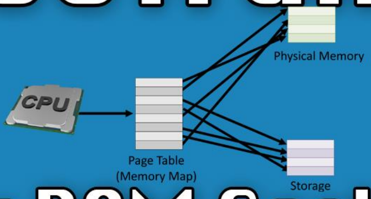


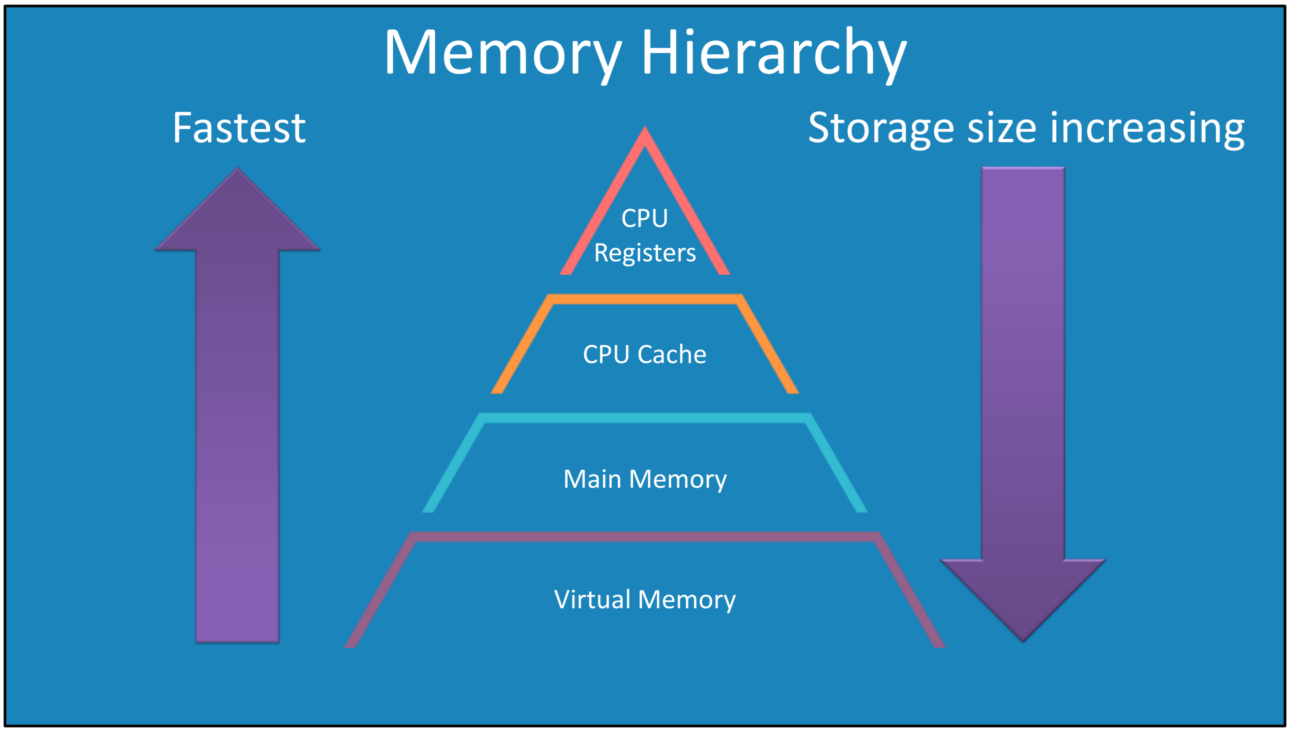
ITFreeTraining



How RAM Caches, Buses and Virtual Memory Work

For the free video please see
http://itfreetraining.com/ap/3a30

In this video from ITFreeTraining, I will look at how RAM caches, buses and virtual memory work in a computer system. Understanding how these work, will give you a better understanding of how to fix performance problems and to understand what sort of performance you may expect to receive.



0:19 Before I start looking at how caches and buses work in a computer system, I will first look at the memory hierarchy. This will give you an understanding of why these systems are designed the way they are.

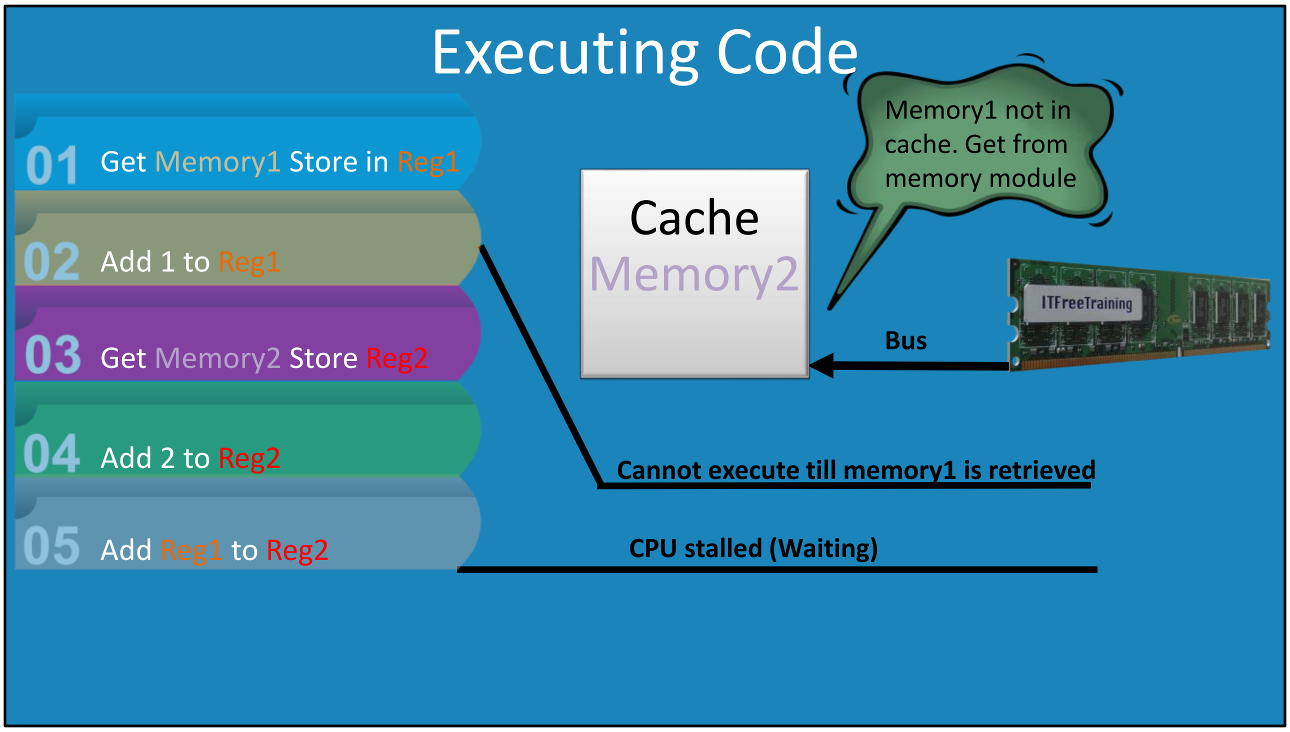
At the top of the hierarchy are CPU registers. When a computer wants to perform an operation, for example adding numbers together, the data is stored in the CPU registers. CPU registers are small in number. They are extremely fast and only used to temporarily store data used when the computer is performing calculations or running code.

At the next level is CPU cache. CPU cache is very fast but there is not much of it. CPU cache is measured in megabytes and varies depending upon the CPU.

If the data cannot be found in the CPU cache, the next step is to access the main memory of the computer. This is slower again, but the advantage is that there is more of it.

Lastly there is virtual memory. This is usually stored on a device like a hard disk. This is slower again, but very cheap.

So what does this all mean? Essentially the top of the pyramid is the fastest and as you go down, the speed decreases. The advantage, however, is that it becomes cheaper and, thus, generally there is more of it. It comes down to a trade-off between cost and performance. Now let’s have a look at how this all work together.


Executing Code
01 Get Memory1 Store in Reg1
02 Add 1 to Reg1
03 Get Memory2 Store Reg2
04 Add 2 to Reg2
05 Add Reg1 to Reg2
Cache
Memory2
Memory1 not in cache. Get from memory module
ITFreeTraining
Bus
Cannot execute till memory1 is retrieved
CPU stalled (Waiting)

1:51 First of all, I will look at how a CPU may execute code. Different CPU's work in different ways, but what I will do is give a very simplified version of how a CPU may execute code that illustrates the steps that a modern CPU may go through in order to execute code.

First consider that the CPU has some cache. The CPU has multiple levels of cache, but for this example, let's consider there is only the single cache to make it simple. In the cache is the contents of some memory that I will call memory2.

Let's consider that the CPU is executing some instructions. Rather than using real computer code, I have simplified the instructions down to make them easy to understand. The first instruction will be to obtain the data in memory1 and store the data in register1. A register holds data temporarily so the data can be used in calculations. The CPU cache however does not have this data. The CPU will thus retrieve the contents of memory1 from the memory modules.

This will take some time. In older CPUs the CPU would need to stop executing instructions until it had the data. When this occurs, the CPU has stalled and is effectively waiting. Waiting means no work is being performed and thus is not ideal.

To help get around this, modern CPUs can execute instructions out of order. While the CPU is waiting for the memory to be retrieved, the CPU will look at the next instruction. In this case, the instruction is to add 1 to regsiter1. The instruction however cannot be executed until the

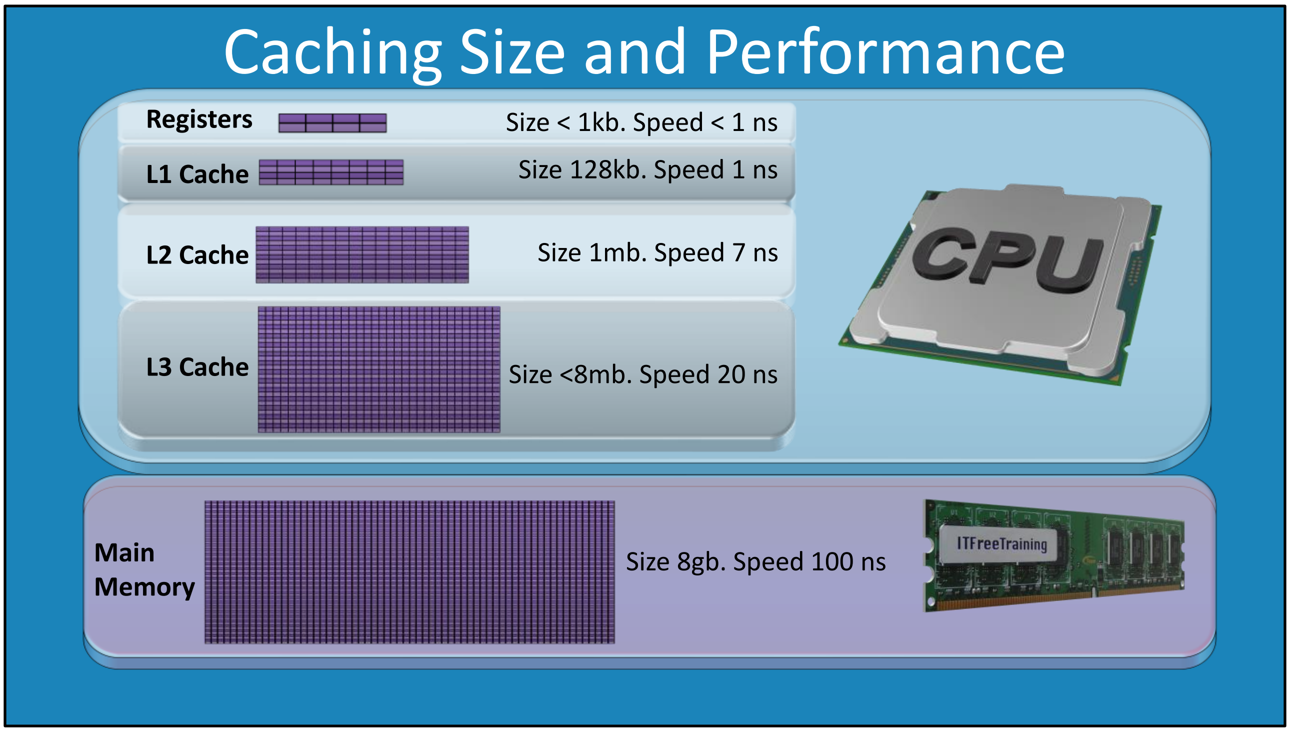
CPU has obtain the data from memory1.

Rather than wait, the CPU can often look at a number of instructions in advance. The CPU will next look at the third instruction. The next instruction will be to get the data from memory2 and place it in register2. The contents of memory2 are already in the cache, so this instruction can be executed immediately.

Looking then at the next instruction, this instruction is adding 2 to register2. Since memory2 was available in the cache and has been placed in register2, this instruction will execute immediately.

The next instruction will add the contents of register1 to register2. In this example, we will consider that the CPU had looked ahead as much as possible and thus was not able to process any more instructions until memory1 is retrieved from the memory module. When this occurs the CPU stalls, that is, it is effectively stopped until the data is obtained.

A CPU can execute a large amount of instructions in the time it takes to retrieve a single byte from a memory module. For this reason, this is why the CPU uses a cache - so the majority of the time, data can be obtained quickly from the cache. Depending on what software you are running, this will determine how many cache misses will occur. A cache miss is when the data is not in the cache and must be obtained from the memory module. Computer software will often utilize loops to process data and data is grouped together. For this reason, the CPU can execute quite a few instructions before a cache miss will occur. If a cache miss does occur, as we have seen, the CPU can execute instructions out of order in an attempt to get some work done before it has to stall. This increases the overall amount of work the CPU can perform.



5:27 The cache in the CPU is divided up into parts referred to as levels; they differ in size and performance. Let's consider a typical CPU. In this example, I will use some approximate speeds and sizes. Every CPU is going to be different so take the figures given here as a rough example of what to expect.

At the top are the registers. These operate approximately at one nanosecond or less. Different CPUs have different registers. For example, a 64-bit CPU will require at least a 64-bit register to contain data. Some individual registers may even be larger than 64 bits. Generally, the total size of the registers will be less than one kilobyte.

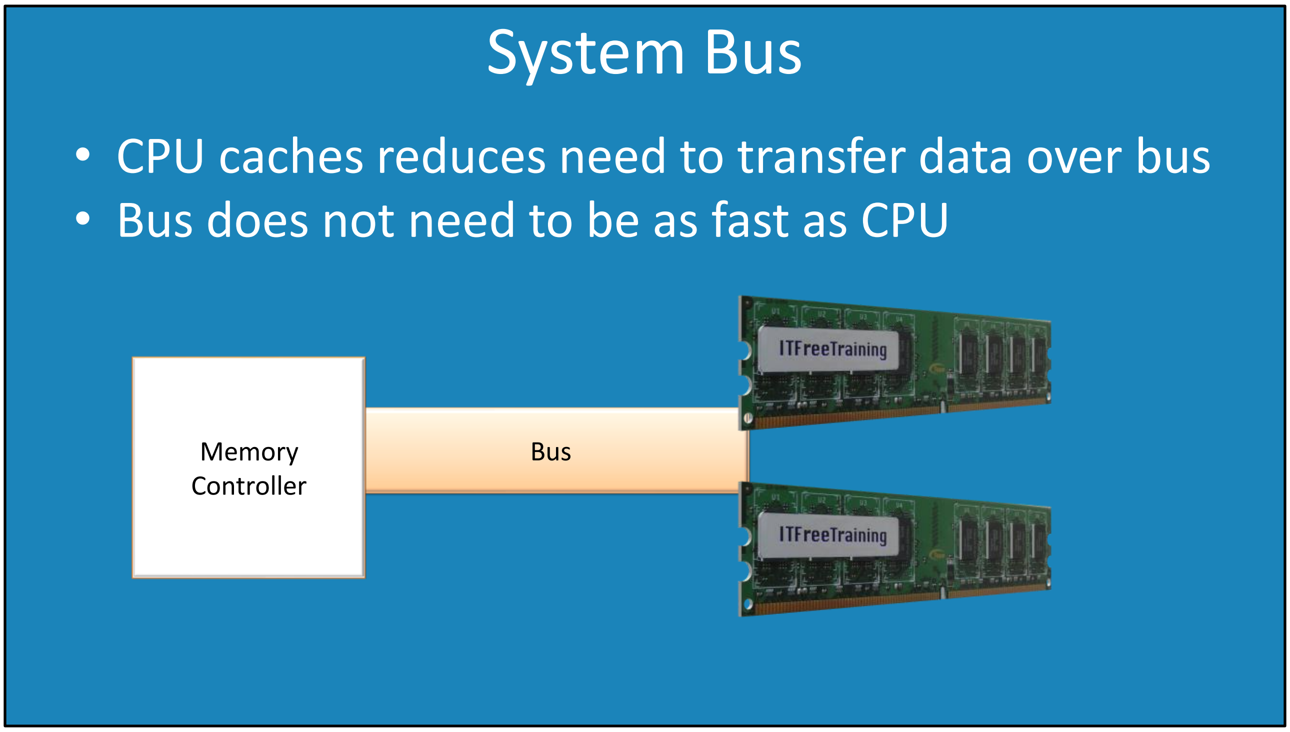
The first level of cache is referred to as Level 1 cache or L1 cache for short. Different CPUs have different amounts of L1 cache. Generally, L1 cache is divided up between the number of cores. In this example, if there are 128 kilobytes of cache and the CPU has four cores, each core would get 32 kilobytes of cache. The cache may also be divided into half, one half for data and the other half for instructions. Different CPUs vary in speed, but L1 cache is the fastest, generally around one nanosecond.

The next level of cache is Level 2 or L2 cache. This cache is generally about a megabyte in size, but again, different CPUs have varying sizes. The CPU cache is also generally divided up between each core. So, in this example, a one megabyte L2 cache in a four core CPU would be divided up into 256 kilobytes per core.

The next cache is Level 3 or L3 cache. This cache is generally much larger than the other caches. In this example the cache is eight megabytes in size. L3 cache varies in size, with high performance CPUs on the market having as much as 32 megabytes or more.

The L3 cache may also be external to the CPU and contained on the motherboard rather than in the CPU itself. Although I have not covered it here, it is possible to have Level 4 cache or L4 cache. This cache is not as common and can be in the CPU or on the motherboard. If it is present, it is shared between the CPU and GPU. L3 cache is still quite fast. In this example around 20 nanoseconds, but your results may vary.

If the memory the CPU requires is not in the cache, the next step is to get the data from the memory modules. Memory modules are much larger than the cache and nowadays are generally measured in gigabytes. Accessing the memory modules is not as quick as the CPU cache and your performance will vary depending on your system. In this example, the memory modules take about 100 nanoseconds to transfer data to the CPU cache. This does not seem a lot, but when you consider that, in this example, this is five times slower than L3 cache, 100 nanoseconds is a long time in computing terms. You can start to see the need for the CPU to execute instructions out of order in an attempt to utilize any idle time effectively.



System Bus

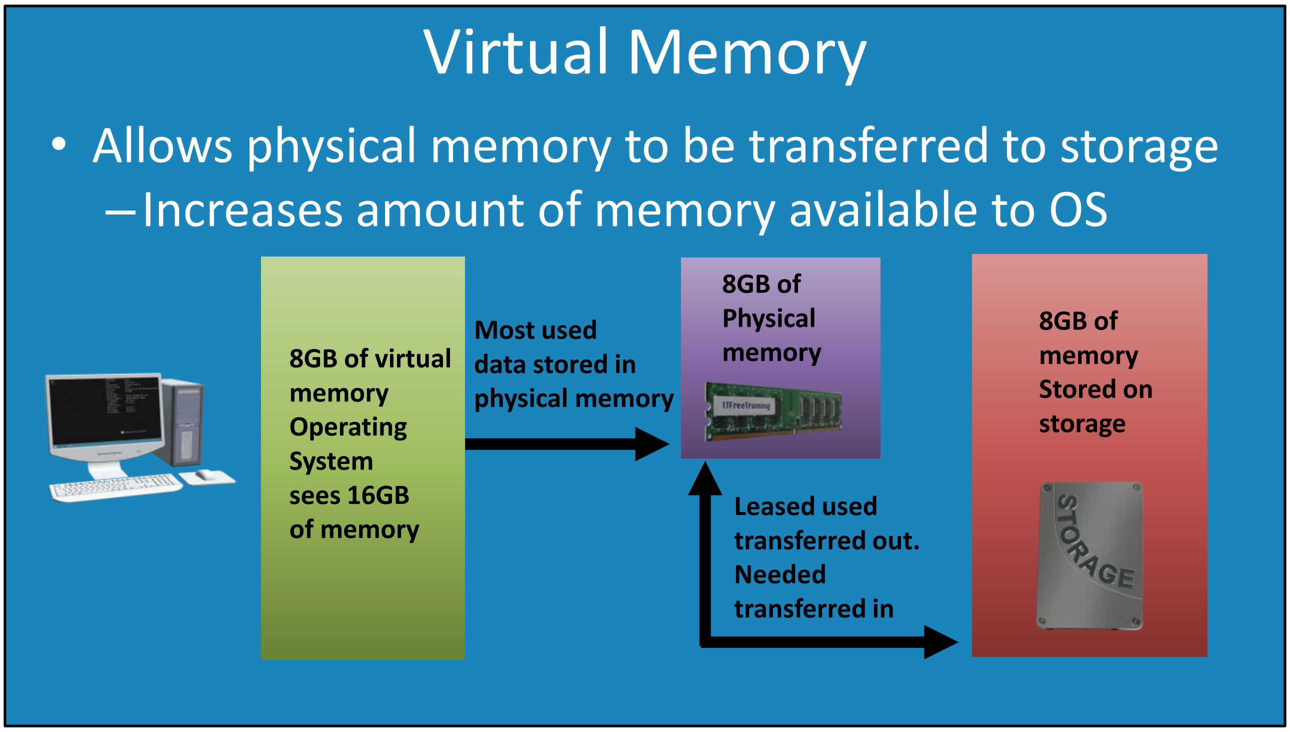
- CPU caches reduces need to transfer data over bus
- Bus does not need to be as fast as CPU

8:55 The next point to look at is, what happens when a cache miss occurs and the CPU needs memory from the memory modules? When this occurs, the memory controller is required to fetch the memory from the memory modules. The memory controller will either be part of the CPU or a separate chip on the motherboard.

The transfer of memory to the memory controller is done using the bus. There have been a number of different buses that have been implemented. As time goes on, the speed of these buses has increased. CPU speed has not increased at the same rate and nowadays the bus speed is getting close to being the same speed as the CPU.

Now consider that, by using a CPU cache, this reduces the need to transfer data over the bus. This is because, in a lot of cases, the data will be found in the cache and thus there will not be a need to transfer data over the bus. Because of this, the bus speed has not needed to keep up with the speed of the CPU.

More memory-intensive software will put more load on the bus then others; but you should be able to see that the speed of the bus, although important, does not have a big effect on the overall system. Of course, I would never say no to a faster system bus, but a small increase in system bus speed generally won’t have a big effect on a computer system.



Virtual Memory

- Allows physical memory to be transferred to storage
  - Increases amount of memory available to OS

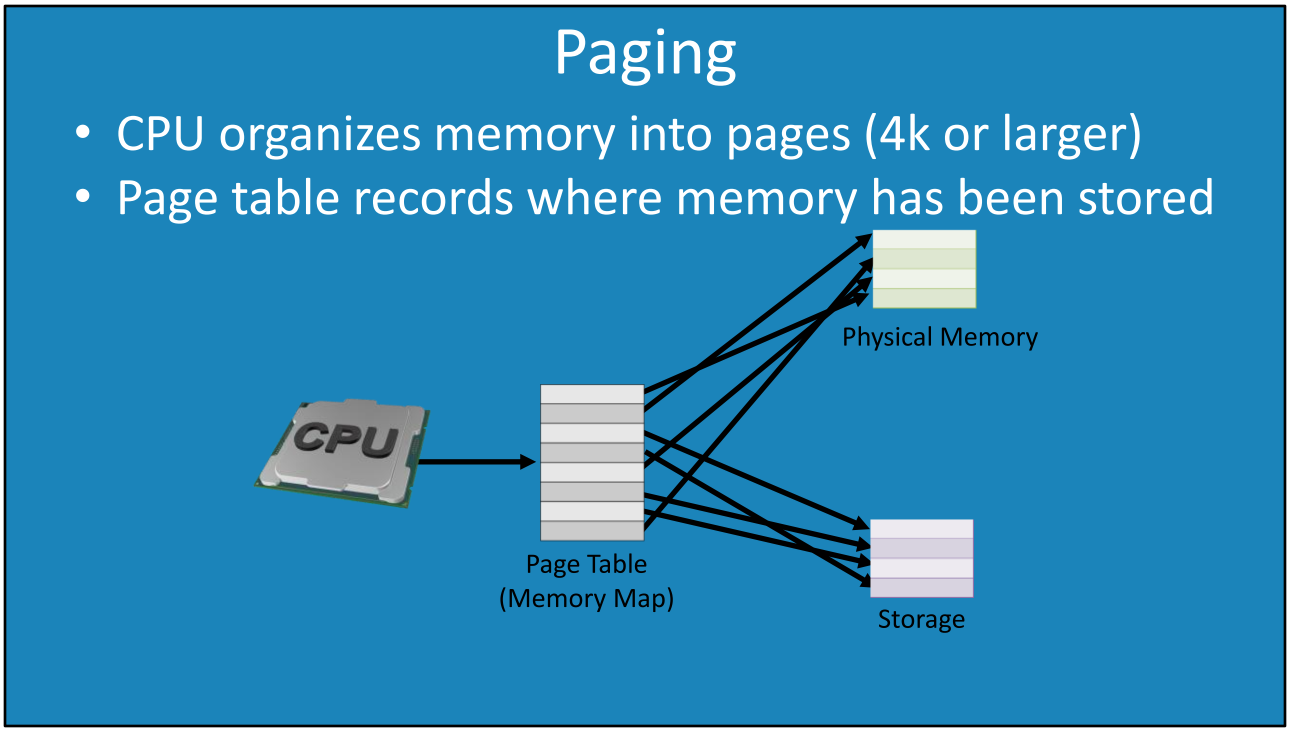
8GB of virtual memory Operating System sees 16GB of memory
Most used data stored in physical memory
8GB of Physical memory
ITFreeTraining
Leased used transferred out. Needed transferred in
8GB of memory Stored on storage
STORAGE

10:15 The next point that I will cover is virtual memory. Virtual memory allows physical memory to be transferred to storage like a solid-state drive. This allows the amount of memory available to an operating system to be increased.

To understand this better, consider this example. A computer has installed in it eight gigabytes of memory. The administrator is attempting to run a memory intensive application that requires more physical memory than what is installed. The administrator thus configures the operating system with eight gigabytes of virtual memory. This takes the total amount of memory the operating system has access to, to 16 gigabytes.

The operating system will attempt to use physical memory where possible. If more than eight gigabytes of memory is required the most frequently used memory is stored in physical memory. In this example, eight gigabytes of storage has been allocated on a solid-state drive to be used with virtual memory. The least used memory is transferred from physical memory to storage and the memory requested is transferred into physical memory.

Essentially what is happening is that memory is being swapped from the memory modules to storage. Different operating systems will use different algorithms to determine what stays in memory and what is transferred to storage. Most algorithms will look at what memory is most likely to be used and transfer memory to storage that is less likely to be used. For example, looking at how often a particular part of memory is accessed and when it was last accessed may help to determine which memory to keep in physical memory and what memory to swap out to storage. So, how does the operating system determine how much data to transfer from memory to storage?



Paging

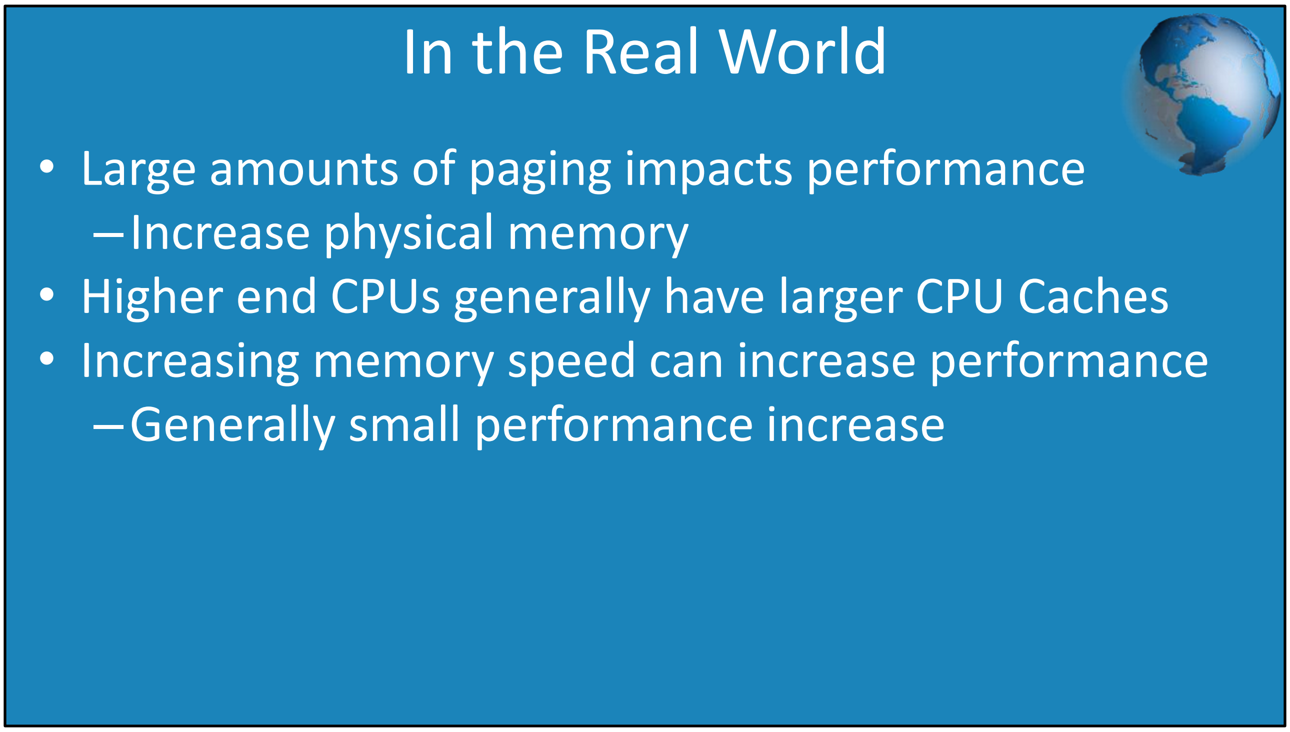
- CPU organizes memory into pages (4k or larger)
- Page table records where memory has been stored

12:02 This brings us to our next topic, paging. In order to swap data from memory to storage, the data is organized into blocks called pages. Different CPUs support different size pages. With most CPUs the smallest page size is four kilobytes. The operating system can change the page size depending on its needs. Some CPUs also support multiple differently-sized pages being used at the same time, but for most, once it configured, it is set to one page size only and then only that page size can be used.

A CPU will have inside it a page table. When the CPU attempts to access memory that is not in the CPU cache, it will refer to this page table. The page table will tell the CPU where the page has been stored. Essentially this is indicating where the page is currently stored, either in physical memory or if it is stored in secondary storage like a hard disk.

Given this example, looking at the page table will tell the operating system if a particular page is stored in physical memory or in storage. When the CPU requests a page that is currently stored in storage, it is up to the operating system to transfer the page from the storage into physical memory. When this occurs, often the operating system will need to remove a page from the physical memory to make room for the new page. To ensure the data is not lost, the page will be written to storage. When this occurs, it is referred to as swapping. Think of swapping like getting a file out of a filing cabinet, but the desk you are trying to put it on is cluttered. In order to make room for the new file, something from the desk needs to be removed and placed back in the filing cabinet.

Different operating systems handle this in different ways. Some operating systems will store the data in a file on the local storage. This file will often be called a page file or swap file. When a computer starts transferring data to and from storage this is often referred to as paging. For example, if an application starts requesting a lot of memory, more than the computer physically has, the operating system may start using the local storage for memory. This will cause a lot of local storage activity and an administrator may say the computer is paging. If the computer is paging a lot, this will have an effect on the overall performance of the computer. This is because local storage is a lot slower than memory modules.



In the Real World

- Large amounts of paging impacts performance
  - Increase physical memory
- Higher end CPUs generally have larger CPU Caches
- Increasing memory speed can increase performance
  - Generally small performance increase

14:31 In computing, there are a lot of facts that may impact system performance. In general, if your computer system is paging a lot, this will have a big impact on performance. Increasing the amount of physical memory will help to fix this problem. Not having enough memory in a computer system and therefore causing it to page is quite noticeable. This will be noticeable in that there will be a lot of local storage activity, for example a lot of hard disk activity. Also, you will notice the computer's performance will slow down and may become noticeably sluggish.

When purchasing a CPU, a lot of factors need to be taken into account. For example, the speed of the CPU and the features it has. Generally, you will find that higher-end CPUs will have larger CPU caches. CPU cache is expensive to manufacture compared to other memory types. For this reason, more of it will generally be found in more expensive CPUs.

Increasing the memory speed of a computer can increase performance. It is debatable how much it will increase the overall performance, as it depends on so many factors, such as what software you are running. Generally, performance increases will be small. This is because CPU caches are designed to reduce the amount of access to the memory modules. If you don't need to access the memory modules too often then the speed that they run at becomes less important. It should also be remembered that, in recent years, CPU speed has not increased as much as memory speed has. This means the gap between CPU and memory speed is not as big as it used to be. In my opinion, I would consider upgrading components other than memory speed, but I still would never say no to an increase in memory speed.

This concludes this video from ITFreeTraining. I hope you have found it useful and I look forward to seeing you in more videos from us. Until the next video, I would like to thank you for watching.

References
https://en.wikipedia.org/wiki/CPU_cache “CPU cache”
https://arstechnica.com/gadgets/2002/07/caching/2/ “Understanding CPU caching and performance”
https://gist.github.com/jboner/2841832 “Latency Numbers Every Programmer Should Know”
https://techtalk.pcpitstop.com/2016/10/05/average-pc-memory-ram-continues-climb/ “Average PC Memory (RAM) Continues to Climb”
https://en.wikipedia.org/wiki/Page_(computer_memory) “Page (computer memory)”

Credits
http://ITFreeTraining.com Trainer: Austin Mason
http://hplewis.com Voice Talent: HP Lewis
http://www.pbb-proofreading.uk Quality Assurance: Brett Batson