**GET239 – Enterprise Technologies Lecture Notes**

# CPU and Memory: Design, Enhancement and Implementation Week 5

**CPU and Memory: Design, Enhancement and Implementation**

**-----------------------------------------------------------------------------------------------------------**

**1. Major Ideas and Concepts**

This week we’ll take a look various CPU and memory designs and architectures. Some of the major concepts we’ll explore will be:

* Describe the major concepts embedded in superscalar processing: pipelining instructions, the instruction unit/execute unit model
* Describe parallel instruction processing
* Explain the issues with handling out-of-order processing
* Describe the purpose of cache memory
* The different types of memory and how they are used
* The memory hierarchy and how each type of memory varies in speed and physical makeup
* The importance of cache and how it is used
* The concept of virtual memory and how paging facilitates it
* The introduction to the concept and importance of memory management

Let’s take a look at different kinds of CPU architecture first then memory.

**0. Modern CPU Methods**

We discussed how computer scientists have wrestled with ways to squeeze more performance out of a CPU without adding more expensive CPUs. The purpose of a computer is to accomplish a unit work by executing instructions and that work is measured by the number of instructions the computer can execute in a given unit of time. There are a number of ways that the number of instructions executed per unit of time can be increased. One way is simply increasing the number of CPUs, called multiprocessing.

Unfortunately this method is costly and has inherent timing and operating system issues. So designers came up with a solution to improve single CPU’s performance. The solution to increase CPU performance was to provide a means for executing instructions in parallel. This technique is called pipelining or scalar processing.

# Pipelining

A pipeline is the continuous and somewhat overlapped movement of instructions to the processor or in the arithmetic steps taken by the processor to perform all of the steps needed to fetch and execute an instruction. Pipelining is a technique that uses an “assembly line.” Similar to the assembly concept used in manufacturing. Without a pipeline, a computer processor gets the first instruction from memory, performs the operation it calls for, and then goes to get the next instruction from memory and so forth until all instructions are executed. While fetching the instruction, the arithmetic part of the processor is idle. It must wait until it gets the next instruction. With pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing arithmetic operations, holding them in a [buffer](http://WhatIs.techtarget.com/definition/0,,sid9_gci211713,00.html) close to the processor until each instruction operation can be performed. The staging of instruction fetching is continuous. The result is an increase in the number of instructions that can be performed during a given time period.

Pipelining is sometimes compared to a manufacturing assembly line in which different parts of a product are being assembled at the same time although; ultimately there may be some parts that have to be assembled before others. Even if there is some sequential dependency, the overall process can take advantage of those operations that can proceed concurrently.

Computer processor pipelining is sometimes divided into an instruction pipeline and an arithmetic pipeline. The instruction pipeline represents the stages in which an instruction is moved through the processor, including its being fetched, perhaps buffered, and then executed. The arithmetic pipeline represents the parts of an arithmetic operation that can be broken down and overlapped as they are performed. If one pipeline is good for improved performance could more pipelines be better? The use of multiple pipelines is known as superscalar.

# Superscalar

Superscalar design is sometimes called second generation RISC that describes a CPU design that makes it possible for more than one [instruction](http://WhatIs.techtarget.com/definition/0,,sid9_gci212356,00.html) at a time to be executed during a single clock cycle. In a superscalar design, the processor or the instruction compiler is able to determine whether an instruction can be carried out independently of other sequential instructions, or whether it has a dependency on another instruction and must be executed in sequence with it. The processor then uses multiple execution units (meaning more circuitry) to simultaneously carry out two or more independent instructions at a time. It effectively does this by dividing the CPU into two parts: the instruction fetch/decode unit and the execution unit then allowing these units to operate independently.

Superscalar processing on the surface sounds great, but it complicates the CPU design considerably. There are number of sticky issues that affect its performance:

* Out of order processing – executing later instructions before the results of previous instructions are known.
* Change in program flow - conditional branch instructions cause this to happen.
* The pipeline could be filled with instructions that will be branched around.
* Resource conflicts – instructions often use the same physical registers as either assigned by the programmer or the compiler.

**1. Memory Hierarchy**

The chart below gives approximate data access times. These ranges are for order of magnitude comparisons and will differ depending on the source. The storage components are ranked on a “per unit” from most expensive to least.

|  |  |
| --- | --- |
| **Storage Medium** | Average Access Time |
| CPU registers | Nil to 2 ns |
| Cache memory L1 | 3 – 10 ns |
| Cache memory L2 | 25 – 50 ns |
| Conventional memory | 30 – 90 ns |
| Expanded storage | 75 – 500 ns |
| Flash Memory | 10 - 20 ms |
| Hard disk | 5 – 20 ms |
| Floppy disk | 95 ms |
| Optical disk | 100 – 600 ms |
| Magnetic tape | .5 sec & greater |

**2. Types of memory**

There are many different types of memory and the most common question that arises is why so many? Do we need all of these different hardware implementations for storing data? I suppose the same question could be posed for a variety of other products that businesses and consumers use. In the case of memory computer scientists continue to explore new methods of making memory liking for reliability, improved performance and reduced manufacturing costs and standard methods to integrate memory with other computer components. So given the principles of Moore’s Law, it stands to reason that memory production materials and methods will continue to evolve giving us faster, better cheaper memory and mostly packaged in smaller containers.

Let’s take a look at the alphabet soup, that is, the acronyms used to identify the various memory types and implementation circuits in use today. Keep in mind that some of these may no longer be used or are used in niche applications. Also, keep in mind this is not intended to be an exhaustive list:

* Cache - usually very high speed, high cost, special purpose memory that serves as a buffer for frequently accessed data
* Random access memory (RAM) - not a great name since this form of memory can be written to and read from. Usually referred to as the primary memory for most modern computers
* Read only memory (ROM) - also a misnomer in that ROM can be written to, but more on this later
* Virtual memory - not real memory but a way into tricking the CPU into thinking it has more real memory than it really has.

Let’s take a look at these in more detail.

# Major Memory Differences

Memory architecture has evolved as production techniques allowed scientists and engineers to produce memory circuits that have different properties and cost structures. Sometimes what appears to be a great product may not be economically feasible for the general population of computer users. Let’s take a look at each major memory type in more detail get a better understanding for how and why they are used.

**Cache** – cache memory is often misunderstood. People hear the term cache and they think of the classic definition in Webster, “*a hiding place especially for provisions*.” Well if you think about this definition it’s close to how cache memory is used. The CPU uses cache as a hiding place for frequently used data values and instructions. In 1974, John Cocke of IBM Research studied the frequency and types of computer instructions that a typical general purpose computer used. He discovered that out of an instruction set of over 200 instructions, the same 10 instructions were executed 71% of the time. As an assembler programmer I found this to be true as well. I rarely used more than a dozen or so instructions in the instruction set. So the concept of “hiding” these 10 instructions close to the CPU’s execution unit makes allot of sense.

For cache memory to do its job it needs two things. It needs to be close to the fetch and execution units and it needs to be very fast. The sizes and number of cache memory units vary greatly. Just a few years ago some computer architectures had none or one cache unit. The costs were simply too high to mass produce them. As the cost of production them came down it became economically viable to add additional as well as larger cache units. We refer to these units as levels. You may have heard the term L1 cache and L2 cache. The cache level refers to its relative size and location with in the computer system. Typically L1 is smaller than L2 but L1 is located within the processing unit where as L2 may reside on a separate circuit between the CPU and main memory. The cache levels provide staging areas for data blocks that are commonly used. The fetch unit will look in L1 cache first if it doesn’t find what it is looking for it will then look in L2 if the data is not there it will look in main memory. Keep in mind that caching is a technique that is scalable and the number and sizes of cache can and will be different from CPU and computer manufacturer.

There are a number of methods employed that allow the CPU to get the data it needs to and from the various levels of cache. These are called cache mapping and replacement schemes.

These schemes rely on a variety of hashing, first in first out (FIFO), last in first out (LIFO), and least recently used (LRU) algorithms. The textbook does a nice job covering these mechanics so I won’t repeat them here. The important point to remember about caching is its contribution to computer performance and the principle in which it operates - locality of reference.

Here is an example I like to use. Susan D asked me to replace the toilet in our basement bathroom. I need tools to perform this task and my tools are in our detached garage about 30 feet from the basement. So one way to approach this task is to get the first tool I need and when I need a different tool walk up the basement steps and another 30 feet to the garage and back again. So think of the garage as memory and the tool as the instruction. You can imagine how long it would take to continuously going back and forth to the garage every time I needed a different tool. So another an approach is to cache all of the tools a few feet outside of the basement bathroom. Now when I need a different tool I just reach out a few feet and grab the next tool. Certainly a more efficient process … so you can see the CPU can executing more instructions per time unit if it can retrieve instructions from cache.

**Random access memory** – though not a great name for it since this form of memory can be written to and read from. It is most commonly used as primary memory for most modern computers. It is the primary storage for two main reasons: it is volatile and it is inexpensive. RAM is primarily used as a temporary staging area for programs and data that are currently active. The idea is when a program is needed to be run it is loaded from a more permanent storage area like disk and placed in memory. We refer to this state as being active. It may not be executing yet but it is actively consuming some of the computers resources.

Think about what happens when you double-click on a desktop icon, like Microsoft Word. The Word executable program (Word.exe code) is retrieved from disk and loaded into memory. When it is time to execute the program (i.e. instructions have been fetched - decoded - executed) the program goes from active mode or state to an execution state.

Here Word stays in memory until the program becomes inactive and the operating system recovers the memory areas allocated to Word or your computer loses power and the contents at all of the RAM addresses are all reset to binary zeroes. You might wonder why it is important that RAM be volatile. Imagine that a program is hung up or your computer “freezes” so that nothing works. Or a program like a virus seizes control of your computer. By simply turning off the power all programs that were active or in the state of execution are now gone.

There are two basic types of RAM: dynamic and static.

* Dynamic - constructed with both transistors and capacitors. The capacitors feed electricity to the transistors but require a recharge every few milliseconds to maintain the data.
* Static – constructed using flip flogs (not the sandals you see on the beach).Static RAM is denser, uses less power and produces heat but unfortunately is more expensive to produce.

# Read only memory

This form of memory creates quite a debate as to what it really is. Is it really “read only?” When ROM was first developed it was used as a form of memory that could be used for special purposes. The idea was that software (i.e. special purpose programs) could be recoded (“etched or flashed”) on it and that these programs remain there for the life of the computer. There are two industry terms used when using ROM in this way - firmware or microcode. What we learned was that the programs recorded on ROM needed to be maintained so the notion of read only is now more for historical significance.

There are a variety of flavors of ROM and they essentially work the same way but with slight variations: PROM (programmable read only memory), EPROM, (erasable read only memory), EEPROM (electrically erasable read only memory) and Flash memory which works like EEPROM but without the erasing byte at a time limitation.

The important principle to remember with ROM is that it is non-volatile so when power is not supplied the data remains recorded.

**Virtual memory**

Virtual memory is not memory at all. It is a technique for temporarily using disk space in lieu of real memory. In a multi-user system, the operating system creates the illusion that there is more “real” memory than actually exists. Real memory is the amount of hardware memory (RAM) contained on the memory cards plugged onto the motherboard. For example, my computer has 8 gigabyte of memory or approximately 8,000,000,000 eight-bit bytes for my Windows operating system to use for running programs. The System z13 can support up to 30 TB of real memory. The good news is that with proper memory management using a technique called “paging”, the operating system can create what is called virtual memory by moving infrequently used program segments, called pages, to disk and put in its real memory place a new program segment; thereby, creating an illusion that there is more memory than is really available.

As other processes become inactive, i.e. waiting for other resources such as I/O, the pages that were originally written to disk are then brought back into real memory to be executed. To the user, it looks as though the system is capable of processing many more programs concurrently than there is real memory available to support them.

The mechanism that makes virtual memory work is paging. The notion of paging is fairly straightforward. The real memory area is mapped into equal-size chunks called pages (we will call these frames later in the semester when we cover virtualization.) The size of these pages is adjustable as parameters supplied to the operating system so there is no one standard optimum page size but 4k size pages is the norm. But keep in mind this number based on the operating system, number of programs, program sizes and the amount of available real memory and real disk space.

Once memory is mapped into pages or page frames the information about which program segments reside in each segment is kept in a page table. Each program has its own page table allowing the operating system to find each page whether in real memory or on disk and whether it is active, inactive or in the state of execution.

The algorithms for paging are similar to those used for caching (FIFO, LIFO, HASH, and LRU) and vary from operating system to operating system. So Windows Server’s paging algorithms may be different than those used in Linux or z/OS but the underlying paging principles are the same. Your textbook does a nice job discussing one technique so I won’t restate that here.

**3. Memory Management**

Memory management is a way for assigning memory resources so that processes and data have the appropriate amount of memory for adequate loading and execution. For a multi-user system to be effective, each user’s processes need memory resources to be allocated equability. Memory is the first resource required before a process can be executed. So there is a strong correlation between the scheduling and memory management functions.

The amount of available memory limits the number of processes that can be scheduled and dispatched. Think of it this way, if there is only enough memory for one process, then a multi-user system effectively becomes a single-tasking system.

As a general rule the more processes that can be executed concurrently the better the overall computer system throughput – effectively the computer is doing more work during a given time interval. However, any one task or process may actually take longer in duration to complete then if it were executing in the system by itself.

Memory management is also one of the weaker components of some of the popular operating systems. Some operating systems will allow the user to keep requesting program execution even after all real and virtual memory have been consumed causing the systems to come to a standstill. You can demonstrate this on your home computer. Select a program like Internet Explorer or Word and keep double clicking on the icon 20 or 30 times or so. What happens? Did the operating systems queue these requests waiting for resources to become available or did it try to satisfy each request even when the resources were exhausted?

The more comprehensive operating systems manage memory more effectively by only running programs where there are available resources to run them then placing all other requests in a queue until resources become available.

**###**