

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)

Copyright © 2008 by Ted Kosan

This work is licensed under the Creative Commons
Attribution-ShareAlike 3.0 License. To view a copy of
this license, visit <http://creativecommons.org/licenses/by-sa/3.0/>

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

1 The Golden Age of Personal Computers

2 One morning teacher and I were troubleshooting an electrical malfunction
3 on a friend's automobile. "Teacher," I said "When did you know what you
4 wanted to be when you grew up?" Teacher pulled the oscilloscope probe
5 away from the computer circuit it was probing, smiled and said "I have
6 been grown up for a while now and I still do not know what I want to be!
7 I can remember, however, the day I had finally saved enough money to
8 purchase my first computer. I took it home, set it up and within an
9 hour I had written my first computer program. At that point I laughed
10 and thought 'I don't know what I want to be when I grow up, but I am
11 absolutely certain I want these wonderful machines to be a part of it!'.
12

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

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

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

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



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

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

56 That Golden Age of computers occurred a while ago, however, and an
57 amazing amount has changed since then. The changes include personal
58 computers that are orders of magnitude faster than the PCs of the early
59 1980s, the Internet, the World Wide Web, Cell Phones and enormous
60 amounts of educational materials on computers and computing that is
61 expanding at an increasing rate.

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

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

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

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

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

97 If you are lucky, the machine has the operating system already installed on
98 it and, if it does not, perhaps one half to one hour of time is required to do

99 so. We will not even bring up the “fun” involved with locating and making
100 sure all of the needed device drivers are installed correctly...

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

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

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

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

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

146 Computer Technologists Must Be Motivated Self-Learners

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

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

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

168 Finally (for the first time in my life) I picked up a technical book (the
169 User's manual) and I started to actually read it... Nobody had told me to do
170 this. Nobody had assigned this work for me to do and nobody was going to
171 test me over what I had read. I started reading the book because I
172 desperately wanted to learn how to use my computer and reading the book

173 was the only way I had to do this. After reading the first few pages of the
174 book, however, a surprising thing started to happen. I became so interested
175 in what I was reading that I lost track of time and before I knew it an hour
176 had passed. During that hour I had learned how to write my first BASIC
177 program and, while it was not easy to do, there was little pain involved
178 because I was learning this information because I really wanted to.

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

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

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

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

203 Learning How To Learn Technical Subjects

204 Many of the technical books I have read in my life include a message in the
205 preface of the book that informs the student that if they do not **read the**
206 **book carefully, do the assigned problems, ask questions** and generally
207 **get actively involved**, they will not learn the subject material. I have
208 found this advice to be very true and I recommend that you follow it. In

209 addition to this sound advice, however, I am going to add some more
210 thoughts of my own that you may find helpful.

211 **Learn even the uninteresting parts**

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

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

224 **Increase your capacity for reading as much as possible**

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

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

242 **Find a quite place to study and use it**

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

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

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

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

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

266 **Minimize distractions**

267 To sacrifice means to surrender or give up something for the
268 attainment of some higher advantage or dearer object. [http://miriams-](http://miriams-well.org/Glossary/)
269 [well.org/Glossary/](http://miriams-well.org/Glossary/)

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

277 What kind of activities might you want to sacrifice? How about watching
278 television, surfing the Internet, talking to friends on the phone and (the
279 big one) playing computer and video games. During my first semester

280 attending college I nearly flunked out because I spent most of my time
281 playing computer games instead of attending class and doing my
282 assignments. A significant number of my students over the years have
283 fallen into the same trap and I have noticed that the problem is getting
284 progressively worse.

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

288 **Figure out what math is all about**

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

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

305 The Von Neumann Architecture

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

- 312 ■ CPU
- 313 ■ RAM
- 314 ■ ROM
- 315 ■ Hard drive
- 316 ■ Flash drive
- 317 ■ CDROM drive
- 318 ■ Network card
- 319 ■ Motherboard"

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

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

333 How Does A Computer Work?

334 "How does a computer work, professor?" Pat asked one day while visiting
335 me at my shop. "I have had a computer since I was a kid, all my friends

336 have computers and my parents have two of them, but computers seem like
337 magic to me because I do not really understand them."

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

341 "Cyberspace?" asked Pat "What is cyberspace and where is it?"

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

347 "Will you teach me how a computer works," asked Pat "I really want to
348 know."

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

357 "I understand" said Pat.

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

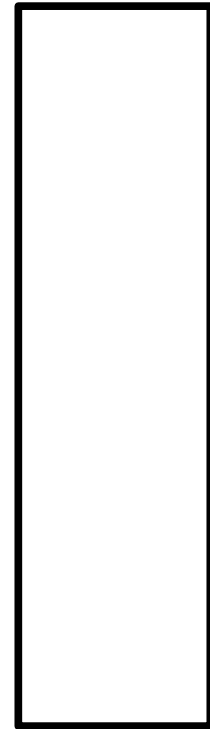


Figure 1

368 (From the movie *2001 A space Odyssey*)

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

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

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

383 “What!?” said Pat “Oh come on!”

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

385 "No" said Pat "That's ridiculous!"

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

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

396 "Von Neumann Martians"

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

399 'The Curve of Binding Energy' by John McPhee (1973, Farrar, Straus
400 and Giroux, pp. 104-105):

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

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

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

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

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

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



Figure 2

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

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

456 **255**, I said as I started writing numbers between 0 and 255 in the boxes
457 "and while the computer is running, there is never a time that a box does
458 not have a number in it. Another name for a number between 0 and 255 is
459 a **byte**. (see Fig. 3) If a number larger than 255 needs to be worked with,
460 it is spread across two or more boxes. These boxes are called **memory**
461 **locations** and this vertical rectangle is called a **memory map** because it
462 shows where the memory locations in a computer are located in relation to
463 each other. Some computers have a small amount of memory locations and
464 some computers have an enormous amount."

465 Pat studied the memory map I had drawn then said
466 "How many memory locations are there in this
467 computer?" while pointing to the computer under my
468 desk.

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

470 "Hmmm" said Pat while thinking for a few moments. "A
471 hundred?"

472 "More..."

473 "A thousand?"

474 "More..."

475 "A million !?"

476 I smiled and said "More!"

477 "A billion !!?"

242
7
199
36
227
175
175
117
255
98
22
151
0
200
48
12

Figure 3

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

484 "A billion memory locations!" cried Pat "That's a lot of numbers. How does a
485 computer keep track of which numbers are in which memory locations?"

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

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

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

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

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

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

Memory Map	
242	15
7	14
199	13
36	12
227	11
15	10
175	9
117	8
255	7
98	6
22	5
151	4
0	3
200	2
48	1
12	0

Figure 4

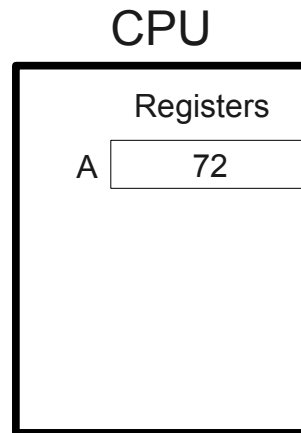
523 memory locations?"

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

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

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

Figure 5



Memory Map

242	15
7	14
199	13
36	12
227	11
175	10
175	9
117	8
255	7
98	6
22	5
151	4
0	3
200	2
48	1
12	0

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

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

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

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

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

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

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



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

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

```
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
PRINT "HELLO"
HELLO
READY.
PRINT 2+3
5
READY.

```

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

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

```
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
PRINT PEEK(0)
47
READY.
PRINT PEEK(1)
55
READY.
PRINT PEEK(2)
0
READY.

```

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

```

10 M=0
20 PRINT PEEK(M);
30 M=M+1
40 IF M <= 200 GOTO 20
RUN
55 0 178 177 145 179 0 0 0
76 0 0 4 0 0 0 0 0
22 0 2 0 0 1 0 0 0 55
105 0 0 0 0 0 0 0 0 2
8 0 60 8 0 160 0 0 160 77
20 0 8 8 0 0 0 0 0 0
55 8 55 8 255 0 0 0 0 0
76 13 184 0 60 7 53 7 0
53 8 135 0 0 101 76 0 0
84 0 2 0 92 94 0 238 1
176 2 238 123 173 19 56 201 33
156 208 96 128 79 199 33 2
0 0 0 0 14 140 60 3
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
160 48 253 64 0 0 3
READY.

```

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

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

619 **What Do The Numbers In Memory Locations Mean?**

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

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

626 “I remember” replied Pat.

627 “Well, the way it does this,” I said “is by having the number that is in any

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

634 Pat thought for a moment and then said "A red apple."

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

643 Pat thought for a little bit then said "Boat".

644 I said "We can associate the number 47 with the idea of a boat. Come up
645 with another idea."

646 Pat said "Cat".

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

649 **Contextual Meaning**

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

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

658 "Contextual what?" Asked Pat.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

707 **What Provides The Context For The Numbers In A Computer's** 708 **Memory?**

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

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

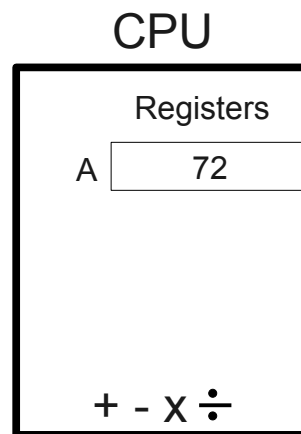
720 Pat looked at the numbers on the Commodore 64's screen a while longer

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.**"

Figure 6



Memory Map

242	15
7	14
199	13
36	12
227	11
15	10
175	9
175	8
255	7
98	6
22	5
151	4
0	3
200	2
48	1
12	0

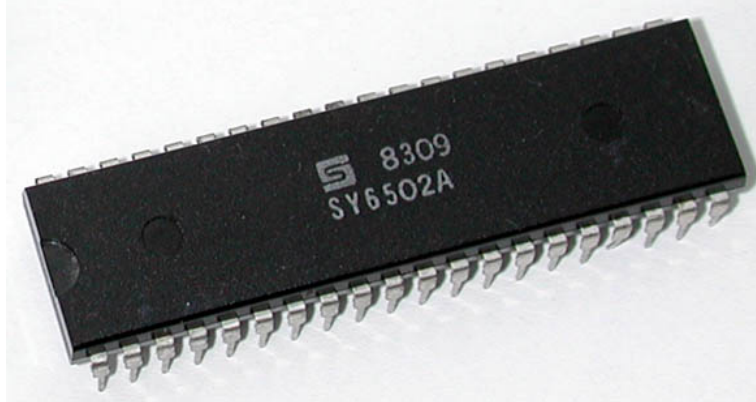
"It sounds to me like a CPU is not much more than a calculator" said Pat.

"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)

"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

758 careful to lightly touch Pat's left hand with my pinky finger before placing
759 the CPU there).

760 "The CPU in your left hand is similar to the one that is in the Commodore 64
761 and it was widely used in the personal computers of the late 1970s and
762 1980s. Its capabilities are similar to that of the calculator in your other
763 hand in that they both can do simple mathematical operations on numbers
764 and they both need to be told exactly what to do, step by step, by a human."



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

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

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

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

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

775 "What answer did you receive?" I asked.

776 "15" replied Pat.

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

778 Perhaps 10 seconds went by then Pat said "The calculator is not doing

779 anything. What are we waiting for?"

780 "Are you sure it is not doing anything?" I said.

781 "No, nothing," said Pat "am I missing something?"

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

788 "Now, 'tell' the CPU in your other hand to add $10 + 5$." I said.

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

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

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

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

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

805 I continued "Lets proceed by putting together a sequence of numbers
806 representing button presses, or **instructions**, that will tell the CPU to add
807 the numbers 10 and 5 together. The first thing we are going to need is an
808 instruction that copies a number from the memory map to a CPU register,
809 specifically register 'A'. Hmmm, we have to pick a number between 0 and
810 255 to represent this instruction, how about the number 169?"

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

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

817 **The Program Counter And The Instruction Register**

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

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

833 **Running A Machine** 834 **Language Program**

835 "Now that we have written the
836 first part of our small program,
837 and placed the extra registers
838 in the CPU needed to run it,
839 should we go ahead and run it
840 to see what it does?"

841 "Yes!" said Pat "This is fun!"

842 "It is fun, isn't it?" I said "I am
843 going to place a small button
844 next to the CPU, label it 'Run
845 One Instruction' and when it is
846 pressed, the CPU will run the
847 instruction that the PC register
848 is pointing to." I drew a small
849 pushbutton switch next to the
850 CPU then said "Ready?" (see
851 Fig. 7)

852 "Ready!" Pat replied.

853 "Okay," I said "lets go!"

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

Figure 7

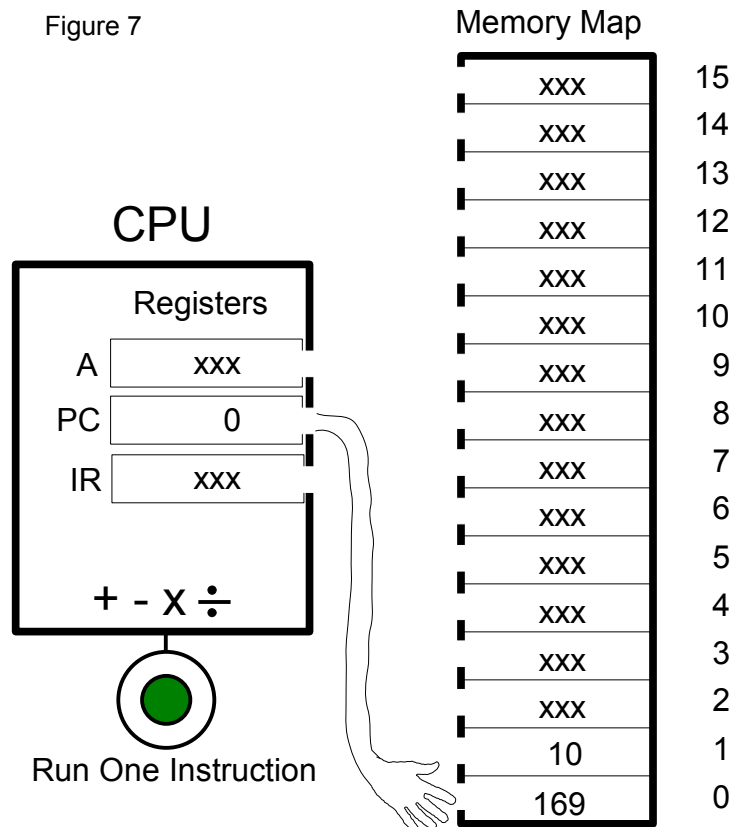
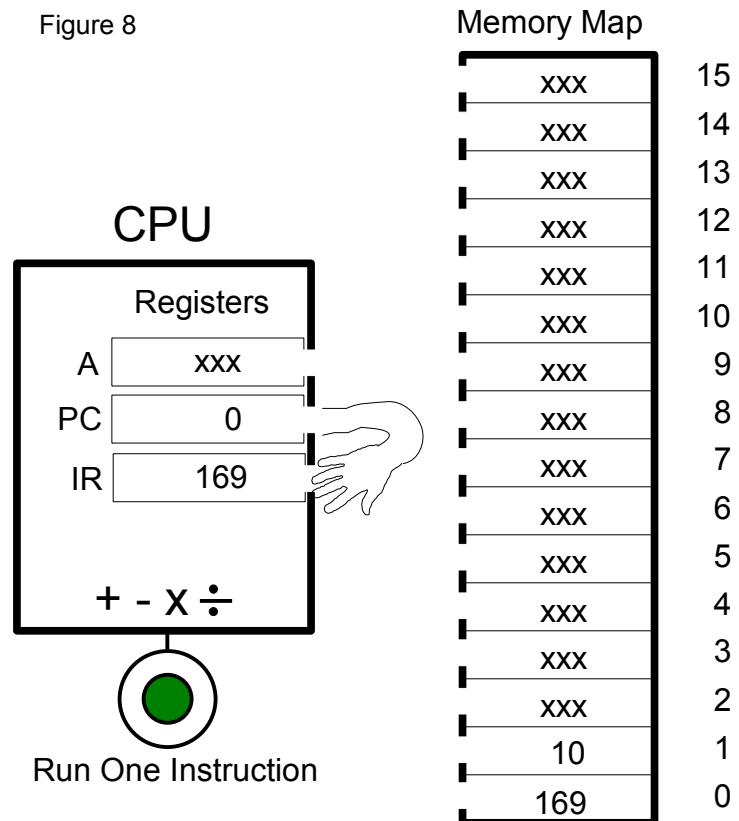


Figure 8

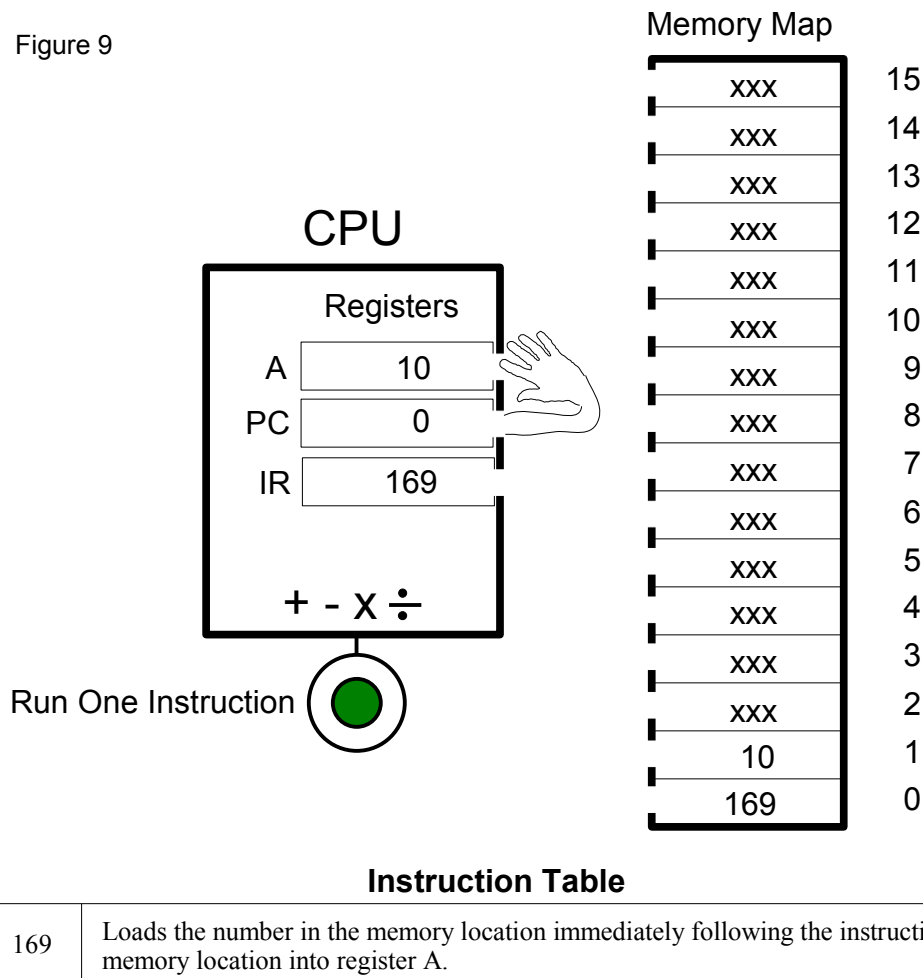


862 “The number 169, which is
 863 now in the Instruction
 864 Register, represents an
 865 instruction or **operation** that
 866 the CPU must perform. In
 867 this case, the number 169
 868 instruction means 'go to the
 869 next memory location
 870 immediately after the one
 871 that held this instruction and
 872 copy the number that is there
 873 into the A register'." (see Fig.
 874 8)

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

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

Figure 9



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

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

903 I smiled and said “I agree, lets come up with another number between 0
 904 and 255, say 105, and this will represent an addition instruction.” I then
 905 wrote the number 105 in the next row of the Instruction Table and also

906 wrote it in memory location 2 in the memory map. "How do you think this
907 addition operation should work?"

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

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

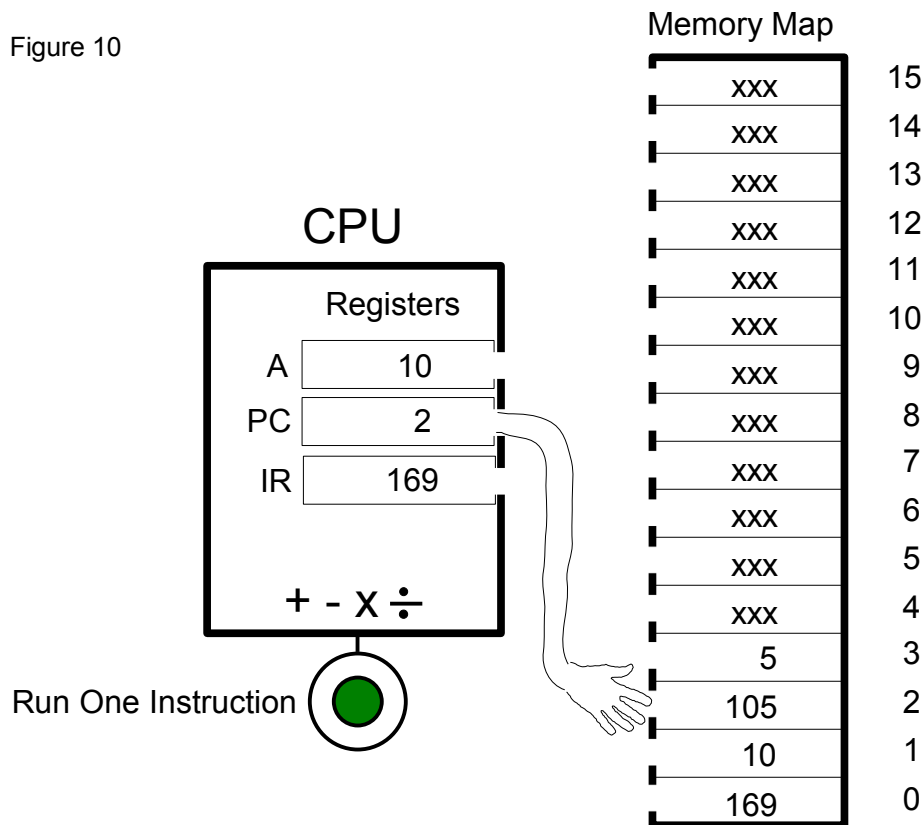
917 "Hmmm" said Pat "that is a good question. I am not sure where the answer
918 should go."

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

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

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

Figure 10



Instruction Table

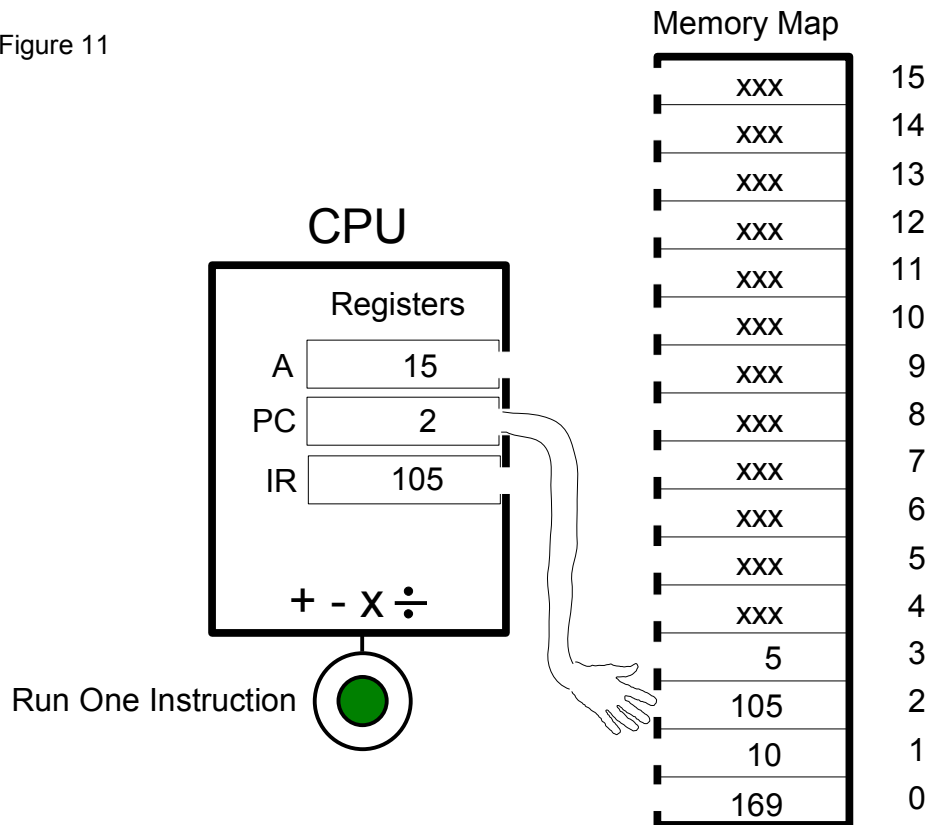
Opcode	Operation Description
169	Loads the number in the memory location immediately following the instruction's memory location into register A.
105	Adds the number that is in register A with the number in the memory location immediately following the instruction's memory location. The answer is placed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

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

936 Pat pressed the imaginary run button on the whiteboard and I proceeded.
 937 "The CPU looks at the Program Counter and sees that the next instruction
 938 that it should execute is in memory location 2 so it copies the number that is
 939 in that memory location, which is 105, into the Instruction Register." I then
 940 erased the 169 that was in the Instruction Register and replaced it with
 941 105. "The CPU then looks at the 105 that is in the Instruction Register,
 942 matches it with 105 that is in the Instruction Table and performs that
 943 operation that is associated with this opcode. The CPU then adds the 5

944 which is in memory location 3 with the 10 that is in register 'A' and then the
 945 answer 15 is placed into register 'A'. The 10 that was already in register 'A'
 946 is overwritten." I then erased the 10 that was in register 'A' and replaced it
 947 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.

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

951 "What we need now," I said "is a third instruction that copies the number
 952 that is in register 'A' to a memory location so that we can use register 'A' to
 953 do other work. Since we used a **load register 'A'** instruction to copy a

954 number from a memory location to register 'A', how about a **store register**
955 **'A'** instruction to copy a number from register 'A' to a memory location? We
956 can give it an opcode of, say, 141." I then started a new row in the
957 Instruction Table and wrote a 141 in the opcode column.

958 Mnemonics

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

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

964 "Neh-moniks," said Pat "What's that?"

965 "Mnemonics," I replied "are aids that help people remember things that are
966 difficult to remember. One example is the color bands that are on the
967 resistors you are going to sort tomorrow." I said with a smile. "Each color
968 represents a different number between 0 and 9 and the colors are **Black**,
969 **Brown**, **Red**, **Orange**, **Yellow**, **Green**, **Blue**, **Violet**, **Grey** and **White**. These
970 colors can be remembered with the phrase **Black Beetles Running On Your**
971 **Grass Bring Very Good Weather**."

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

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

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

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

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

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

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

999 I replied "I agree, computers work very differently than most people would
1000 expect. When I first learned about how a computer works, I was very
1001 surprised and also amazed that humans were capable of developing such a
1002 wonderful design. In fact, I am still amazed!"

1003 After a few moments I said "Lets finish the STA instruction. I am going to
1004 write your description of how the STA instruction works in the Instruction
1005 Table" which I did. I then asked Pat "which memory location should we tell
1006 the STA instruction to copy the number in register 'A' to?"

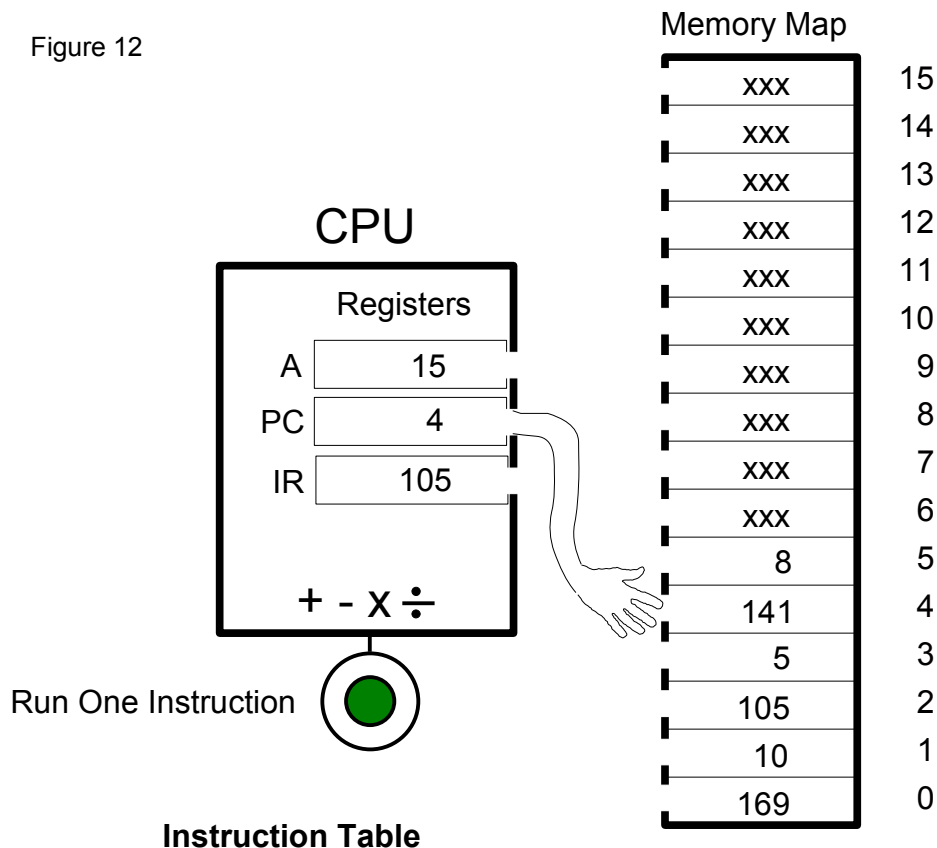
1007 Pat looked at the memory map and said "how about putting it into memory
1008 location 8?"

1009 "Okay," I replied "we will place the STA instruction's opcode, which is 141,
1010 into memory location 4 and place the address that it should write to, which
1011 is 8, into memory location 5." (see Fig. 12)

1012 "Would you like to run this last instruction Pat?" I said.

1013 "Sure" said Pat who then reached out a hand and pressed the run button on
1014 the whiteboard. "The first thing that the CPU does is to look at the Program
1015 Counter to see what the address is of the next instruction to execute. Our
1016 Program Counter contains the address 4 so it goes to memory location 4
1017 and copies the number it finds there to the Instruction Register. The
1018 number that is now in the Instruction Register is 141 and the CPU matches
1019 this number with the one in the Instruction Table to determine what
1020 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.

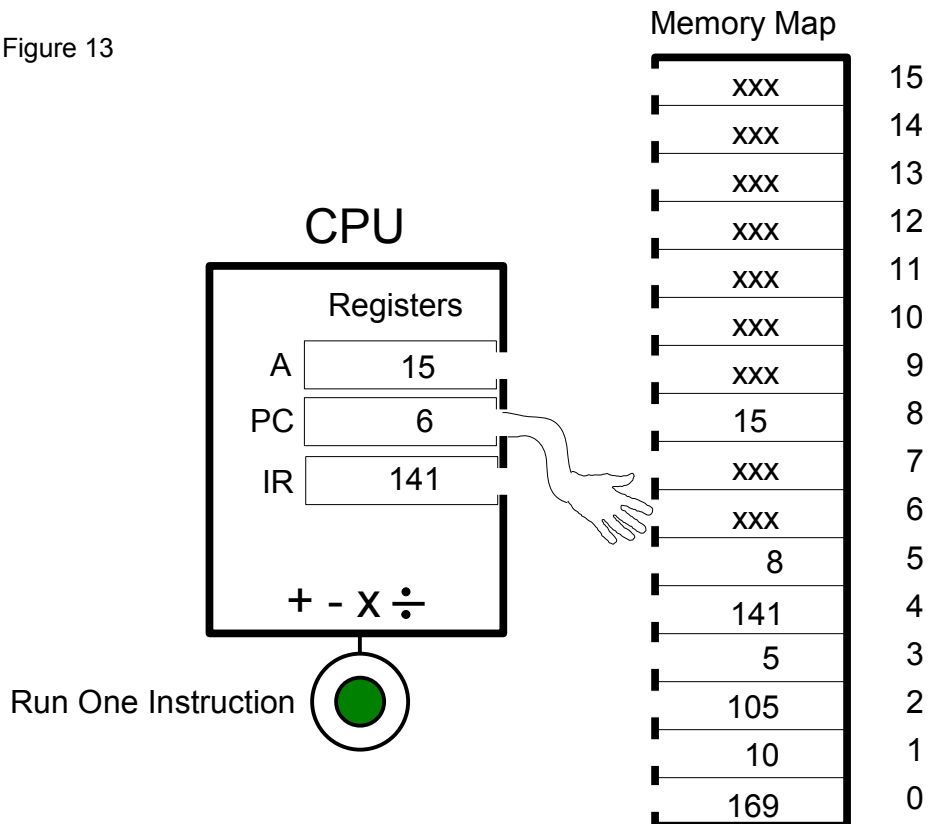
1021 tells the CPU to get the address of where it is going to store to from the
 1022 next memory location after the instruction itself. The CPU looks in this
 1023 location, which is location 5, and finds an 8 there. Finally, the CPU copies
 1024 the number which is currently in register 'A', which is 15 (our answer) to
 1025 memory location 8." Pat then picked up the marker and wrote a 15 in
 1026 memory location 8.

1027 "Very good Pat," I said "but you need to do one more thing before the
 1028 instruction is finished."

1029 Pat looked at the whiteboard for a few moments and then said "Oops, I

1030 forgot to update the Program Counter!" Pat then erased the number that
 1031 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.

1032 I smiled and said "We have successfully completed a small program, what
 1033 do you think?"

1034 Pat said "I am still fuzzy about a number of things, but I am really enjoying
 1035 this so far!"

1036 "I am glad you are enjoying this information, Pat. There are thousands of

1037 careers in the world that have this information at their core and, if you
1038 continue to study computers, perhaps you will work with them some day.
1039 What are some of the things you are fuzzy about?"

1040 Machine Language

1041 Pat pointed at the Commodore 64's screen and asked "If a CPU is
1042 programmed with opcodes, how can it also be programmed in BASIC?"

1043 "To answer that question perhaps it would be best if we went back to the
1044 early days of computers. When the first modern computers were created in
1045 the late 1940s and early 1950s, the only way they could be programmed
1046 was using opcodes. The programmers back then would create a program
1047 by drawing a memory map, like we did on the whiteboard, and then write
1048 CPU opcodes, and needed data, into the memory locations. They would
1049 then enter the series of numbers they had written into the physical
1050 computer's memory using switches and buttons. Programs that are written
1051 directly with a CPU's opcodes are called **machine language** programs."

1052 Assembly Language

1053 "The early programmers soon found out, however, that remembering what
1054 all of the opcodes did was difficult and that is when they created mnemonics
1055 for each instruction. After this, they discovered that developing programs
1056 using the mnemonics was much easier than using the CPU's opcodes and so
1057 programming evolved from using opcodes to using mnemonics. After the
1058 mnemonic version of a program was developed, the programmer would then
1059 use documentation, similar to our Instruction Table, to look up what opcode
1060 went with each mnemonic and they would then write these opcodes next to
1061 each mnemonic in their program. The mnemonic equivalent of the small
1062 program we made would look like this:

	Address	Opcode	Operand	Mnemonic & Operand
1063	000	169	010	LDA #010
1064	002	105	005	ADC #005
1065	004	141	008	STA 008

1067 "The Address column holds the beginning address of each opcode, the
1068 Opcode column holds the opcode and the Operand column contains the data
1069 an opcode may need. The Mnemonic & Operand column contains the
1070 mnemonic version of the program which the programmer writes first and
1071 then fills in the appropriate machine language numbers in the left three
1072 columns. The number sign next to the numbers 10 and 5 means that these
1073 numbers are placed in memory immediately after the instruction's opcode."

1074 The 6502 CPU's Instruction Set

1075 Pat looked at the mnemonic version of the small program we had written
1076 then asked "How many instructions does the CPU in the Commodore 64
1077 have?"

1078 "The 6510 CPU that is in the Commodore 64 is based on the 6502 CPU and
1079 they both have 56 instructions." I replied "That may seem like a large
1080 number of instructions, but most of them are as simple as the LDA, ADC and
1081 STA instructions we have been working with. Lets do an Internet search
1082 and find the complete list of instructions that the CPU in the Commodore 64
1083 uses." I did this and found the following list:

1084	ADC	ADD memory to accumulator with Carry.
1085	AND	AND memory with accumulator.
1086	ASL	Arithmetic Shift Left one bit.
1087	BCC	Branch on Carry Clear.
1088	BCS	Branch on Carry Set.
1089	BEQ	Branch on result EQUAL to zero.
1090	BIT	test BITS in accumulator with memory.
1091	BMI	Branch on result MINus.
1092	BNE	Branch on result Not Equal to zero.
1093	BPL	Branch on result PLUS).
1094	BRK	force Break.
1095	BVC	Branch on oVerflow flag Clear.
1096	BVS	Branch on oVerflow flag Set.
1097	CLC	CLear Carry flag.
1098	CLD	CLear Decimal mode.
1099	CLI	CLear Interrupt disable flag.
1100	CLV	CLear oVerflow flag.
1101	CMP	CoMPare memory and accumulator.
1102	CPX	ComPare memory and index X.
1103	CPY	ComPare memory and index Y.
1104	DEC	DECrement memory by one.
1105	DEX	DEcrement register S by one.
1106	DEY	DEcrement register Y by one.
1107	EOR	Exclusive OR memory with accumulator.
1108	INC	INCrement memory by one.
1109	INX	INcrement register X by one.
1110	INY	INcrement register Y by one.
1111	JMP	JuMP to new memory location.
1112	JSR	Jump to SubRoutine.
1113	LDA	LoaD Accumulator from memory.
1114	LDX	LoaD X register from memory.
1115	LDY	LoaD Y register from memory.
1116	LSR	Logical Shift Right one bit.
1117	NOP	No OPeration.
1118	ORA	OR memory with Accumulator.
1119	PHA	PusH Accumulator on stack.

1120 PHP PuSH Processor status on stack.
1121 PLA PuLl Accumulator from stack.
1122 PLP PuLl Processor status from stack.
1123 ROL ROtate Left one bit.
1124 ROR ROtate Right one bit.
1125 RTI ReTurn from Interrupt.
1126 RTS ReTurn from Subroutine.
1127 SBC SuBtract with Carry.
1128 SEC SEt Carry flag.
1129 SED SEt Decimal mode.
1130 SEI SEt Interrupt disable flag.
1131 STA STore Accumulator in memory.
1132 STX STore Register X in memory.
1133 STY STore Register Y in memory.
1134 TAX Transfer Accumulator to register X.
1135 TAY Transfer Accumulator to register Y.
1136 TSX Transfer Stack pointer to register X.
1137 TXA Transfer register X to Accumulator.
1138 TXS Transfer register X to Stack pointer.
1139 TYA Transfer register Y to Accumulator.

1140 “Look at all of those instructions!” said Pat “That would sure take a lot of
1141 time to look up the opcodes for all of them after the mnemonic version of a
1142 program was finished. Hmm, couldn't the mnemonic version of a program
1143 be given to the computer so that it could do the opcode lookup
1144 automatically?”

1145 “Yes it could,” I replied “and this is what the early programmers thought of
1146 too!” The type of program they developed to do this is called an **assembler**
1147 and what it does is take the mnemonic version of a program and convert it
1148 into its machine language equivalent. The name they then gave the
1149 mnemonic version of a program is **assembly language** and it is the **source**
1150 **code** that the assembler takes as its input information. Very few
1151 programmers develop programs in machine language today, but a number
1152 still write programs in assembly language.”

1153 “Will the machine language for one CPU run on another CPU?” Pat asked.

1154 “That depends on a number of things that we will not get into now, but the
1155 short answer is that if the second CPU is the same model, or in the same
1156 'family', as the first CPU then it would. If the second CPU is a different
1157 model, or in a different CPU 'family', then no it wouldn't. For example, the
1158 assembly language for the 6510 CPU, which is the CPU that the
1159 Commodore 64 contains, will not run on an x86 family processor which
1160 most personal computers use.”

1161 **Low Level Languages And High Level Languages**

1162 “To get back to your question about how a computer can be programmed in
1163 machine language and in BASIC, one has to understand that even though
1164 assembly language was easier to use than machine language, it was still
1165 somewhat difficult for humans to develop programs with.

1166 The early programmers wanted to develop programs in a language that was
1167 more like a human language, English for example, than the machine
1168 language that CPUs understand. Both machine language and assembly
1169 language are considered to be **low level languages** because the thing that
1170 gives the numbers in these languages their contextual meaning is the CPU's
1171 hardware. Programmers wanted to work with computer languages that
1172 have much of their contextual meaning derived from human languages so
1173 that the ideas that the programs worked with were more natural for humans
1174 to use. They then figured out ways to use the low level languages they
1175 could already program in to create the **high level languages** that they
1176 wanted to program in.

1177 “This is when languages like FORTRAN (in 1957), ALGOL (in 1958), LISP
1178 (in 1959), COBOL (in 1960), BASIC (in 1964) and C (1972) were
1179 created. Ultimately, a CPU is only capable of understanding machine
1180 language and, just like assembly language needs to be converted to
1181 machine language before a CPU can understand it, so it is with all
1182 computer languages.”

1183 **Compilers And Interpreters**

1184 “How is a high level language converted into machine language?” asked
1185 Pat.

1186 “There are two types of programs that are commonly used to convert a
1187 higher level language into machine language.” I replied. “The first kind of
1188 program is called a **compiler** and it takes a high-level language's source
1189 code (which is usually in typed form) as its input and converts it into
1190 machine language. After the machine language equivalent of the source
1191 code has been generated, it can be loaded into a computer's memory and
1192 run. The compiled version of a program can also be saved on a storage
1193 device and loaded into a computer's memory whenever it is needed.”

1194 The second type of program that is commonly used to convert a high-level
1195 language into machine language is called an **interpreter**. Instead of
1196 converting source code into machine language like a compiler does, an
1197 interpreter reads the source code (usually one line at a time), determines

1198 what actions this line of source code is suppose to accomplish, and then it
1199 performs these actions. It then looks at the next line of source code
1200 underneath the one it just finished interpreting, it determines what actions
1201 this next line of code wants done, it performs these actions, and so on."

1202 "An example of an interpreter is the BASIC interpreter that is in the
1203 Commodore 64. When we typed in the line of BASIC code that asked the
1204 Commodore to print the contents of a memory location, and pressed the
1205 Return key, the Commodore's BASIC interpreter read the line we typed,
1206 determined which memory location we wanted to see the contents of, and
1207 then printed this number to the screen."

1208 "How many computer languages are there?" asked Pat.

1209 "Thousands of computer languages have been created since the 1940's," I
1210 replied "but there are currently around 2 to 3 hundred historically
1211 important languages. Lets see if we can find a list of them." I brought up a
1212 browser on my PC, did a search on 'computer languages' and located a page
1213 that listed the historically important ones." (
1214 http://en.wikipedia.org/wiki/Timeline_of_programming_languages)

1215 **The Three Types Of Computer Memory**

1216 Pat looked at the list of historically important computer languages for a
1217 while then asked "Earlier you said that a compiled program can be stored
1218 for later use. I know that a computer usually stores programs on its 'hard
1219 drive' but where does something like a hard drive fit into this model of a
1220 computer that is on the whiteboard?"

1221 "Now that we have gone through the work of figuring out how a computer
1222 operates at its lowest levels," I replied "it is easier to explain how devices
1223 like hard drives are attached to one. Instead of using the detailed model of
1224 a computer that we have developed on this whiteboard, though, I am going
1225 to draw a similar diagram that is more general."

1226 I picked up a blank whiteboard and started whistling the theme to '2001 A
1227 Space Odyssey' again as I drew another memory map. Instead of drawing
1228 the individual memory locations, however, I left the memory map unfilled
1229 but still labeled it 'memory map' at the top. I also drew an empty square to
1230 the left of the memory map and labeled it 'CPU'."

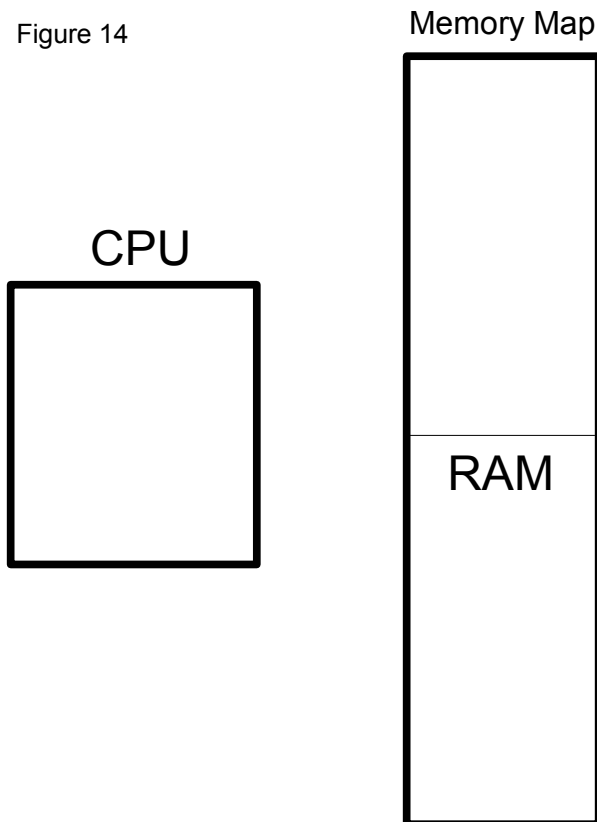
1231 **RAM (Random Access Memory)**

1232 I then asked Pat "What does the word RAM mean to you?"

1233 Pat thought for a few moments then replied "I think RAM has something to
1234 do with how much memory a computer has. I know that my Mom's
1235 computer did not have enough RAM to run a new program she bought so
1236 she had a friend add more RAM to it."

1237 "That is correct," I said "**RAM** is one of three types of memory that can be
1238 present in a memory map. RAM stands for Random Access Memory but a
1239 better name for it would have been RWM or Read Write Memory because
1240 numbers can be both copied into this kind of memory and copied out of it.
1241 All the numbers in RAM memory locations will keep whatever numbers they

Figure 14



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 loses the numbers it contains when the power is 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)

1264 "As long as a computer is powered
1265 up," I continued "every memory location will always contain a number
1266 between 0 and 255. There is no such thing as a blank memory location
1267 when the power is on. When the power is off, however, all of the RAM
1268 memory locations are blank. When the computer is first turned on, each
1269 RAM memory location has a number between 0 and 255 randomly appear in
1270 it during the time that the system's power rises to its operating level."

1271 “Since these RAM memory locations come up with random numbers in
1272 them, there is no contextual meaning associated with these numbers so they
1273 do not hold any meaningful information. Computer programmers
1274 sometimes say that memory locations that do not have any contextual
1275 meaning associated with them contain **garbage**. After a computer has gone
1276 through its power-up cycle, its RAM memory locations are ready to have
1277 numbers copied into these locations that have contextual meaning
1278 associated with them. The numbers that represent machine language
1279 programs have contextual meaning associated with them and an example of
1280 this was the small machine language program we developed a little while
1281 ago.”

1282 “But now we have a problem,” I said “because when the power-up cycle on
1283 a computer is finished, a small electronic circuit senses this then sends a
1284 signal to the CPU that says 'the power is on now, start running!' **Most**
1285 **CPUs have an address built into them at the factory which is the**
1286 **address in the memory map where they should look for their first**
1287 **machine language instruction immediately after power-up.** In the
1288 Commodore 64, this address is 65532.”

1289 “If a machine language instruction has not been purposefully placed into
1290 this memory location, the computer will lock up and everyone has had a lot
1291 of experience with their computers locking up!”

1292 Pat laughed and said “Oh yes! My computer locks up all the time!”

1293 I smiled and continued “After this first machine language instruction has
1294 been executed, the Program Counter is set to the next machine language
1295 instruction in the sequence, it is then executed and so on.” This next part
1296 was important so I dropped the level of my voice a little and said “**if there**
1297 **is ever an instant in time when the CPU is ready to execute a**
1298 **machine language instruction, and the number it pulls from the**
1299 **memory location that the Program Counter is pointing to is not part**
1300 **of the program that is running, the computer will also lock up...**
1301 Most of the time that a computer locks up, this is the cause.”

1302 “You mean something as simple as that can lock up a computer?” Pat said
1303 “Why is that?”

1304 **A CPU Is A Very Dumb Device**

1305 “Do you remember when I said earlier that a CPU was one of the dumbest
1306 things in the world?” I asked.

1307 “Yes” said Pat “it was when we were talking about the CPU being like a
1308 simple calculator.”

1309 “The reason that a CPU is so stupid,” I continued “is that it needs to be told
1310 exactly what to do, step by step, the whole time it is running. In order to
1311 get a feel for how stupid this is, imagine that you had to be told exactly
1312 what to do, step by step, from the time you woke up in the morning until the
1313 time you went to sleep at night. Your instructions might look something like
1314 this:

- 1315 1) Open your left eye.
- 1316 2) Open your right eye.
- 1317 3) Take your left hand and pull your covers down until they are below
1318 your feet.
- 1319 4) Turn your whole body 90 degrees so that your legs are hanging off
1320 the side of the bed.
- 1321 5) Place your left foot on the floor.
- 1322 6) Place your right foot on the floor.
- 1323 7) Raise your back 90 degrees so that you are sitting straight up.
- 1324 8) Put your left hand on the edge of the bed.
- 1325 9) Put your right hand on the edge of the bed.
- 1326 10) Push yourself up with your arms into a standing position...”

1327 As I said these instructions, I acted some of them out and Pat began
1328 laughing.

1329 “You see,” I said “this is pretty stupid. Now imagine that your instructions
1330 were suppose to say 'turn left 45 degrees. Walk forward 8 steps', but
1331 instead they said 'turn right 180 degrees. Walk forward 1000 steps'. These
1332 look like legitimate instructions but they are really garbage instructions
1333 because they told you to turn around and face your bed then walk forward
1334 1000 steps!”

1335 “A similar thing can happen with a computer. Through a programming
1336 error, a machine language instruction (or data for an instruction) can be
1337 placed into a program that does not mean anything in the context of the
1338 program. This can cause the computer to attempt to do something just as
1339 silly as you trying to walk through your bed. Once a garbage instruction
1340 has been executed, the CPU usually loses track of where it was suppose to

1341 be in the program and it continues to execute garbage instructions in
1342 memory until somebody pushes the reset button. Do you see now how easy
1343 it can be to lock up a computer, Pat?"

1344 "Yes" said Pat "In fact, I was thinking that it seems so easy to lock up a
1345 computer that it is a wonder that they do not lock up more often than they
1346 do."

1347 "I agree," I said "and if your PC locks up, you just need to restart it. If the
1348 engine computer on something like a passenger jet locks up, however, that
1349 could be a big problem!"

1350 "Wow" said Pat "I wouldn't want to be on a passenger jet if that happened!"
1351 Pat looked at the ceiling, thought for a few moments then said "I hear about
1352 PC's locking up all the time, but I have never heard about the engine
1353 computer on a passenger jet, or even a car, locking up. How come one kind
1354 of computer locks up a lot, but other kinds don't lock up very much at all?"

1355 "That is a difficult question to answer completely at this point in our
1356 discussion." I replied. "If you ever take me up on my offer to help you to
1357 learn how to program a computer, though, ask this question again and I will
1358 try to explain it to you."

1359 "Okay" said Pat.

1360 "Now I have a question for you." I said "If all the RAM memory locations in
1361 a computer come up with 'garbage' numbers in them, where in memory
1362 does the CPU go to get its first instructions when it first powers up?"

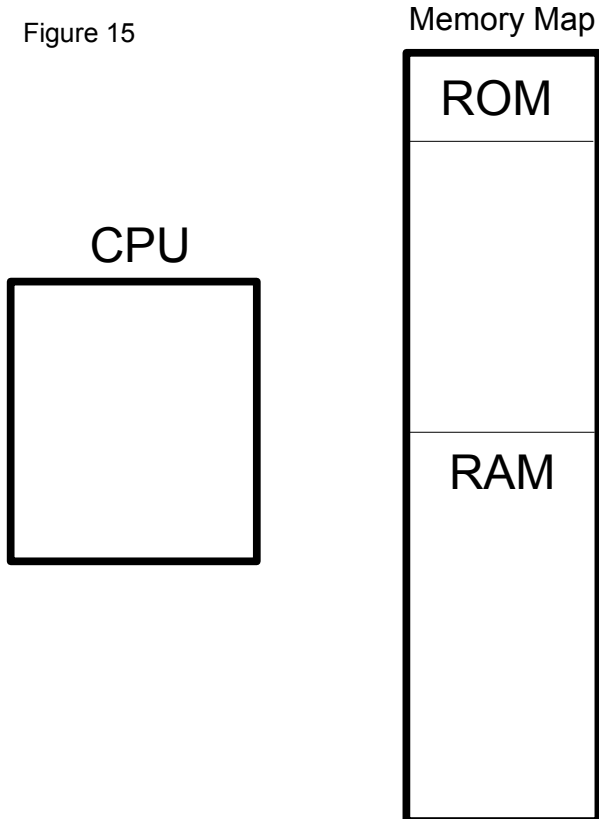
1363 "Hmmm" said Pat, while looking at the memory map. "It can't get its first
1364 instructions from RAM because RAM contains garbage data in it right after
1365 it powers up. It seems that we need a kind of memory that remembers its
1366 numbers even when the power is off. You had said that there are three
1367 basic types of memory in a memory map, does one of the other two types
1368 work like this?"

1369 **ROM (Read Only Memory)**

1370 "Yes," I said "very good! One of the other two types of memory in a memory
1371 map is called **ROM** memory and it stands for **Read Only Memory**. Another
1372 name for this memory is **non-volatile** memory. The name ROM fits this
1373 type of memory a little better than RAM's name does because the numbers

1374 in this second type of memory are meant to mostly be copied, or read, from.
1375 The special thing about ROM memory is that after numbers have been
1376 placed into it, they will be held there even after the power is turned off.” As
1377 I was saying this, I drew a second horizontal line about one eighth of the
1378 way down from the top of the memory map then labeled the topmost
1379 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

1401 reprogrammed 'in the field' (which means where they are being used) lead
1402 to the development of a chip called a **PROM** which is a **Programmable**
1403 **Read Only Memory**. These chips had little patterns of fuses in them that
1404 would be burned when the devices were programmed. They were more
1405 flexible than the early ROMs but the disadvantage of these chips was that
1406 they could only be programmed once.”

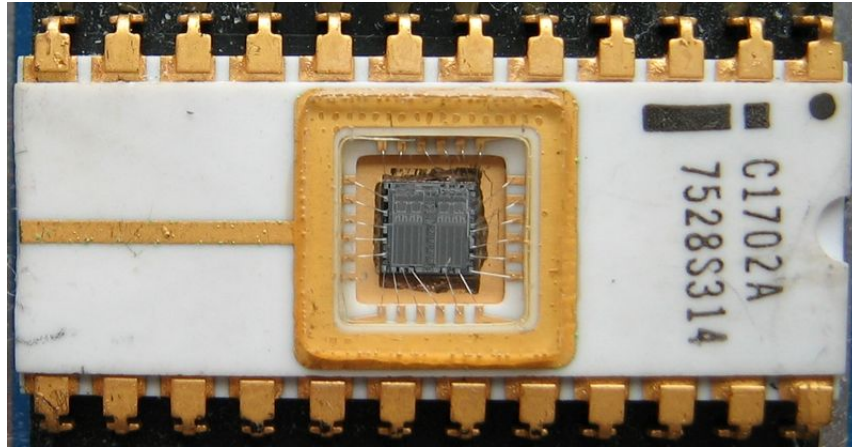
1407 The need to have a ROM that could be reprogrammed many times lead to
1408 the development of the **EPROM**, which stands for **Erasable**
1409 **Programmable Read Only Memory**. EPROMs can have programs placed
1410 into them by anyone having a device called an EPROM programmer. What
1411 is even more interesting is that these chips have a small round window on

1412 their top that allows light to shine into them. If ultra violet (or UV) light is
1413 shined into this window for perhaps 10 minutes, the numbers that were last
1414 programmed into the chip are erased by this light.”

1415 “Aside from their use as ROMs, EPROMs are a very interesting kind of
1416 computer chip to have because they allow people to see what the inside of a
1417 chip looks like. I have some old EPROMs around here somewhere, would
1418 you like me to give you one?”

1419 “Yes please!” said Pat

1420 So I searched through my collection of electronic parts until I found an
1421 EPROM which I then gave to Pat.



1422 As I handed Pat the EPROM chip, I was again careful to lightly touch Pat's
1423 hand with my pinky before I placed the chip into it.

1424 **ESD (Electro Static Discharge)**

1425 “Why do you keep touching my hand with your pinky finger before you give
1426 me a chip?” Pat asked.

1427 I replied “Have you ever walked across a carpeted room during the Winter,
1428 reached out your hand to open a door and then received a shock of static
1429 electricity from the metal door knob?”

1430 “Oh yes.” answered Pat “Sometimes I have even seen a blue spark jump
1431 between my hand an the door knob and those shocks really hurt!”

1432 “Believe it or not,” I said “little sparks like that often move between your
1433 fingers and the things you touch even if you cannot feel them. Another
1434 name for these sparks is **ESD** or **Electro Static Discharge** and it is caused
1435 by static electricity. Most of the time ESD sparks do not cause any harm,
1436 but if you allow sparks like that to hit a computer chip, the chip can easily
1437 be damaged. I have a couple of stories about ESD and computer chips that
1438 you may find interesting.”

1439 “The first story happened when I was younger and working for a company
1440 called Tire Tele as an electronics technician. Tire Tele manufactured low
1441 tire pressure warning systems for automobiles and these systems consisted
1442 of sensor units, which were placed inside each tire of an automobile, and a
1443 receiver which was placed on the dash board. The sensor units would
1444 periodically send pressure information to the receiver and, if any of the tires
1445 was losing pressure, the receiver would alert the driver.”

1446 “Tire Tele began selling their product to people and everything was going
1447 fine. Then, about 6 months after the first units were sold, they started to
1448 fail and people began returning them for repair or for a refund. The Tire
1449 Tele engineers determined that the computer chips in these devices were
1450 failing and so they sent a few of the dead chips back to the chip
1451 manufacturer for analysis. The chip manufacturer disassembled the chips,
1452 looked at them under a high power microscope and discovered that the little
1453 electronic circuits in the chip were being damaged by ESD.”

1454 “The chip manufacturer sent some of their engineers to the Tire Tele plant
1455 to observe how the units were being assembled and they discovered that
1456 none of the people on the assembly line were using anti-static protection
1457 devices or procedures. One anti-static procedure that all people who work
1458 with computer chips use is to make skin-to-skin contact with a person
1459 before handing a computer chip to them. The skin-to-skin contact allows
1460 the static electricity level between the two people to equalize which will
1461 prevent an ESD spark from traveling into the computer chip when it is
1462 handed over. As soon as anti-static equipment and procedures were put
1463 into place in the Tire Tele facility, their ESD problems disappeared.”

1464 “Who would have thought that such a small thing as little sparks could
1465 cause such a problem?” said Pat “What is your second story?”

1466 “The second story happened when I was visiting a High School,” I replied
1467 “in order to demonstrate a computer interface board I had built. I was
1468 placed in a large carpeted room, along with other people who were

1469 demonstrating things to the students, and the carpet caused a great deal of
1470 static electricity to accumulate in the room. The computer interface board I
1471 had made contained a speech synthesis chip on it that would take numbers
1472 as input and turn these numbers into various words."

1473 "I had written a program that made the chip recite the letters of the
1474 alphabet over and over again. The computer would say 'A, B, C, D...' in a
1475 mechanical voice that sounded like a robot. As students would come to my
1476 display, I would point to each chip on the board, explain what it did and I
1477 would end by saying 'this last chip allows the computer to talk'. About half
1478 way through the day, a group of students came to my table, I went through
1479 my explanations and, as I pointed at the speech chip, a huge blue spark
1480 jumped from the end of my finger into the chip and it immediately went
1481 from saying 'A, B, C, D' to mumbling 'MWA BLA VLAZ DAUP'!. That chip
1482 never did work correctly again!"

1483 Pat started laughing and so did I! "It wasn't very funny at the time," I said
1484 "but it certainly seems funny now!"

1485 "Anyway, that is an EPROM that you have in your hand and after they were
1486 invented in 1971, computer development in general moved forward at a
1487 quicker pace because of the shorter time it took to reprogram these devices.
1488 Even though the EPROM was a very useful device, it was still somewhat
1489 difficult to work with because it needed to be placed in a UV eraser before it
1490 could be reprogrammed. This led to the **EEPROM**, or **Electrically**
1491 **Erasable Programmable Read Only Memory**, being developed in 1981.
1492 Instead of UV light being needed to erase these devices, they could be
1493 erased and reprogrammed electronically one memory location at a time"

1494 "One of the more recent types of ROM memory chips is called **Flash**
1495 memory and it also can be reprogrammed electronically. Unlike EEPROMs,
1496 however, Flash memory has to be reprogrammed in blocks of memory
1497 locations but, since it is less expensive to make than EEPROM memory, it
1498 has become very popular where large amounts of storage are needed.
1499 Flash memory is not only used in personal computers, it is also used as
1500 storage memory for digital audio players, USB drives, mobile phones and
1501 digital cameras."

1502 "I have an MP3 player" said Pat "if it has Flash ROM in it, does this mean
1503 that the player has a computer in it?"

1504 "There is a good chance that it does," I said "and if it has a computer in it,

1505 then that computer is going to work in a similar manner to the models of a
1506 computer that we have been drawing on the whiteboards. Once you
1507 understand how this model works, you understand how most of the
1508 computers in the world work. That is very powerful knowledge to have.”

1509 “Amazing!” said Pat “I feel like I am stepping into a whole new world! I had
1510 never thought too much about computers before, but now that I am starting
1511 to see how they work, I want to know more about them.” after a pause, Pat
1512 continued “I am beginning to understand how numbers can be placed into
1513 the various ROM memories. What I want to know now is what the numbers
1514 usually mean that are put into these ROMs.”

1515 **BIOS And POST**

1516 “Lets go back then,” I said “to the question I asked you about what happens
1517 when a computer first powers up. I asked 'If all the RAM memory locations
1518 in a computer come up with 'garbage' numbers in them, where in memory
1519 does the CPU go to get its first instructions when it first powers up?' We
1520 determined that some type of ROM chip needs to be placed into the section
1521 of memory that contains the address that a CPU first goes to when it is
1522 turned on. A machine language program is placed into this ROM chip and
1523 the machine language program usually tells the CPU to check the various
1524 parts of the computer system to make sure they are operating correctly.”

1525 “On a typical PC, the ROM that is placed in the part of memory that the
1526 CPU first looks at for its initial instructions is called the **BIOS** or **Basic**
1527 **Input Output System**. The part of a PC's BIOS that tells the CPU to check
1528 the computer system for correct operation is called the **POST** or **Power On**
1529 **Self Test** code. When you first turn on your PC, Pat, what kinds of things
1530 do you notice?”

1531 “Well” said Pat “the first thing that happens is that the screen flashes on
1532 and changing numbers are then shown at the upper left of the screen. After
1533 this, the lights on my keyboard blink, my hard drive starts making noise and
1534 then a little later my graphic desktop is shown.”

1535 “These are all a result of the machine language POST code telling your CPU
1536 to check each of these devices.” I said “The changing numbers are shown as
1537 the CPU checks the system's RAM chips, the keyboard lights are blinked
1538 when it is checked and the hard drive makes noise when it is checked.
1539 There are many more devices in the PC that the POST code also checks but
1540 these do not make noise nor do they flash lights or print to the screen.”

1541 “This should answer your question about the meaning of the numbers that
1542 are typically placed into ROM memory. A significant amount of these
1543 numbers represent a machine language program that tests the computer
1544 system when it is first turned on. Other parts of the same ROM also usually
1545 contain machine language code that controls various pieces of hardware
1546 that are attached to the system. We will talk about this other kind of code
1547 later.”

1548 “For now we have another a more pressing problem. The amount of ROM
1549 in a PC is usually much smaller than the amount of RAM it has. After the
1550 CPU has finished executing the POST code in the BIOS ROM, it needs more
1551 machine language instructions to run. The BIOS ROM, however (being
1552 relatively small) has very little room for extra instructions. The CPU's
1553 Program Counter could be reset back to the beginning of the ROM and the
1554 POST code could be re-executed, but this would result in the CPU re-
1555 executing the POST code over and over again and the computer could not
1556 be used to do any useful work.”

1557 “After the POST code, there is only room for a small number of final
1558 instructions for the CPU and, if it cannot find any more instructions, its
1559 Program Counter will run off the end of the ROM memory into garbage
1560 memory and the numbers in the garbage memory will quickly lock the CPU
1561 up. Therefore, the remaining instructions in the ROM should be used to tell
1562 the CPU where to find more instructions, but where is it going to get them
1563 from Pat?”

1564 Pat studied the memory map for a while then said “The CPU can't get more
1565 machine language instructions from RAM because the computer was just
1566 turned on and all the RAM locations contain garbage numbers. It also can't
1567 get more machine language instructions from the ROM because the ROM is
1568 fairly small and it has already used most of the instructions in there.
1569 Hmm, compiled programs consist of numbers that represent machine
1570 language instructions and, from what I know, the programs that a PC can
1571 run are stored on its hard drive. My guess is that the CPU can get the
1572 machine language instructions it needs from the programs on its hard
1573 drive.”

1574 “You are right.” I said “After it has finished running its POST code, a PC
1575 usually obtains the machine language instructions it needs from its hard
1576 drive. But how does the PC's CPU talk to a hard drive? We have not placed
1577 a hard drive into our whiteboard model of a computer yet, where do you

1578 think it should go?"

1579 **I/O Memory**

1580 Pat looked at the whiteboard model while thinking about this then said "I
1581 am not sure where it should go. Earlier, though, you said that there were 3
1582 kinds of memory in a computer and we have only talked about two of them,
1583 which are RAM and ROM. Maybe the hard drive is attached to this third
1584 kind of memory."

1585 "That is a good guess," I said "the hard drive in a computer is attached to
1586 the third kind of memory." As I said this I pointed at the whiteboard to the
1587 the area of the memory that was between the ROM memory and the RAM
1588 memory. "As with RAM and ROM, this third kind of memory, which is
1589 called Input/Output (or I/O) memory, also consists of memory locations
1590 that can hold a number between 0 and 255." As I was saying this, I drew
1591 evenly spaced horizontal lines in the I/O memory part of the memory map to
1592 represent its memory locations. I then erased a little opening on the left
1593 side of each of these I/O memory locations. "Notice that I have put an
1594 opening in the left side of each of these memory locations to show that the
1595 CPU has access to each one of them, just like it does with the RAM and
1596 ROM locations."

1597 "Instead of starting with a hard drive, though, lets see how something
1598 simpler, like a keyboard, is attached to a computer. The first thing we need
1599 to do is to show on our model what is 'inside' of the computer and what is
1600 'outside' of it. By 'inside' and 'outside' I do not mean inside and outside the
1601 box that the PC is in. I mean what is included in the core part of the
1602 computer system and what is outside of this core." I then drew a vertical
1603 dashed line to the right of the memory map and said "Everything to the left
1604 of this vertical dashed line can be considered to be inside the core of the
1605 computer and everything to the right of it is outside."

1606 I then drew a small horizontal rectangle to the right of the dashed line and
1607 wrote the word 'Keyboard' inside of it. Finally, I pointed at this rectangular
1608 model of a keyboard and said "when you press a key on a keyboard, Pat,
1609 what do you think happens?"

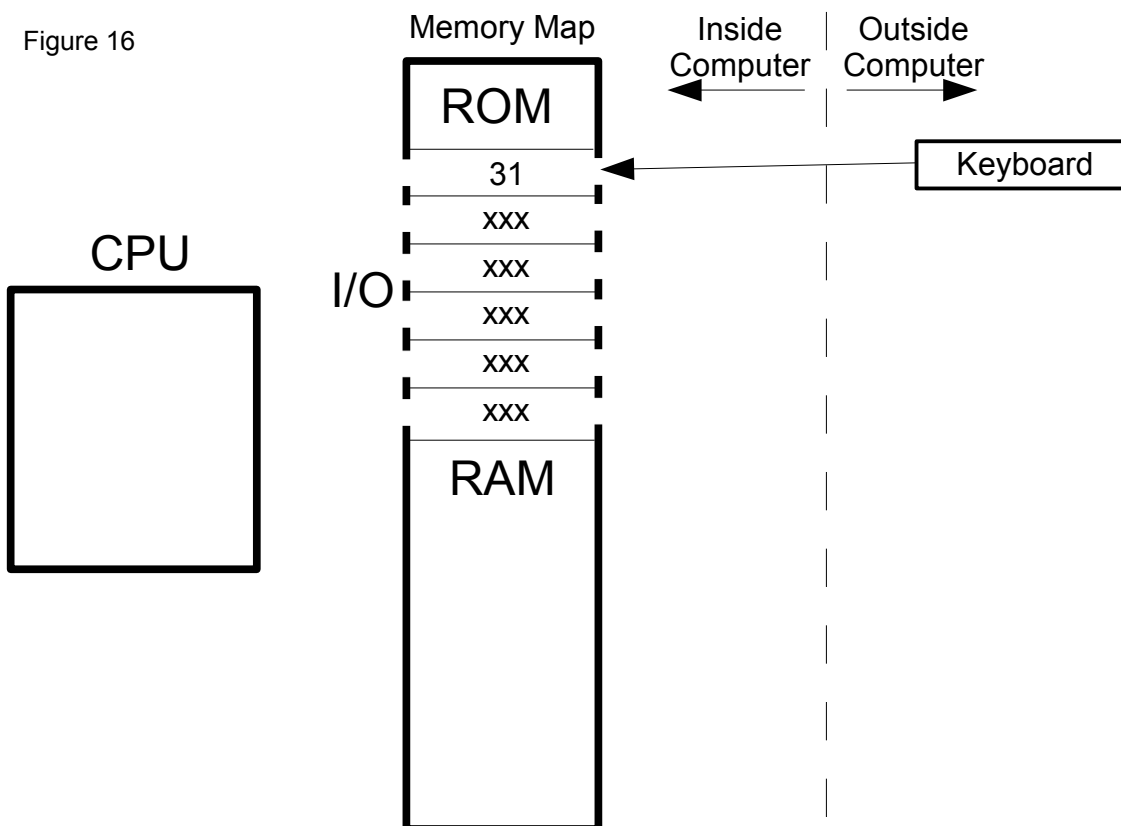
1610 Pat thought about this then replied, "The key is turned into electronic
1611 signals and sent to the computer through the keyboard's cable."

1612 "This is true," I said "but the interesting part is what the electronic signals

1613 represent. When a key is pressed on a keyboard, say the 'A' key, the idea of
 1614 the capital letter 'A' is turned into a number, and the electronic signals that
 1615 are sent through the keyboard's cable represent this number."

1616 "But where does the other end of a keyboard's wire attach to the computer
 1617 at? This is where the I/O memory locations come in. I/O memory locations
 1618 are special memory locations because, not only do they have an opening
 1619 that faces towards the CPU, they also have another opening that faces the
 1620 outside of the computer! The way that a device that is outside the core of a
 1621 computer sends information into the computer is through one of these I/O
 1622 memory locations." I then drew a line from the left side of the keyboard
 1623 through the vertical dashed line and into the right side opening of one of the
 1624 I/O memory locations. (see Fig. 16)

Figure 16



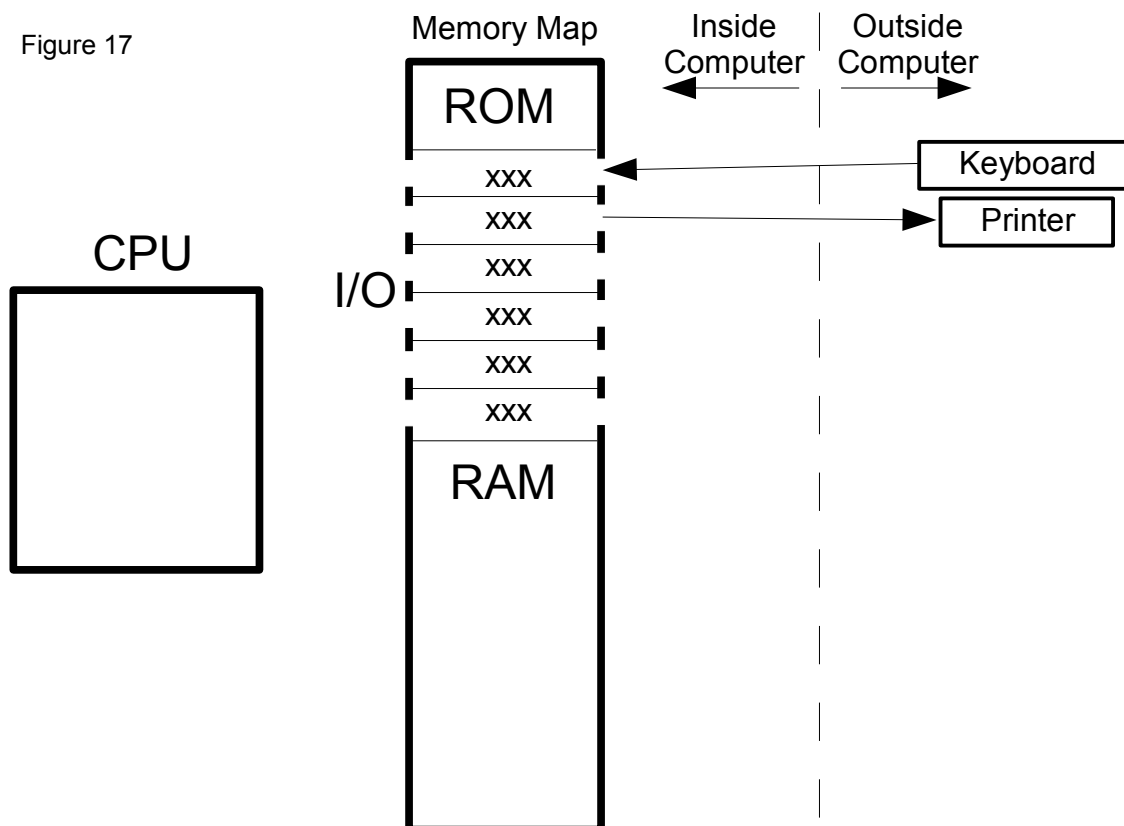
1625 "This line represents the keyboard's cable and, when a key is pressed on the
 1626 keyboard, a number between 0 and 255 (that represents that key) is sent
 1627 through the cable. This number then appears, as if by magic, in the I/O
 1628 location that the cable is attached to. After the number appears in the I/O
 1629 location, the CPU can access this same I/O location from its left-facing
 1630 opening in order to determine which key had been pressed."

1631 “Thats cool!” shouted Pat “I never would have guessed that a keyboard
1632 worked that way, but it makes sense!”

1633 “I agree,” I said “it does make sense and when I first learned how I/O
1634 memory locations worked, I was as excited as you are! The last thing I am
1635 going to do with the model of the keyboard is to place an arrow at the end
1636 of the cable that is attached to the I/O location to show that the keyboard
1637 send data into the computer.” which I did.

1638 “The next device I am going to draw is a printer.” Underneath the printer I
1639 drew another horizontal rectangle and wrote the word 'Printer' in it. I then
1640 drew a line between the printer and another one of the I/O memory
1641 locations. (see Fig. 17) “This is a simplified model of how a printer
1642 attaches to a computer. Now that you know how a keyboard sends a letter,
1643 like a capital 'A', to a computer, see if you can explain how a capital letter
1644 'A' might be printed on a printer.”

Figure 17



1645 Pat looked at the model and said “Lets see, the CPU places a number that
1646 represents a capital letter 'A' into the I/O location that the printer is

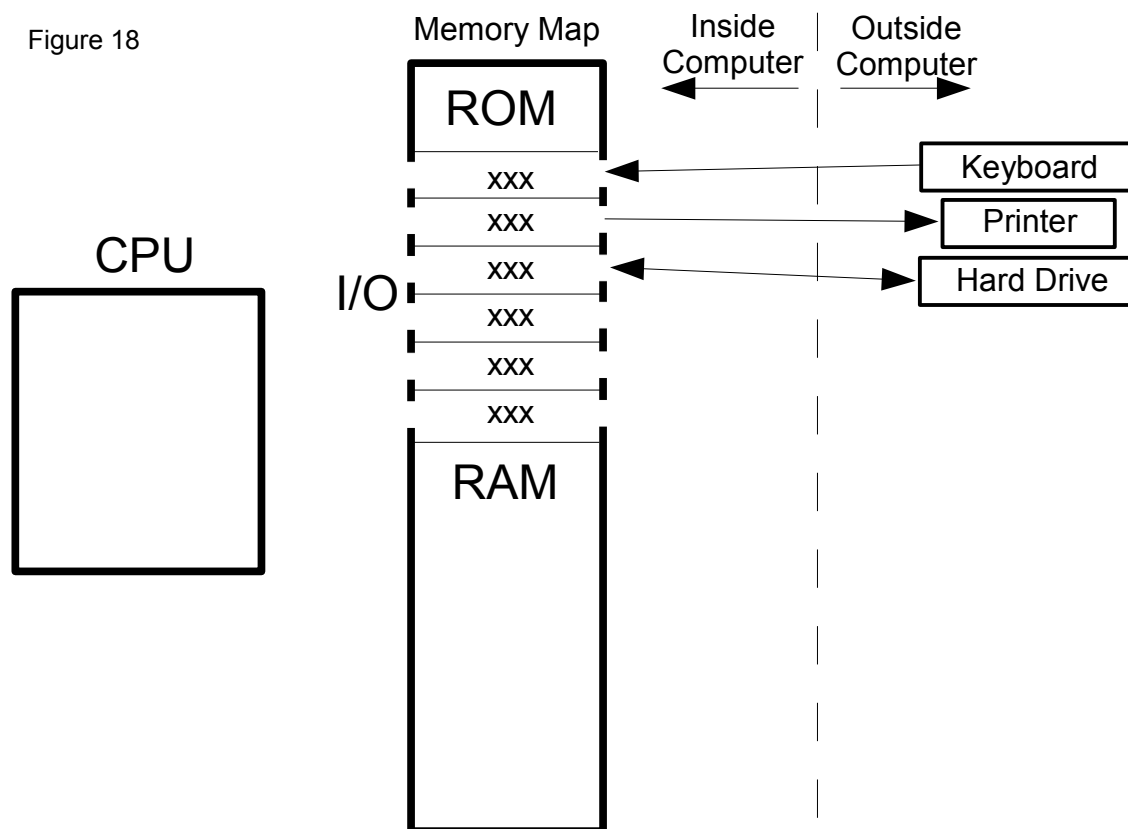
1647 attached to, this number is then converted into an electronic signal that
1648 represents the 'A' and the electronic signal is sent to the printer. The
1649 printer converts the electronic signal back into a number, determines that it
1650 represents a capital letter 'A', and then it prints it."

1651 "Very good Pat!" I said "Finally, which way should I point the arrow on the
1652 printer's cable?"

1653 "Point the arrow towards the printer, because information goes from the
1654 computer out to the printer."

1655 "Correct." I said, and I drew an arrow on the printer's cable that pointed
1656 towards the printer. "Now Pat, I think we know enough about how I/O
1657 memory locations work to go back to the hard drive." I drew another
1658 horizontal box (underneath the box that represented the printer) and
1659 wrote the words 'Hard Drive' in it. I then drew a line from the hard drive's
1660 rectangle to an unused I/O location then I said "which way should the arrow
1661 point on the hard drive's cable?"

Figure 18



1662 Pat thought about this for a moment then said "You should draw an arrow

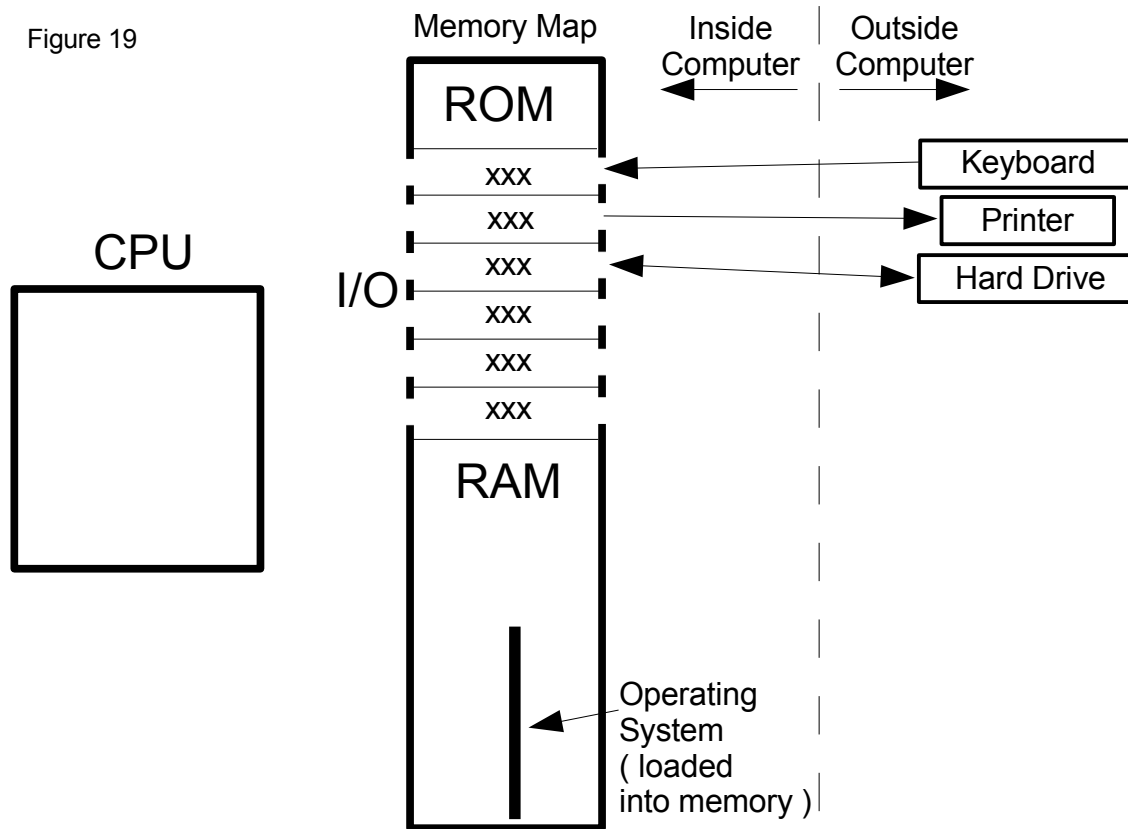
1663 on both ends of a hard drive's cable because a computer can send
1664 information to a hard drive and it can also read information from a hard
1665 drive." (see Fig. 18)
1666 "I thought I was going to trick you with that question Pat," I said "but I was
1667 wrong!" I then drew arrows on both ends of the hard drive's cable to show
1668 that information can be sent both ways along it.

1669 **Loading An Operating System**

1670 "Now," I said "lets go back to discussing what happens when the CPU is
1671 finished running the POST code and it needs to find more instructions to
1672 execute or it will lock up. The remaining code in the ROM tells the CPU to
1673 talk to a storage device (like a hard drive, a Flash drive or a CDROM)
1674 through the I/O location that the storage device is attached to. Actually, in
1675 a real computer, more than one I/O location is used to talk to a storage
1676 device, but for now a one I/O location example is simpler to work with. The
1677 CPU talks to a storage device and asks it if it has a program called an
1678 **operating system** stored on it."

1679 "If the storage device does have an operating system program stored on it,
1680 the CPU requests that the device send the numbers that represent the
1681 operating system's machine language instructions to the I/O location that
1682 the device is attached to, one number at a time. As each number arrives in
1683 the storage device's I/O location, the CPU copies this number to a register
1684 and then it copies the number from the register to RAM. An operating
1685 system consists of thousands and thousands of machine language
1686 instructions so I can not show them all being placed in RAM individually.
1687 Instead, I am going to draw a vertical bar from the bottom of the RAM
1688 upwards which represents RAM being filled with the numbers that
1689 represent the operating system," which I did. (see Fig. 19)

Figure 19



1690 Operating Systems: Bridges To Cyberspace

1691 "What exactly is a computer operating system?" Pat asked "I have always
1692 been confused by this."

1693 I thought about Pat's question for a few moments then said "Do you
1694 remember earlier when I said that, in a way, computers were magic
1695 because part of a computer exists in the physical world and the other part
1696 exists in a non-physical realm called cyberspace?"

1697 "I remember." said Pat.

1698 "Do you also remember what **context** and **contextual meaning** are?" I
1699 asked.

1700 "Yes," said Pat "context means the circumstances within which an event
1701 happens or the environment within which something is placed. Contextual
1702 meaning is the meaning that the context gives to the events that happen, or

1703 things that are placed, within it."

1704 "Very good, Pat. Now it is time to give you my explanation for what
1705 cyberspace is." I said. **"Cyberspace consists of all of the ideas that are
1706 currently bound to numbers in any computer, anywhere in the
1707 physical universe, through contextual meaning."**

1708 Pat sat quietly for a while, with eyes staring off into space, thinking about
1709 what I just said. Finally, Pat blinked, looked at me and said "When you
1710 described what cyberspace was, a picture came into my mind and in it were
1711 millions of computer memory locations laid out across the Earth, with
1712 millions of ideas floating above them, and they were connected to each
1713 other by threads of contextual meaning."

1714 "Some people," I said "think that there really is a world of ideas that is
1715 separate from the physical world and perhaps some day we will know what
1716 ideas actually are. Even if we do not know exactly what ideas are yet,
1717 though, we do know that it is the computer's ability to easily move ideas
1718 into and out of cyberspace, and easily manipulate ideas when they are
1719 there, that gives the computer its great power."

1720 "I will explain cyberspace more fully as we continue our discussion, but lets
1721 use our current understanding of cyberspace to help us understand what a
1722 computer operating system is. A computer operating system is a special
1723 kind of program that acts as a bridge between the physical world and
1724 cyberspace. Most of the sophisticated computers in the world (including
1725 PCs, servers, ATM machines, car computers and cell phones) have an
1726 operating system in them and it is the operating system in a device that
1727 enables it to access the resources of cyberspace."

1728 Systems Within Systems

1729 "We have been using the word 'system' quite a bit, for example 'computer
1730 system' and 'operating system', but what is a good definition of a system?
1731 Before we continue, lets find a definition for system on the Internet." I went
1732 to my computer, searched for a definition of the word 'system' and found
1733 the following:

1734 **System:** A group of interacting, interrelated, or interdependent elements
1735 or parts that function together as a whole to accomplish a goal.
1736 <http://www.doe.mass.edu/frameworks/scitech/2001/resources/glossary.html>

1737 "This definition," I said "indicates that the purpose of a system is to

1738 accomplish a goal and that a system is made up of parts that work together
1739 to accomplish this goal. Examples of systems include the water system that
1740 provides water to your home, a skyscraper and an automobile engine. The
1741 parts in a system can also be arranged into groups that form systems of
1742 their own and these smaller systems are often called **subsystems**. The
1743 prefix 'sub' means 'under' so another way to think about a subsystem is as
1744 an 'undersystem'. Subsystems can contain subsystems of their own and
1745 many of the things in the world contain multiple levels of systems within
1746 systems. A skyscraper's subsystems include its heating and cooling system,
1747 lighting system, telephone system, elevator system and cleaning system
1748 (which include janitors)."

1749 "People can be part of a system?" asked Pat.

1750 "Yes," I replied "there are many kinds of systems that have people as parts.
1751 A building's cleaning system fits our definition of a system because it has a
1752 goal (to keep the building clean) and it contains parts that interact
1753 together to attain this goal. The mops, dusters and garbage cans in a
1754 building are parts in its cleaning system, but so are the janitors that interact
1755 with these parts."

1756 "I had never thought that people could be parts in a system" said Pat "but
1757 you are right, they can."

1758 "There are many types of parts that can be used in a system," I said
1759 "including metal and plastic parts, rubber hoses, water, air, electricity,
1760 people, and information. It is this last kind of part, information, that I would
1761 like to focus on."

1762 "Information!?" said Pat "How can information be a part in a system if you
1763 can't even touch it?"

1764 "Information is present in every system," I said "no matter what kind of
1765 system it is, but sometimes it is not obvious how a system uses the
1766 information that is contained within it because information is not physical.
1767 It cannot be seen or touched."



1768 “Lets take a simple system, like a combination lock, and see if we can
1769 determine how information is being used in this system.” I went to my
1770 storage room and returned with a combination lock. “What is the goal of
1771 this system, Pat?” I said, and I gave Pat the lock.

1772 Pat studied the lock for a while, turned the dial a few times, pulled up on its
1773 shackle then said “The goal of a combination lock is to prevent people from
1774 stealing your stuff.”

1775 “And how does it do this?” I asked

1776 Pat studied the lock some more then said “The lock's shackle is placed
1777 through something that has a hole in it, like a locker, then the shackle is
1778 closed. The lock will not release the shackle until the correct combination
1779 is entered on the dial, and the thing that the lock is attached to will not
1780 open until the shackle is removed from the hole it was put in.”

1781 “Lets assume that the thing that the lock is attached to is a locker.” I said
1782 How does the locker know whether or not it has a lock attached to it?”

1783 Pat replied “A person must first try to lift the locker's handle. If there is not
1784 a lock in the handle's hole, the handle will lift and the locker's door will
1785 open. If there is a lock in the handle, though, the metal around the hole will
1786 bump against the metal of the shackle when the handle is lifted and this
1787 bumping will prevent the handle from lifting far enough to open the door.”

1788 “That is correct.” I said “One of the laws of physics states that 'two pieces of
1789 physical matter cannot occupy the same space at the same time'. The
1790 locker is a system that contains a lock as a subsystem and the lock uses this
1791 law of physics to inform the locker that it is not permitted to open. This
1792 kind of physical 'informing' or communication, the bumping together of two
1793 pieces of physical matter, is a common example of how information is used
1794 in a system. Another bumping-related example is the accelerator pedal in
1795 an automobile. If a driver wants to make a car go faster, they press their
1796 foot against the accelerator pedal, the pedal pushes against a lever, the
1797 lever usually pulls on a cable that has some kind of system at its other end

1798 that allows more air/fuel mixture to enter the engine which results in the
1799 engine turning faster.”

1800 “Moving back to the lock, a human has the number sequence that will open
1801 the lock stored in their mind. This number sequence represents information
1802 and the way that this information is communicated to the lock is by turning
1803 the lock's dial. As the dial is turned, bumping-type information is
1804 communicated between the parts of the lock. If the correct combination is
1805 entered, the piece of matter that is informing the shackle that it cannot
1806 open is allowed to move freely and, when the shackle is pulled, this piece of
1807 matter moves away from the shackle as it is lifted and the lock opens.”

1808 **Physical Parts Are Costly And Constrained**

1809 “I am starting to see how information can be used as a part of a system,”
1810 Pat said “but what does all of this have to do with computer operating
1811 systems and cyberspace?”

1812 I smiled and replied “Parts made of physical matter have all kinds of
1813 constraints associated with them, Pat. If the parts are made of metal, for
1814 example, the ore for the metal has to be mined out of the Earth, then the
1815 ore has to be transported to a mill where it is transformed into metal
1816 shapes, like rods and bars, that are suitable for processing by machine
1817 tools. These metal shapes then have to be transported to the manufacturing
1818 facilities that contain these machine tools so that the machines can create
1819 parts from them. The parts are then transported to assembly facilities that
1820 assemble the parts into subsystems and these subsystems are often
1821 transported to yet other assembly facilities where they are assembled into
1822 final products. Each step along the way, from ore to finished product, takes
1823 time and energy to accomplish and time and energy translate into cost.”

1824 “Another kind of constraint that parts made from physical matter are under
1825 are the laws of physics. These laws dictate that parts made from physical
1826 matter can be moved back and forth only so fast, they can only handle so
1827 much force applied to them and there are limits to how large or small they
1828 can be. Beyond this, once the parts are formed into a given shape to serve
1829 a given function, they cannot be easily reformed into other shapes to serve a
1830 different function. For example, if we wanted our combination lock to have
1831 a 4 number combination instead of a 3 number combination, it would be
1832 extremely difficult to reshape all of the parts in the lock to accomplish this.”

1833 **Moving Parts Into Cyberspace**

1834 "High cost and low flexibility are two of the main constraints that are
1835 associated with parts made of physical matter. But what if it were possible
1836 to take the parts of a system that only consist of information and move them
1837 into cyberspace?"

1838 "Move them into cyberspace!?" Pat said. "Is this possible?"

1839 "Yes Pat!" I replied "Any part of any physical system that stores or
1840 communicates information can be moved into cyberspace using a computer.
1841 As soon as a part is moved into cyberspace, it becomes an idea and ideas
1842 existing in cyberspace are constrained much less by the laws of the physical
1843 world than their physical counterparts are. Beyond this, parts can be made
1844 in cyberspace that would be extremely difficult, or even impossible, to make
1845 in the physical world. Cyberspace parts can even be created, assembled
1846 into systems, disassembled and destroyed quicker than you can blink your
1847 eyes. They can also be sent anywhere in the world through computer
1848 networks at the speed of light."

1849 Pat's mouth dropped open in amazement. "Cyberspace does sound like a
1850 magical place," Pat finally said "but I don't fully understand how it works."

1851 **A Lock Made Of Physical Parts and Cyberspace Parts**

1852 "I do not think that anybody completely understands cyberspace yet, Pat," I
1853 said "but we are learning more about it all the time. Lets go back to the
1854 combination lock and see if we can describe a lock that has some physical
1855 parts and some cyberspace parts. This lock will still have a shackle, so that
1856 it can lock a locker, and it will still have a 3 number combination. Instead
1857 of having a dial, though, it will have a keypad so that a human can enter the
1858 combination. Instead of metal parts controlling whether the shackle is able
1859 to be opened or not, it will have a solenoid."

1860 "A solenoid?" Pat said "What is a solenoid?"

1861 "Have you ever turned a nail into a magnet by wrapping a bunch of wire
1862 around it then putting electricity through the wire?" I asked.

1863 "Yes," said Pat "I read how to do that in a science book and, when I turned
1864 it on, I was able to pick up paper clips with it."

1865 I continued "A magnet made using this technique is called an
1866 electromagnet. A solenoid uses an electromagnet to pull on a metal arm in

1867 one direction and a spring is usually used to pull the metal arm in the
1868 opposite direction when the electricity is turned off. The result of this is
1869 that when the electricity is turned on, the arm moves up against a stop in
1870 one direction and when the electricity is removed, the spring moves it
1871 against a stop in the opposite direction. In our lock, the end of a solenoid's
1872 arm can be used to either allow the shackle to lift or to prevent it from
1873 lifting. Solenoids are on/off devices and computers love to control devices
1874 that are either on or off."

1875 "Why is that?" asked Pat.

1876 "I will make a deal with you Pat." I said "Do some research about computers
1877 on the Internet when you go home. The next time you come over, if you can
1878 tell me why computers love things that are either on or off, I will give you a
1879 solenoid. Is it a deal?"

1880 "Its a deal!" said Pat "But are you saying that we are going to put a
1881 computer inside of a lock?"

1882 "Sure, why not?" I said.

1883 **Microcontrollers: Computers On A Chip**

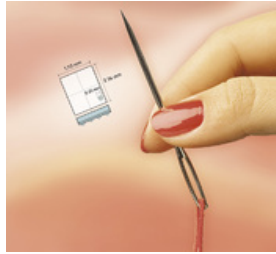
1884 "Isn't a computer too big to fit into a lock?" Pat replied.

1885 "Some computers are too big to fit into a lock," I said "but some computers
1886 are very small. Have you ever heard of a microcontroller?"

1887 "No," said Pat "what is a microcontroller?"

1888 "A microcontroller is a complete computer on a chip. Inside the chip is the
1889 same model of a computer that we have been developing on the whiteboard.
1890 It has a CPU, RAM, ROM and I/O in it." I then went to one of my drawers
1891 and returned with a small microcontroller. "Open your hand, Pat." I said. I
1892 then lightly touched Pat's hand with my pinky finger, to equalize our
1893 charges, and then placed the microcontroller in it."

1894 Pat studied the microcontroller for a while then said "Its so small. Is there
1895 really a complete computer in this chip?"



1896 "Yes," I said "and some microcontrollers are even smaller than that one!"

1897 Pat studied the chip a bit longer then gave it back to me.

1898 I held the chip between my fingers and said "If there is a program in this
1899 chip when it is turned on, part of the chip enters cyberspace. A computer
1900 program is what is used to create parts in cyberspace and, by programming,
1901 we will be able to create cyberspace lock parts and place them inside this
1902 microcontroller. The microcontroller can be interfaced to the keypad and to
1903 the solenoid using the I/O locations in its memory map. It can then accept a
1904 combination from a human and open the lock if the combination is correct."

1905 "Is the cyberspace lock better than the normal lock?" Pat asked.

1906 "In some ways it is." I replied. "For example, the combination cannot be
1907 changed in the normal lock, but it can easily be changed in the cyberspace
1908 one. The cyberspace lock can have additional numbers added to the
1909 combination with little effort and it can even keep track of how many times
1910 the lock was opened and at what time. The cyberspace lock is also capable
1911 of having additional information added to it, like a user ID, so that it can
1912 record who is opening it. Since cyberspace holds ideas, almost any idea you
1913 can think of can be placed into this microcontroller, as long as the idea is
1914 not too big to fit!"

1915 "Thats amazing!" said Pat "Computers are becoming more interesting to me
1916 all the time. I understand a little better now what cyberspace is but it is
1917 still kind of fuzzy to me though."

1918 "The only way to gain a better understanding of how cyberspace works," I
1919 said "is to learn how to program computers. Learning how to program is
1920 very hard work, but it is one of the most useful skills a person can have in
1921 our modern world."

1922 "The more I learn about computers" said Pat "the more I want to learn how
1923 to program them. I am definitely going to take you up on the offer you

1924 made earlier to help me. For now, though, I still don't understand what a
1925 computer operating system is or how it acts as a bridge between the
1926 physical world and cyberspace."

1927 **Operating System: The Part Of A Computer Which Is Made From**
1928 **Cyberspace Parts**

1929 "Now that we have discussed both systems in general and cyberspace," I
1930 said "we are in a good position to explain what a computer operating
1931 system is. We just saw how a normal combination lock can be made more
1932 capable and flexible by moving some of its parts into cyberspace. There are
1933 many systems in the physical world that are made more capable and flexible
1934 by having some of their parts exist in cyberspace, including automobiles,
1935 televisions, aircraft, microwave ovens, heating and cooling systems,
1936 machine tools and telephones. As we move into the future, traditional
1937 physical systems of all kinds are being redesigned to include cyberspace
1938 parts, and systems that already have cyberspace parts are being redesigned
1939 to increase their percentage of cyberspace parts. One might even imagine
1940 that all systems would be made 100% of cyberspace parts if it were
1941 possible."

1942 Pat thought about this for a while then asked "What type systems today
1943 have the greatest percentage of cyberspace parts?"

1944 I smiled and said "The type of systems that currently have the greatest
1945 percentage of cyberspace parts are sophisticated computers, like PCs and
1946 servers. Usually, over half of a sophisticated computer system is built from
1947 cyberspace parts and the portion of a computer that is built from
1948 cyberspace parts is called its **operating system!**"

1949 "That's what an operating system is!?" cried Pat "The portion of a computer
1950 that is made from cyberspace parts?"

1951 "Yes," I replied "this is one way to look at it".

1952 **Application Programs**

1953 "What about applications programs that run on a computer, like a word
1954 processor?" asked Pat "aren't those part of the computer too?"

1955 "I am making a distinction," I said "between the cyberspace parts that are
1956 needed to make a computer a complete, functioning system, and the
1957 cyberspace parts that are added to this functioning system to make it do a

1958 given kind of work. For example, when you first turn on a typical personal
1959 computer, it spends some time booting up (which means loading the
1960 operating system into memory and running it) and then you are presented
1961 with a graphical user interface or GUI. This GUI allows you to do things
1962 like move a mouse pointer around on the screen, select menus and look at
1963 the contents of a hard drive. Your computer is now a complete, functioning
1964 system but it has not been specialized for any given kind of work yet. In
1965 order to do work, you must select an application program with the mouse
1966 (like a word processor) and when this application is loaded and running,
1967 the computer can then do specialized work with it. The physical parts of a
1968 computer are called its **hardware** and the cyberspace parts of a computer,
1969 including both the operating system and application programs, is called
1970 **software.**"

1971 **Computers Without Operating Systems**

1972 Pat sat quietly for a few moments then said "Earlier you said that most
1973 sophisticated computers have operating systems. Are there some
1974 computers that do not have operating systems?"

1975 "Yes," I replied "some smaller microcontrollers do not have a separate
1976 operating system. They usually just run one dedicated program and the
1977 cyberspace components that are needed to perform tasks that a separate
1978 operating system would perform are built into the program itself. These
1979 microcontrollers usually run their dedicated program directly from ROM.
1980 Some computers with operating systems, such as cell phones and
1981 automotive computers, will also run their operating system from ROM but
1982 computers like PCs and servers load their operating systems into RAM each
1983 time they are powered up."

1984 **Back To Loading An Operating System**

1985 "I understand now that an operating system is made using cyberspace
1986 parts" said Pat "but what does an operating system actually do?"

1987 "We will talk about what an operating system does in a later discussion," I
1988 replied "but for now, lets continue our discussion about what happens in a
1989 PC when the POST code in its BIOS ROM is nearly finished running and the
1990 last machine language instructions in the ROM tell the CPU to talk to a
1991 storage device in order to obtain the numbers that represent an operating
1992 system. Do you remember how a CPU is able to communicate with devices
1993 that are outside of itself?"

1994 Pat looked at the model of a computer that we had been drawing on the
1995 whiteboard and replied "A CPU is able to communicate with devices outside
1996 itself using the special I/O memory locations in its memory map."

1997 "Yes," I said "and what makes the I/O locations special?"

1998 Pat replied "The I/O memory locations are special because they can can
1999 have numbers copied into them and out of them by both the CPU and by a
2000 device that is outside the computer. That is why the I/O locations have a
2001 hole in them that face the CPU on one side and a hole that face the outside
2002 world on the other side."

2003 "Very good Pat." I said "Now, the CPU talks to the storage device and asks
2004 if it contains numbers that represent an operating system. If the storage
2005 device replies that it does, the CPU requests that the device send the
2006 operating system's numbers into the I/O location, one number at a time, and
2007 the CPU in our model copies each of these numbers into RAM. Most of
2008 these numbers represent machine language instructions." As I said this I
2009 pointed to the vertical bar at the bottom of the RAM section of the memory
2010 map which represented the operating system's numbers.

2011 "After the core of the operating system has been copied into RAM," I said
2012 "the last instructions in the ROM sets the CPU's Program Counter to the
2013 beginning of the operating system's machine language instructions in RAM
2014 and then the operating system starts to run. The process of copying the
2015 operating system's numbers from a storage device into RAM, and running
2016 the core of the operating system after it has been loaded, is called **booting**
2017 the computer system. Assuming that this is a model of PC, the operating
2018 system will show a GUI (or a command line interface, like the Commodore
2019 64 has) to the user when it is finished booting and it is then ready to run
2020 applications."

2021 **Primary Storage And Secondary Storage**

2022 "The purpose of a storage device is to hold numbers?" Pat asked.

2023 "Yes." I replied. "The amount of RAM and ROM in a computer's memory
2024 map is limited and, as we talked about earlier, when the power on the
2025 computer is turned off the numbers in the RAM memory locations
2026 disappear. Therefore, in order for a computer like a PC or a server to be
2027 useful, it must have the ability to hold numbers outside its memory map for
2028 later use. The storage that is in a computer's memory map is called

2029 **primary storage** and the storage that is held in devices outside of the
2030 computer is called **secondary storage** or **mass storage**. Examples of
2031 secondary storage devices include hard drives, flash drives, CDRoms, DVDs
2032 and magnetic tapes. To give you a feel for the amount of primary vs.
2033 secondary storage in a typical PC, we can take this PC on the table as an
2034 example. This PC has 1 gigabyte of RAM and 100 gigabytes of hard drive
2035 space."

2036 "That's a big difference," said Pat "but if the hard drive can hold so many
2037 more bytes than the RAM can, why not just get rid of the RAM and use the
2038 hard drive's storage instead?"

2039 "That is a good question, Pat." I said. "The reason that secondary storage
2040 can not be used in place of primary storage is that primary storage, like
2041 RAM, is able to have numbers copied into and out of it **much** faster than
2042 secondary storage can. Primary storage is faster than secondary storage,
2043 but it is also more expensive per byte. Both types of storage are needed,
2044 however, in a general purpose computer like a PC."

2045 **General Purpose Computers Vs. Specific Use Computers**

2046 "A PC is a general purpose computer?" Pat asked "Why is that?"

2047 "General purpose computers," I replied "are designed to maximize their
2048 flexibility so that they can be configured as needed to perform various kinds
2049 of work. The way that a computer is configured to perform a given kind of
2050 work is with a program. Examples of programs that allow a computer to do
2051 a given kind of work include word processors, games, browsers and media
2052 players. Since a PC can easily run numerous kinds of programs, it is
2053 considered to be a general purpose computer. If the programs are large, or
2054 if more then one program is going to be running on the computer at the
2055 same time, then the amount of RAM needs to be large. If one has many
2056 programs, or the amount of data the programs use is large, then the amount
2057 of secondary storage needs to be large. The more general purpose a
2058 computer needs to be, the more RAM and secondary storage it needs."

2059 "Does a general purpose computer also need a large amount of ROM?" Pat
2060 asked.

2061 "No," I replied "a general purpose computer only needs enough ROM to
2062 hold instructions that test the hardware during power up, instructions that
2063 allow the operating system to more easily talk to the hardware, data that

2064 configures the motherboard and instructions that help load the operating
2065 system into RAM. In a general purpose computer, the amount of ROM in its
2066 memory map is significantly less than the amount of RAM. Specific purpose
2067 computers, on the other hand, usually have significantly more ROM than
2068 RAM."

2069 "What is a specific purpose computer," asked Pat "and why do they have
2070 more ROM than RAM?"

2071 "A specific purpose computer is a computer that has been designed to
2072 perform one dedicated task. Examples of specific purpose computers
2073 include computers that control automobile engines, televisions, DVD
2074 players, heating and cooling systems, audio systems, elevators and security
2075 systems. The reason that specific purpose computers usually have more
2076 ROM than RAM is that they typically only run one program, or a small
2077 number of programs. This program or programs, along with perhaps a
2078 small operating system, take up a comparatively small amount of space
2079 which can usually fit into ROM primary storage. Microcontrollers are the
2080 kind of computer system that are most widely used as specific purpose
2081 computers."

2082 "That makes sense." said Pat "Maybe tonight I will ask my Mom if I can take
2083 her car computer apart so that I can see how much ROM and RAM it has!"

2084 We both laughed at this!

2085 "I would not do that on a running car, if I were you," I said "at least not
2086 until you have learned how to do some computer interfacing. In the mean
2087 time, I have some old cars in my recycle yard that have engine computers in
2088 them. You are welcome to take one of those computers apart if you would
2089 like."

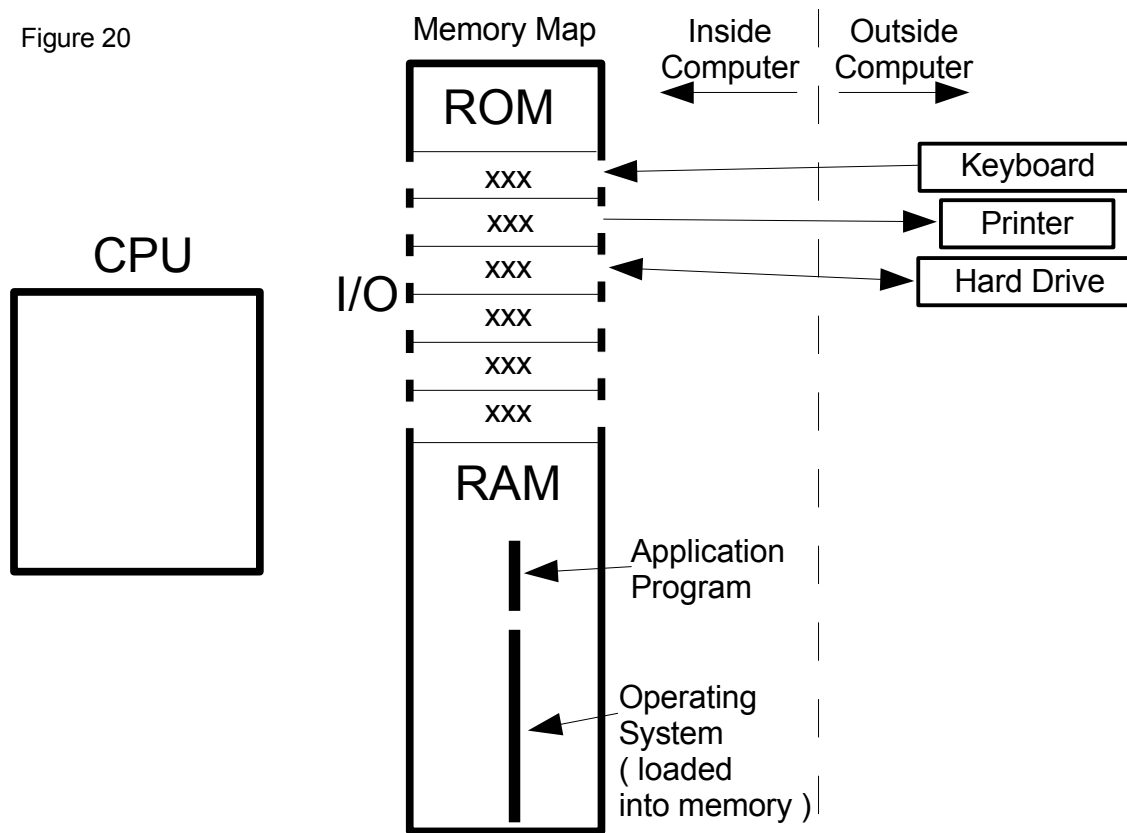
2090 "Thanks!" said Pat "I think I will do that soon."

2091 **Running An Application Program**

2092 "Lets continue our discussion of what happens when a PC boots up." I said
2093 "We made it to the point where the operating system was loaded into RAM
2094 and was presenting a GUI, or command line interface, to the user. Can you
2095 use the model on the whiteboard to describe what happens when the user
2096 runs an application program?"

2097 "I will try." said Pat. "If the operating system has a GUI, then the user will
 2098 use the mouse pointer to double click on a program, or select it from a pull
 2099 down menu. The machine language instructions in the operating system
 2100 will then ask the secondary storage device that holds the program to send it
 2101 one number at a time to the I/O location that the storage devices is attached
 2102 to. The CPU will then take each of these numbers and place them into an
 2103 unused section of RAM. After the CPU has finished copying the program
 2104 into RAM, the operating system will then tell the CPU to start running the
 2105 machine language instructions of this program." Pat then took a marker
 2106 and added another vertical line above the line that represented the
 2107 operating system. This second vertical line represented the application
 2108 program after it was loaded into RAM. (see Fig. 20)

Figure 20



2109 "That is correct Pat," I said "you seem to be learning this information rather
 2110 well! One thing I have not mentioned until now is that a PC usually has an
 2111 extra piece of hardware in it called a **Direct Memory Access controller** or
 2112 **DMA controller** for short. This controller is able to directly copy numbers
 2113 from one section of memory to another independent of the CPU. Devices

2114 like hard drives, graphic chips, sound chips and network interfaces often
2115 use a DMA controller to copy numbers to and from other parts of memory in
2116 order to free the CPU to do other work. This makes the overall computer
2117 system work faster. There are other simplifications I have made during our
2118 discussion in order to present the core ideas of how a computer works as
2119 clearly as possible. As we get deeper into how a computer works, though, I
2120 will add this more detailed information."

2121 **Something Is Missing From The Models**

2122 "The models of a computer that we have created on the whiteboard have
2123 enough detail to show the core ideas of how most computers in the world
2124 work. As I said before, in our modern computer-based world this is
2125 extremely valuable knowledge to possess. Furthermore, as computers
2126 become applied to increasingly more aspects of our society, this knowledge
2127 will become even more valuable. What do you think of these models of how
2128 a computer works?"

2129 "I think its amazing!" said Pat "The models explain so many things about a
2130 computer that I had no clue about before. But something still seems to be
2131 missing."

2132 "Oh?" I said "That is curious because the models are fairly accurate. What
2133 do you think is missing?"

2134 "It appears to me," said Pat "that the computer spends almost all of its time
2135 copying numbers from memory into the CPU, copying numbers from the
2136 CPU into memory and doing simple mathematical operations. Beyond this,
2137 not much more seems to be happening."

2138 "No, you are right Pat," I said "at its lowest levels, a computer does not do
2139 much more than this."

2140 "But," said Pat "what about all of the cool things a computer can do!? Take
2141 a space game program, for example. A typical space game may have
2142 dozens of ships on the screen, all shooting lasers and missiles at each other
2143 while avoiding all kinds of spinning asteroids and space junk. Explosions
2144 and collisions are happening everywhere and the sounds from the ship's
2145 engines, the lasers, missiles, collisions and explosions are being projected
2146 from the computer's speakers. At the same time, the game is taking all
2147 kinds of quickly typed input from the keyboard and it may also be in
2148 communication with one or more other computers playing the same game

2149 over a network. How do these simple models of how a computer works
2150 explain all of the intense action that a game like this has?"

2151 **Wink Of An Eye**

2152 "I was wondering if you would notice that some critical information was
2153 missing from the models," I said " and I will try to explain what it is. There
2154 is an episode from the original Star Trek TV show, called 'Wink of an Eye',
2155 that can help explain the missing piece. In this episode the **Enterprise** is
2156 exploring an outer part of the galaxy when it receives a distress call from a
2157 nearby planet. The ship is placed into orbit around the planet and a landing
2158 party (consisting of captain Kirk, Mr. Spock, Dr. McCoy and some red
2159 shirted crew members) is beamed down to the planet. There are buildings
2160 and other signs of civilization on the planet, but no humans can be found
2161 and the landing party's instruments can not even detect any animal life.
2162 They keep hearing insect buzzing sounds, though, which is strange because
2163 there are no insects. Have you ever watched any of the original Star Trek
2164 episodes, Pat?"

2165 "Sure," said Pat, "I have seen a number of them."

2166 "Do you know what usually happens to red shirted crew members?" I asked.

2167 "Something bad usually happens to them!" Pat said.

2168 "Right," I said "something bad usually happens to them and this episode is
2169 no exception. Soon after beaming down to the planet, something bad
2170 happens to one of the red shirted crew members. In this episode, the
2171 people that had sent the distress signal (who are called Scalosians) are
2172 still on the planet, but their metabolisms have been vastly increased by
2173 radiation which was emitted from a volcano. Their metabolisms have been
2174 increased so much, in fact, that the atoms in their bodies are vibrating too
2175 quickly for the landing party to see. This faster vibration results in the
2176 Scalosians living in a faster time frame than the crew of the enterprise are.
2177 At this point I am going to deviate from the story line of this episode in
2178 order to better explain the part that is missing from our models of a
2179 computer."

2180 "Lets assume that a red shirted crew member screamed and captain Kirk
2181 started running across the landscape to investigate. Imagine him taking
2182 two steps then freezing like a statue in a running position and the other
2183 members of the landing party also become frozen. At the same time, people

2184 suddenly appear and they seem to be acting normally. They are walking
2185 around the frozen landing party members and discussing what they should
2186 do about them. What has happened is that the perspective has been
2187 switched to the faster time frame and we are seeing the world as the
2188 Scalosians see it."

2189 One Scalosian looks at captain Kirk and says "we can not let him reach his
2190 fallen crew member. Let us build a brick wall to stop him," and the other
2191 Scalosian agrees. Have you ever seen a brick wall built, Pat?"

2192 "No." answered Pat.

2193 "Brick walls can not be built by magic," I said "they must be built using very
2194 specific techniques. Lets assume that the Scalosians are going to build a
2195 brick wall that is 50 meters long, 5 meters high and 1/2 meter thick. The
2196 first thing that needs to be done is to hire a backhoe crew to dig a trench 50
2197 meters long and make it deep enough to put the bottom of the wall below
2198 the frost line. This digging might take 2 days. Next, a cement form making
2199 crew needs to be hired and it might take them another 2 days to put
2200 together the forms in the bottom of the trench for something called a footer,
2201 which is what the bricks will sit on. A cement crew is then contracted to fill
2202 the form with cement and it might take a week for the cement to cure to the
2203 point where bricks can be placed on it. Finally, brick layers are hired to
2204 slowly and carefully assemble the wall brick-by-brick. This might take
2205 another two weeks."

2206 "During all this time captain Kirk, in his slower time frame, has moved
2207 perhaps an inch or two. Now imagine that captain Kirk looks up and he
2208 sees... what?"

2209 "A brick wall, suddenly appeared out of nowhere!" cried Pat "The wall was
2210 not built using magic, but to captain Kirk in his slower time frame, it looks
2211 like it was."

2212 "Yes," I said "to captain Kirk it looks like a brick wall suddenly appeared in
2213 front of him as if by magic. This time frame difference is the part that is
2214 missing from our models of a computer. A computer is only capable of
2215 doing very simple operations (like copying numbers from memory into the
2216 CPU, copying numbers from the CPU into memory and simple mathematical
2217 operations) but it can do millions of these operations every second!"

2218 "A computer can do millions of operations a second!?" said Pat.

2219 "Yes," I replied "which means that computers work in a much faster time
2220 frame than humans do. Imagine that a computer can look at you from
2221 inside its screen. It looks at you and what does it see?"

2222 "If it is running millions of times faster than we are," said Pat "then we look
2223 like statues to it."

2224 "That is correct." I said "Imagine that this PC is watching me as I use my
2225 index finger to press the 'A' key on the keyboard." As I said this I started
2226 slowly moving my finger towards the 'A' key. "From the computer's point of
2227 view, it might take a hundred years for my finger to reach the top of the 'A'
2228 key and another 5 years to press it down enough for the key to click. At
2229 soon as the key clicks, the letter 'A' is quickly converted into a number, this
2230 number is encoded as electronic signals and these signals are sent into the
2231 computer at the speed of light."

2232 "The computer must get very bored," said Pat "while waiting for humans to
2233 interact with it."

2234 "I agree," I said "a computer usually spends thousands of years in its time
2235 frame waiting for humans to interact with it. This also answers your
2236 question about how computers are able to do all of the amazing things they
2237 do, including games like your space game. A computer screen is made up of
2238 little dots called **picture elements** or **pixels** for short. The computer puts
2239 each ship, asteroid, laser, and missile in your space game together pixel-by-
2240 pixel, just like the Scalosians had to put their wall together brick-by-brick.
2241 But the computer also has thousands of years in its time to do this. From
2242 our point of view, it appears that the computer is doing all of this work by
2243 magic."

2244 **I Want To Learn More About Computer Software And Hardware**

2245 Pat sat quietly for a while then said "I have enjoyed our discussion about
2246 how a computer works and now I really want to learn more about computer
2247 software and hardware. You said that you would help me learn how to
2248 program so when can we start?"

2249 "Come back soon," I replied "and we will begin."