Week 5: Main memory-3

The cache memory is a fast, convenient temporary storage area for frequently-accessed data; however it is usually tiny compared to the amount of memory required to fit complete program and data sets. Let's take into consideration a typical Intel Core i5-based desktop computer system:

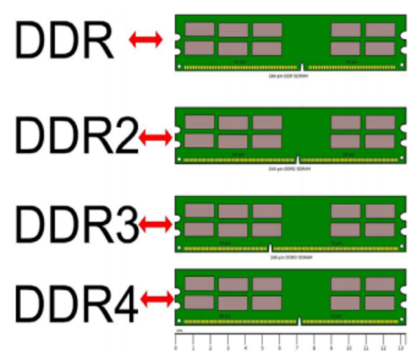
Memory Type Size	Proportion of Total
Main Memory 8 GB	100%
Level 3 Cache 8 MB	0.1%
Level 2 Cache 256 KB	0.364%
Level 1 Cache 64 KB	7.6x10-6 %

Despite the cache memory making up for the performance shortfalls of main memory, constant developments are being made to its speed, in order to keep up with ever increasing processor performance.

Memory chips are arranged on circuit boards in DIMM (Dual In-Line Memory Module) format
– edge card connectors on both sides of board

each SDRAM type has different electrical properties are incompatible

keyed' so that they will only fit in the socket corresponding to that specific memory technology.



Memory bandwidth

The primary factor in desktop computing performance is *memory bandwidth*, a measure of how quickly data can be transferred: all else being equal, the faster the memory can deliver data, the faster the computer will perform.

Memory latency

Memory latency refers to how long it takes from a request for data being issued, and that data finally being retrieved from memory. For almost all desktop computing applications, memory latency does not have a significant effect on performance. (It is, however, more critical in server systems.)

Memory technologies

Here is a summary of common memory technologies in use, and their peak bandwidth abilities of their fastest speed ratings (some historical processor requirements listed in italics for comparison)

Technology	**Speed Rating**	**Peak Bandwidth**
Intel Pentium 4	*1066MHz FSB*	*8. 5 GB/ sec*
Intel Core 2 series	*1600MHz FSB*	*12. 8 GB/ sec*
AMD Athlon64	*Hyper Transport*	*14. 4 GB/ sec*
SDRAM	PC133	1.06 GB/sec
DDR SDRAM	DDR-400	3.2 GB/sec
DDR2 SDRAM	DDR2-800	6.4 GB/sec
DDR3 SDRAM	DDR3-1600	12.8 GB/sec
DDR4 SDRAM	DDR4-3200	25.6 GB/sec

Some memory technologies can be run in "dual channel" mode; this means that two sets of memory are interleaved together, effectively doubling the peak bandwidth. For example, "dual channel DDR-400" is capable of 6.4GB/sec, twice the 3.2GB/sec of standard "single channel" DDR-400.

A note about these large numbers; they are *peak* memory bandwidth requirements. That is, the *highest theoretical* requirement. In normal use, the processors are much less demanding.

The knife cuts both ways, though; memory systems often can only work at a fraction of their peak bandwidth efficiency.

SDRAM

The oldest machines still capable of running modern software use memory known as ***SDRAM*** **(http://en.wikipedia.org/wiki/SDRAM)** (Synchronous Dynamic Random Access Memory). This memory was introduced in 1997 and often used with Intel Pentium II and III and early AMD Athlon-based computers. Most of Apple's "G3" and "G4" systems also use this memory.

DDR SDRAM

DDR **(http://en.wikipedia.org/wiki/DDR_SDRAM)** (Double Data Rate) SDRAM is a development above normal SDRAM which allowed for the memory bandwidth to be doubled, by allowing it to perform twice the amount of data transfer per clock tick -- hence the name "double data rate".

This memory technology was cheap to manufacture and was massively popular amongst all desktop computing systems in the early-to-mid 2000s.

DDR2 SDRAM

Standard DDR memory chips encountered a design limit as to how fast they could be run; this prompted the development of the **DDR2 SDRAM** **(http://en.wikipedia.org/wiki/DDR2_SDRAM)** standard, which allowed for faster clock speeds. Another side-benefit of the design was that with the use of a lower voltage, power consumption was also reduced.

DDR3 SDRAM

DDR3 **(http://en.wikipedia.org/wiki/DDR3_SDRAM)** is yet faster and more power efficient. Nearly all modern desktop and notebook computers use DDR3 for their main system memory.

DDR3 vs. DDR2 **(http://www.anandtech.com/show/2232)** : an article on **anandtech.com** **(http://www.anandtech.com/)** comparing DDR3 and DDR2 memory.

RDRAM (Ram bus)

RDRAM **(http://en.wikipedia.org/wiki/RDRAM)** (Rambus Dynamic Random Access Memory, named after "Rambus", the company that invented it) was a memory technology designed in the late 1990s with a focus on high memory bandwidth, at the expense of latency.

This happened to suit well with Intel's then-new Pentium 4 processor, which had memory bandwidth requirements that were unheard of at the time.

However, due to licencing and manufacturing-difficulty issues, RDRAM remained an expensive and unpopular choice in the general computing market, and has since been left behind by cheaper DDR SDRAM memory (explained below). It has, however, found a niche in some specialist processing markets: many game consoles, such as the Sony Playstation 2 and 3, use Rambus memory.

Non-volatile memory

Most memory in a computer is *volatile*: when the power is turned off, the memory is erased.

Although appropriate for most processing situations, it does not lend well to permanent storage. All computers contain a small core program, called the BIOS (Basic Input/Output System), which provides a very primitive way for software to access various parts of the hardware.

The BIOS must stay present at all times, and cannot be erased. A memory type known as *ROM* (Read Only Memory) is used for this purpose. Some forms of ROM can be reprogrammed by the computer, known as *PROM* (Programmable Read Only Memory).

An EPROM chip.

Legacy PROMs were only erasable using ultraviolet light; these are known as EPROM (Erasable-Programmable Read Only Memory). These chips had a little window on the top onto which UV light had to be shined to erase. This meant that:

1. the entire chip had to be erased; it wasn't possible to erase a portion
2. it was necessary to put a sticker over the window, lest UV light from the sun slowly erase the

Nowadays, the most common form in industry is called *Flash EEPROM* (the *EE* stands for "Electrically Erasable"), as it allows the contents of the memory chip to be changed relatively quickly and conveniently. Almost all computers have a BIOS chip made of Flash EEPROM, however the most common consumer use of this type of memory is in portable devices such as music players, smartphones and tablets. As memory chips are *solid- state* (no moving parts), they are much more reliable and abuse-resistant than other storage technologies.

Memory addressing

In high-level circumstances, such as when writing a program in a high-level programming language, memory addressing is a simple flat file of addresses which can be allocated and used at will.

However, in reality it is very unlikely that this is the case. Memory controller hardware, and the operating system that provides the interface between application software and the hardware, translate these simple high-level memory operations into what is really involved in managing the memory.

Memory addressing in hardware

In the software world, pages can be of variable size as there are no physical limitations outside the addressing range of the hardware it's running on - and with modern 64-bit systems this is no longer an issue. In the hardware world, however, the world is very different: there isn't just "one" memory chip, there are dozens that work together to build the total amount of memory in the system.

Something has to translate this flat-file software world to the discrete chips of the hardware world, and this is the job of the memory controller, a part of the CPU. It is aware of the memory chips under its management, and distributes the address space across them.

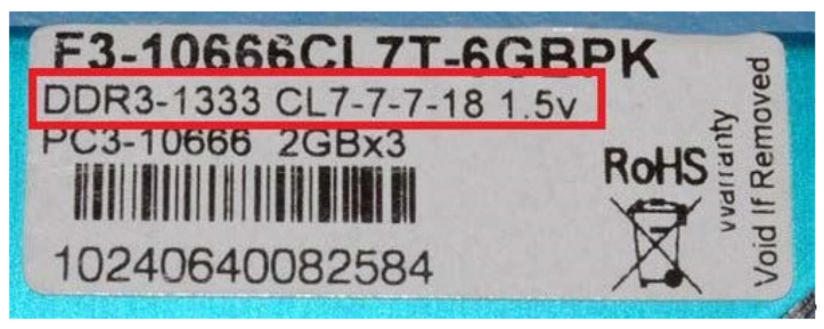
Hardware memory access

Memory chips themselves aren't a flat file - they are a two-dimensional matrix that is arranged into rows and columns. When a memory address is requested, the memory controller needs to:

- look up which memory chip is responsible for storing that part of the address space
- locate the exact row and column of that memory chip's storage matrix

Accessing the row and column of the memory chip takes some time. This is a significant contributor to memory latency - how long it takes to get the data out of a particular memory area.

It is typical for quality memory modules to have not only their memory clock speed written on the label, but also its latency with a series of numbers. For example:



A sticker from a DDR3 memory module. Photo: hardwaresecrets.com

This DDR3 memory module has a series of timing information. They are typically of the form:

DDRx-yyyy CLa- b- c- d

The first figure is the **memory speed. DDR3 - 1333 CL7- 7- 7- 18** refers to DDR3 memory that is running at an effective clock speed of 1333 MHz.

Four more data points follow:

The **first figure**, often prefixed with CL, stands for "**CAS latency** (http://en.wikipedia.org/wiki/CAS_latency)". This is the time it takes, in clock ticks, for the memory module to start finding the data requested of it. In this instance, it will take 7 clock ticks.

The **second figure** is the "RAS to CAS delay". This is the time taken between accessing the row of the memory matrix, and the column. For this memory module, it takes 7 clock ticks.

The **third figure** is the "RAS precharge". This is how long it takes to go from one row in the memory matrix, to another. This memory module takes 7 clock ticks to perform this function.

The **fourth figure** is "Active to precharge delay". This is the time the memory controller has to wait from one memory access instruction to another.

hardwaresecrets.com (http://www.hardwaresecrets.com/) have a more in-depth **guide to understanding RAM timings** (http://www.hardwaresecrets.com/article/Understanding-RAM-Timings/26), and these performance metrics.

The reason why it takes a while for these processes to take place is due to internal propagation delays within the memory chip itself: not every step to complete each task can (or will) take exactly the same time. These figures specify how long the manufacturer rates the chip to complete these operations, in a stable manner.

Fortunately, most of this complication is hidden from the hardware technician: the memory module will have a piece of "**Serial Presence Detect** (http://en.wikipedia.org/wiki/Serial_presence_detect)" data, from which the system can fetch these rated timings so that the memory can be configured automatically.

Hardware 'tweakers' (those who have a hobby of getting maximum performance out of their systems) will often push the memory chips over the rated stable timings, in the hope of squeezing extra performance.

Multi-channel memory

One way of minimising memory latency (and as such, improving performance) is to interleave the memory requests over multiple channels. That is, instead of relying on a single bank of memory for accessing data, these operations are distributed over a number of memory banks, and the instructions split between them.

Memory interleaving is the technique behind **multi-channel memory architecture** (http://en.wikipedia.org/wiki/Multi-channel_memory_architecture) (e.g. "dual channel") in modern computer systems.

For example, all odd-numbered memory addresses are stored in one bank of memory (bank A), and even- numbered address stored in another (bank B). Then, when a block of memory is accessed, each memory access instruction is split between the two banks, significantly reducing the effective latency of the memory operation as a whole.

Memory management in software

As we have seen, for a program to run - that is, for a process to execute - its instructions have to be in memory. It is then clear that existing memory has to be shared by all the users and all the processes at the same time. Except in exceptional circumstances, there is not enough memory to keep all this in memory all the time. Operating Systems employ several techniques to solve this problem:

The Operating System allocates sections of memory to processes at their request, and quickly frees them when they are no longer needed. This means that only some fragments of a process will be in memory at a given time, the remainder most likely waiting its turn to be brought in on disk. This is called ***swapping*** **(http://en.wikipedia.org/wiki/Paging#Terminology%23Terminology)** , because process information is swapped out to disk to make up some space, and then swapped in again when required.

Given the previous point, there can be no guarantee that a program section will be allocated to the same portion of memory each time it is swapped in. The Operating System looks for some free space in memory where it can put the information, and places it there. Thus, the memory addresses as seen by a process are different from the actual physical addresses in the computer memory.

In addition, the Operating System manages allocations for all the users to share the same memory. The memory management system of the Operating System manages all transparently for the users, who are unaware of this. The system very efficiently translates all the process addresses to physical addresses as the process executes.

As a consequence of this swapping and use of free space, the memory space for a process is not contiguous, but the process address space is actually fragmented like a mosaic. The process thinks that is using a continuous range of addresses, but the actual addresses are scattered all over memory, and some of them are even in secondary memory.

Unlimited memory (?)

As a consequence of virtual memory, a computer “never” runs out of memory to run applications, since the Operating System swaps out things in memory when necessary and copies them onto disk. This frees up space to load the new application, or at least part of it. Programmers write programs as if there is unlimited RAM space.

This facility, however, has a price. The speed at which a hard drive manages information is much slower than RAM speed, so if the system is using virtual memory too heavily there will be a significant performance drop. That’s why it is important to have enough RAM to keep a significant portion of your processes in memory to avoid the Operating System swapping all the time in and out of disk. The system will run very slow when this happens. This phenomenon is called thrashing, where the Operating System swaps out something out of memory, only to need it immediately after, and vice versa. As we can see, having enough memory is a key to maintain the performance of a computer system.

In some systems some parameters may be tuned to run virtual memory more efficiently. In multi-user systems typically there is an administrator (called root in UNIX, for example) to do that, and in personal computers this is done by the user. You should investigate this facility for your computer system.

Virtual memory

The size of modern programs means that usually a whole program does not fit in memory all at once. Hence, only the relevant parts of a process (a running program) are loaded into main memory, with the remainder residing on disk. When a program requests data that is not in main memory - e.g. an instruction, or data from a file - it has to get it from disk. This is organised transparently by the OS and the corresponding software and hardware, so programmers write programs as if the size of the computer memory is unlimited. Since this arrangement provides programmers with a virtual computer memory to work with, it is called ***virtual memory*** **(http://en.wikipedia.org/wiki/Virtual_memory)** .

To see how this works, assume a word is requested by the CPU; the word:

- is first searched for in the cache/s (primary level memory)
- if the word is in the cache (a cache hit), it is loaded immediately;
- if it is not in the cache (a cache miss), it is loaded from main memory (secondary level memory) if it is there;
- if it is not in main memory either, it must be brought in from disk.

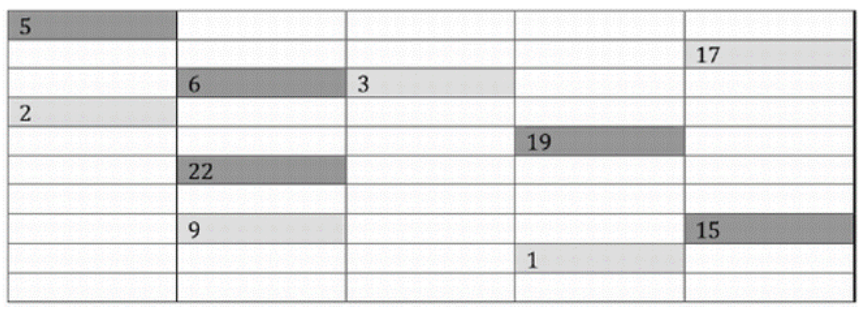
When the required data is not in memory there is a significant performance penalty since I/O devices are several orders of magnitude slower than the processor and memory.

The most common way to implement virtual memory is by dividing up main memory into fixed-size blocks called *page frames* usually 1K, 2K or 4K in size, and dividing programs and data also into pages of the same size. When the required data is not in memory, the corresponding page where the data is in secondary memory is brought in, and placed in one of the frames, and accessed there. Data transfers between disk and memory are always done in terms of pages.

When the data requested is in memory, the process waits for the transfer via the data bus. When data is required from disk however, the CPU cannot block processing until the data comes in, since the CPU would be idle for too long. Instead, the running process is put to sleep - it cannot run anyhow since the data it needs is not available - waiting for the transfer of the data from disk, as follows:

- a whole page is requested to be transferred from disk;
- the incoming page is loaded into memory where it is now available for access.

Of course, this means that contiguous program pages not necessarily result in contiguous page frames in memory. Figure 4.4 shows two processes pages in memory, the missing pages are in secondary memory.



Virtual memory works by translating each logical (*virtual*) memory address into a *physical* memory address. This is performed at run time, taking into account where in memory is each block located. In this way it is possible to:

- write programs as if there was only one user using the computer
- write programs independently of the memory capacity of the computer

All this operation is managed by the Operating System, which is in charge of determining whether the data is in memory and, if not, of locating where the data is on disk, and:

1. putting the process to sleep
2. if there isn’t a page frame free in memory, deciding which page is to be replaced to make room for the incoming page
3. issuing the read transfer
4. loading the page into memory (this accepts variations)
5. waiting until the transfer has finished

When the transfer is finished, the OS decides which process is going to run in the CPU next - it may not be the original process now sleeping - and dispatches this process to run in the CPU by copying all the process information into the CPU structures (registers, PC and IP).