Final Paper

Alicia Piavis

CPT 301 Computer Organization and Architecture

Sassan Sheedvash

9/4/2018

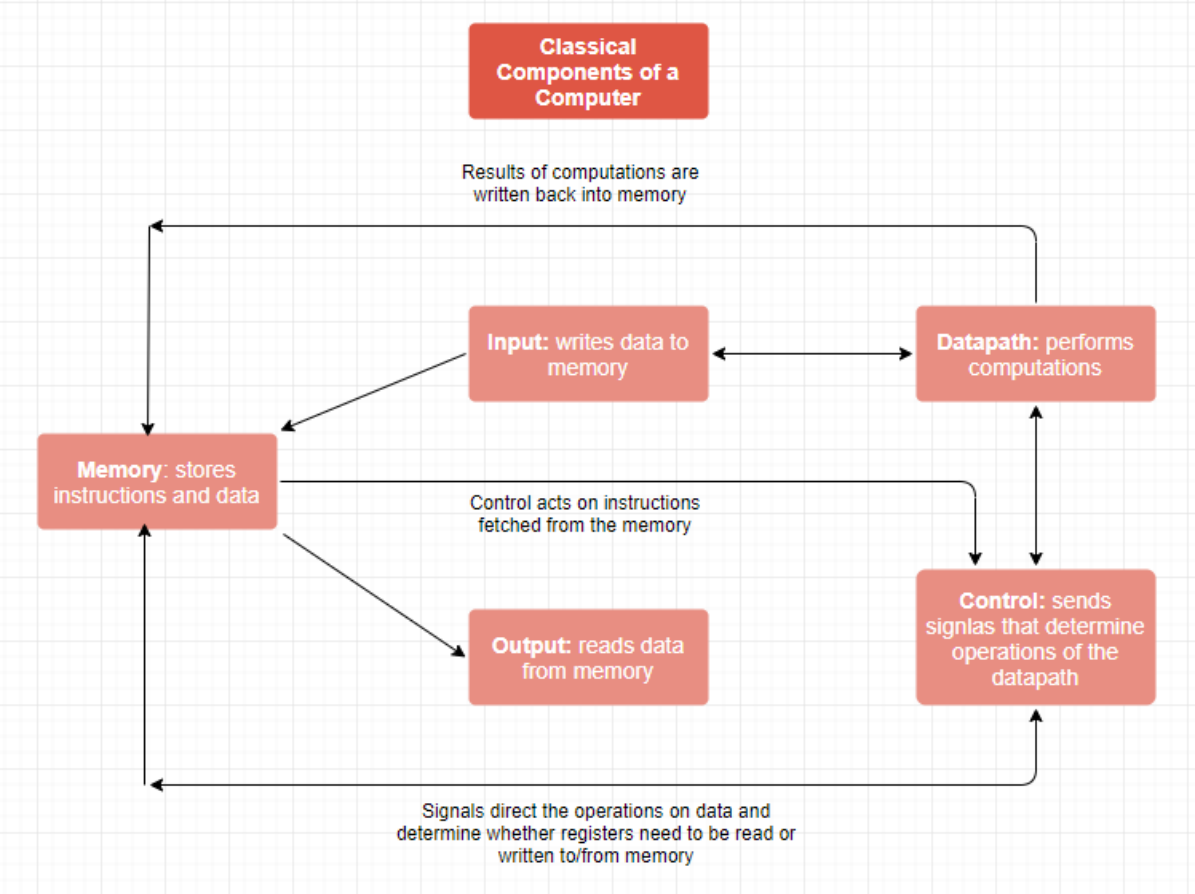
Week 1 Interactive Assignment: Computer Technology and Instructions

The five classical components of a computer are the input, output, control, datapath and memory. These components work together to allow for full functioning of a computer. For example, for a speech to text application, the program needs to access instructions for the program from the memory of the computer. Then it may require input through a microphone either embedded in the computer or connected to the computer. Once input is received, control directs how that data is manipulated. Datapath executes the computations to translate the input into text, and then output prints the data from the memory to the screen.

Computer performance is a broad term used to describe a relationship when comparing the operation efficiency of different devices. Before evaluating performance, a specific metric must be determined. The most common metrics used to describe computer performance are execution time (how quickly a single task is performed), and throughput (the rate at which multiple tasks are simultaneously completed over a given period of time).

At the most basic level, computers perform computations by following sets of instructions. An instruction is a line that includes an operator and operands. Possible operations include addition, subtraction, etc. Operands are the values or registers referred to in the instruction line, and are used in the computation. Hardware and software interface through sets of instructions. At the lowest level, software instructions are represented as a series of ones and zeros called machine language. These ones and zeros interact with the hardware through electric signals.

Link to Concept Map: Classical Components of a Computer



Week 2 Interactive Assignment: Arithmetic for Computers

Binary addition involves adding columns of bit pairs, in a right to left direction. The base TEN sum of the first vertical bit pair needs to be converted to base TWO. For example, if the sum of a vertical bit pair in base TEN is 2, this value represented in base TWO (or binary) is 10. The least significant (right) bit of the sum is dropped below the vertical bit pair that were added, while the most significant bit (left) is carried up over the next pair of vertical bits. Then the process of adding vertical bit pairs continues from right to left.

Subtraction is a "bit" different in the sense that vertical bit pairs can be subtracted very simply, unless a vertical bit pair results in a need to subtract 0-1. In this case, it may be easier to convert the second 32-bit binary number in the equation to the two's complement representation of the number. Once it is represented in two's complement, the two 32-bit numbers would

simply get added (since two's complement creates a negative version of the second 32-bit number in the equation). The process of adding again proceeds from right to left.

Multiplication and division operations through the ALU are fairly similar to elementary multiplication and division that can be done by hand. For example, in multiplication, the first operand is the multiplicand, and the second is the multiplier (Patterson & Hennessy, 2014). The process starts by multiplying the multiplicand by the least significant digit of the multiplier. The product is placed underneath. Then the multiplicand is multiplied by the second least significant digit of the multiplier, and the product is placed under the first product, but shifted to the left one bit. The process continues with bits from the multiplier moving right to left and the product always shifting one to the left from the product above. At the end, the sum of all of the products is the final product. Division is also performed in a similar fashion to long division, and may sometimes result in a remainder, in addition to the quotient.

Example of Multiplication:

```
                    1000
x                   1001
                    1000
                   0000
                  0000
                 1000
Product =        1001000
```

Example of Division:

```
      1001    Quotient
1000 |1001010
  -  1000
       10
       101
       1010
     - 1000
       10     Remainder
```

Floating point numbers “represent numbers in which the binary point is not fixed” (Patterson & Hennessy, 2014). For example, the number 5.8×10^{15} is written in scientific notation, and can be represented in a program as a floating-point number. As discussed previously, computations with floating point numbers can occasionally cause problems, especially when there are limitations in the register size. Limitations in hardware can cause undesirable outcomes due to lack of precision. As exemplified by Patterson and Hennessy (2014): “let's see if c + (a + b) = (c + a) + b. Assume c = -1.5ten × 1038, a = 1.5ten × 1038, and b = 1.0, and that these are all single precision numbers.”

c + (a + b) = -1.5E38 + (1.5E38 + 1.0)

= -1.5E38 + (1.5E38)

= 0

(c + a) + b = (-1.5E38 + 1.5E38) + 1.0

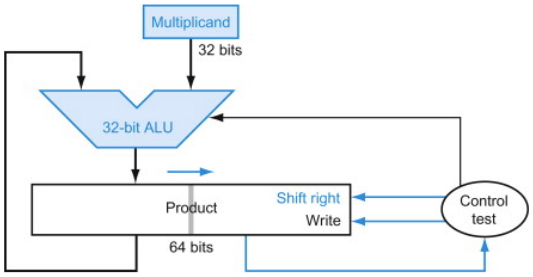
= (0.0) + 1.0 = 1.0

= 1.0

According to Patterson and Hennessy (2014), parallelism is used “to perform simultaneous operations on short vectors of sixteen 8-bit operands, eight 16-bit operands, four 32-bit operands, or two 64-bit operands. This allows operations to occur more efficiently: “parallelism occurs within a wide word, the extensions are classified as subword parallelism” (Patterson & Hennessy, 2014).

Link to Flowchart: [Multiplication Algorithm]

Hardware Diagram (COD Figure 3.3)



Multiplication Algorithm Flow Chart

Week 3 Interactive Assignment: The Processor

Computer performance is a hot topic in technology today, due to the growing need to process large amounts of data as quickly as possible. Performance can be affected by instruction count, clock cycle time, and clock cycles per instruction. While the instruction count is determined by the compiler and instruction set architecture (ex. MIPS), the processor affects the clock cycle time and cycles per instruction (CPI) (Patterson & Hennessy, 2014).

Implementation depends on both the instruction set, as well as the processor and associated elements (hardware components). Basic implementation of an instruction involves fetching the instruction from the correct address in memory, reading the register(s) expressed in the instruction, performing a computation in the ALU (either an operation for an R-type instruction, calculating the next memory address for a memory instruction, or comparing for a branch instruction), accessing an operand in the data memory, and writing the result to a register.

All instruction types need some basic elements in the datapath. In order to fetch instructions, a memory unit is required, which stores instructions for the desired program. In addition, the program counter (PC) is used to fetch the instruction from the correct memory address. An adder (part of the ALU) is necessary to compute the address for the next instruction. Other elements utilized in the datapath depend on the instruction type. For example, R-type instructions require the register file (which contains registers holding operands), and the ALU to perform operations. Memory instructions require the data memory unit (to load from or store words to), as well as the sign extension unit. Branch instructions use the ALU to evaluate a branch condition, and require an additional adder to compute the new branch target if the conditional evaluates to true.

Patterson and Hennessy (2014) provide a great example of the datapath for an R-type instruction: add $t1, $t2, $t3. In order to build this datapath the processor would need the memory unit (which contains the instructions for the program), the PC (which contains the memory address of the instruction), the adder (to compute the address of the next instruction if there was one in the program), the ALU (to execute the operation: add $t1, $t2, $t3), and then the result would be written back into the register file in register $t1.

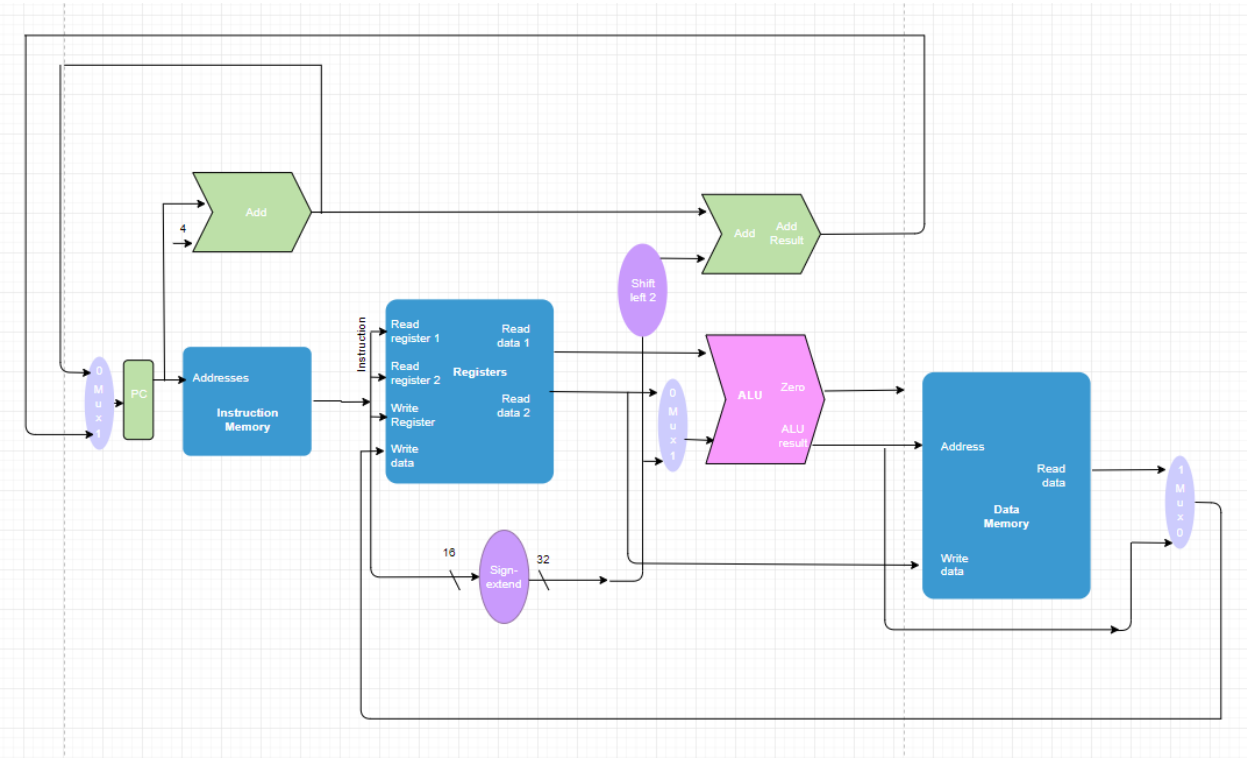
Pipelining increases the performance of a processor by allowing multiple instructions to be executed simultaneously. Historically, a single-cycle *without* pipelining only allowed for one instruction to be executed per clock cycle. However, in single-cycle *with* pipelining, modern processors allow datapaths for multiple instructions to run simultaneously, as long as the elements needed for a subsequent data path are available. Therefore, rather than a single cycle resulting in the execution of one instruction, a single cycle results in the execution of multiple instructions. Even though the execution time for a single instruction is not affected, the throughput is much higher. Pipelining is accompished by organizing the execution of instructions into five stages. Once a datapath is past a certain stage, elements from the previous stage become available for use by the next instruction. Therefore, instructions are executed in a staggered fashion in the pipeline approach.

To provide an example, the five stages in a pipelined approach are: 1) IF- Instruction Fetch; 2) ID- Instruction decode and register file read; 3) EX- Execution or address calculation; 4) MEM- Data memory access; and 5) WB: Write back. In single-cycle *without* pipelining, an instruction would go through each of the five stages (IF, ID, EX, MEM, and WB) completely, before the next instruction begins execution. On the other hand, in single-cycle *with* pipelining, new instructions begin execution before the previous instruction has completed the cycle. In

other words, if there are three instructions in a program, instruction one goes through stage IF in the first clock cycle. During the second clock cycle, instruction one is in stage ID, while instruction two is in IF. During the third clock cycle, instruction one is in EX, instruction two is in ID, and instruction three is in IF, and so on. The differences in single-cycle without versus with pipelining can be seen in the figures below.

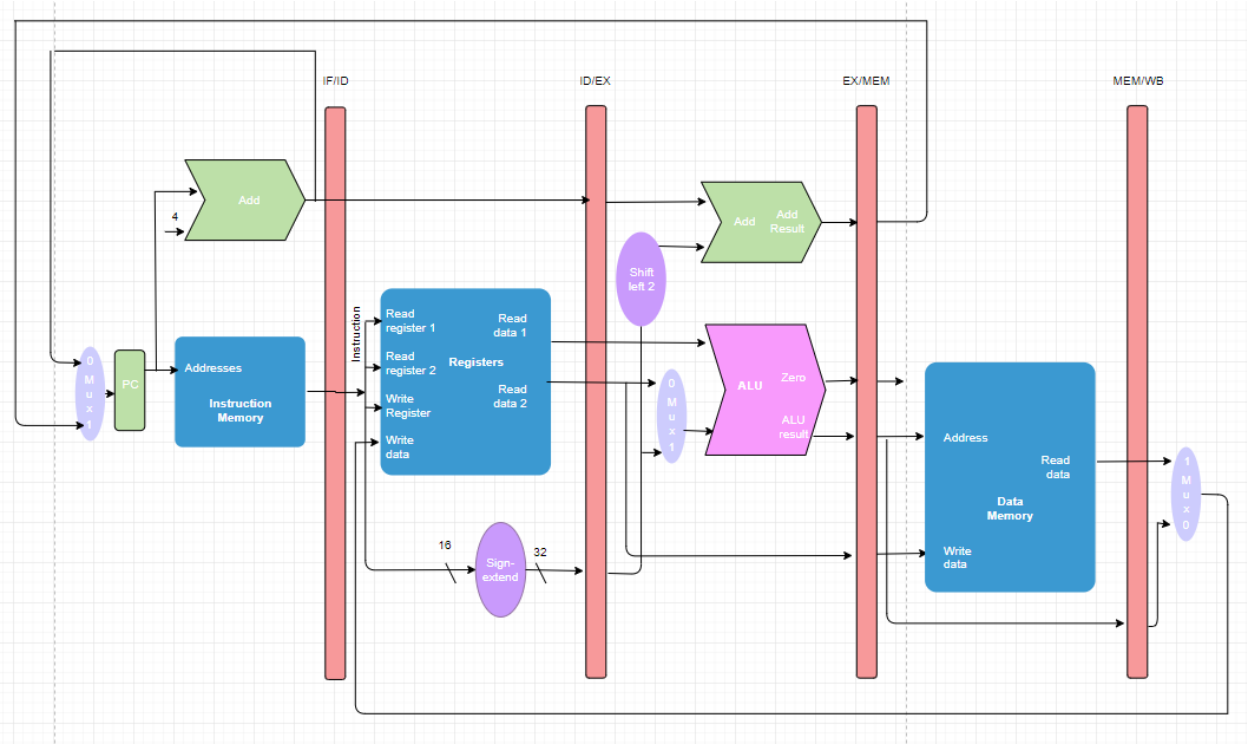
Link to Diagram: [Single Clock Cycle Without Pipelined Performance]

The Single-Cycle Datapath Without Pipelining



Link to Diagram: [Single Clock Cycle with Pipelined Performance]

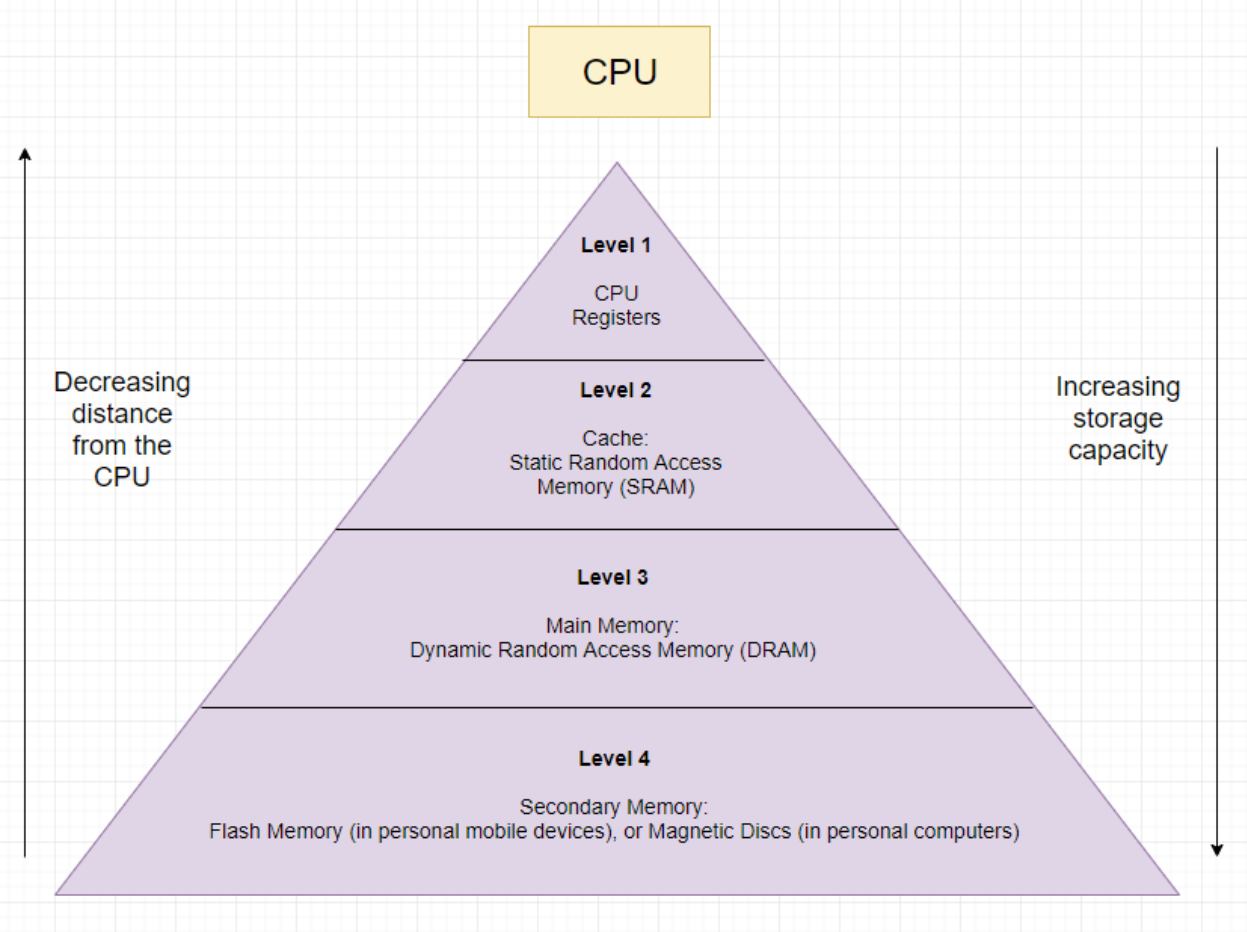
The Pipelined Version of the Single-Cycle Datapath



Week 4 Interactive Assignment: Exploiting Memory Hierarchy

This week's diagram (COD Figure 5.3 from Patterson & Hennessy below) reflects the organization of the memory hierarchy. A memory hierarchy is used in the architecture of computers so that, "the user has the illusion of a memory that is as large as the largest level of the hierarchy, but can be accessed as if it were all built from the fastest memory". The hierarchy is organized in such a way that the fastest memory (cache) has the smallest storage capacity and is located in the closest proximity to the processor, while the slowest memory (secondary memory) has the largest storage capacity, and is the farthest away from the processor. In addition, the fastest memory is the most expensive, while the slowest memory is the least expensive (Patterson & Hennessy).

Link to Diagram: Memory Hierarchy



At the top of the memory hierarchy is the CPU registers (Level 1 above). The next level is the cache (Level 2 above). The cache is SRAM (static random access memory). It is closest to the processor, more expensive than DRAM (dynamic random access memory), uses more area per bit of memory, needs minimal power, and doesn't need to refresh. Next in the hierarchy is the main memory (Level 3 above). The main memory is DRAM, and it is farther from the processor than SRAM, denser and cheaper, uses less area per bit of memory, needs more power, and needs to be refreshed. The next level in the hierarchy is secondary memory (Level 4 above). Personal mobile devices use flash memory, which can get worn about by writes, so flash memory utilizes wear leveling to shift block around to prevent wear. The secondary memory in most PC's use magnetic disc hardware. Magnetic discs are the cheapest and largest hardware.

They are a collection of metal platters covered in magnetic recording material, and a read-write head aligns tracks and then sectors to read and write data (Patterson & Hennessy).

According to Patterson & Hennessy (2014), cache is "a term is also used to refer to any storage managed to take advantage of locality of access." The simplest form of cache is direct-mapped, meaning that each memory location is mapped to one location in the cache (Patterson & Hennessy, 2014). In order to determine whether an address reference is a hit or a miss, the address is broken down into two parts, the index and the tag. The index consists of the lower bits of the address and correspond to a location in the cache. If the index reference results in a hit in the cache, then the tag (the upper bits in the address) are used to identify whether the word in the cache matches the requested word. The first few references to an empty cache will come back as misses. Then the cache fills up with data that has been copied from the memory to fulfill those misses. When a reference results in a miss that prompts the retrieval of data from a memory address that conflicts with a full index in the cache, the new data overwrites the contents at that cache index.

Cache performance is measured by a number of different indicators including memory-stall cycles (from cache misses), read-stall cycles, write-stall cycles, miss rate, and average memory access time (AMAT), which accounts for both hits and misses. These indicators of performance can be improved in a number of ways. For example, "Larger blocks exploit spatial locality to lower miss rates" (Patterson & Hennessy, 2014). However, if blocks become too large relative to the cache, the number of blocks decreases, and there is competition between blocks, which can slow performance. A write buffer can also improve performance by storing data that is waiting to be written to memory. Write-back schemes can increase performance by waiting to modify blocks in lower regions of the hierarchy until the block is replaced. A split cache (separate

caches for instructions and data) in conjunction with increased bandwith can allow blocks to be transferred more efficiently. Other methods of improving performance include increasing the clock rate, allowing for more flexible placement of blocks, and utilizing a second-level cache to reduce the miss penalty from the first cache.

Dependable memory hierarchy utilizes the principle of redundancy. Losing data can be catastrophic. In an attempt to prevent this from happening, and to make memory more dependable, computer architects have found three ways to improve Mean Time to Failure (MTTF). Mean Time to Failure is a measure of "the percentage of devices that would be expected to fail in a year" (Patterson & Hennessy, 2014). The three ways to improve MTTF are:

1) Fault avoidance: Preventing fault occurrence by construction.
2) Fault tolerance: Using redundancy to allow the service to comply with the service specification despite faults occurring.
3) Fault forecasting: Predicting the presence and creation of faults, allowing the component to be replaced before it fails. (Patterson & Hennessy, 2014)

Through the consideration of these guidelines in the design process for modern computers, architects have been able to establish a dependable memory hierarchy for businesses and consumers.

Virtual machines are, "emulation methods that provide a standard software interface" (Patterson & Hennessy, 2014). For example, I used a virtual machine when I completed the Programming Concepts course here at Ashford University and we utilized a virtual Java environment. Virtual machines have become desirable for a number of reasons. They provide isolation and security, avoid failures in security and the reliability of operating systems, allow sharing of one computer (especially in cloud computing), and have an overhead that is more

acceptable than in the past (Patterson & Hennessy, 2014). Virtual machines run on a host hardware platform and provide the illusion of a fully functional operating system independent of the user's own.

Virtual memory is desired because of "efficient and safe sharing of memory among multiple programs, such as for the memory needed by multiple virtual machines for Cloud computing, and to remove the programming burdens of a small, limited amount of main memory" (Patterson & Hennessy, 2014). Virtual memory overcomes the limitations of traditional memory and allows users to share a machine without interfering with the activities or devices of other users. Reading and writing is done independently with no affect to other users. In contrast to traditional memory, virtual memory blocks are known as pages and virtual memory misses are known as page faults (Patterson & Hennessy, 2014).

Week 5 Interactive Assignment: Computer Architecture Parallelism

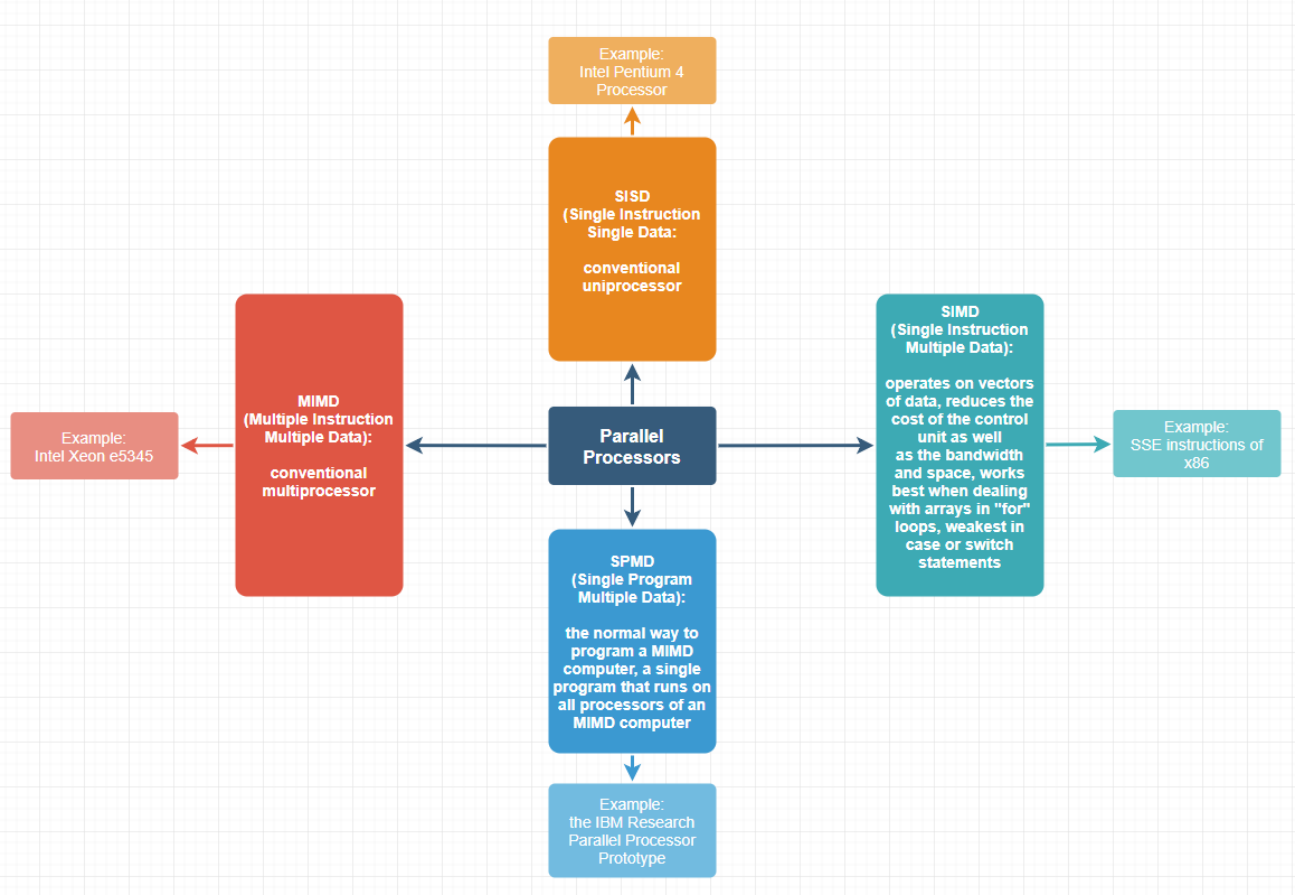
The overall goal of computer architecture parallelism is to increase performance and throughput. It is important to recognize that there are different types of parallelism: task-level parallelism (or process-level parallelism) and parallel processing programs. Task-level parallelism utilizes "multiple processors by running independent programs simultaneously" (Patterson & Hennessy, 2014). A parallel-processing program is, "a single program that runs on multiple processors simultaneously" (Patterson & Hennessy, 2014). Regardless of the type of parallelism used, the goal is to design software that runs more efficiently on a microprocessor than on a uniprocessor implementing instruction-level parallelism.

Part of the drive for increased performance from processors is due to the need for solutions to scientific and business problems that require extreme processing power at reduced costs and low energy. According to Patterson & Hennessy (2014), the energy problem "means

that future increases in performance will primarily come from explicit hardware parallelism rather than much higher clock rates or vastly improved CPI." In addition to hardware parallelism, programmers now need to design and build code that optimizes these advancements in hardware. For example, "sequential code now means slow code" (Patterson & Hennessy, 2014). In the past, software was designed for uniprocessors, but today's developers design for the future in anticipation that the cores per chip will be scaled.

In addition to the evolution of multicore microprocessors, computer architects have used these CPU's as inspiration for a different type of processor, Graphics Processing Units (GPU's). GPU's emerged as a result of an increased demand for gaming. There are a number of differences between GPU's and CPU's. For example, GPU's only execute a subset of the functions that a CPU can. However, this "allows them to dedicate all their resources to graphics" (Patterson & Hennessy, 2014). Therefore, what a GPU cannot handle, the CPU makes up for. In addition, "The GPU problems sizes are typically hundreds of megabytes to gigabytes, but not hundreds of gigabytes to terabytes" (Patterson & Hennessy, 2014). GPU's also have a different approach to hiding memory latency. Instead of utilizing multi-level caches, they rely on hardware threading. Finally, memory in GPU's emphasizes an increase in bandwith, and GPU's have more processors than CPU's. For example, the new NVidia Titan V GPU has 5,120 cores (Nvidia Corporation, 2018)!

Link to Concept Map: Parallel Processors



Fundamental Concepts that Influence Computer Organization and Architecture

The fundamental concepts that influence computer organization and architecture include the instruction set architecture, arithmetic and operations, parallelism, hardware elements, software, the processor, control, datapaths, pipelining, and memory hierarchy. Even when writing software, it is important for programmers to understand the fundamentals of computer architecture, since a smooth interface between the hardware and software can either increase or degrade the performance of a system. Organizations will utilize these concepts to increase the life of devices, anticipate additional cores in future processors, avoid errors in performance, make software migrations from uniprocessor to multiprocessor systems more seamless, and increase performance while maintaining minimal weight, energy consumption, and cost in future devices.

Resources

Multiplication. (n.d.). Retrieved 2018, from http://www.massey.ac.nz/~mjjohnso/notes/59304/l5.html

Nvidia Corporation. (2018). Introducing NVIDIA TITAN V: The World's Most Powerful PC Graphics Card. (n.d.). Retrieved September 1, 2018, from https://www.nvidia.com/en-us/titan/titan-v/

Patterson, D. A., & Hennessy, J. L. (2014). Computer organization and design: The hardware/software interface (5th ed.). Retrieved from https://zybooks.zyante.com/#/zybook/jCx8rOUvAL/gettingstarted