Computer Systems: Gateways To Cyberspace

A Story About How Computers Work For the Absolute Beginner

by Ted Kosan

Part of The Professor And Pat series
(professorandpat.org)

Table of Contents

The Golden Age of Personal Computers....4
Computer Technologists Must Be Motivated Self-Learners....8
Learning How To Learn Technical Subjects....9
  Learn even the uninteresting parts....10
  Increase your capacity for reading as much as possible....10
  Find a quite place to study and use it....10
  Minimize distractions....11
  Figure out what math is all about....12
The Von Neumann Architecture....13
How Does A Computer Work?....14
What Do The Numbers In Memory Locations Mean?....22
Contextual Meaning....23
What Provides The Context For The Numbers In A Computer's Memory?. .25
CPUs: Calculators Without Buttons....25
The Program Counter And The Instruction Register....28
Running A Machine Language Program....29
Mnemonics....35
Machine Language....39
Assembly Language....39
The 6502 CPU's Instruction Set....39
Low Level Languages And High Level Languages....41
Compilers And Interpreters....42
The Three Types Of Computer Memory....43
RAM (Random Access Memory)....43
A CPU Is A Very Dumb Device....45
ROM (Read Only Memory)....47
ESD (Electro Static Discharge)....49
BIOS And POST....51
I/O Memory....53
Loading An Operating System....57
Operating Systems: Bridges To Cyberspace....58
Systems Within Systems....60
Physical Parts Are Costly And Constrained....62
Moving Parts Into Cyberspace....62
A Lock Made Of Physical Parts and Cyberspace Parts....63
Microcontrollers: Computers On A Chip....64
Operating System: The Part Of A Computer Which Is Made From Cyberspace Parts....65

Application Programs....66
Computers Without Operating Systems....67
Back To Loading An Operating System....67
Primary Storage And Secondary Storage....68
General Purpose Computers Vs. Specific Use Computers....69
Running An Application Program....70
Something Is Missing From The Models....71
Wink Of An Eye....72
I Want To Learn More About Computer Software And Hardware....75

The Golden Age of Personal Computers

> One morning teacher and I were troubleshooting an electrical malfunction on a friend's automobile. “Teacher," I said “When did you know what you wanted to be when you grew up?" Teacher pulled the oscilloscope probe away from the computer circuit it was probing, smiled and said “I have been grown up for a while now and I still do not know what I want to be! I can remember, however, the day I had finally saved enough money to purchase my first computer. I took it home, set it up and within an hour I had written my first computer program. At that point I laughed and thought 'I don't know what I want to be when I grow up, but I am absolutely certain I want these wonderful machines to be a part of it!'.

I consider myself to be an aspiring universal technologist which means that I am interested in all aspects and areas of technology and I strive each day to learn something new about technology that I did not know before. I assume you are reading this book because you too are interested in technology and specifically there is something about computers that you find attractive. You and I have something in common, then, because I fell in love with computers when I was a Senior in High School in 1982 and I have become ever more deeply involved with them since that time.

The early 1980s was a wonderful time to become involved with computers because inexpensive PCs (Personal Computers) became available for the first time. These machines generated a great deal of excitement and this excitement resulted in a wide-range of what I call “first generation” educational materials being written for these machines.

What I mean by **first generation educational materials** are the educational materials that are created for a technology just after it becomes available. When a technology first appears, the people who write about that technology assume that almost nobody knows anything about it and therefore authors are especially careful to move slowly and not miss critical information. Beyond this, many of these authors were beginners with the new technology themselves not too long before creating their educational materials and so all the critical little pieces of information, that expert authors with years of experience tend to forget, are still fresh in their minds.

My first computer was a **Commodore 64** (http://en.wikipedia.org/wiki/Commodore_64) and I took full advantage of the excellent educational materials that were available for it.

The User's manual for the Commodore 64 taught the beginner how to program the BASIC programming language from scratch and the Commodore 64's Programmer's Reference manual contained the machine's complete electrical schematics, a description of each main chip's function, specifications for all of the Input/Output (I/O) ports and explanations of its Central Processing Unit's (CPU) machine and assembly language.

At that time, I thought the Commodore 64 was wonderfully complex and intriguing and it was not until much later that I realized how truly simple the Commodore 64 (and its contemporaries) were. Most of the personal computers from that time had educational materials available similar to what the Commodore 64 had and I think this unique mix of inexpensive price, relative simplicity, excellent first generation educational materials and high community excitement level created the ideal conditions under which to learn how computers truly worked. Many of the great software developers, hardware engineers and hackers of today first learned how computers worked during the 1980s and it is my opinion that the unique conditions existing at that time enabled them to gain the deep understanding of computers that is the core of their success today.

That Golden Age of computers occurred a while ago, however, and an amazing amount has changed since then. The changes include personal computers that are orders of magnitude faster than the PCs of the early 1980s, the Internet, the World Wide Web, Cell Phones and enormous amounts of educational materials on computers and computing that is expanding at an increasing rate.

There is more of everything in the world of computers now than was available in the 1980s, but unfortunately this 'more of everything' in computing that we have now provides an unfriendly environment within which to learn about computers if you are a beginner. Here an example of what I mean.

When I purchased my Commodore 64 I brought it home, attached it to my television (it did not need a special monitor) applied power to it and waited the 5 seconds (!) it took to boot. The operating system was stored in the system's main board so it did not have to load from an external storage device. The keyboard was built into the computer itself so the only wires that needed to be attached were the power cord and the cable that went to the television. The main screen (actually the only screen...) consisted of a command line interface that waited to accept **BASIC** (Beginner's All Purpose Symbolic Instruction Code) language commands and programs as input.

Right there on day one, within 5 seconds of booting, the 1980 era machine led the beginner directly and naturally to learning their first programming language. Learning their first programming language is critical for a beginner because it gives invaluable insights into what a computer is and how it works. The knowledge gained from learning that initial programming language makes it much easier to learn further programming languages. It also opens doors for learning about all the other aspects of computers.

Another subtle benefit of the early 1980s era computers is that they guided the beginner into learning how to touch type because typing was the only way to communicate with these machines. If you have observed an excellent programmer at work you can appreciate how valuable the skill of touch typing can be.

In contrast to the 1980s PC, a typical modern PC provides a horrible educational experience for the beginner. After unpacking the main unit and the monitor, you have to plug in the power cord, figure out which of the 10+ connectors on the back of the machine the monitor plugs into and plug it in without bending any of the little, delicate pins. Then you have to plug in the keyboard, plug in the mouse and plug in the network connection or the phone line.

If you are lucky, the machine has the operating system already installed on it and, if it does not, perhaps one half to one hour of time is required to do so. We will not even bring up the “fun” involved with locating and making sure all of the needed device drivers are installed correctly...

We will make this “easy” and assume that the operating system is already installed. More likely than not, the computer is using the Windows™ operating system so lets power up and wait for it to boot... and wait... and wait... for over a minute in most cases. The GUI (Graphical User Interface) that finally comes up is nice, but the level of complexity that the user is presented with is astounding when compared to the simple user interface that a machine like the Commodore 64 presents.

Up to this point, things are bad enough but unfortunately they are about to become worse. Search as much as you like but you will not find the software tools needed to write even a simple program on the machine! The early 1980s computers presented a programming language as the first tool that a user encountered after booting the machine. With most current computers, however, a programming language is not even included with the computer!

What this means, in my opinion, is that a beginner who wants to learn about the fundamentals of computers today will find this task significantly more challenging than the beginner did in the early 1980s. The task is more challenging but, then again, the returns on one's investment of labor are significantly greater too. Computer technologies have moved into almost all aspects of society and their growth continues to expand at an increasing rate. However, for those who are willing to discipline themselves, focus their efforts and invest the hard work it takes to master the fundamentals of computer technology, the rewards are well worth the effort.

The Golden Age of personal computers is part of the past now and the unique environment it provided for deep, natural learning of computer fundamentals is part of the past too. While I can not bring that age back, I did live through it and I think I can pass some of that age's magic along to you if you are willing to work hard to learn the information that I am going to be guiding you through in this document. You see, one of the secrets of success that age taught us was not what we learned but rather, the way we learned it.

> One summer afternoon Teacher and I were installing a sonar system on a boat at the lake. “Teacher," I said “What is the secret to effective learning?" Teacher looked at me, cocked an eyebrow, paused and then grabbed me by the back of the neck and pushed my head under the water. Teacher's reaction surprised me so much that I did not have time to take a deep breath before hitting the water and I was soon struggling. Teacher finally pulled me up and, after I had recovered somewhat, asked me what the thing I wanted most was when I was under the water. “Air!” I replied “The only thing I wanted was Air!" Teacher then said “In order for your learning to be effective, you must want to learn the thing you are learning as much as you wanted air when your head was under the water. That which is learned without desire is soon forgotten. That which is learned with great desire, however, is knowledge that will be remembered forever.” (A modification of an old parable).

Computer Technologists Must Be Motivated Self-Learners

When I was a senior in High School in 1982, inexpensive personal computers were so new that our school only had a few and none of the teachers in the school knew how to program them. Since none of the teachers knew how to program these computers, no programming classes were offered. If we wanted to learn about these machines, we had to do so on our own. While some schools in the world did have classes on programming, many did not and even the ones that did were not very in depth.

At the time, I thought I was very unfortunate to be in a school that did not have classes on computers but I was to eventually find out that this misfortune was actually a wonderful blessing in disguise. I think that this blessing was one of the significant benefits that the golden age of personal computers provided for the people who learned about computers during that time.

Since I could not take a class on computers at school, I would go home at night and sit in my bedroom and look at my computer. There I was, there was the computer and next to it on the desk were the computer's User's manual and its 2 inch thick Reference manual. I looked at the computer, looked at the User's manual then looked at the Reference manual. There was nobody to teach me about the computer. There was nobody to ask questions to about the computer. And it was so utterly quiet...

Finally (for the first time in my life) I picked up a technical book (the User's manual) and I started to actually read it... Nobody had told me to do this. Nobody had assigned this work for me to do and nobody was going to test me over what I had read. I started reading the book because I desperately wanted to learn how to use my computer and reading the book was the only way I had to do this. After reading the first few pages of the book, however, a surprising thing started to happen. I became so interested in what I was reading that I lost track of time and before I knew it an hour had passed. During that hour I had learned how to write my first BASIC program and, while it was not easy to do, there was little pain involved because I was learning this information because I really wanted to.

It took a month or so to work my way through the User's manual and, believe it or not, it took me years to master all of the material that was contained in the Reference manual. While the information I was learning greatly fascinated me (and still does) the surprising lesson that I learned was that it was not only possible to learn deep technical information on one's own, it was actually a very efficient method for doing so. Even later I discovered that self-motivated learning is the only kind of learning that is effective. As the Teacher said earlier, “That which is learned without desire is soon forgotten.”

And guess what? Today technology is changing the world at such a fast (and ever increasing) rate that the only way to keep up with this constant change is to be a **continuous self-motivated learner**. The following quote, from Computer Scientist and futurist Ray Kurzweil, supports this statement:

> "An analysis of the history of technology shows that technological change is exponential, contrary to the common-sense 'intuitive linear' view. So we won't experience 100 years of progress in the twenty first century—it will be more like 20,000 years of progress (at today's rate).”

The implications of this passage are that technology is changing so quickly that it is becoming impossible for teachers to learn the new knowledge fast enough to then pass it on to their students. Therefore, like it or not, if you have the desire to become a computer technologist, then you have no choice but to become a self-motivated learner yourself.

Learning How To Learn Technical Subjects

Many of the technical books I have read in my life include a message in the preface of the book that informs the student that if they do not **read the book carefully**, **do the assigned problems**, **ask questions** and generally **get actively involved**, they will not learn the subject material. I have found this advice to be very true and I recommend that you follow it. In addition to this sound advice, however, I am going to add some more thoughts of my own that you may find helpful.

Learn even the uninteresting parts

In an earlier section we already addressed the fact that learning without desire is not very effective. What little is learned it usually retained only long enough to pass a test and then it is quickly forgotten. The problem here is that not every part of a subject you are studying is going to deeply interest you. You might be tempted to skip these parts but unfortunately **these uninteresting parts usually contain information that is necessary for fully comprehending the parts of the subject that do interest you**.

It may not be pleasant, but you have no choice but to force yourself to learn even the uninteresting parts of a subject. A hidden benefit is that sometimes the material that interested you the least in the beginning turns into a passionate subject sometime later.

Increase your capacity for reading as much as possible

Until engineers create a way to plug a high-speed network connection directly into your brain, reading is the most concentrated means available for pumping deep, detailed knowledge into your mind. Take a few moments and think about all of the wonderful inventions that have been developed over the past 100 years. From the automobile and airplane to radio, television, computers, skyscrapers and satellites, practically all inventions are the direct result of the inventor's ability to read and comprehend technical literature.

If you desire to become a computer technologist of some type, you absolutely have to be a strong and continuous reader. If you are already a strong reader but have not acquired the habit of reading technical literature, then make an adjustment to your reading mix and start as soon as possible. If you are not currently a strong reader, make the resolution that this is a skill you are going to begin to develop right now. If deep technical literature is too much of a challenge to start with, then pick an area of literature that interests you (such as science fiction, fantasy, etc.) and start there.

Find a quite place to study and use it

This piece of advice cannot be stressed enough. **If you do not have a**

quiet place to study, you are never going to learn any technical subject at a deep enough level in order to succeed. Most technical information is absorbed by the mind very slowly and it will require you to study and restudy it during frequent blocks of quiet time measured in hours in order for you to understand it.

You may have to be very creative in order to solve this problem, but solve it you must. If your house is not quiet during the day, think about getting up earlier to study or study later after everyone has gone to bed. If you have easy access to a public library, or other quiet facility, make use of it. If you have to go to a relative's or friend's house, then do it.

You may even have to resort to drastic measures like a garage, a barn or deep in the woods in order to find a quiet place to study. It is said that in Tibet, monks voluntarily allow themselves to be sealed in caves high up in the Himalaya mountains for years at a time so that they can meditate in peace and quiet. A monk will enter a small cell that has been carved in the back of a cave and the opening is then sealed with bricks and mortar. Every day an attendant passes food and water to the monk through a small hole in the wall, but other than that they are completely alone for perhaps 5 years at a time.

If some Tibetan monks are able to mediate in a silent cave for 5 years straight, certainly you can work yourself up to quietly studying for an hour or two at a time!

Minimize distractions

> To sacrifice means to surrender or give up something for the attainment of some higher advantage or dearer object. http://miriams-well.org/Glossary/

Certain occupations require greater amounts of focus, effort and devotion than others and most areas of computer technology fall into this category. It is an unpleasant but obvious fact that time spent doing a given activity is time that cannot be spent doing another activity. If the hours in a day were unlimited, then this would not be a problem. Since this is not the case, however, certain activities will need to be sacrificed on a daily basis in order to devote that time to studying computers.

What kind of activities might you want to sacrifice? How about watching television, surfing the Internet, talking to friends on the phone and (the big one) playing computer and video games. During my first semester attending college I nearly flunked out because I spent most of my time playing computer games instead of attending class and doing my assignments. A significant number of my students over the years have fallen into the same trap and I have noticed that the problem is getting progressively worse.

If you want to succeed as a computer technologist, then you absolutely have to sacrifice the activities in your life that waste your time and squander your energies.

Figure out what math is all about

If you are already proficient in math, then you can skip this section. If you are a person who struggles with math, however, you are going to need to do something about this. The solution is not easy, but at least it is straight-forward. What you need to do is to start from square one, locate a good arithmetic book and work through it cover to cover. This means starting at page one, reading each chapter until you understand the material and then work ALL of the problems at the end of the chapter.

When you have finished the arithmetic book, locate a good algebra book and work your way through that one too. Keep working through increasingly more advanced mathematics books, indefinitely. Visit used book stores on a regular basis and start accumulating math books of all types. The Internet is the ultimate way to obtain inexpensive used math books so make good use of this resource too. Never get rid of your math books, even after you have worked through them, because you will want to use them as a reference later.

The Von Neumann Architecture

Imagine that you were sitting at your PC working on a document and a friend came over to you, pointed at the PC and asked “What is in that box?” What would you say? You might be tempted to throw some **buzzwords** (http://en.wikipedia.org/wiki/Buzzwords) at them, hoping that they would be satisfied and move on. You could say "That box is a computer and it contains the following items:

- CPU
- RAM
- ROM
- Hard drive
- Flash drive
- CDROM drive
- Network card
- Motherboard"

Most people, however, would not be content with this weak substitute for a true explanation and they would want to know what each of the above items did and how they all fit together. At this point you are stuck. You are going to have to set aside your work for awhile and try to explain how a computer works in a way that is as understandable as possible.

The fortunate thing is that the primary set of ideas upon which most computers are based are relatively simple. Once you understand these ideas, and how they interact with each other, you will be able to look at almost any computer (from the computer that controls a car's engine, to the computer that runs your cell phone, up through the computers that run the Internet) and you will understand how it works. The following is an explanation of how a computer works as told by a mysterious friend of mine called the professor to a young person called Pat.

How Does A Computer Work?

“How does a computer work, professor?” Pat asked one day while visiting me at my shop. “I have had a computer since I was a kid, all my friends have computers and my parents have two of them, but computers seem like magic to me because I do not really understand them.”

I smiled and replied “In a way, Pat, computers *are* magic because part of a computer exists in the physical world, and the other part exists in a non-physical realm called **cyberspace**.”

“Cyberspace?” asked Pat “What is cyberspace and where is it?”

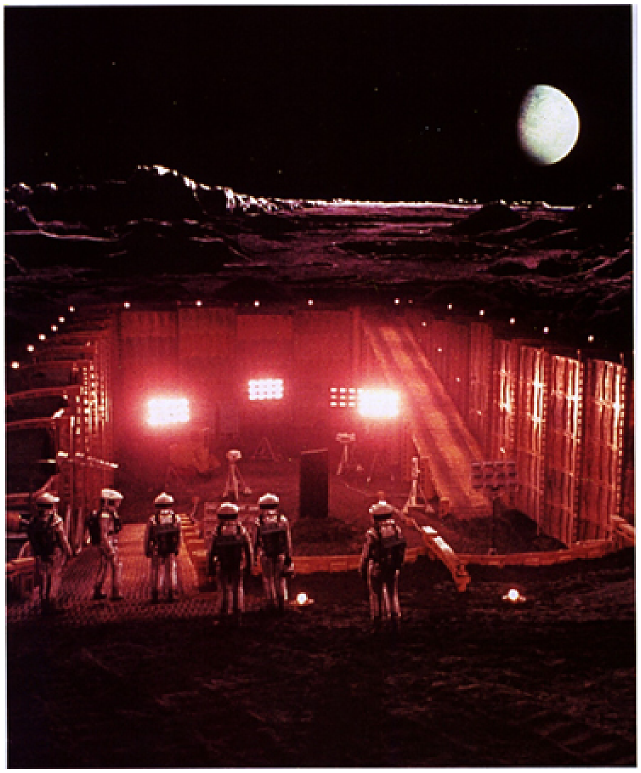
“Where is cyberspace?” I said “Cyberspace is everywhere, and nowhere. Each time you surf the Internet on your computer you enter cyberspace, but you also enter it when you make a telephone call or play a video game. As for what cyberspace is, this would be difficult to explain without first understanding how a computer works.”

“Will you teach me how a computer works,” asked Pat “I really want to know.”

I looked at Pat for a long while before I replied. “I can teach you a bit about computers, Pat, but this explanation would only be a beginning and you will need to continue studying computers on your own if you want to really understand them. A teacher is mainly a guide, and not a substitute for taking responsibility for you own learning. I can open some doors for you, but it will be up to you to walk through those doors to find out where they lead. As long as you understand this, I am willing to spend some time explaining how a computer works to you. Do you understand?”

“I understand” said Pat.

“Pull up a chair then," I said “while I fetch some small whiteboards and a marker.” When I returned, I placed a whiteboard on the table and carefully drew a tall vertical rectangle towards the center of the board. As I drew I slowly whistled three progressively higher notes followed by two quicker and even higher notes followed by a low “BOM bom BOM bom BOM bom BOM bom" I noticed Pat looking at me sideways under raised eyebrows. “Have you ever seen a movie called '**2001 a Space Odyssey'**?," I said. No? Well, many people consider it to be one of the best science fiction movies ever made and in the movie scientists find a tall black monolith that had been buried under the moon's surface by someone, or something...”



(From the movie *2001 A space Odyssey*)

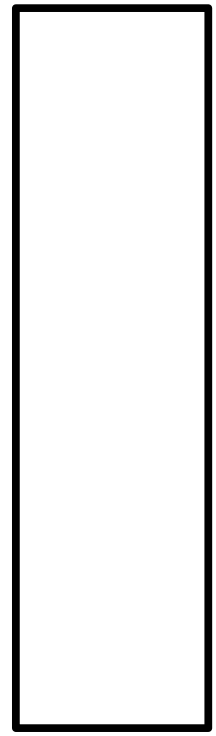


Figure 1

“What's a monolith and who or what buried it there?” Said Pat, wondering where I was going with all of this.

“A monolith is a vertical stone monument or marker," I replied “and in the movie aliens from a distant planet buried a monolith under the moon's surface, waiting for the day when people from earth would be evolved enough to find it. This rectangle I am drawing reminds me of the monolith from the movie because that monolith was also shaped like a tall vertical rectangle.” (see Fig. 1)

“That's eerie” said Pat “What is the monolith from '**2001 a Space Odyssey**' doing in a computer?" I moved my head a little closer to Pat and in a hushed tone said “believe it or not, a number of scientists have said that one of the people who was on the team that invented the first modern computers in the late 1940s, **John Von Neumann**, was actually an alien from Mars...”

“What!?” said Pat “Oh come on!"

“You don't believe me?” I said in a hurt tone.

“No” said Pat “That's ridiculous!"

“I'll make a bet with you.” I said “If I am wrong I will give you a piece of junk electronic equipment from my storage room to take apart but if you are wrong you have to sort all of the resistors in this drawer.” I then pulled a plastic drawer from one of my storage cases, which was filled with a bunch of miscellaneous resistors, and placed it on the table.
Pat studied the tangle of resistors in the drawer for a few moments then said “I don't know what resistors are, but I will sort them if I lose. You will have to show me how though. But there is no way I can lose this one!

I smiled and said “Bring up a browser on my computer, locate a search engine and type the following:

“Von Neumann Martians”

Pat proceeded to do this and included in the search results was a link to a web page that contained the following passage:

> 'The Curve of Binding Energy' by John McPhee (1973, Farrar, Straus and Giroux, pp. 104-105):
>
> "Not all the Los Alamos theories could be tested. Long popular within the Theoretical Division was, for example, a theory that the people of Hungary are Martians. The reasoning went like this: The Martians left their own planet several aeons ago and came to Earth; they landed in what is now Hungary; the tribes of Europe were so primitive and barbarian it was necessary for the Martians to conceal their evolutionary difference or be hacked to pieces. Through the years, the concealment had on the whole been successful, but the Martians had three characteristics too strong to hide: their wanderlust, which found its outlet in the Hungarian gypsy; their language (Hungarian is not related to any of the languages spoken in surrounding countries); and their unearthly intelligence. One had only to look around to see the evidence: Teller, Wigner, Szilard, Von Neumann -- Hungarians all. Wigner had designed the first plutonium-production reactors. Szilard had been among the first to suggest that fission could be used to make a bomb. Von Neumann had developed the digital computer. Teller -- moody, tireless, and given to fits of laughter, bursts of anger -- worked long hours and was impatient with what he felt to be the

excessively slow advancement of Project Panda, as the hydrogen-bomb development was known. ... Teller had a thick Martian accent. He also had a sense of humor that could penetrate bone."

Pat's face slowly turned from skepticism to surprise while reading this passage. When finished, Pat looked at the tall rectangular “monolith” I had drawn on the board with a new sense of awe." I said “Sometime soon I will explain what resistors are and show you how to sort them but for now, lets continue with our discussion.”

I picked up the marker and started drawing evenly-spaced horizontal lines across the rectangle, starting from its bottom and working my way towards the top. “One of the primary things that computers have in them are a bunch of boxes all lined up next to one another. Each box is the same size as all the other boxes and, just like normal boxes, these boxes hold something. But you cannot go into a computer, open the tops of these boxes, turn the computer over and expect things like paper clips or marbles to fall out.” (see Fig. 2)

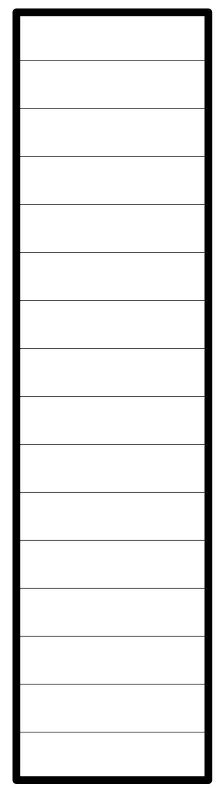


Figure 2

“These boxes are very special. They cannot hold physical objects and yet they can contain anything a human mind can think of! This is a paradox that I will try to explain in a little while but for now, if these boxes do not hold physical objects, can you guess what they *do* contain?”

Pat thought for a little while and then said “I read somewhere that computers are good at something called 'crunching numbers' so I guess these boxes have something to do with numbers.”

I smiled and said “Very good!" Each of these boxes can hold a number and that is all they can hold. There must be something very special about numbers if the main purpose of these boxes is to hold them. The way that a computer uses numbers is one of the main sources of its incredible power and it seems fitting that John Von Neumann, one of the greatest mathematicians of all time, had a hand in placing them there.”

“The boxes in most computers can each hold a number between **0 and**

255," I said as I started writing numbers between 0 and 255 in the boxes “and while the computer is running, there is never a time that a box does not have a number in it. Another name for a number between 0 and 255 is a **byte**. (see Fig. 3) If a number larger than 255 needs to be worked with, it is spread across two or more boxes. These boxes are called **memory locations** and this vertical rectangle is called a **memory map** because it shows where the memory locations in a computer are located in relation to each other. Some computers have a small amount of memory locations and some computers have an enormous amount.”

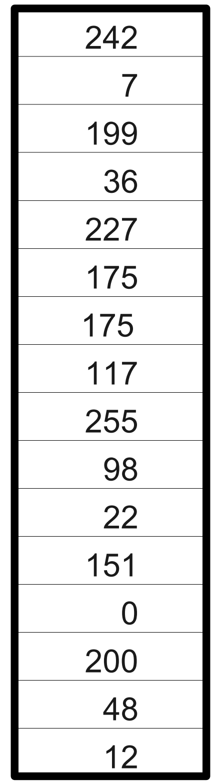


Figure 3

Pat studied the memory map I had drawn then said “How many memory locations are there in this computer?” while pointing to the computer under my desk.

"How many do you think there are?" I asked.

“Hmmm” said Pat while thinking for a few moments. “A hundred?"

“More...”

“A thousand?”

“More...”

“A million !?”

I smiled and said “More!"

“A billion !!?”

“Yes!” I said “This computer has around a billion memory locations, each holding 1 byte, and some computers have significantly more than this! The metric prefix for a billion is **giga** and so this computer has a **gigabyte** of storage in is memory map. If it took 1 second to count each of these memory locations it would take you over 30 years to count them all!"

“A billion memory locations!” cried Pat “Thats a lot of numbers. How does a computer keep track of which numbers are in which memory locations?”

“That is an excellent question." I said. " One certainly could not give them their own names, like Bill or Lisa or Tom, because one would run out of names long before running out of memory locations. Even the early computers had too many memory locations to give each location its own name and therefore the inventors of the modern computer had to solve this problem right from the start. How do you think they did it? Perhaps if you think of some examples in the physical world that have a similar problem, a lot of items that need to be uniquely identified, that may help.”

Pat looked out of the window for a while, trying to think of something in the physical world that was similar to the memory locations. The professor lived on a very tall, wooded hill and from it one could see great distances. On a road on a distant hill, a mail truck was delivering mail and Pat watched the carrier place letters into one mail box after another.”

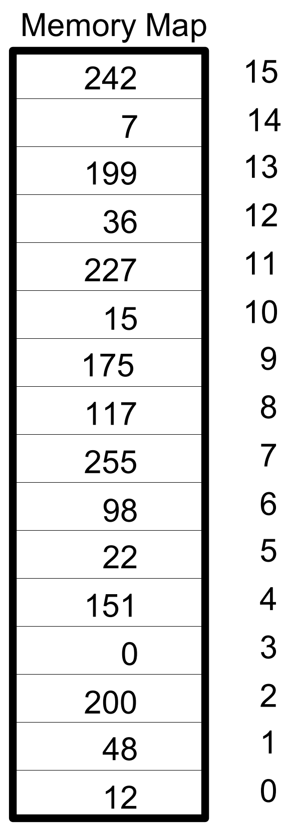


Figure 4

“I got it!” cried Pat. “Those memory locations are similar to house addresses! All of the houses on the street on that hill have their own address, and the houses on my street are the same way. Did the inventors of the computer give each memory location its own address?"

“Yes!” I said “You figured it out! Each memory location has its own unique address and all computers give the first memory location an address of 0, the next memory locations receives an address of 1 and so on all the way to the top of the memory map.” As I said this I started placing an address next to each of the memory locations starting at the bottom of the memory map and working up. At the top of the rectangle I wrote the words 'Memory Map'." (see Fig. 4)

“One way to think of a memory map is that it is a very long street with thousands and thousands of houses on it, each 'house' or memory location can hold a number between 0 and 255 and each house has its own address.”

Pat thought about the mail carrier then said “Physical houses have mail carriers who deliver mail to them and retrieve mail from them. If memory locations are like houses, what 'delivers' and picks up the numbers from the memory locations?"

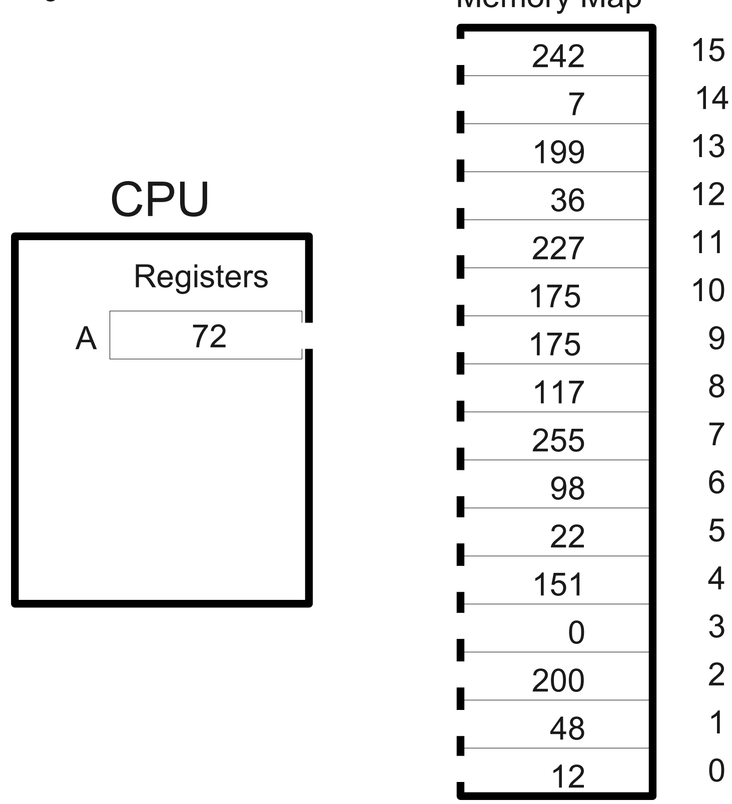
“That is another good question!” I said. “You can think of a computer as a strange kind of world with one long street on it that only has one mail carrier. Instead of letters and packages, this mail carrier can only deliver and retrieve numbers. Another remarkable thing is that this mail carrier has one rubber arm that is able to stretch for very long distances. Instead of walking from house to house, the mail carrier sits in the post office, which is placed off to one side of the street, and stretches the rubber arm to each house.”

As I described this, I drew a square off to the left side of the memory map to represent the post office and then I erased an opening in the left side of each memory location. “To show that the mail carrier can access all the houses with that rubber arm,” I said “I am placing an opening on the left side of each of the memory locations, the side that faces towards the post office. In a computer, its 'post office' is called a **CPU** which stands for **Central Processing Unit** and another name for it is **microprocessor**.”

“The CPU also has a small number of memory locations in it, some of which are the same size as the memory locations that are in the memory map. These CPU memory locations can also hold a number between 0 and 255 but, instead of being given a unique address number, the memory locations that are inside of a CPU are usually labeled with one or more letters. To distinguish them from the memory locations that are in the memory map, these memory locations are called **registers** and our example computer has a register which I am going to label A.”

As I said this I drew a register in the CPU, labeled it 'A' and created an opening on its right side to show that the carrier had access to its contents. Finally, I placed a number between 0 and 255 into the register saying "Registers, like memory locations, must always contain a number in them while the computer is running." (see Fig. 5)

Figure 5



The diagram was starting to take shape. Pat studied it with great interest then asked "If the memory locations and registers can never be empty, how can the carrier remove numbers from them?"

"I was wondering if you would notice that." I said. "In a computer, the numbers do not actually move. The mail carrier is able to reach into any memory location and **copy** the number that is there into a register, or copy a number from a register to a memory location, but the original number is never moved. When a number is copied to a register or memory location, however, the number that was already there is overwritten.""

Pat looked up from the whiteboard to the PC's computer monitor and said "Can we see some of the numbers that are in the memory locations in this computer?".

"We could," I said "but I have a better idea. When I was a kid, the first computer I had was a Commodore 64 and it was a wonderful machine for learning about computers. I still have it and, if you would like, I will get it from the storage room, set it up and we can play with it. What do you think?"

"Sure!" said Pat "I'd love to see what an old computer looks like!."

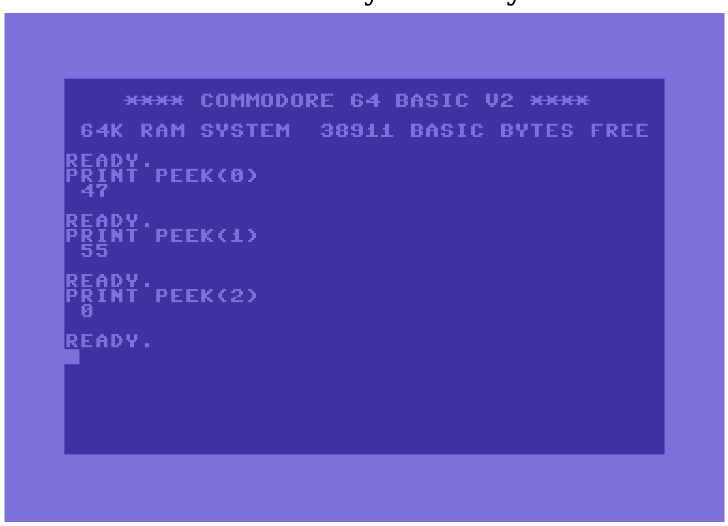
I retrieved the Commodore 64 (http://en.wikipedia.org/wiki/Commodore_64) from my storage room, plugged it into a television and powered it up. Within 5 seconds the following friendly blue screen appeared.

Pat said “Hey, that came up fast! Our PC at home takes much longer to come up.” After reading the screen for a little bit, Pat asked “What is BASIC?”

“BASIC," I replied “is a typed language that a computer programmer uses to tell a computer what to do in a step-by-step manner. It consists of a set of commands along with rules for how to use them. For example, if I type 'PRINT “HELLO”' and then press the <Enter> key, BASIC understands that I want it to print the word HELLO on the screen. BASIC can also act like a calculator. If I type 'PRINT 2+3', BASIC will add the numbers 2 and 3 together and give the result 5”

Pat experimented by typing in a few more simple math operations then asked “Can we tell BASIC to show us the numbers that are in the computer's memory locations?”

“Sure," I said “the command that BASIC uses to peek into a memory location is PEEK(<address>) and we can use it together with the PRINT statement to print the contents of any memory location to the screen.”



I had BASIC show us the contents of memory locations 0, 1 and 2 which contained the numbers 47, 55 and 0 respectively. “Notice that these three numbers are between 0 and 255. We could continue typing in PRINT PEEK() statements to check the contents of higher memory locations, but BASIC can also do this automatically if we write a program that tells it to do this.” I then typed in a short program and had BASIC run it.

"What I just typed on the screen is called a BASIC program." I told Pat "A **program** consists of instructions that tells a computer exactly what to do step-by-step and this specific program tells the computer to peek into memory locations 0 through 200 and print the number that it finds in each location to the screen. Again, notice that there is no memory location that has a number that is less than 0 in it and none that have a number greater than 255."

"At this point, I am not going to explain how BASIC works," I said "but someday I will help you to learn to program in BASIC if you would like. We will, however, be discussing more about what a computer program is in a little while."

What Do The Numbers In Memory Locations Mean?

Pat looked at all the numbers on the screen that were obtained from the Commodore 64's memory map and then asked "What do all of these numbers mean?"

"That," I told Pat "is one of the great secrets behind the power of a computer! Remember when I told you that a computer's memory locations can contain anything a human mind can think of?"

"I remember" replied Pat.

"Well, the way it does this," I said "is by having the number that is in any given memory location represent an idea that is in a human's mind. For example, lets say that we write a program that works with apples. In our program, we are going to have the number 1 represent red apples and the number 2 represent green apples. We will use memory location 5 to hold the type of apple we are currently working with. If I place a 1 into memory location 5, what kind of an apple is it now holding?”

Pat thought for a moment and then said “A red apple.”

“Correct," I said “and if we placed a number 2 into memory location 5, that memory location would then 'contain' a green apple. Of course, we can not place a physical red apple into memory location 5, but by having a number (in this case the number 1) **represent** a red apple, we can place a reference to the **idea** of a red apple into that memory location. Any idea you can think of, no matter what it is, we can associate a number with that idea and thereby enable a computer to work with it. Go ahead, come up with an idea Pat.”

Pat thought for a little bit then said “Boat”.

I said “We can associate the number 47 with the idea of a boat. Come up with another idea.”

Pat said “Cat”.

“234.” I said “See, no matter what idea you think of, I can think of a number to represent it!”

Contextual Meaning

After this explanation, Pat's eyes lit up and one could almost see wheels and gears turning behind them. “That's amazing!” cried Pat “I never would have guessed that a computer works like this!” After thinking a while longer, though, Pat asked “But if a memory location can only hold a number between 0 and 255, how can it possibly be capable of representing all of the millions of ideas that a human can have?”

“That is a wonderful question Pat," I said “and the answer is a concept called **contextual meaning**"

“Contextual what?” Asked Pat.

“Contextual meaning.” I said “I will give you an example that will help explain what it is.” I stood up, walked out of the room, waited a few moments and then walked back in and said “Give me five” in a very calm voice.

“Give you 5 what?” asked Pat, with a look of confusion.

“Can you think of some things I could mean by that statement?” I said.

“Well,” said Pat after a few moments “if we were in the store buying candy and you had just asked the clerk behind the counter for some chocolate bars, the clerk might ask you 'how many do you want?' and you could say 'Give me five'"

“That is a good example," I said “can you think of another?”

“Hmmmm” said Pat “you could be asking a friend to loan you some money and when the friend asks you how many dollars you need, you could say 'Give me five'"

“Good,” I said “now give me one more.”

Pat thought for a while, smiled, stuck out a hand palm-up and said “Give me five!” I smiled in return and slapped the upturned hand.”

“Okay Pat," I said “in each of those three examples the same phrase 'Give me 5' was used. How did the people in each example know what was meant when the phrase was said?”

Pat pondered this question then responded “the meaning of 'Give me five' **depended on what the people were doing**. In the first example, some candy bars were being purchased and in the second example, money was being borrowed from a friend.”

“What about the third example?” I said “We were not doing anything special when you said 'Give me five' and yet I knew exactly what you meant.”

“But I didn't just say 'Give me five' in a calm voice like you did, I said 'Give me 5!' in a loud voice and put my hand out. Everyone knows that when a person says 'Give me five' in a loud voice and puts their hand out, that they want you to slap it.”

“Yes," I said “everyone knows this because by saying 'Give me five!' in a loud voice, and putting your hand out, you provided what is called a **context** for the phrase 'Give me five!'” **Context** means the circumstances within which an event happens or the environment within which something is placed. In the first example, the purchasing of the candy bars provided the context for 'Give me five' and in the second example the borrowing of some money provided the context. **Contextual meaning**, therefore, is the meaning that a context gives to the events or things that are placed within it.”

“I had never looked at things this way before,” said Pat “but now that I think about it, contextual meaning seems like it is used all the time.”

“Yes," I said “most people use contextual meaning every day, but they are not aware of it. Contextual meaning is a very powerful concept and it is what enables a computer's memory locations to reference any idea that a human can think of. Each memory location can only hold a number between 0 and 255, but a human can have those numbers mean anything they wish. Larger numbers than 255 can also be spread across more than one memory location.”

What Provides The Context For The Numbers In A Computer's Memory?

“I am beginning to understand contextual meaning” said Pat “but what provides the context for the numbers that are in a computer's memory locations?”

“When a program is loaded into a computer's memory locations,” I replied “it is the **program** that provides the **context**. The **person** who creates most of this context is the **programmer** who wrote the program. When a programmer creates a program, the ideas that are in a programmer's mind become linked to the numbers that represent the information that the program works with. Each time the program is loaded into the computer's memory, the program's numbers are loaded along with the ideas that are linked to these numbers.”

Pat looked at the numbers on the Commodore 64's screen a while longer and then went back to studying the model of a computer that I was drawing on the whiteboard. Pat then said “The CPU can copy a number from a memory location to a register and it can copy a number from a register to a memory location. How does it know what numbers to copy where and what does it do with the numbers other than copy them?”

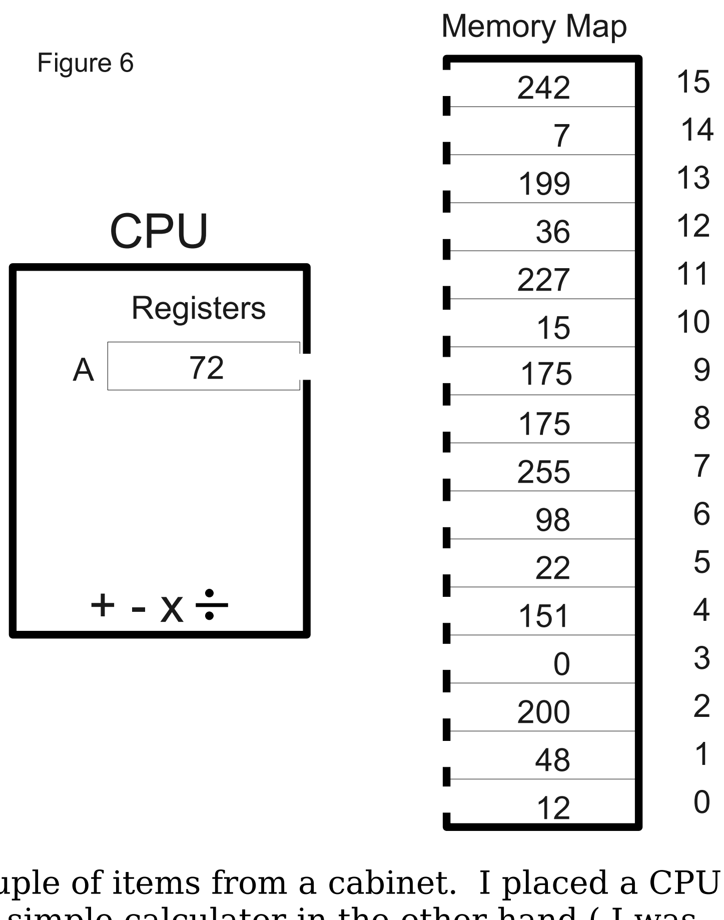
CPUs: Calculators Without Buttons

I thought about this question for a few moments then said “Lets start with the second part of your question first. Many people who do not know very much about computers think of a CPU as a kind of brain. In one way they are correct because it is the main place in a computer where operations can be performed on numbers. But the **only operations on numbers that most CPUs can perform are to add, subtract, multiply and divide them**. It can also compare the size of two numbers but most CPUs can not do too much more beyond these operations. In truth, **a CPU is one of the dumbest things in the world**. In fact **it is so dumb that it has to be told exactly what to do thousands of times a second**.”

“It sounds to me like a CPU is not much more than a calculator” said Pat.

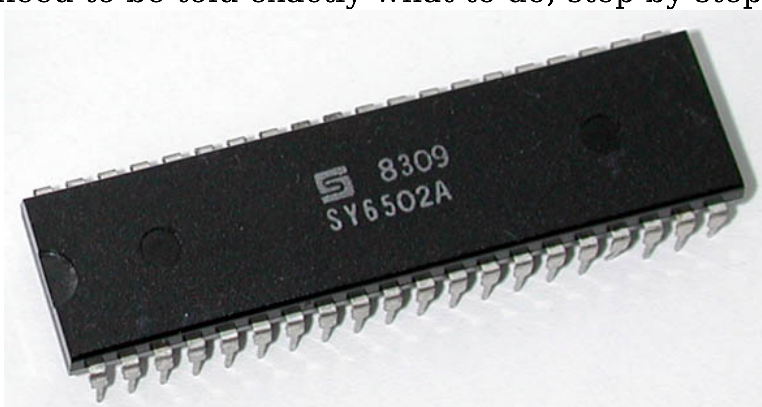
“That is an excellent observation Pat," I said “**a CPU is not much more than a simple calculator, the kind that can only add, subtract, multiply and divide**.” I then drew the symbols for addition, subtraction, multiplication and division in the CPU box on the whiteboard. (see Fig. 6)

Figure 6



“There is a significant difference between a CPU and a calculator though. Put out both of your hands palm up Pat.” I said while I fetched a couple of items from a cabinet. I placed a CPU in Pat's left hand and I placed a simple calculator in the other hand (I was careful to lightly touch Pat's left hand with my pinky finger before placing the CPU there).

“The CPU in your left hand is similar to the one that is in the Commodore 64 and it was widely used in the personal computers of the late 1970s and 1980s. Its capabilities are similar to that of the calculator in your other hand in that they both can do simple mathematical operations on numbers and they both need to be told exactly what to do, step by step, by a human.”



“Told what to do?” said Pat “I can't tell a calculator to do something, it doesn't have any ears!”

I laughed “You are right, you do not actually tell most calculators what to do, not quite yet anyway, but you do indicate to it what you want it to do. How do you do this?”

Pat looked at the calculator then said “You 'tell' it what you want it to do by pressing its buttons.”

“Exactly!” I said “Go ahead and 'tell' the calculator that you want it to add the numbers 10 and 5 together and to give you their sum.”

Pat typed '10 + 5 =' on the calculator.

“What answer did you receive?” I asked.

“15” replied Pat.

“Okay, now look at the calculator and tell me what it is doing.” I said.

Perhaps 10 seconds went by then Pat said “The calculator is not doing anything. What are we waiting for?”

“Are you sure it is not doing anything?” I said.

“No, nothing,” said Pat “am I missing something?”

“It does not look like the calculator is doing anything," I replied “but it is actually waiting for you to tell it what to do next. Most calculators will wait for instructions from a human for a few minutes and, if an instruction is not received during this time, they will turn off in order to conserve battery power. When a human turns a calculator on, it will quickly enter a mode where it is waiting for instructions again.”

“Now, 'tell' the CPU in your other hand to add 10 + 5.” I said.

Pat looked at the CPU, turned it upside down then said “I can't because there aren't any buttons.”

“No, there are not any buttons on a CPU," I replied “so how does a human tell a CPU what to do?”

“I don't know,” said Pat “and I can't even come up with a guess.”

“Lets go back to the model of a computer that we have been drawing on the whiteboard. The CPU is sitting off to the side of the memory map and it is able to copy numbers from the memory map into its registers and from its registers to the memory map. It does not have any buttons on it so a human cannot tell it what to do this way. What would happen, though, if we were to use the concept of contextual meaning to associate the equivalent of button presses with certain numbers and then placed these numbers into the memory map. Could the CPU access these numbers?”

“Yes, it could!” said Pat “Instead of physical buttons, numbers that represented buttons could be placed into memory and this would be just as good.”

I continued “Lets proceed by putting together a sequence of numbers representing button presses, or **instructions**, that will tell the CPU to add the numbers 10 and 5 together. The first thing we are going to need is an instruction that copies a number from the memory map to a CPU register, specifically register 'A'. Hmmm, we have to pick a number between 0 and 255 to represent this instruction, how about the number 169?”

“That sounds as good as any number to me.” replied Pat.

I wrote the number 169 in memory location 0 on the whiteboard model then said “In order to make it easy for the 169 '**load register A**' instruction to find the number it is suppose to load, we will have it always copy the number that is one memory location higher in memory than the instruction itself.” I then wrote a number 10 into memory location 1.

The Program Counter And The Instruction Register

“Now we have a couple more problems to solve before we can proceed. The CPU is going to need to know where in memory to find the current instruction and it is going to have to have a place to copy it to in the CPU before it can use it. The way that most CPUs solve the first problem is with a special register called a **Program Counter** or **Instruction Pointer**. The Program Counter holds the memory address of the current instruction.” I drew a register box underneath the A register and labeled it 'PC'. I then wrote the address '0' in this register and drew an arm with a hand on the end of it from the right side of the Program Counter to memory location 0.

The second problem is solved with another register called the **Instruction Register** and it is the register that the number that represents the current instruction is copied to inside the CPU.” I drew another box in the CPU underneath the program counter register and labeled it IR. The last thing I did was to place X's in all of the memory locations that we were not focusing on at the moment.

Running A Machine Language Program

“Now that we have written the first part of our small program, and placed the extra registers in the CPU needed to run it, should we go ahead and run it to see what it does?”

“Yes!” said Pat “This is fun!”

"It is fun, isn't it?” I said “I am going to place a small button next to the CPU, label it 'Run One Instruction' and when it is pressed, the CPU will run the instruction that the PC register is pointing to.” I drew a small pushbutton switch next to the CPU then said “Ready?” (see Fig. 7)

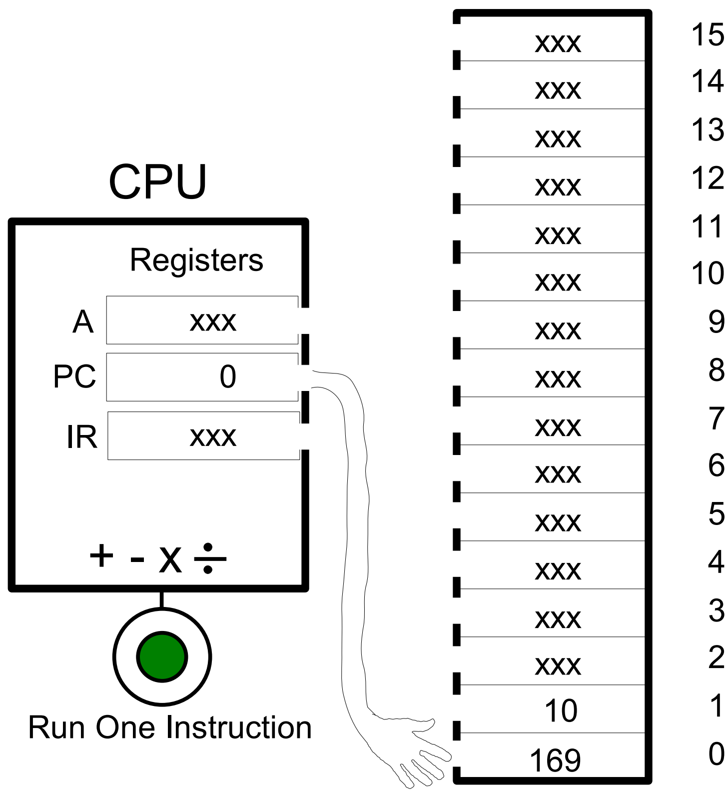


Figure 7

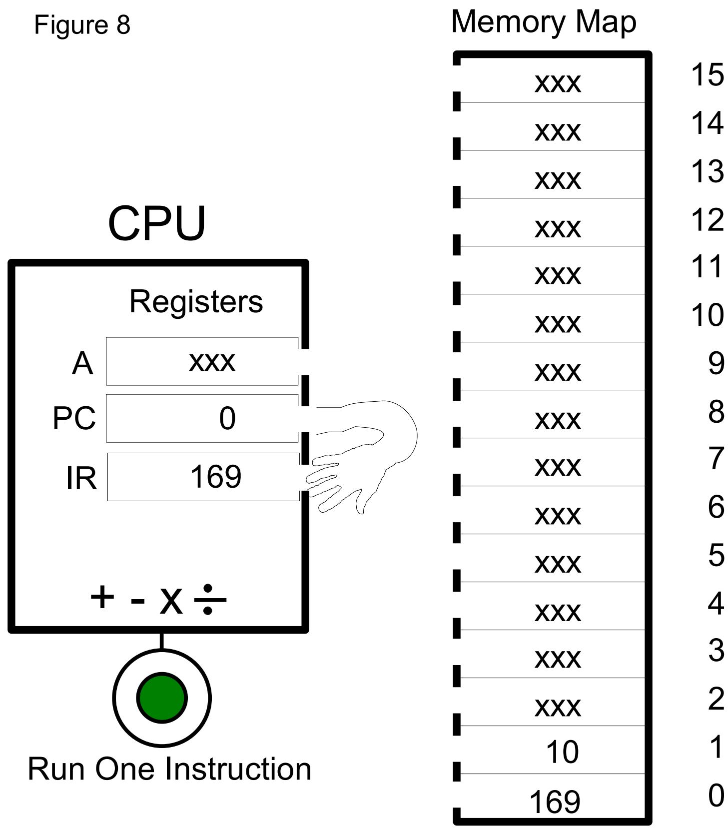
“Ready!” Pat replied.

“Okay," I said “lets go!”

I pushed the run button then said “The first thing that the CPU does when we tell it to execute the next instruction is to look at the Program Counter in order to determine where in memory the instruction is located. In this case the Program Counter has the number 0 in it so the CPU, which is like the mail carrier with the long rubber arm, goes to memory location 0, finds the number 169 that is located there, and copies it into the Instruction Register.” As I say this I write the number 169 into the Instruction Register box in the CPU.

"The number 169, which is now in the Instruction Register, represents an instruction or **operation** that the CPU must perform. In this case, the number 169 instruction means 'go to the next memory location immediately after the one that held this instruction and copy the number that is there into the A register'." (see Fig. 8)

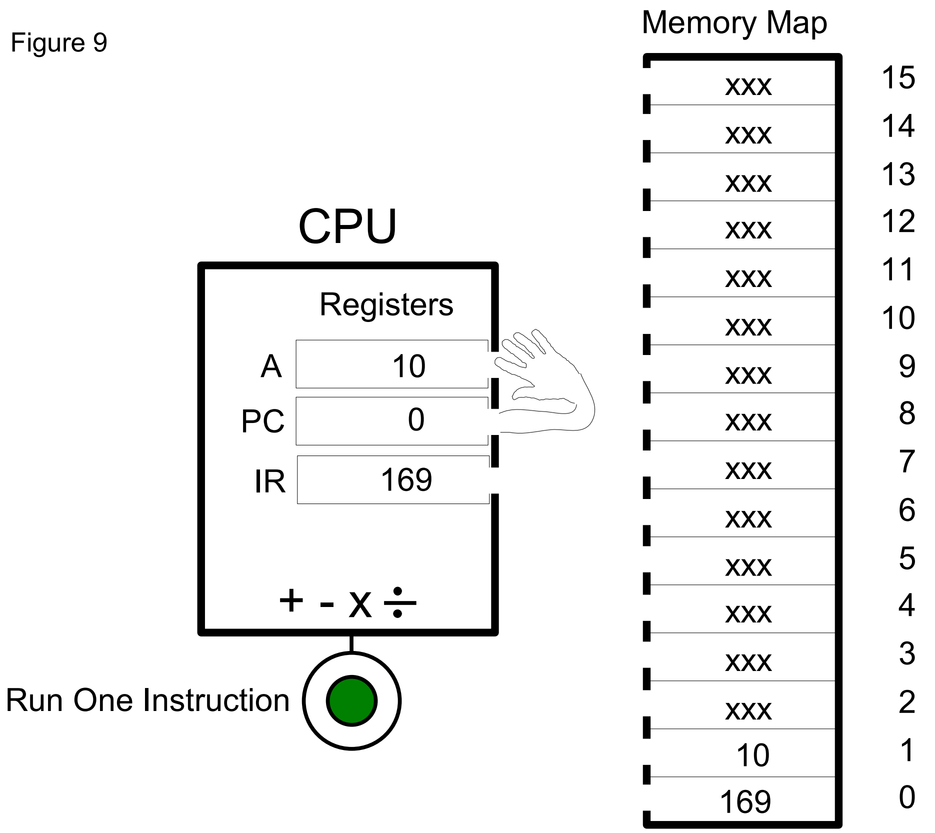
Figure 8



In order for the CPU to determine what a given instruction should do, it must have the equivalent of a map or table that associates the instruction's number with the actions that should be performed when this instruction is run." I said. "Below the CPU I drew a rectangle and labeled it Instruction Table. Towards the top of this rectangle I wrote the number 169 followed by the sentence **'Loads the number in the memory location immediately following the instruction's memory location into register 'A''**.

"In this case the CPU looks at the number 169 which is in the Instruction Register," I said "matches this number in the Instruction Table and then performs the operation that has been associated with this number. The contents of the next memory location after the one that holds the instruction is then copied to register 'A'." I erased the old value that was in register 'A' and replaced it with the number 10. (see Fig. 9)

Figure 9



Instruction Table

169	Loads the number in the memory location immediately following the instruction's memory location into register A.

“We have just successfully run, or **executed**, our first instruction," I said “and now the number 10 is in register 'A' waiting to be added to the number 5. The last thing we need to do is to update the Program Counter register to point to the address of the memory location that will hold the next instruction.” I then erased the old value that was in the Program Counter and replaced it with the number 2. I also made the program counter point to memory location 2.

Pat said “It seems that the next instruction we need is one that tells the CPU to add 2 numbers together.”

I smiled and said “I agree, lets come up with another number between 0 and 255, say 105, and this will represent an addition instruction.” I then wrote the number 105 in the next row of the Instruction Table and also wrote it in memory location 2 in the memory map. “How do you think this addition operation should work?”

“Well” said Pat “we can have this instruction assume that the first number to be added is already in register 'A', and the second number can be placed immediately after the address of the addition instruction in memory, just like with the load instruction.” Pat pointed to memory location 3 and said “Place the number 5 into memory location 3, right after the 105 that represents the addition instruction.”

I said “I like that idea” and I wrote a 5 in memory location 3. “After the addition instruction adds the 10 and the 5 together, where should it place the answer?”

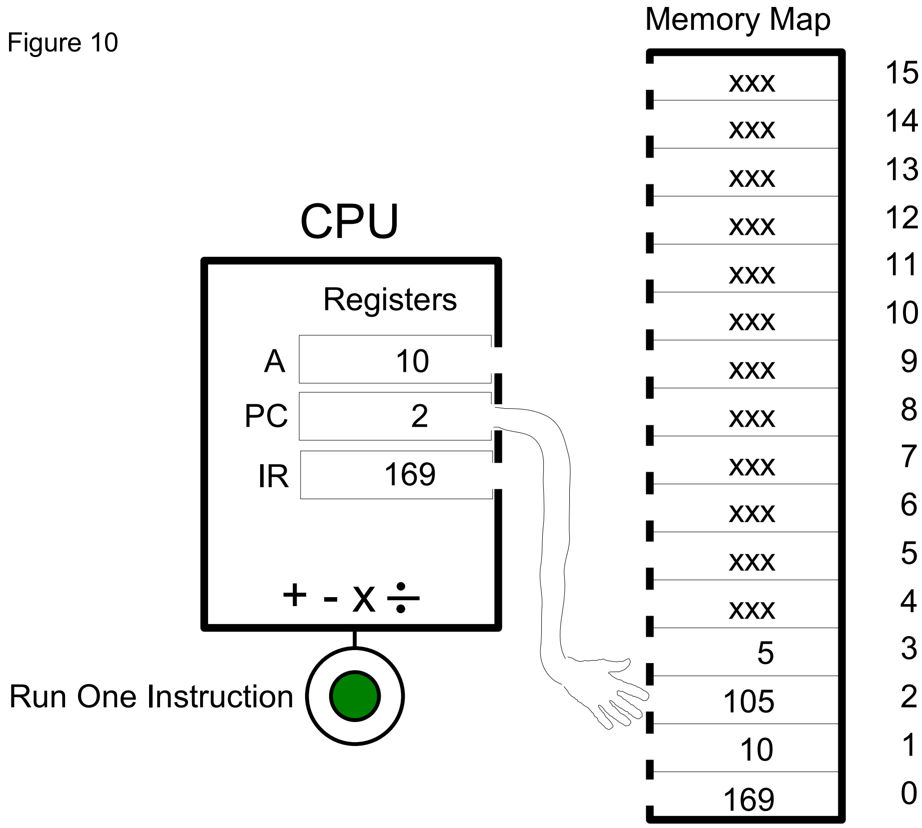
“Hmmm” said Pat “that is a good question. I am not sure where the answer should go.”

“What we could do is to place the answer back into register 'A' since we do not need the number that is there any more. What do you think?”

“That sounds okay.” replied Pat “The operation description that is placed next to the 105 in the Instruction Table can say something like **'Adds the number that is in register 'A' with the number in the memory location immediately following the instruction's memory location. The answer is placed into register 'A''**"

“Very good!” I said and I wrote this operation description next to the number 105 in the Instruction Table. “By the way, another name for a register that is able to have numbers added with it is an **accumulator** so we can refer to register 'A' as **accumulator A** if we would like. Also, numbers like 169 and 105 that represent CPU instructions or operations are called **operation codes** or **opcodes**.” I then wrote the word 'Opcode' at the top of the column that contained the instruction numbers and above the descriptions column I wrote 'Operation Description'" (see Fig. 10)

Figure 10



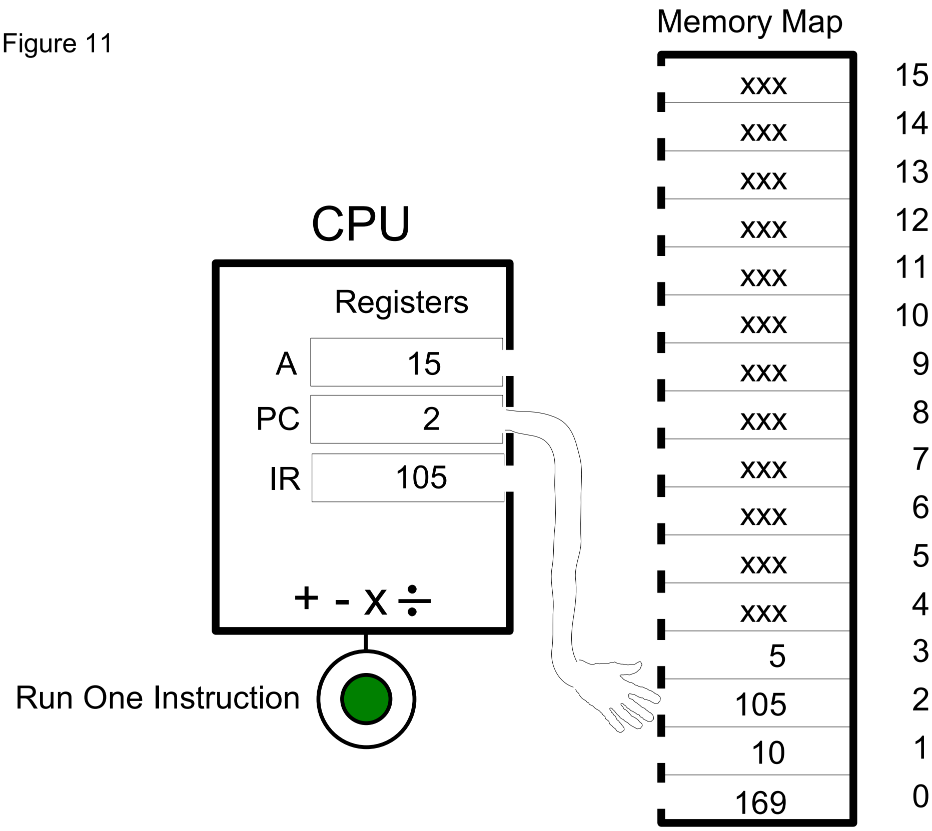
Instruction Table

Opcode	Operation Description
169	Loads the number in the memory location immediately following the instruction's memory location into register A.
105	Adds the number that is in register A with the number in the memory location immediately following the instruction's memory location. The answer is placed into register A.

After this was done I said "Press the run button and we will walk through executing the next instruction."

Pat pressed the imaginary run button on the whiteboard and I proceeded. "The CPU looks at the Program Counter and sees that the next instruction that it should execute is in memory location 2 so it copies the number that is in that memory location, which is 105, into the Instruction Register." I then erased the 169 that was in the Instruction Register and replaced it with 105. "The CPU then looks at the 105 that is in the Instruction Register, matches it with 105 that is in the Instruction Table and performs that operation that is associated with this opcode. The CPU then adds the 5 which is in memory location 3 with the 10 that is in register 'A' and then the answer 15 is placed into register 'A'. The 10 that was already in register 'A' is overwritten." I then erased the 10 that was in register 'A' and replaced it with a 15. (see Fig. 11)

Figure 11



Instruction Table

Opcode	Operation Description
169	Loads the number in the memory location immediately following the instruction's memory location into register A.
105	Adds the number that is in register A with the number in the memory location immediately following the instruction's memory location. The answer is placed into register A.

"Finally," I said "we need to update the Program Counter so that it contains the address of the opcode of the next instruction to execute, which will be address 4." And I did this.

"What we need now," I said "is a third instruction that copies the number that is in register 'A' to a memory location so that we can use register 'A' to do other work. Since we used a **load register 'A'** instruction to copy a number from a memory location to register 'A', how about a **store register 'A'** instruction to copy a number from register 'A' to a memory location? We can give it an opcode of, say, 141.” I then started a new row in the Instruction Table and wrote a 141 in the opcode column.

Mnemonics

“That sounds good to me," Pat said “but if we come up with too many more instructions, I am going to start forgetting which opcodes represent which operations.”

“That is a problem that the first computer programmers had too and the way they solved it was with **mnemonics**.” I said.

“Neh-moniks,” said Pat “What's that?”

“Mnemonics," I replied “are aids that help people remember things that are difficult to remember. One example is the color bands that are on the resistors you are going to sort tomorrow.” I said with a smile. “Each color represents a different number between 0 and 9 and the colors are **B**lack, **B**rown, **R**ed, **O**range, **Y**ellow, **G**reen, **B**lue, **V**iolet, **G**rey and **W**hite. These colors can be remembered with the phrase **B**lack **B**eetles **R**unning **O**n **Y**our **G**rass **B**ring **V**ery **G**ood **W**eather.”

“A different type of mnemonic is the one that mechanics use to remember which way nuts and bolts tighten and loosen. 'Righty tighty, lefty loosey' means that a nut or bolt should be turned to the right (or clockwise) to tighten it and to the left (or counter clockwise) to loosen it.”

“For our CPU instructions, we might use **LDA** to represent the **load register 'A'** instruction, **ADC** to represent the **add to register 'A'** instruction and **STA** to represent the **store register 'A'** instruction.” As I said each mnemonic I wrote it to the left of its opcode in the Instruction Table and, when I was done, I wrote the word 'Mnemonic' at the top of the new column.

“Now we need to figure out how the STA instruction is going to work. We know that the number we want to copy to memory is already in register 'A', but how is the instruction going to know which memory location to copy this number into?”

Pat thought about this problem for a while then said “Since the LDA and

ADC instructions both needed to use the numbers that were just after them in memory, could we have the STA instruction also look at the number in the memory location that is just after it in memory to determine where to copy the contents of register 'A' to? The memory location immediately after the location that holds the STA instruction can contain the destination address that it needs"

I blinked and then stared at Pat for a few moments. “Uhh, yes Pat, that is a very good idea," I finally said “in fact, most CPUs use the technique you just described in their store instructions. Are you sure you have never studied computers before?”

“No” said Pat “I have used them, but I have never studied how they work. They certainly work a lot differently than I would have expected.”

I replied “I agree, computers work very differently than most people would expect. When I first learned about how a computer works, I was very surprised and also amazed that humans were capable of developing such a wonderful design. In fact, I am still amazed!”

After a few moments I said “Lets finish the STA instruction. I am going to write your description of how the STA instruction works in the Instruction Table” which I did. I then asked Pat “which memory location should we tell the STA instruction to copy the number in register 'A' to?”

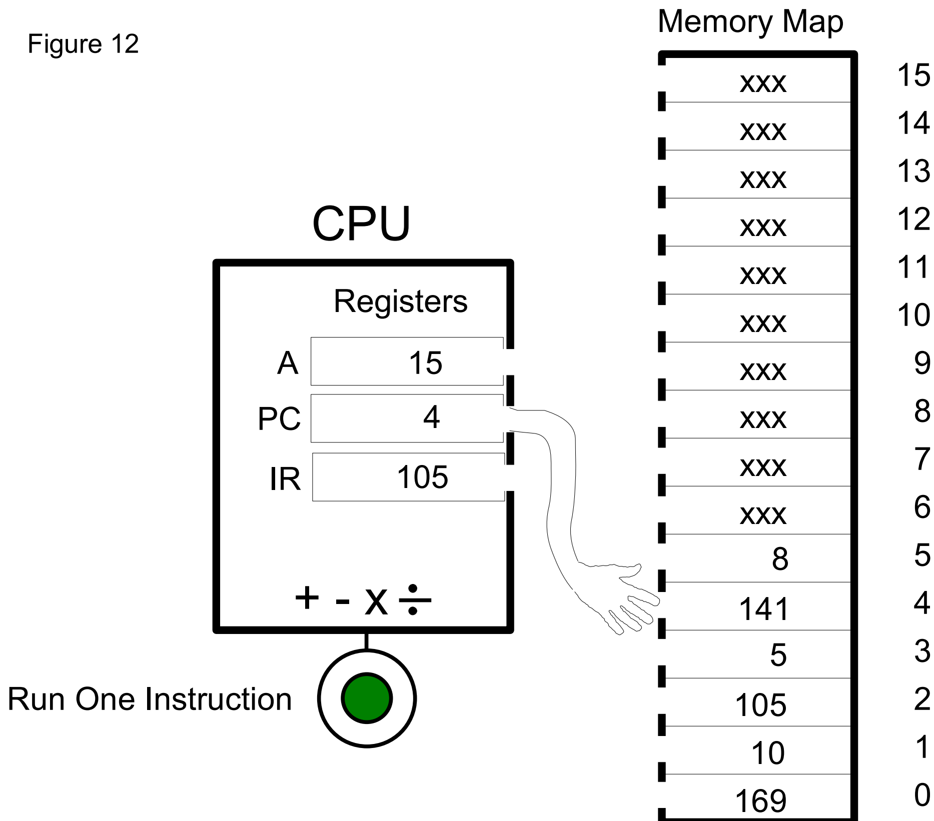
Pat looked at the memory map and said “how about putting it into memory location 8?”

“Okay," I replied “we will place the STA instruction's opcode, which is 141, into memory location 4 and place the address that it should write to, which is 8, into memory location 5.” (see Fig. 12)

“Would you like to run this last instruction Pat?” I said.

“Sure” said Pat who then reached out a hand and pressed the run button on the whiteboard. “The first thing that the CPU does is to look at the Program Counter to see what the address is of the next instruction to execute. Our Program Counter contains the address 4 so it goes to memory location 4 and copies the number it finds there to the Instruction Register. The number that is now in the Instruction Register is 141 and the CPU matches this number with the one in the Instruction Table to determine what operation it needs to do. The operation description for the STA instruction

Figure 12



Instruction Table

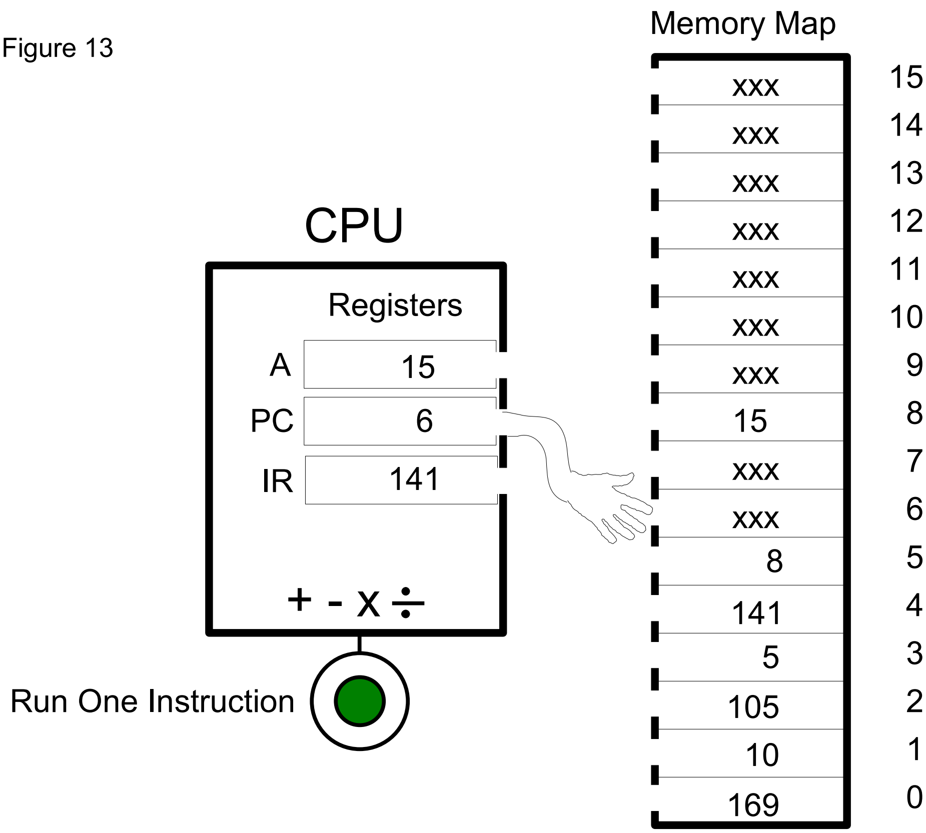
Mnemonic	Opcode	Operation Description
LDA	169	Loads the number in the memory location immediately following the instruction's memory location into register A.
ADC	105	Adds the number that is in register A with the number in the memory location immediately following the instruction's memory location. The answer is placed into register A.
STA	141	Stores the number in register A into a memory location. The address of the memory location is represented by the number that is in the memory location just after the instruction.

tells the CPU to get the address of where it is going to store to from the next memory location after the instruction itself. The CPU looks in this location, which is location 5, and finds an 8 there. Finally, the CPU copies the number which is currently in register 'A', which is 15 (our answer) to memory location 8.” Pat then picked up the marker and wrote a 15 in memory location 8.

“Very good Pat," I said “but you need to do one more thing before the instruction is finished.”

Pat looked at the whiteboard for a few moments and then said “Oops, I forgot to update the Program Counter!" Pat then erased the number that was in the program counter and wrote a 6 there. (see Fig. 13)

Figure 13



Instruction Table

Mnemonic	Opcode	Operation Description
LDA	169	Loads the number in the memory location immediately following the instruction's memory location into register A.
ADC	105	Adds the number that is in register A with the number in the memory location immediately following the instruction's memory location. The answer is placed into register A.
STA	141	Stores the number in register A into a memory location. The address of the memory location is represented by the number that is in the memory location just after the instruction.

I smiled and said "We have successfully completed a small program, what do you think?"

Pat said "I am still fuzzy about a number of things, but I am really enjoying this so far!"

"I am glad you are enjoying this information, Pat. There are thousands of careers in the world that have this information at their core and, if you continue to study computers, perhaps you will work with them some day. What are some of the things you are fuzzy about?”

Machine Language

Pat pointed at the Commodore 64's screen and asked “If a CPU is programmed with opcodes, how can it also be programmed in BASIC?”

“To answer that question perhaps it would be best if we went back to the early days of computers. When the first modern computers were created in the late 1940s and early 1950s, the only way they could be programmed was using opcodes. The programmers back then would create a program by drawing a memory map, like we did on the whiteboard, and then write CPU opcodes, and needed data, into the memory locations. They would then enter the series of numbers they had written into the physical computer's memory using switches and buttons. Programs that are written directly with a CPU's opcodes are called **machine language** programs.”

Assembly Language

“The early programmers soon found out, however, that remembering what all of the opcodes did was difficult and that is when they created mnemonics for each instruction. After this, they discovered that developing programs using the mnemonics was much easier than using the CPU's opcodes and so programming evolved from using opcodes to using mnemonics. After the mnemonic version of a program was developed, the programmer would then use documentation, similar to our Instruction Table, to look up what opcode went with each mnemonic and they would then write these opcodes next to each mnemonic in their program. The mnemonic equivalent of the small program we made would look like this:

Address	Opcode	Operand	Mnemonic & Operand
000	169	010	LDA #010
002	105	005	ADC #005
004	141	008	STA 008

“The Address column holds the beginning address of each opcode, the Opcode column holds the opcode and the Operand column contains the data an opcode may need. The Mnemonic & Operand column contains the mnemonic version of the program which the programmer writes first and then fills in the appropriate machine language numbers in the left three columns. The number sign next to the numbers 10 and 5 means that these numbers are placed in memory immediately after the instruction's opcode.”

The 6502 CPU's Instruction Set

Pat looked at the mnemonic version of the small program we had written then asked “How many instructions does the CPU in the Commodore 64 have?”

“The 6510 CPU that is in the Commodore 64 is based on the 6502 CPU and they both have 56 instructions.” I replied “That may seem like a large number of instructions, but most of them are as simple as the LDA, ADC and STA instructions we have been working with. Lets do an Internet search and find the complete list of instructions that the CPU in the Commodore 64 uses.” I did this and found the following list:

```
ADC  ADd memory to accumulator with Carry.
AND  AND memory with accumulator.
ASL  Arithmetic Shift Left one bit.
BCC  Branch on Carry Clear.
BCS  Branch on Carry Set.
BEQ  Branch on result EQual to zero.
BIT  test BITs in accumulator with memory.
BMI  Branch on result MInus.
BNE  Branch on result Not Equal to zero.
BPL  Branch on result PLus).
BRK  force Break.
BVC  Branch on oVerflow flag Clear.
BVS  Branch on oVerflow flag Set.
CLC  CLear Carry flag.
CLD  CLear Decimal mode.
CLI  CLear Interrupt disable flag.
CLV  CLear oVerflow flag.
CMP  CoMPare memory and accumulator.
CPX  ComPare memory and index X.
CPY  ComPare memory and index Y.
DEC  DECrement memory by one.
DEX  DEcrement register S by one.
DEY  DEcrement register Y by one.
EOR  Exclusive OR memory with accumulator.
INC  INCrement memory by one.
INX  INcrement register X by one.
INY  INcrement register Y by one.
JMP  JuMP to new memory location.
JSR  Jump to SubRoutine.
LDA  LoaD Accumulator from memory.
LDX  LoaD X register from memory.
LDY  LoaD Y register from memory.
LSR  Logical Shift Right one bit.
NOP  No OPeration.
ORA  OR memory with Accumulator.
PHA  PusH Accumulator on stack.
```

```
PHP  PusH Processor status on stack.
PLA  PuLl Accumulator from stack.
PLP  PuLl Processor status from stack.
ROL  ROtate Left one bit.
ROR  ROtate Right one bit.
RTI  ReTurn from Interrupt.
RTS  ReTurn from Subroutine.
SBC  SuBtract with Carry.
SEC  SEt Carry flag.
SED  SEt Decimal mode.
SEI  SEt Interrupt disable flag.
STA  STore Accumulator in memory.
STX  STore Register X in memory.
STY  STore Register Y in memory.
TAX  Transfer Accumulator to register X.
TAY  Transfer Accumulator to register Y.
TSX  Transfer Stack pointer to register X.
TXA  Transfer register X to Accumulator.
TXS  Transfer register X to Stack pointer.
TYA  Transfer register Y to Accumulator.
```

“Look at all of those instructions!” said Pat “That would sure take a lot of time to look up the opcodes for all of them after the mnemonic version of a program was finished. Hmmm, couldn't the mnemonic version of a program be given to the computer so that it could do the opcode lookup automatically?”

“Yes it could," I replied “and this is what the early programmers thought of too!” The type of program they developed to do this is called an **assembler** and what it does is take the mnemonic version of a program and convert it into its machine language equivalent. The name they then gave the mnemonic version of a program is **assembly language** and it is the **source code** that the assembler takes as its input information. Very few programmers develop programs in machine language today, but a number still write programs in assembly language.”

“Will the machine language for one CPU run on another CPU?” Pat asked.

“That depends on a number of things that we will not get into now, but the short answer is that if the second CPU is the same model, or in the same 'family', as the first CPU then it would. If the second CPU is a different model, or in a different CPU 'family', then no it wouldn't. For example, the assembly language for the 6510 CPU, which is the CPU that the Commodore 64 contains, will not run on an x86 family processor which most personal computers use.”

Low Level Languages And High Level Languages

“To get back to your question about how a computer can be programmed in machine language and in BASIC, one has to understand that even though assembly language was easier to use than machine language, it was still somewhat difficult for humans to develop programs with.

The early programmers wanted to develop programs in a language that was more like a human language, English for example, than the machine language that CPUs understand. Both machine language and assembly language are considered to be **low level languages** because the thing that gives the numbers in these languages their contextual meaning is the CPU's hardware. Programmers wanted to work with computer languages that have much of their contextual meaning derived from human languages so that the ideas that the programs worked with were more natural for humans to use. They then figured out ways to use the low level languages they could already program in to create the **high level languages** that they wanted to program in.

“This is when languages like FORTRAN (in 1957), ALGOL (in 1958), LISP (in 1959), COBOL (in 1960), BASIC (in 1964) and C (1972) were created. Ultimately, a CPU is only capable of understanding machine language and, just like assembly language needs to be converted to machine language before a CPU can understand it, so it is with all computer languages.”

Compilers And Interpreters

“How is a high level language converted into machine language?” asked Pat.

“There are two types of programs that are commonly used to convert a higher level language into machine language.” I replied. “The first kind of program is called a **compiler** and it takes a high-level language's source code (which is usually in typed form) as its input and converts it into machine language. After the machine language equivalent of the source code has been generated, it can be loaded into a computer's memory and run. The compiled version of a program can also be saved on a storage device and loaded into a computer's memory whenever it is needed.”

The second type of program that is commonly used to convert a high-level language into machine language is called an **interpreter**. Instead of converting source code into machine language like a compiler does, an interpreter reads the source code (usually one line at a time), determines what actions this line of source code is suppose to accomplish, and then it performs these actions. It then looks at the next line of source code underneath the one it just finished interpreting, it determines what actions this next line of code wants done, it performs these actions, and so on.”

“An example of an interpreter is the BASIC interpreter that is in the Commodore 64. When we typed in the line of BASIC code that asked the Commodore to print the contents of a memory location, and pressed the Return key, the Commodore's BASIC interpreter read the line we typed, determined which memory location we wanted to see the contents of, and then printed this number to the screen.”

“How many computer languages are there?” asked Pat.

“Thousands of computer languages have been created since the 1940's,” I replied “but there are currently around 2 to 3 hundred historically important languages. Lets see if we can find a list of them.” I brought up a browser on my PC, did a search on 'computer languages' and located a page that listed the historically important ones.” (http://en.wikipedia.org/wiki/Timeline_of_programming_languages)

The Three Types Of Computer Memory

Pat looked at the list of historically important computer languages for a while then asked “Earlier you said that a compiled program can be stored for later use. I know that a computer usually stores programs on its 'hard drive' but where does something like a hard drive fit into this model of a computer that is on the whiteboard?”

“Now that we have gone through the work of figuring out how a computer operates at its lowest levels," I replied “it is easier to explain how devices like hard drives are attached to one. Instead of using the detailed model of a computer that we have developed on this whiteboard, though, I am going to draw a similar diagram that is more general.”

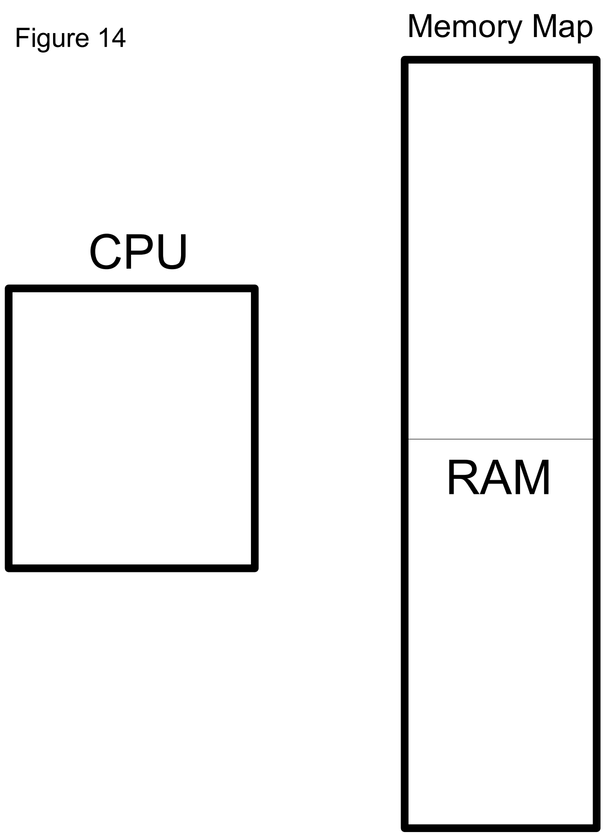
I picked up a blank whiteboard and started whistling the theme to '2001 A Space Odyssey' again as I drew another memory map. Instead of drawing the individual memory locations, however, I left the memory map unfilled but still labeled it 'memory map” at the top. I also drew an empty square to the left of the memory map and labeled it 'CPU'.”

RAM (Random Access Memory)

I then asked Pat “What does the word RAM mean to you?”

Pat thought for a few moments then replied “I think RAM has something to do with how much memory a computer has. I know that my Mom's computer did not have enough RAM to run a new program she bought so she had a friend add more RAM to it.”

“That is correct,” I said “**RAM** is one of three types of memory that can be present in a memory map. RAM stands for Random Access Memory but a better name for it would have been RWM or Read Write Memory because numbers can be both copied into this kind of memory and copied out of it. All the numbers in RAM memory locations will keep whatever numbers they hold as long as the computer is on but, when the computer is turned off, all the numbers in all RAM memory locations are lost. Memory that looses the numbers it contains when the power it turned off is called **volatile memory**.” As I said this I drew a horizontal line across the middle of the memory map and then label the bottom half of the rectangle RAM. “In this new model of a computer, I am having the bottom half of the memory map represent RAM memory locations. In a PC, there are millions of RAM memory locations which is too many to show in this model. Instead of drawing all of the RAM locations individually, I am representing them with this rectangle labeled 'RAM'” (see Fig. 14)



“As long as a computer is powered up," I continued “every memory location will always contain a number between 0 and 255. There is no such thing as a blank memory location when the power is on. When the power is off, however, all of the RAM memory locations are blank. When the computer is first turned on, each RAM memory location has a number between 0 and 255 randomly appear in it during the time that the system's power rises to its operating level.”

“Since these RAM memory locations come up with random numbers in them, there is no contextual meaning associated with these numbers so they do not hold any meaningful information. Computer programmers sometimes say that memory locations that do not have any contextual meaning associated with them contain **garbage**. After a computer has gone through its power-up cycle, its RAM memory locations are ready to have numbers copied into these locations that have contextual meaning associated with them. The numbers that represent machine language programs have contextual meaning associated with them and an example of this was the small machine language program we developed a little while ago.”

“But now we have a problem," I said “because when the power-up cycle on a computer is finished, a small electronic circuit senses this then sends a signal to the CPU that says 'the power is on now, start running!' **Most CPUs have an address built into them at the factory which is the address in the memory map where they should look for their first machine language instruction immediately after power-up**. In the Commodore 64, this address is 65532.”

“If a machine language instruction has not been purposefully placed into this memory location, the computer will lock up and everyone has had a lot of experience with their computers locking up!”

Pat laughed and said “Oh yes! My computer locks up all the time!”

I smiled and continued “After this first machine language instruction has been executed, the Program Counter is set to the next machine language instruction in the sequence, it is then executed and so on.” This next part was important so I dropped the level of my voice a little and said “**if there is ever an instant in time when the CPU is ready to execute a machine language instruction, and the number it pulls from the memory location that the Program Counter is pointing to is not part of the program that is running, the computer will also lock up…** Most of the time that a computer locks up, this is the cause.”

“You mean something as simple as that can lock up a computer?” Pat said “Why is that?”

A CPU Is A Very Dumb Device

“Do you remember when I said earlier that a CPU was one of the dumbest things in the world?” I asked.

“Yes” said Pat “it was when we were talking about the CPU being like a simple calculator.”

“The reason that a CPU is so stupid," I continued “is that it needs to be told exactly what to do, step by step, the whole time it is running. In order to get a feel for how stupid this is, imagine that you had to be told exactly what to do, step by step, from the time you woke up in the morning until the time you went to sleep at night. Your instructions might look something like this:

1) Open your left eye.
2) Open your right eye.
3) Take your left hand and pull your covers down until they are below your feet.
4) Turn your whole body 90 degrees so that your legs are hanging off the side of the bed.
5) Place your left foot on the floor.
6) Place your right foot on the floor.
7) Raise your back 90 degrees so that you are sitting straight up.
8) Put your left hand on the edge of the bed.
9) Put your right hand on the edge of the bed.
10) Push yourself up with your arms into a standing position...”

As I said these instructions, I acted some of them out and Pat began laughing.

“You see," I said “this is pretty stupid. Now imagine that your instructions were suppose to say 'turn left 45 degrees. Walk forward 8 steps', but instead they said 'turn right 180 degrees. Walk forward 1000 steps'. These look like legitimate instructions but they are really garbage instructions because they told you to turn around and face your bed then walk forward 1000 steps!”

“A similar thing can happen with a computer. Through a programming error, a machine language instruction (or data for an instruction) can be placed into a program that does not mean anything in the context of the program. This can cause the computer to attempt to do something just as silly as you trying to walk through your bed. Once a garbage instruction has been executed, the CPU usually looses track of where it was suppose to be in the program and it continues to execute garbage instructions in memory until somebody pushes the reset button. Do you see now how easy it can be to lock up a computer, Pat?”

“Yes” said Pat “In fact, I was thinking that it seems so easy to lock up a computer that it is a wonder that they do not lock up more often than they do.”

“I agree," I said “and if your PC locks up, you just need to restart it. If the engine computer on something like a passenger jet locks up, however, that could be a big problem!”

“Wow” said Pat “I wouldn't want to be on a passenger jet if that happened!” Pat looked at the ceiling, thought for a few moments then said “I hear about PC's locking up all the time, but I have never heard about the engine computer on a passenger jet, or even a car, locking up. How come one kind of computer locks up a lot, but other kinds don't lock up very much at all?”

“That is a difficult question to answer completely at this point in our discussion.” I replied. “If you ever take me up on my offer to help you to learn how to program a computer, though, ask this question again and I will try to explain it to you.”

“Okay” said Pat.

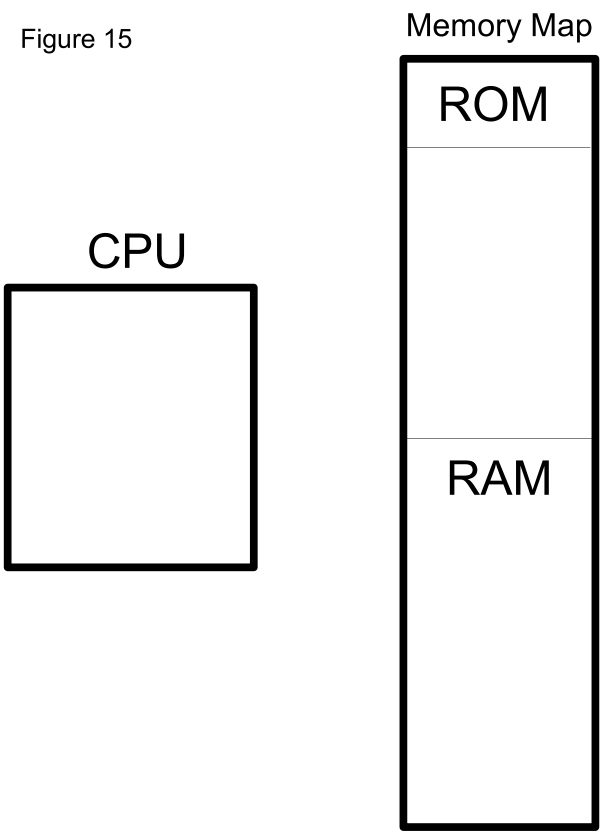
“Now I have a question for you.” I said “If all the RAM memory locations in a computer come up with 'garbage' numbers in them, where in memory does the CPU go to get its first instructions when it first powers up?”

“Hmmm” said Pat, while looking at the memory map. “It can't get its first instructions from RAM because RAM contains garbage data in it right after it powers up. It seems that we need a kind of memory that remembers its numbers even when the power is off. You had said that there are three basic types of memory in a memory map, does one of the other two types work like this?”

ROM (Read Only Memory)

“Yes," I said “very good! One of the other two types of memory in a memory map is called **ROM** memory and it stands for **Read Only Memory**. Another name for this memory is **non-volatile** memory. The name ROM fits this type of memory a little better than RAM's name does because the numbers in this second type of memory are meant to mostly be copied, or read, from. The special thing about ROM memory is that after numbers have been placed into it, they will be held there even after the power is turned off." As I was saying this, I drew a second horizontal line about one eighth of the way down from the top of the memory map then labeled the topmost rectangle 'ROM'. (see Fig. 15)

Figure 15



"If ROM memories are read only, how do the numbers get into them in the first place?" asked Pat.

"There are different kinds of ROM chips," I said "and there are various ways that the numbers can be placed into them. The earliest ROM chips had the numbers placed into them during the manufacturing process. These ROMs are inexpensive to make but the numbers that are placed into them can never be changed. This means that if different numbers were needed in the area of memory that this type of ROM was in, the old ROM chip would have to be removed and thrown away and a new ROM chip put in its place."

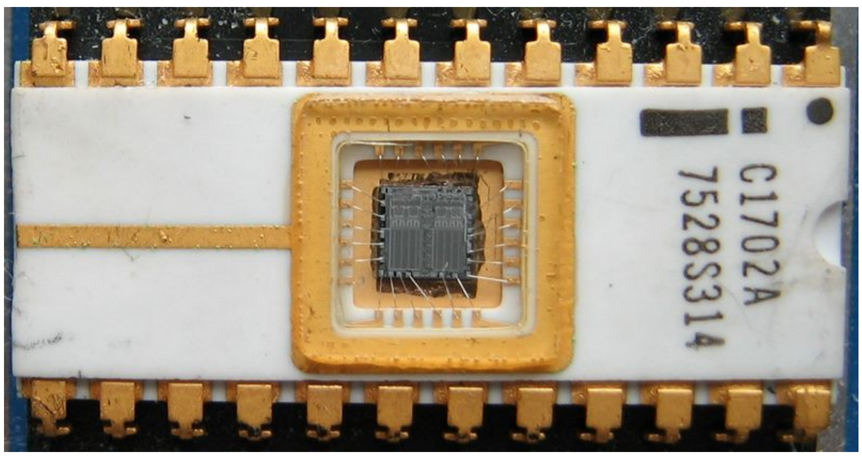
"The need for ROMs to have the ability of having their numbers reprogrammed 'in the field' (which means where they are being used) lead to the development of a chip called a **PROM** which is a **Programmable Read Only Memory**. These chips had little patterns of fuses in them that would be burned when the devices were programmed. They were more flexible than the early ROMs but the disadvantage of these chips was that they could only be programmed once."

The need to have a ROM that could be reprogrammed many times lead to the development of the **EPROM**, which stands for **Erasable Programmable Read Only Memory**. EPROMs can have programs placed into them by anyone having a device called an EPROM programmer. What is even more interesting is that these chips have a small round window on their top that allows light to shine into them. If ultra violet (or UV) light is shined into this window for perhaps 10 minutes, the numbers that were last programmed into the chip are erased by this light.”

“Aside from their use as ROMs, EPROMs are a very interesting kind of computer chip to have because they allow people to see what the inside of a chip looks like. I have some old EPROMs around here somewhere, would you like me to give you one?”

“Yes please!” said Pat

So I searched through my collection of electronic parts until I found an EPROM which I then gave to Pat.



As I handed Pat the EPROM chip, I was again careful to lightly touch Pat's hand with my pinky before I placed the chip into it.

ESD (Electro Static Discharge)

“Why do you keep touching my hand with your pinky finger before you give me a chip?” Pat asked.

I replied “Have you ever walked across a carpeted room during the Winter, reached out your hand to open a door and then received a shock of static electricity from the metal door knob?”

“Oh yes.” answered Pat “Sometimes I have even seen a blue spark jump between my hand an the door knob and those shocks really hurt!”

“Believe it or not,” I said “little sparks like that often move between your fingers and the things you touch even if you cannot feel them. Another name for these sparks is **ESD** or **Electro Static Discharge** and it is caused by static electricity. Most of the time ESD sparks do not cause any harm, but if you allow sparks like that to hit a computer chip, the chip can easily be damaged. I have a couple of stories about ESD and computer chips that you may find interesting.”

“The first story happened when I was younger and working for a company called Tire Tele as an electronics technician. Tire Tele manufactured low tire pressure warning systems for automobiles and these systems consisted of sensor units, which were placed inside each tire of an automobile, and a receiver which was placed on the dash board. The sensor units would periodically send pressure information to the receiver and, if any of the tires was loosing pressure, the receiver would alert the driver.”

“Tire Tele began selling their product to people and everything was going fine. Then, about 6 months after the first units were sold, they started to fail and people began returning them for repair or for a refund. The Tire Tele engineers determined that the computer chips in these devices were failing and so they sent a few of the dead chips back to the chip manufacturer for analysis. The chip manufacturer disassembled the chips, looked at them under a high power microscope and discovered that the little electronic circuits in the chip were being damaged by ESD.”

“The chip manufacturer sent some of their engineers to the Tire Tele plant to observe how the units were being assembled and they discovered that none of the people on the assembly line were using anti-static protection devices or procedures. One anti-static procedure that all people who work with computer chips use is to make skin-to-skin contact with a person before handing a computer chip to them. The skin-to-skin contact allows the static electricity level between the two people to equalize which will prevent an ESD spark from traveling into the computer chip when it is handed over. As soon as anti-static equipment and procedures were put into place in the Tire Tele facility, their ESD problems disappeared.”

“Who would have thought that such a small thing as little sparks could cause such a problem?” said Pat “What is your second story?”

“The second story happened when I was visiting a High School," I replied “in order to demonstrate a computer interface board I had built. I was placed in a large carpeted room, along with other people who were demonstrating things to the students, and the carpet caused a great deal of static electricity to accumulate in the room. The computer interface board I had made contained a speech synthesis chip on it that would take numbers as input and turn these numbers into various words.”

“I had written a program that made the chip recite the letters of the alphabet over and over again. The computer would say 'A, B, C, D...' in a mechanical voice that sounded like a robot. As students would come to my display, I would point to each chip on the board, explain what it did and I would end by saying 'this last chip allows the computer to talk'. About half way through the day, a group of students came to my table, I went through my explanations and, as I pointed at the speech chip, a huge blue spark jumped from the end of my finger into the chip and it immediately went from saying 'A, B, C, D' to mumbling 'MWA BLA VLAZ DAUP'!. That chip never did work correctly again!”

Pat started laughing and so did I! “It wasn't very funny at the time," I said “but it certainly seems funny now!”

“Anyway, that is an EPROM that you have in your hand and after they were invented in 1971, computer development in general moved forward at a quicker pace because of the shorter time it took to reprogram these devices. Even though the EPROM was a very useful device, it was still somewhat difficult to work with because it needed to be placed in a UV eraser before it could be reprogrammed. This lead to the **EEPROM**, or **Electrically Erasable Programmable Read Only Memory**, being developed in 1981. Instead of UV light being needed to erase these devices, they could be erased and reprogrammed electronically one memory location at a time”

“One of the more recent types of ROM memory chips is called **Flash** memory and it also can be reprogrammed electronically. Unlike EEPROMs, however, Flash memory has to be reprogrammed in blocks of memory locations but, since it is less expensive to make than EEPROM memory, it has become very popular where large amounts of storage are needed. Flash memory is not only used in personal computers, it is also used as storage memory for digital audio players, USB drives, mobile phones and digital cameras.”

“I have an MP3 player” said Pat “if it has Flash ROM in it, does this mean that the player has a computer in it?”

“There is a good chance that it does," I said “and if it has a computer in it, then that computer is going to work in a similar manner to the models of a computer that we have been drawing on the whiteboards. Once you understand how this model works, you understand how most of the computers in the world work. That is very powerful knowledge to have.”

“Amazing!” said Pat “I feel like I am stepping into a whole new world! I had never thought too much about computers before, but now that I am starting to see how they work, I want to know more about them.” after a pause, Pat continued “I am beginning to understand how numbers can be placed into the various ROM memories. What I want to know now is what the numbers usually mean that are put into these ROMs.”

BIOS And POST

“Lets go back then," I said “to the question I asked you about what happens when a computer first powers up. I asked 'If all the RAM memory locations in a computer come up with 'garbage' numbers in them, where in memory does the CPU go to get its first instructions when it first powers up?' We determined that some type of ROM chip needs to be placed into the section of memory that contains the address that a CPU first goes to when it is turned on. A machine language program is placed into this ROM chip and the machine language program usually tells the CPU to check the various parts of the computer system to make sure they are operating correctly.”

“On a typical PC, the ROM that is placed in the part of memory that the CPU first looks at for its initial instructions is called the **BIOS** or **Basic Input Output System**. The part of a PC's BIOS that tells the CPU to check the computer system for correct operation is called the **POST** or **Power On Self Test** code. When you first turn on your PC, Pat, what kinds of things do you notice?”

“Well” said Pat “the first thing that happens is that the screen flashes on and changing numbers are then shown at the upper left of the screen. After this, the lights on my keyboard blink, my hard drive starts making noise and then a little later my graphic desktop is shown.”

“These are all a result of the machine language POST code telling your CPU to check each of these devices.” I said “The changing numbers are shown as the CPU checks the system's RAM chips, the keyboard lights are blinked when it is checked and the hard drive makes noise when it is checked. There are many more devices in the PC that the POST code also checks but these do not make noise nor do they flash lights or print to the screen.”

“This should answer your question about the meaning of the numbers that are typically placed into ROM memory. A significant amount of these numbers represent a machine language program that tests the computer system when it is first turned on. Other parts of the same ROM also usually contain machine language code that controls various pieces of hardware that are attached to the system. We will talk about this other kind of code later.”

“For now we have another a more pressing problem. The amount of ROM in a PC is usually much smaller than the amount of RAM it has. After the CPU has finished executing the POST code in the BIOS ROM, it needs more machine language instructions to run. The BIOS ROM, however (being relatively small) has very little room for extra instructions. The CPU's Program Counter could be reset back to the beginning of the ROM and the POST code could be re-executed, but this would result in the CPU re-executing the POST code over and over again and the computer could not be used to do any useful work.”

“After the POST code, there is only room for a small number of final instructions for the CPU and, if it cannot find any more instructions, its Program Counter will run off the end of the ROM memory into garbage memory and the numbers in the garbage memory will quickly lock the CPU up. Therefore, the remaining instructions in the ROM should be used to tell the CPU where to find more instructions, but where is it going to get them from Pat?”

Pat studied the memory map for a while then said “The CPU can't get more machine language instructions from RAM because the computer was just turned on and all the RAM locations contain garbage numbers. It also can't get more machine language instructions from the ROM because the ROM is fairly small and it has already used most of the instructions in there. Hmmm, compiled programs consist of numbers that represent machine language instructions and, from what I know, the programs that a PC can run are stored on its hard drive. My guess is that the CPU can get the machine language instructions it needs from the programs on its hard drive.”

“You are right.” I said “After it has finished running its POST code, a PC usually obtains the machine language instructions it needs from its hard drive. But how does the PC's CPU talk to a hard drive? We have not placed a hard drive into our whiteboard model of a computer yet, where do you think it should go?”

I/O Memory

Pat looked at the whiteboard model while thinking about this then said “I am not sure where it should go. Earlier, though, you said that there were 3 kinds of memory in a computer and we have only talked about two of them, which are RAM and ROM. Maybe the hard drive is attached to this third kind of memory.”

“That is a good guess," I said “the hard drive in a computer is attached to the third kind of memory.” As I said this I pointed at the whiteboard to the the area of the memory that was between the ROM memory and the RAM memory. “As with RAM and ROM, this third kind of memory, which is called Input/Output (or I/O) memory, also consists of memory locations that can hold a number between 0 and 255.” As I was saying this, I drew evenly spaced horizontal lines in the I/O memory part of the memory map to represent its memory locations. I then erased a little opening on the left side of each of these I/O memory locations. “Notice that I have put an opening in the left side of each of these memory locations to show that the CPU has access to each one of them, just like it does with the RAM and ROM locations.”

“Instead of starting with a hard drive, though, lets see how something simpler, like a keyboard, is attached to a computer. The first thing we need to do is to show on our model what is 'inside' of the computer and what is 'outside' of it. By 'inside' and 'outside' I do not mean inside and outside the box that the PC is in. I mean what is included in the core part of the computer system and what is outside of this core.” I then drew a vertical dashed line to the right of the memory map and said “Everything to the left of this vertical dashed line can be considered to be inside the core of the computer and everything to the right of it is outside.”

I then drew a small horizontal rectangle to the right of the dashed line and wrote the word 'Keyboard' inside of it. Finally, I pointed at this rectangular model of a keyboard and said “when you press a key on a keyboard, Pat, what do you think happens?”

Pat thought about this then replied, “The key is turned into electronic signals and sent to the computer through the keyboard's cable.”

“This is true," I said “but the interesting part is what the electronic signals represent. When a key is pressed on a keyboard, say the 'A' key, the idea of the capital letter 'A' is turned into a number, and the electronic signals that are sent through the keyboard's cable represent this number.”

“But where does the other end of a keyboard's wire attach to the computer at? This is where the I/O memory locations come in. I/O memory locations are special memory locations because, not only do they have an opening that faces towards the CPU, they also have another opening that faces the outside of the computer! The way that a device that is outside the core of a computer sends information into the computer is through one of these I/O memory locations.” I then drew a line from the left side of the keyboard through the vertical dashed line and into the right side opening of one of the I/O memory locations. (see Fig. 16)

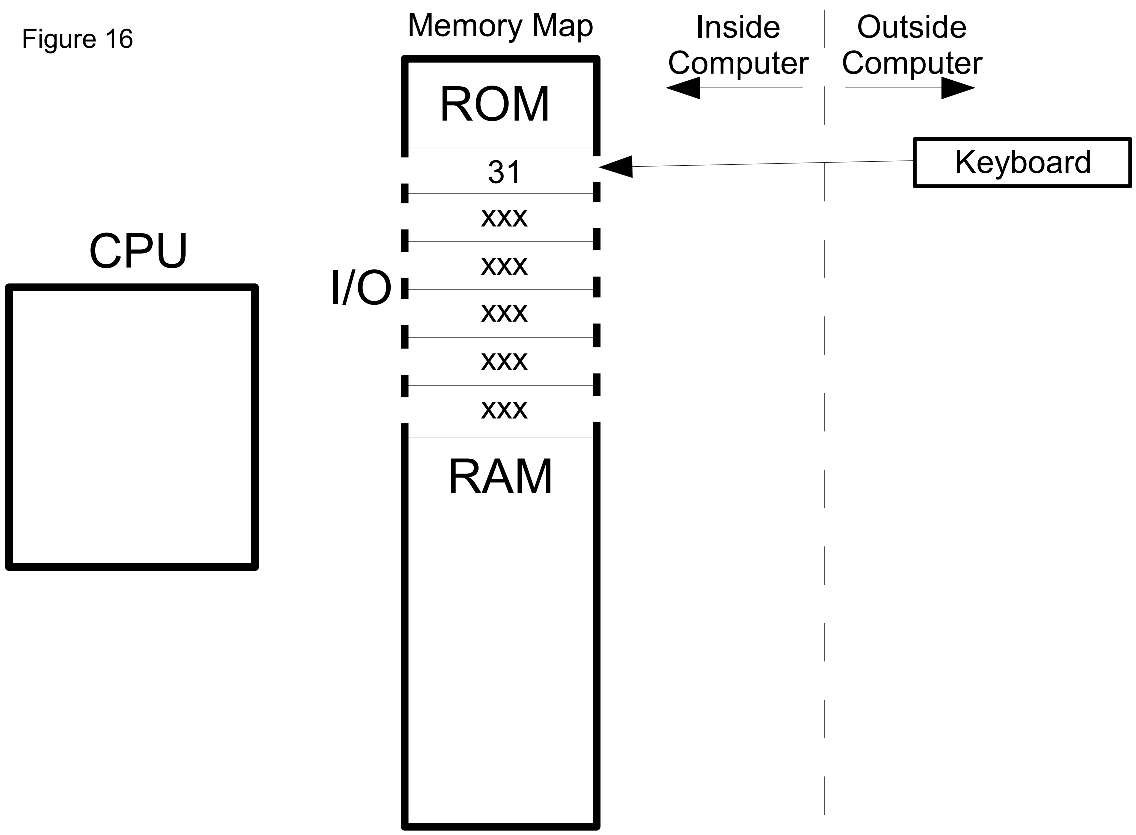


Figure 16

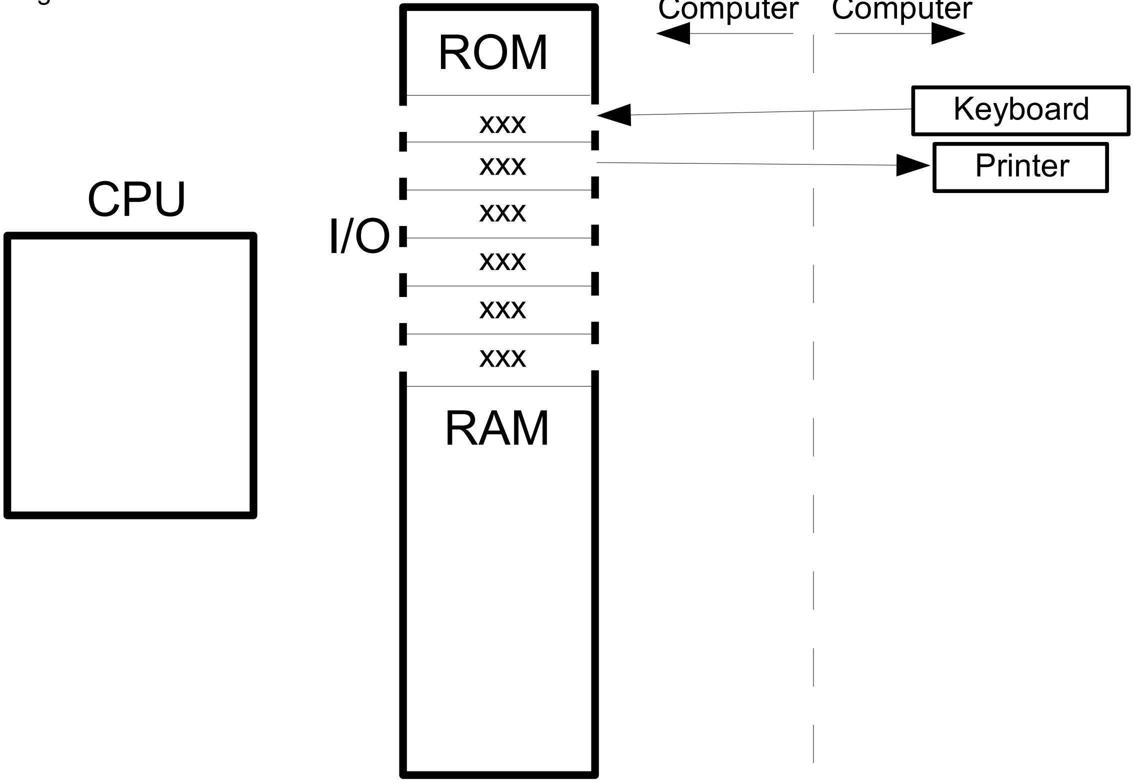
“This line represents the keyboard's cable and, when a key is pressed on the keyboard, a number between 0 and 255 (that represents that key) is sent through the cable. This number then appears, as if by magic, in the I/O location that the cable is attached to. After the number appears in the I/O location, the CPU can access this same I/O location from its left-facing opening in order to determine which key had been pressed.”

“Thats cool!” shouted Pat “I never would have guessed that a keyboard worked that way, but it makes sense!”

“I agree," I said “it does make sense and when I first learned how I/O memory locations worked, I was as excited as you are! The last thing I am going to do with the model of the keyboard is to place an arrow at the end of the cable that is attached to the I/O location to show that the keyboard send data into the computer.” which I did.

“The next device I am going to draw is a printer.” Underneath the printer I drew another horizontal rectangle and wrote the word 'Printer' in it. I then drew a line between the printer and another one of the I/O memory locations. (see Fig. 17) “This is a simplified model of how a printer attaches to a computer. Now that you know how a keyboard sends a letter, like a capital 'A', to a computer, see if you can explain how a capital letter 'A' might be printed on a printer.”

Figure 17



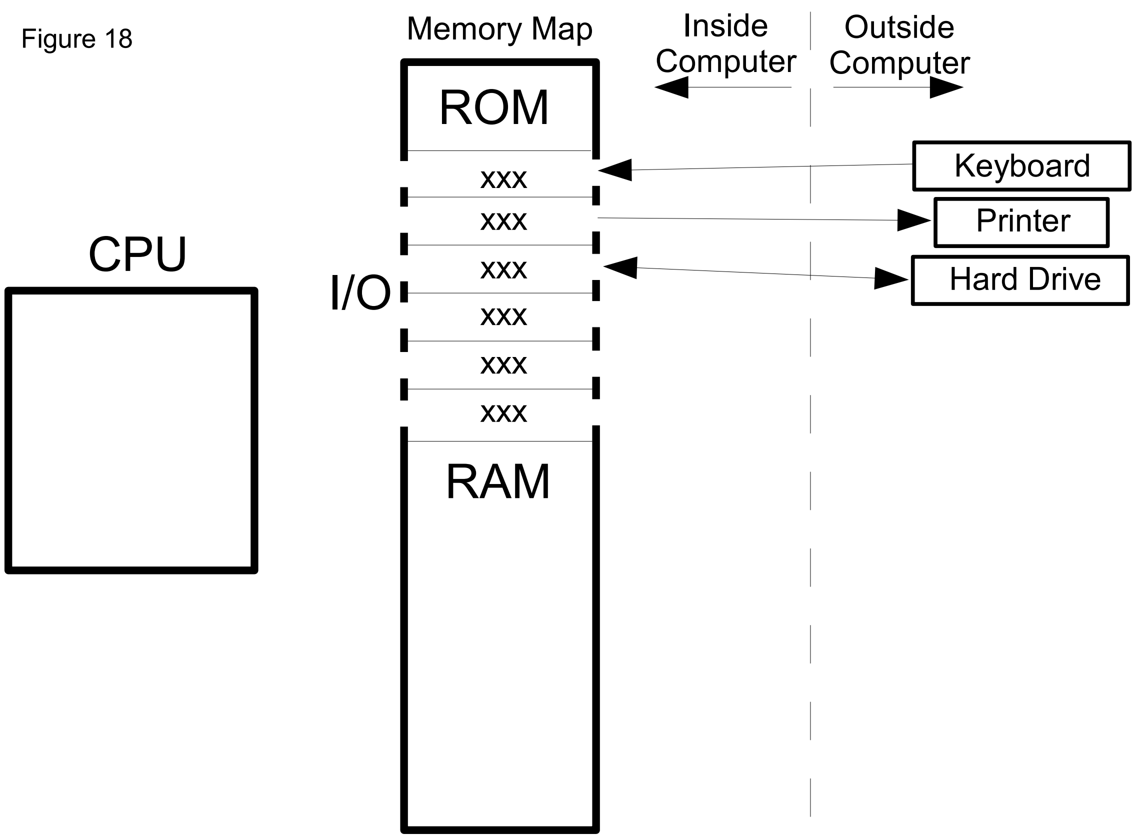
Pat looked at the model and said “Lets see, the CPU places a number that represents a capital letter 'A' into the I/O location that the printer is attached to, this number is then converted into an electronic signal that represents the 'A' and the electronic signal is sent to the printer. The printer converts the electronic signal back into a number, determines that it represents a capital letter 'A', and then it prints it.”

“Very good Pat!” I said “Finally, which way should I point the arrow on the printer's cable?”

“Point the arrow towards the printer, because information goes from the computer out to the printer.”

“Correct.” I said, and I drew an arrow on the printer's cable that pointed towards the printer. “Now Pat, I think we know enough about how I/O memory locations work to go back to the hard drive.” I drew another horizontal box (underneath the box that represented the printer) and wrote the words 'Hard Drive' in it. I then drew a line from the hard drive's rectangle to an unused I/O location then I said “which way should the arrow point on the hard drive's cable?”

Figure 18



Pat thought about this for a moment then said “You should draw an arrow

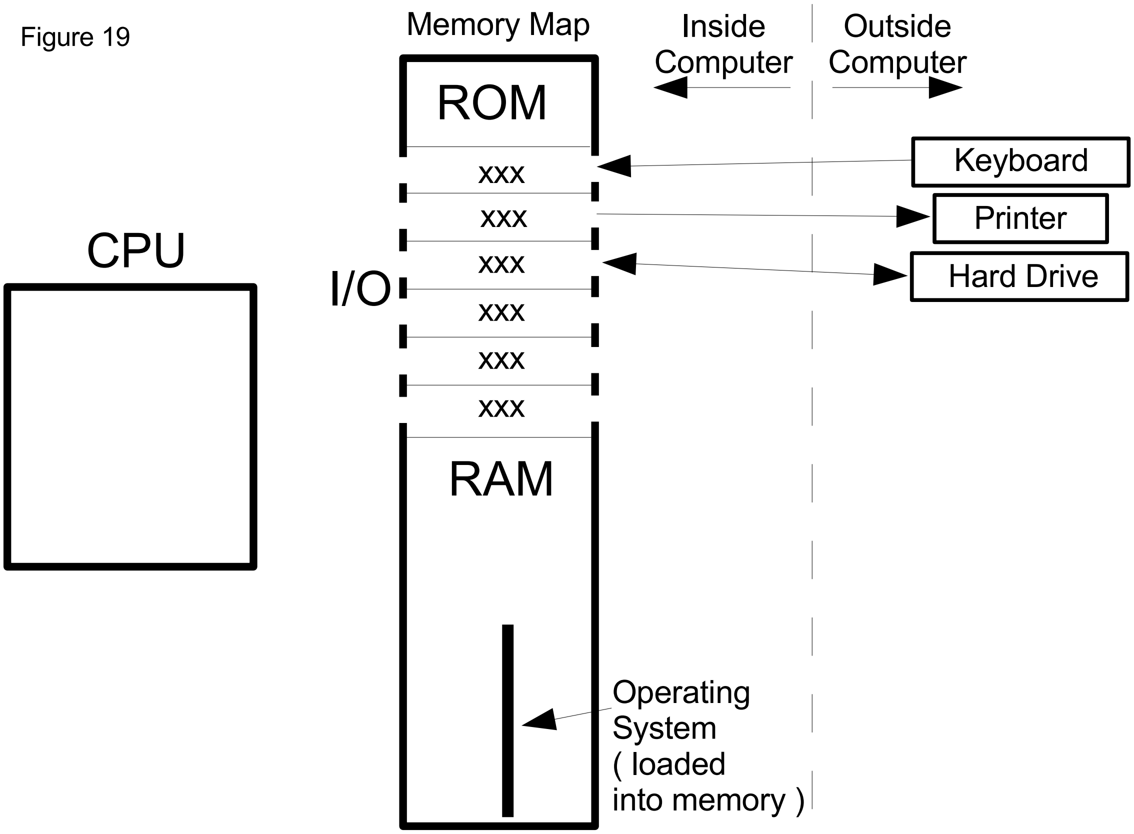
on both ends of a hard drive's cable because a computer can send information to a hard drive and it can also read information from a hard drive.” (see Fig. 18)
“I thought I was going to trick you with that question Pat," I said “but I was wrong!” I then drew arrows on both ends of the hard drive's cable to show that information can be sent both ways along it.

Loading An Operating System

“Now,” I said “lets go back to discussing what happens when the CPU is finished running the POST code and it needs to find more instructions to execute or it will lock up. The remaining code in the ROM tells the CPU to talk to a storage device (like a hard drive, a Flash drive or a CDROM) through the I/O location that the storage device is attached to. Actually, in a real computer, more than one I/O location is used to talk to a storage device, but for now a one I/O location example is simpler to work with. The CPU talks to a storage device and asks it if it has a program called an **operating system** stored on it.”

“If the storage device does have an operating system program stored on it, the CPU requests that the device send the numbers that represent the operating system's machine language instructions to the I/O location that the device is attached to, one number at a time. As each number arrives in the storage device's I/O location, the CPU copies this number to a register and then it copies the number from the register to RAM. An operating system consists of thousands and thousands of machine language instructions so I can not show them all being placed in RAM individually. Instead, I am going to draw a vertical bar from the bottom of the RAM upwards which represents RAM being filled with the numbers that represent the operating system," which I did. (see Fig. 19)

Figure 19



Operating Systems: Bridges To Cyberspace

“What exactly is a computer operating system?” Pat asked “I have always been confused by this.”

I thought about Pat's question for a few moments then said “Do you remember earlier when I said that, in a way, computers were magic because part of a computer exists in the physical world and the other part exists in a non-physical realm called cyberspace?”

“I remember.” said Pat.

“Do you also remember what **context** and **contextual meaning** are?” I asked.

“Yes,” said Pat “context means the circumstances within which an event happens or the environment within which something is placed. Contextual meaning is the meaning that the context gives to the events that happen, or things that are placed, within it."

"Very good, Pat. Now it is time to give you my explanation for what cyberspace is." I said. "**Cyberspace consists of all of the ideas that are currently bound to numbers in any computer, anywhere in the physical universe, through contextual meaning.**"

Pat sat quietly for a while, with eyes staring off into space, thinking about what I just said. Finally, Pat blinked, looked at me and said "When you described what cyberspace was, a picture came into my mind and in it were millions of computer memory locations laid out across the Earth, with millions of ideas floating above them, and they were connected to each other by threads of contextual meaning."

"Some people," I said "think that there really is a world of ideas that is separate from the physical world and perhaps some day we will know what ideas actually are. Even if we do not know exactly what ideas are yet, though, we do know that it is the computer's ability to easily move ideas into and out of cyberspace, and easily manipulate ideas when they are there, that gives the computer its great power."

"I will explain cyberspace more fully as we continue our discussion, but lets use our current understanding of cyberspace to help us understand what a computer operating system is. A computer operating system is a special kind of program that acts as a bridge between the physical world and cyberspace. Most of the sophisticated computers in the world (including PCs, servers, ATM machines, car computers and cell phones) have an operating system in them and it is the operating system in a device that enables it to access the resources of cyberspace."

Systems Within Systems

"We have been using the word 'system' quite a bit, for example 'computer system' and 'operating system', but what is a good definition of a system? Before we continue, lets find a definition for system on the Internet." I went to my computer, searched for a definition of the word 'system' and found the following:

> **System:** A group of interacting, interrelated, or interdependent elements or parts that function together as a whole to accomplish a goal.
> http://www.doe.mass.edu/frameworks/scitech/2001/resources/glossary.html

"This definition," I said "indicates that the purpose of a system is to accomplish a goal and that a system is made up of parts that work together to accomplish this goal. Examples of systems include the water system that provides water to your home, a skyscraper and an automobile engine. The parts in a system can also be arranged into groups that form systems of their own and these smaller systems are often called **subsystems**. The prefix 'sub' means 'under' so another way to think about a subsystem is as an 'undersystem'. Subsystems can contain subsystems of their own and many of the things in the world contain multiple levels of systems within systems. A skyscraper's subsystems include its heating and cooling system, lighting system, telephone system, elevator system and cleaning system (which include janitors)."

"People can be part of a system?" asked Pat.

"Yes," I replied "there are may kinds of systems that have people as parts. A building's cleaning system fits our definition of a system because it has a goal (to keep the building clean) and it contains parts that interact together to attain this goal. The mops, dusters and garbage cans in a building are parts in its cleaning system, but so are the janitors that interact with these parts."

"I had never thought that people could be parts in a system" said Pat "but you are right, they can."

"There are many types of parts that can be used in a system," I said "including metal and plastic parts, rubber hoses, water, air, electricity, people, and information. It is this last kind of part, information, that I would like to focus on."

"Information!?" said Pat "How can information be a part in a system if you can't even touch it?"

"Information is present in every system," I said "no matter what kind of system it is, but sometimes it is not obvious how a system uses the information that is contained within it because information is not physical. It cannot be seen or touched."

“Lets take a simple system, like a combination lock, and see if we can determine how information is being used in this system.” I went to my storage room and returned with a combination lock. “What is the goal of this system, Pat?” I said, and I gave Pat the lock.

Pat studied the lock for a while, turned the dial a few times, pulled up on its shackle then said “The goal of a combination lock is to prevent people from stealing your stuff.”

“And how does it do this?” I asked

Pat studied the lock some more then said “The lock's shackle is placed through something that has a hole in it, like a locker, then the shackle is closed. The lock will not release the shackle until the correct combination is entered on the dial, and the thing that the lock is attached to will not open until the shackle is removed from the hole it was put in.”

“Lets assume that the thing that the lock is attached to is a locker.” I said How does the locker know whether or not it has a lock attached to it?”

Pat replied “A person must first try to lift the locker's handle. If there is not a lock in the handle's hole, the handle will lift and the locker's door will open. If there is a lock in the handle, though, the metal around the hole will bump against the metal of the shackle when the handle is lifted and this bumping will prevent the handle from lifting far enough to open the door.”

“That is correct.” I said “One of the laws of physics states that 'two pieces of physical matter cannot occupy the same space at the same time'. The locker is a system that contains a lock as a subsystem and the lock uses this law of physics to inform the locker that it is not permitted to open. This kind of physical 'informing' or communication, the bumping together of two pieces of physical matter, is a common example of how information is used in a system. Another bumping-related example is the accelerator pedal in an automobile. If a driver wants to make a car go faster, they press their foot against the accelerator pedal, the pedal pushes against a lever, the lever usually pulls on a cable that has some kind of system at its other end that allows more air/fuel mixture to enter the engine which results in the engine turning faster.”

“Moving back to the lock, a human has the number sequence that will open the lock stored in their mind. This number sequence represents information and the way that this information is communicated to the lock is by turning the lock's dial. As the dial is turned, bumping-type information is communicated between the parts of the lock. If the correct combination is entered, the piece of matter that is informing the shackle that it cannot open is allowed to move freely and, when the shackle is pulled, this piece of matter moves away from the shackle as it is lifted and the lock opens.”

Physical Parts Are Costly And Constrained

“I am starting to see how information can be used as a part of a system," Pat said “but what does all of this have to do with computer operating systems and cyberspace?”

I smiled and replied “Parts made of physical matter have all kinds of constraints associated with them, Pat. If the parts are made of metal, for example, the ore for the metal has to be mined out of the Earth, then the ore has to be transported to a mill where it is transformed into metal shapes, like rods and bars, that are suitable for processing by machine tools. These metal shapes then have to be transported to the manufacturing facilities that contain these machine tools so that the machines can create parts from them. The parts are then transported to assembly facilities that assemble the parts into subsystems and these subsystems are often transported to yet other assembly facilities where they are assembled into final products. Each step along the way, from ore to finished product, takes time and energy to accomplish and time and energy translate into cost.”

“Another kind of constraint that parts made from physical matter are under are the laws of physics. These laws dictate that parts made from physical matter can be moved back and forth only so fast, they can only handle so much force applied to them and there are limits to how large or small they can be. Beyond this, once the parts are formed into a given shape to serve a given function, they cannot be easily reformed into other shapes to serve a different function. For example, if we wanted our combination lock to have a 4 number combination instead of a 3 number combination, it would be extremely difficult to reshape all of the parts in the lock to accomplish this.”

Moving Parts Into Cyberspace

”High cost and low flexibility are two of the main constraints that are associated with parts made of physical matter. But what if it were possible to take the parts of a system that only consist of information and move them into cyberspace?”

“Move them into cyberspace!?” Pat said. “Is this possible?”

“Yes Pat!” I replied “Any part of any physical system that stores or communicates information can be moved into cyberspace using a computer. As soon as a part is moved into cyberspace, it becomes an idea and ideas existing in cyberspace are constrained much less by the laws of the physical world than their physical counterparts are. Beyond this, parts can be made in cyberspace that would be extremely difficult, or even impossible, to make in the physical world. Cyberspace parts can even be created, assembled into systems, disassembled and destroyed quicker than you can blink your eyes. They can also be sent anywhere in the world through computer networks at the speed of light.”

Pat's mouth dropped open in amazement. “Cyberspace does sound like a magical place," Pat finally said “but I don't fully understand how it works.”

A Lock Made Of Physical Parts and Cyberspace Parts

“I do not think that anybody completely understands cyberspace yet, Pat," I said “but we are learning more about it all the time. Lets go back to the combination lock and see if we can describe a lock that has some physical parts and some cyberspace parts. This lock will still have a shackle, so that it can lock a locker, and it will still have a 3 number combination. Instead of having a dial, though, it will have a keypad so that a human can enter the combination. Instead of metal parts controlling whether the shackle is able to be opened or not, it will have a solenoid.”

“A solenoid?” Pat said “What is a solenoid?”

“Have you ever turned a nail into a magnet by wrapping a bunch of wire around it then putting electricity through the wire?” I asked.

“Yes,” said Pat “I read how to do that in a science book and, when I turned it on, I was able to pick up paper clips with it.”

I continued “A magnet made using this technique is called an electromagnet. A solenoid uses an electromagnet to pull on a metal arm in one direction and a spring is usually used to pull the metal arm in the opposite direction when the electricity is turned off. The result of this is that when the electricity is turned on, the arm moves up against a stop in one direction and when the electricity is removed, the spring moves it against a stop in the opposite direction. In our lock, the end of a solenoid's arm can be used to either allow the shackle to lift or to prevent it from lifting. Solenoids are on/off devices and computers love to control devices that are either on or off."

"Why is that?" asked Pat.

"I will make a deal with you Pat." I said "Do some research about computers on the Internet when you go home. The next time you come over, if you can tell me why computers love things that are either on or off, I will give you a solenoid. Is it a deal?"

"Its a deal!" said Pat "But are you saying that we are going to put a computer inside of a lock?"

"Sure, why not?" I said.

Microcontrollers: Computers On A Chip

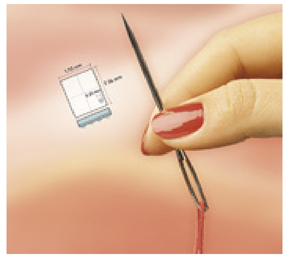
"Isn't a computer too big to fit into a lock?" Pat replied.

"Some computers are too big to fit into a lock," I said "but some computers are very small. Have you ever heard of a microcontroller?"

"No," said Pat "what is a microcontroller?"

"A microcontroller is a complete computer on a chip. Inside the chip is the same model of a computer that we have been developing on the whiteboard. It has a CPU, RAM, ROM and I/O in it." I then went to one of my drawers and returned with a small microcontroller. "Open your hand, Pat." I said. I then lightly touched Pat's hand with my pinky finger, to equalize our charges, and then placed the microcontroller in it."

Pat studied the microcontroller for a while then said "Its so small. Is there really a complete computer in this chip?"



“Yes," I said “and some microcontrollers are even smaller than that one!”

Pat studied the chip a bit longer then gave it back to me.

I held the chip between my fingers and said “If there is a program in this chip when it is turned on, part of the chip enters cyberspace. A computer program is what is used to create parts in cyberspace and, by programming, we will be able to create cyberspace lock parts and place them inside this microcontroller. The microcontroller can be interfaced to the keypad and to the solenoid using the I/O locations in its memory map. It can then accept a combination from a human and open the lock if the combination is correct.”

“Is the cyberspace lock better than the normal lock?” Pat asked.

“In some ways it is.” I replied. “For example, the combination cannot be changed in the normal lock, but it can easily be changed in the cyberspace one. The cyberspace lock can have additional numbers added to the combination with little effort and it can even keep track of how many times the lock was opened and at what time. The cyberspace lock is also capable of having additional information added to it, like a user ID, so that it can record who is opening it. Since cyberspace holds ideas, almost any idea you can think of can be placed into this microcontroller, as long as the idea is not too big to fit!”

“Thats amazing!” said Pat “Computers are becoming more interesting to me all the time. I understand a little better now what cyberspace is but it is still kind of fuzzy to me though.”

“The only way to gain a better understanding of how cyberspace works," I said “is to learn how to program computers. Learning how to program is very hard work, but it is one of the most useful skills a person can have in our modern world.”

“The more I learn about computers” said Pat “the more I want to learn how to program them. I am definitely going to take you up on the offer you made earlier to help me. For now, though, I still don't understand what a computer operating system is or how it acts as a bridge between the physical world and cyberspace.”

Operating System: The Part Of A Computer Which Is Made From Cyberspace Parts

“Now that we have discussed both systems in general and cyberspace,” I said “we are in a good position to explain what a computer operating system is. We just saw how a normal combination lock can be made more capable and flexible by moving some of its parts into cyberspace. There are many systems in the physical world that are made more capable and flexible by having some of their parts exist in cyberspace, including automobiles, televisions, aircraft, microwave ovens, heating and cooling systems, machine tools and telephones. As we move into the future, traditional physical systems of all kinds are being redesigned to include cyberspace parts, and systems that already have cyberspace parts are being redesigned to increase their percentage of cyberspace parts. One might even imagine that all systems would be made 100% of cyberspace parts if it were possible.”

Pat thought about this for a while then asked “What type systems today have the greatest percentage of cyberspace parts?”

I smiled and said “The type of systems that currently have the greatest percentage of cyberspace parts are sophisticated computers, like PCs and servers. Usually, over half of a sophisticated computer system is built from cyberspace parts and the portion of a computer that is built from cyberspace parts is called its **operating system**!”

“That's what an operating system is!?” cried Pat “The portion of a computer that is made from cyberspace parts?”

“Yes,” I replied "this is one way to look at it".

Application Programs

“What about applications programs that run on a computer, like a word processor?” asked Pat ”aren't those part of the computer too?”

“I am making a distinction," I said “between the cyberspace parts that are needed to make a computer a complete, functioning system, and the cyberspace parts that are added to this functioning system to make it do a given kind of work. For example, when you first turn on a typical personal computer, it spends some time booting up (which means loading the operating system into memory and running it) and then you are presented with a graphical user interface or GUI. This GUI allows you to do things like move a mouse pointer around on the screen, select menus and look at the contents of a hard drive. Your computer is now a complete, functioning system but it has not been specialized for any given kind of work yet. In order to do work, you must select an application program with the mouse (like a word processor) and when this application is loaded and running, the computer can then do specialized work with it. The physical parts of a computer are called its **hardware** and the cyberspace parts of a computer, including both the operating system and application programs, is called **software**."

Computers Without Operating Systems

Pat sat quietly for a few moments then said "Earlier you said that most sophisticated computers have operating systems. Are there some computers that do not have operating systems?"

"Yes," I replied "some smaller microcontrollers do not have a separate operating system. They usually just run one dedicated program and the cyberspace components that are needed to perform tasks that a separate operating system would perform are built into the program itself. These microcontrollers usually run their dedicated program directly from ROM. Some computers with operating systems, such as cell phones and automotive computers, will also run their operating system from ROM but computers like PCs and servers load their operating systems into RAM each time they are powered up."

Back To Loading An Operating System

"I understand now that an operating system is made using cyberspace parts" said Pat "but what does an operating system actually do?"

"We will talk about what an operating system does in a later discussion," I replied "but for now, lets continue our discussion about what happens in a PC when the POST code in its BIOS ROM is nearly finished running and the last machine language instructions in the ROM tell the CPU to talk to a storage device in order to obtain the numbers that represent an operating system. Do you remember how a CPU is able to communicate with devices that are outside of itself?"

Pat looked at the model of a computer that we had been drawing on the whiteboard and replied “A CPU is able to communicate with devices outside itself using the special I/O memory locations in its memory map.”

“Yes," I said “and what makes the I/O locations special?”

Pat replied “The I/O memory locations are special because they can can have numbers copied into them and out of them by both the CPU and by a device that is outside the computer. That is why the I/O locations have a hole in them that face the CPU on one side and a hole that face the outside world on the other side.”

“Very good Pat.” I said “Now, the CPU talks to the storage device and asks if it contains numbers that represent an operating system. If the storage device replies that it does, the CPU requests that the device send the operating system's numbers into the I/O location, one number at a time, and the CPU in our model copies each of these numbers into RAM. Most of these numbers represent machine language instructions.” As I said this I pointed to the vertical bar at the bottom of the RAM section of the memory map which represented the operating system's numbers.

“After the core of the operating system has been copied into RAM," I said “the last instructions in the ROM sets the CPU's Program Counter to the beginning of the operating system's machine language instructions in RAM and then the operating system starts to run. The process of copying the operating system's numbers from a storage device into RAM, and running the core of the operating system after it has been loaded, is called **booting** the computer system. Assuming that this is a model of PC, the operating system will show a GUI (or a command line interface, like the Commodore 64 has) to the user when it is finished booting and it is then ready to run applications.”

Primary Storage And Secondary Storage

“The purpose of a storage device is to hold numbers?” Pat asked.

“Yes.” I replied. “The amount of RAM and ROM in a computer's memory map is limited and, as we talked about earlier, when the power on the computer is turned off the numbers in the RAM memory locations disappear. Therefore, in order for a computer like a PC or a server to be useful, it must have the ability to hold numbers outside its memory map for later use. The storage that is in a computer's memory map is called

primary storage and the storage that is held in devices outside of the computer is called **secondary storage** or **mass storage**. Examples of secondary storage devices include hard drives, flash drives, CDROMs, DVDs and magnetic tapes. To give you a feel for the amount of primary vs. secondary storage in a typical PC, we can take this PC on the table as an example. This PC has 1 gigabyte of RAM and 100 gigabytes of hard drive space."

"That's a big difference," said Pat "but if the hard drive can hold so many more bytes than the RAM can, why not just get rid of the RAM and use the hard drive's storage instead?"

"That is a good question, Pat." I said. "The reason that secondary storage can not be used in place of primary storage is that primary storage, like RAM, is able to have numbers copied into and out of it **much** faster than secondary storage can. Primary storage is faster than secondary storage, but it is also more expensive per byte. Both types of storage are needed, however, in a general purpose computer like a PC."

General Purpose Computers Vs. Specific Use Computers

"A PC is a general purpose computer?" Pat asked "Why is that?"

"General purpose computers," I replied "are designed to maximize their flexibility so that they can be configured as needed to perform various kinds of work. The way that a computer is configured to perform a given kind of work is with a program. Examples of programs that allow a computer to do a given kind of work include word processors, games, browsers and media players. Since a PC can easily run numerous kinds of programs, it is considered to be a general purpose computer. If the programs are large, or if more then one program is going to be running on the computer at the same time, then the amount of RAM needs to be large. If one has many programs, or the amount of data the programs use is large, then the amount of secondary storage needs to be large. The more general purpose a computer needs to be, the more RAM and secondary storage it needs."

"Does a general purpose computer also need a large amount of ROM?" Pat asked.

"No," I replied "a general purpose computer only needs enough ROM to hold instructions that test the hardware during power up, instructions that allow the operating system to more easily talk to the hardware, data that configures the motherboard and instructions that help load the operating system into RAM. In a general purpose computer, the amount of ROM in its memory map is significantly less than the amount of RAM. Specific purpose computers, on the other hand, usually have significantly more ROM than RAM."

"What is a specific purpose computer," asked Pat "and why do they have more ROM than RAM?"

"A specific purpose computer is a computer that has been designed to perform one dedicated task. Examples of specific purpose computers include computers that control automobile engines, televisions, DVD players, heating and cooling systems, audio systems, elevators and security systems. The reason that specific purpose computers usually have more ROM than RAM is that they typically only run one program, or a small number of programs. This program or programs, along with perhaps a small operating system, take up a comparatively small amount of space which can usually fit into ROM primary storage. Microcontrollers are the kind of computer system that are most widely used as specific purpose computers."

"That makes sense." said Pat "Maybe tonight I will ask my Mom if I can take her car computer apart so that I can see how much ROM and RAM it has!"

We both laughed at this!

"I would not do that on a running car, if I were you," I said "at least not until you have learned how to do some computer interfacing. In the mean time, I have some old cars in my recycle yard that have engine computers in them. You are welcome to take one of those computers apart if you would like."

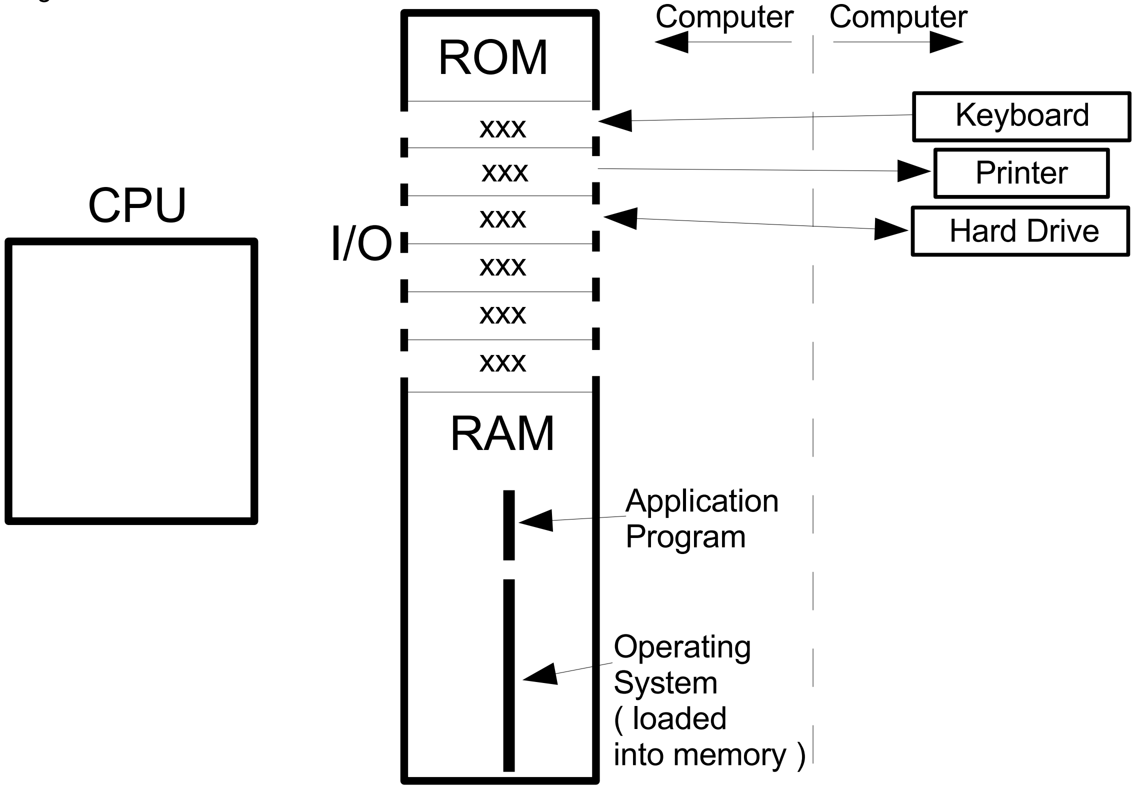
"Thanks!" said Pat "I think I will do that soon."

Running An Application Program

"Lets continue our discussion of what happens when a PC boots up." I said "We made it to the point where the operating system was loaded into RAM and was presenting a GUI, or command line interface, to the user. Can you use the model on the whiteboard to describe what happens when the user runs an application program?"

"I will try." said Pat. "If the operating system has a GUI, then the user will use the mouse pointer to double click on a program, or select it from a pull down menu. The machine language instructions in the operating system will then ask the secondary storage device that holds the program to send it one number at a time to the I/O location that the storage devices is attached to. The CPU will then take each of these numbers and place them into an unused section of RAM. After the CPU has finished copying the program into RAM, the operating system will then tell the CPU to start running the machine language instructions of this program." Pat then took a marker and added another vertical line above the line that represented the operating system. This second vertical line represented the application program after it was loaded into RAM. (see Fig. 20)

Figure 20



"That is correct Pat," I said "you seem to be learning this information rather well! One thing I have not mentioned until now is that a PC usually has an extra piece of hardware in it called a **Direct Memory Access controller** or **DMA controller** for short. This controller is able to directly copy numbers from one section of memory to another independent of the CPU. Devices like hard drives, graphic chips, sound chips and network interfaces often use a DMA controller to copy numbers to and from other parts of memory in order to free the CPU to do other work. This makes the overall computer system work faster. There are other simplifications I have made during our discussion in order to present the core ideas of how a computer works as clearly as possible. As we get deeper into how a computer works, though, I will add this more detailed information.”

Something Is Missing From The Models

“The models of a computer that we have created on the whiteboard have enough detail to show the core ideas of how most computers in the world work. As I said before, in our modern computer-based world this is extremely valuable knowledge to possess. Furthermore, as computers become applied to increasingly more aspects of our society, this knowledge will become even more valuable. What do you think of these models of how a computer works?”

“I think its amazing!” said Pat “The models explain so many things about a computer that I had no clue about before. But something still seems to be missing.”

“Oh?” I said “That is curious because the models are fairly accurate. What do you think is missing?”

“It appears to me," said Pat "that the computer spends almost all of its time copying numbers from memory into the CPU, copying numbers from the CPU into memory and doing simple mathematical operations. Beyond this, not much more seems to be happening.”

“No, you are right Pat,” I said “at its lowest levels, a computer does not do much more than this.”

“But,” said Pat “what about all of the cool things a computer can do!? Take a space game program, for example. A typical space game may have dozens of ships on the screen, all shooting lasers and missiles at each other while avoiding all kinds of spinning asteroids and space junk. Explosions and collisions are happening everywhere and the sounds from the ship's engines, the lasers, missiles, collisions and explosions are being projected from the computer's speakers. At the same time, the game is taking all kinds of quickly typed input from the keyboard and it may also be in communication with one or more other computers playing the same game over a network. How do these simple models of how a computer works explain all of the intense action that a game like this has?”

Wink Of An Eye

"I was wondering if you would notice that some critical information was missing from the models," I said " and I will try to explain what it is. There is an episode from the original Star Trek TV show, called 'Wink of an Eye', that can help explain the missing piece. In this episode the **Enterprise** is exploring an outer part of the galaxy when it receives a distress call from a nearby planet. The ship is placed into orbit around the planet and a landing party (consisting of captain Kirk, Mr. Spock, Dr. McCoy and some red shirted crew members) is beamed down to the planet. There are buildings and other signs of civilization on the planet, but no humans can be found and the landing party's instruments can not even detect any animal life. They keep hearing insect buzzing sounds, though, which is strange because there are no insects. Have you ever watched any of the original Star Trek episodes, Pat?"

"Sure," said Pat, "I have seen a number of them."

"Do you know what usually happens to red shirted crew members?" I asked.

"Something bad usually happens to them!" Pat said.

"Right," I said "something bad usually happens to them and this episode is no exception. Soon after beaming down to the planet, something bad happens to one of the red shirted crew members. In this episode, the people that had sent the distress signal (who are called Scalosians) are still on the planet, but their metabolisms have been vastly increased by radiation which was emitted from a volcano. Their metabolisms have been increased so much, in fact, that the atoms in their bodies are vibrating too quickly for the landing party to see. This faster vibration results in the Scalosians living in a faster time frame than the crew of the enterprise are. At this point I am going to deviate from the story line of this episode in order to better explain the part that is missing from our models of a computer."

"Lets assume that a red shirted crew member screamed and captain Kirk started running across the landscape to investigate. Imagine him taking two steps then freezing like a statue in a running position and the other members of the landing party also become frozen. At the same time, people suddenly appear and they seem to be acting normally. They are walking around the frozen landing party members and discussing what they should do about them. What has happened is that the perspective has been switched to the faster time frame and we are seeing the world as the Scalosians see it."

One Scalosian looks at captain Kirk and says "we can not let him reach his fallen crew member. Let us build a brick wall to stop him," and the other Scalosian agrees. Have you ever seen a brick wall built, Pat?"

"No." answered Pat.

"Brick walls can not be built by magic," I said "they must be built using very specific techniques. Lets assume that the Scalosians are going to build a brick wall that is 50 meters long, 5 meters high and 1/2 meter thick. The first thing that needs to be done is to hire a backhoe crew to dig a trench 50 meters long and make it deep enough to put the bottom of the wall below the frost line. This digging might take 2 days. Next, a cement form making crew needs to be hired and it might take them another 2 days to put together the forms in the bottom of the trench for something called a footer, which is what the bricks will sit on. A cement crew is then contracted to fill the form with cement and it might take a week for the cement to cure to the point where bricks can be placed on it. Finally, brick layers are hired to slowly and carefully assemble the wall brick-by-brick. This might take another two weeks."

"During all this time captain Kirk, in his slower time frame, has moved perhaps an inch or two. Now imagine that captain Kirk looks up and he sees... what?"

"A brick wall, suddenly appeared out of nowhere!" cried Pat "The wall was not built using magic, but to captain Kirk in his slower time frame, it looks like it was."

"Yes," I said "to captain Kirk it looks like a brick wall suddenly appeared in front of him as if by magic. This time frame difference is the part that is missing from our models of a computer. A computer is only capable of doing very simple operations (like copying numbers from memory into the CPU, copying numbers from the CPU into memory and simple mathematical operations) but it can do millions of these operations every second!"

"A computer can do millions of operations a second!?" said Pat.

"Yes," I replied "which means that computers work in a much faster time frame than humans do. Imagine that a computer can look at you from inside its screen. It looks at you and what does it see?"

"If it is running millions of times faster than we are," said Pat "then we look like statues to it."

"That is correct." I said "Imagine that this PC is watching me as I use my index finger to press the 'A' key on the keyboard." As I said this I started slowly moving my finger towards the 'A' key. "From the computer's point of view, it might take a hundred years for my finger to reach the top of the 'A' key and another 5 years to press it down enough for the key to click. At soon as the key clicks, the letter 'A' is quickly converted into a number, this number is encoded as electronic signals and these signals are sent into the computer at the speed of light."

"The computer must get very bored," said Pat "while waiting for humans to interact with it."

"I agree," I said "a computer usually spends thousands of years in its time frame waiting for humans to interact with it. This also answers your question about how computers are able to do all of the amazing things they do, including games like your space game. A computer screen is made up of little dots called **picture elements** or **pixels** for short. The computer puts each ship, asteroid, laser, and missile in your space game together pixel-by-pixel, just like the Scalosians had to put their wall together brick-by-brick. But the computer also has thousands of years in its time to do this. From our point of view, it appears that the computer is doing all of this work by magic."

I Want To Learn More About Computer Software And Hardware

Pat sat quietly for a while then said "I have enjoyed our discussion about how a computer works and now I really want to learn more about computer software and hardware. You said that you would help me learn how to program so when can we start?"

"Come back soon," I replied "and we will begin."