6502 Emulator, Binary, And Hexadecimal

by Ted Kosan

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

Table of Contents

MathRider and the U6502 Emulator

The Sun had just started rising above the trees when I heard a knock at the door of my shop. I opened the door and there stood Pat, wearing a cheerful smile.

"Hello professor," said Pat "do you have time today to begin teaching me how to program? I haven't been able to think of anything else since the last time we talked."

I recalled the recent discussion we had about how a computer worked and my promise to help Pat to learn how to program. I was pleased that Pat had remembered the promise and I took it as a sign that Pat might just have the perseverance needed to become a programmer. "Yes," I replied "I have some time today. Come in, boot up my computer and launch a browser"

We both sat down in front of the monitor and, after Pat launched a browser, I said "Go to http://mathrider.org and download the latest release of the software you find there."

Note: At this point you should also download MathRider onto your PC. There is a book in the download section of the MathRider website called MathRider for Newbies which contains instructions for installing MathRider.

As we were waiting for the software to download, Pat asked "What software are we downloading?"

"We are downloading a program called MathRider which includes a program called U6502 that creates a virtual 6502 CPU. Programs like this are called **emulators** because they emulate the operation of a hardware CPU in software. The unique thing about this specific emulator is that, not only can it run on a PC, it can also run on a microcontroller. This will allow you to learn how to program a 6502 processor on a PC and then easily transfer these programming skills to a microcontroller if you later become interested in monitoring and controlling things in the physical world with software."

"Monitoring and controlling things in the physical world sounds like fun!" said Pat. "I am definitely interested in learning how to do that some day."

The file finished downloading and I had Pat create a directory on the hard drive called pats_files and then unzip the file into it.

"MathRider is written in the Java programming language," I said "so before it can be run, the PC needs to have Java installed on it. I already have Java installed on this PC but if you want to try running the emulator at home, you will need to install Java on your PC too. The MathRider for Newbies book also contains instructions for installing Java on your PC."

"What is Java?" asked Pat "I have heard about it, and have seen the coffee cup Java logo on the web and my Mom's cell phone, but I never knew exactly what it was."

I replied "Java™ consists of a simulated CPU and a language which has been specifically designed to run on this CPU. Another name for a simulated CPU is a **virtual machine** and a CPU simulator that runs Java code is called a **Java Virtual Machine** or **JVM**. Java was created in the early 1990s and it was one of the first software technologies that was specifically designed for use in a globally-networked environment. One of Java's core capabilities is to allow programs to be compiled into the JVM's machine language (which are called Java bytecodes) and then these programs can be executed on any devices that has a JVM on it."

"Do you mean that my Mom's cell phone has a JVM on it?" asked Pat.

"Yes." I replied "The reason for this is because cell phones use many different CPUs, hardware configurations, and operating systems which makes them difficult to write applications for. When a JVM is placed on a cell phone, it allows it to run standard Java bytecodes and this makes it easier to write Java applications that can be run on multiple cell phone types. Currently, Java cell phone application developers still need to account for various difference between cell phones, but there are people who are constantly working on reducing these differences."

"If JVMs can be placed in PCs and cell phones," asked Pat "what other kinds of things can they be placed in?"

"Web browsers, smart cards, video players, television set top boxes, machine controllers, video game consoles, almost anything that contains a CPU." I replied.

Pat thought for a few moments then asked "What is the difference between an **emulator** and a **virtual machine**?"

"The main difference," I replied "is that an emulator usually emulates a physical CPU design while a virtual machine does not necessarily do so. In Java's case, a document has been created that specifies all the requirements a JVM needs to adhere to and this document is called the Java Virtual Machine **Specification**. People are free to implement this specification in a variety of ways and some people have even implemented the specification in hardware. Most people, however, implement the JVM specification in software.

Pat started laughing then said "Perhaps hardware implementations of the JVM specification should be called Java Actual Machines or JAMs because they are actual CPUs!"

I laughed too!

"So, we are going to be running a 6502 emulator on top of a Java Virtual Machine which will itself be running on a physical CPU?" asked Pat.

"Yes," I replied "and you will encounter numerous examples of this kind of 'machine running on top of machine' pattern as you explore the field of computing. For now, though, lets start working with the U6502 emulator.

Using The U6502 Emulator

After MathRider was installed, I ran it.

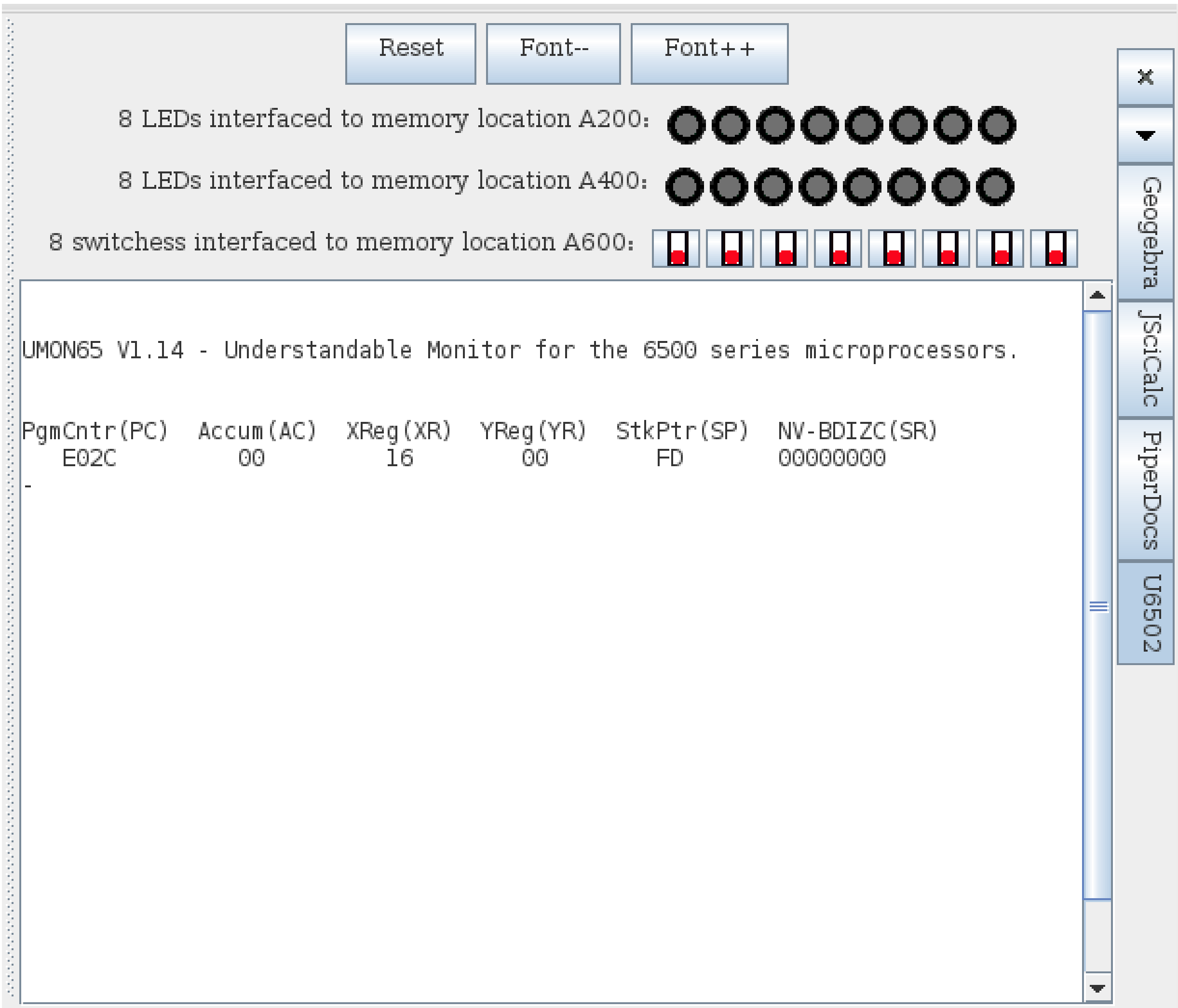
"Wow!" said Pat "MathRider looks cool! What can it do?"

"MathRider can be thought of as an environment which enables people to learn programming and mathematics at the same time." I replied. "It can do an amazing amount of things, such as automatically plot and solve mathematical equations, but for now we will mostly be using MathRider to learn how to program a 6502 microprocessor. I may, however, periodically write short programs in MathRider which will help illustrate the ideas we will be covering."

"So later on you will show me how to use MathRider?" asked Pat.

"Yes." I replied. "But if you would like to start exploring MathRider on your own at home, the Mathrider for Newbies book contains an introduction to MathRider which will help you get started."

I then used the mouse to select the **U6502** tab on the right side of the MathRider application and the emulator was displayed.



After the emulator was shown I said "The emulator's GUI contains three simulated devices that are interfaced to the I/O portion of the emulator's memory map.

The first device contains a set of 8 LEDs (Light Emitting Diodes) and these are interfaced to memory location A200. The interface circuitry that connects these LEDs to memory location A200 automatically uses whatever number is placed into location A200 to determine which LEDs should be turned on.

The second device also contains 8 LEDs and it works just like the first device does except it is interfaced to location A400.

The third device is a set of 8 switches and it is used to input information into the system. It is interfaced to memory location A600 in the emulator's memory map and the interface circuitry automatically converts the state of these switches into a number and places it into location A600.

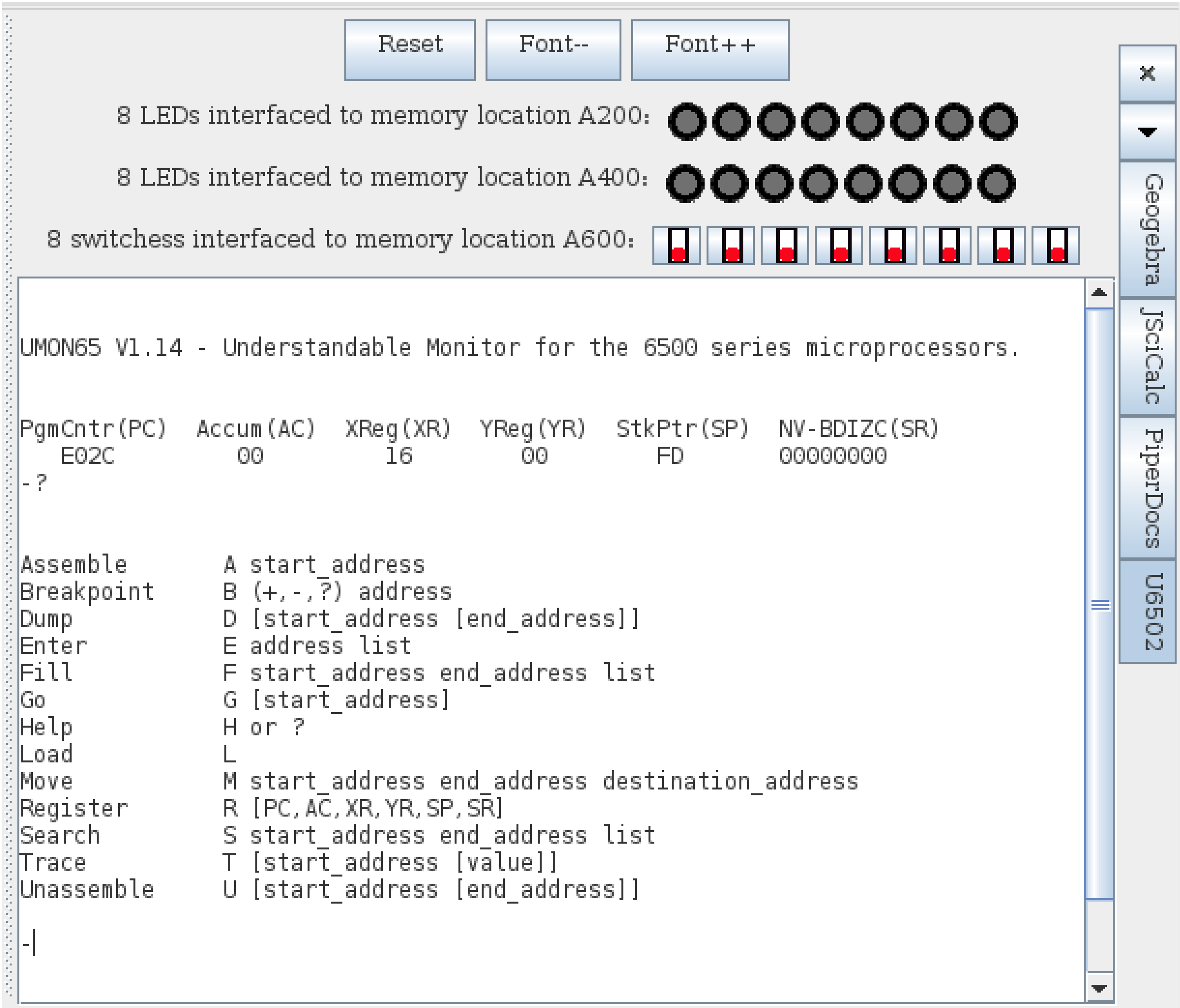
Pat studied the emulator's GUI for a while then said "The first device contains 8 switches, the second device contains 8 LEDs and the third device also contains 8 LEDs. Is there a reason why these devices all have 8 lights or switches in them?"

"Yes," I replied "and it is related to the reason why each memory location can only hold a number between 0 and 255. Do you remember from our earlier discussion what another name is for a number between 0 and 255 is?"

"It is called a **byte**." said Pat. "Is there a way to look at the bytes that are in the memory locations of the emulator?"

"Yes." I said. "Click the mouse pointer just to the right of the dash that is in the emulator's typing area and a blinking cursor will be placed there. Then, type a question mark followed by the <enter> key and watch what happens."

Pat did this and then the following information was displayed in the emulator's text area:


Reset
Font--
Font++
8 LEDs interfaced to memory location A200:
8 LEDs interfaced to memory location A400:
8 switchess interfaced to memory location A600:
UMON65 V1.14 - Understandable Monitor for the 6500 series microprocessors.
PgmCntr(PC) Accum(AC) XReg(XR) YReg(YR) StkPtr(SP) NV-BDIZC(SR)
E02C 00 16 00 FD 00000000
-?
Assemble A start_address
Breakpoint B (+,-,?) address
Dump D [start_address [end_address]]
Enter E address list
Fill F start_address end_address list
Go G [start_address]
Help H or ?
Load L
Move M start_address end_address destination_address
Register R [PC,AC,XR,YR,SP,SR]
Search S start_address end_address list
Trace T [start_address [value]]
Unassemble U [start_address [end_address]]
-|
Geogebra
JSciCalc
PiperDocs
U6502

Note: from here on the output from the emulator will be shown in text form instead of inside of a screen shot.

```
?

Assemble       A start_address
Breakpoint     B (+,-,?) address
Dump           D [start_address [end_address]]
Enter          E address list
Fill           F start_address end_address list
Go             G [start_address]
Help           H or ?
Load           L
Move           M start_address end_address destination_address
Register       R [PC,AC,XR,YR,SP,SR]
Search         S start_address end_address list
Trace          T [start_address [value]]
Unassemble     U [start_address [end_address]]

-
```

"What was just printed in the text area?" asked Pat.

"When you typed a question mark '?' and pressed <enter>," I replied "the question mark was sent to a special program on the emulator which is called a monitor. A **monitor** is a program that gives a person access to all of the memory locations in a computer along with all of the registers in the computer's CPU. When the monitor receives a question mark, it recognizes it as one of the commands it supports (in this case it is the **help** command) and it responds with a list of all of its commands along with the extra information each command accepts. After a command is done executing, a dash '-' is displayed as a command prompt."

Pat looked through the list then asked "Which one of these commands allows us to look at the bytes in the emulator's memory map?"

"Which command do you think it is?" I replied.

Pat looked at each command in the list then said "I think its probably the **Dump** command."

"You are correct." I said. "The purpose of the **Dump** command is to show the contents of a computer's memory locations. Type **d e000 e090** at the command prompt and lets see what happens." Pat did this and the following output was displayed:

```
-d e000 e090

E000  4C 1B E0 4C B5 F3 4C 74 - F3 4C 97 F3 4C 57 F3 4C  L..L..Lt.L..LW.L
E010  A7 F2 4C 03 F3 4C 41 EA - 4C 70 E0 A2 FF 9A D8 20  ..L..LA.Lp.....
E020  37 F3 4C 25 E0 A2 16 A0 - F4 20 57 F3 00 C9 02 D0  7.L%..... W.....
E030  13 29 F0 4A 4A 4A 18 69 - C0 8D 09 00 A9 00 8D 08  .).JJJ.i........
E040  00 6C 08 00 4C 47 E0 A0 - 00 8C 44 00 A2 64 C8 D0  .l..LG....D..d..
E050  FD CA D0 FA EE 44 00 AD - 44 00 4C 4C E0 20 69 E0  .....D..D.LL. i.
E060  20 DB E0 20 33 E1 4C 5D - E0 A2 66 A0 F4 20 57 F3   .. 3.L]..f.. W.
E070  A0 00 8C 5E 00 8C CD 00 - 8C 5F 00 8C 73 00 8C 87  ...^....._..s...
E080  00 A0 00 A9 00 99 A5 00 - C8 C0 27 D0 F8 A0 00 20  ..........'....
E090  97
-
```

I smiled as Pat's mouth dropped open with surprise.

"What's that ?!" cried Pat "It looks like some kind of strange code!"

My smile turned into a laugh. "When you entered the Dump command, you requested that the monitor show you the bytes that were in locations E000 through E090 and so it did." I said.

"But I thought that bytes were numbers between 0 and 255. Why do these numbers have letters in them?!" asked Pat.

"The is an interesting story behind that, Pat." I replied. "Let me find a small whiteboard and then I will tell it to you."

Decimal and Binary Numerals

When I returned with the whiteboard, I set it down on the table and said "Can you tell me what the difference is between a **number** and a **numeral**, Pat?"

"Hmmm..." said Pat. "I know that a number is used to indicate the amount of something and that numerals have something to do with digits like 1, 2 and 3, but I am not sure exactly what the difference is between a number and a numeral."

"This is not unusual," I said "because most people are not aware that there is a distinction and even those that do often use these terms interchangeably in everyday life. However, in order to properly explain why the bytes in the Dump command's output have letters in them, we need to precisely define what a **number** is and what a **numeral** is and we must also determine how they are **related**. This information lies within the realm of **mathematics** and the deeper one goes into the field of computing, the more mathematics one needs to know."

"I don't know Professor," said Pat "I don't understand mathematics very well and it is not one of my favorite subjects. Are you sure that a person needs to know a lot of mathematics in order to program computers?"

"The more mathematics a person knows," I said "the more effective programmer they will be. Perhaps the reason you do not understand mathematics very well is that you have not been exposed to it in a way that matches your learning style. It is my opinion that one of the best ways to learn mathematics is to study mathematics and programming at the same time. If you would like, I will integrate mathematics into our discussions on programming and maybe you will find that you will learn it easier this way. What do you think?"

"Okay," replied Pat "we can try this and see what happens."

"Very good, now lets start by defining what numbers and numerals are." I said. " The first thing to understand about **numbers** is that they do not exist in the physical world. A **number** is a mathematical concept that exists only in the non-physical world of ideas and its purpose is to describe the **amount** of something."

"People use numbers every day in the physical world," said Pat "so how could they do this if numbers didn't exist here?"

I replied "The only way that numbers can be worked with from within the physical world is by using symbols to represent them which are either made of physical matter or make use of certain properties of physical matter. Any symbol (or set of symbols) that is encoded in physical matter, and used to represent a number, is called a **numeral**. For example, if we wanted to work with the number 5, we could write a '5' on a piece of paper, draw 5 lines next to each other on a piece of paper, tap the top of a table 5 times, say the word 'five', turn on a set of little lights on a computer screen in the shape of a 5, or even arrange a set of transistors in the memory location of a computer into an on/off pattern that represents the number 5..."

Pat looked at me and blinked, and I smiled and raised an eyebrow...

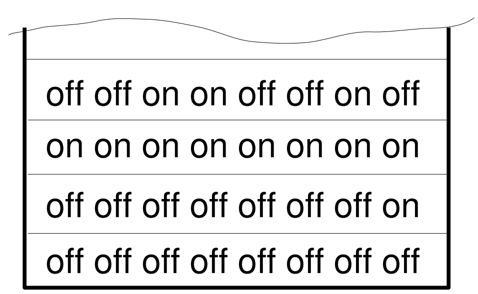
"Numbers do not actually exist inside a computer," asked Pat "only numerals that represent numbers do?"

"That is correct." I replied. "I did not want to make the distinction between numbers and numerals during our earlier discussion on contextual meaning in order to make that discussion simpler."

"I can understand how written symbols, sets of lights, and sounds can be used as numerals, but what are transistors and how can patterns of ons and offs be used as numerals?"

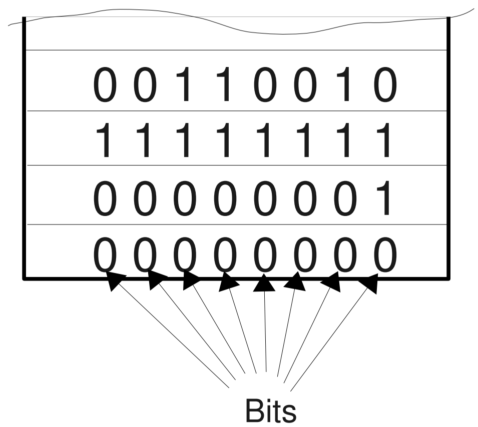
"A **transistor** is an electronic valve that does to the flow of electrons in a conductor what a water valve does to the flow of water in a pipe." I said. "Transistors can also be configured so that they are either all the way **on** or all the way **off**, like the light switch for this room." I reached and flicked the light switch on and off a few times to illustrate this point. "The pattern of ons and offs that can be created by a set of transistors in a memory location is able to represent a range of numbers. This pattern is thought of as being arranged in a row like this." I then wrote patterns of ons and offs inside 4 memory locations on the whiteboard. (see Fig. 1)

Figure 1



"The words 'on' and 'off' are somewhat awkward to use for representing these patterns and therefore a simpler numeral system which has the symbol '0' represent an 'off' state and the symbol '1' represent an 'on' state is commonly used. These two symbols represent the on/off state of what is called a **bit** of information and this is what the memory locations I just drew look like when the words 'on' and 'off' are replaced with bits." (see Fig. 2)

Figure 2



Pat looked at the second diagram then said "These 1's and 0's certainly take up less room than the words 'on' and 'off' do. I don't see yet how they are able to represent numbers, though."

"We will look at that next, then" I said. "Patterns of bits are actually part of a numeral system which is called the **binary numeral system** and this system matches any pattern of bits with a unique number. The binary numeral system only has 2 symbols in it, 0 and 1, and it is therefore called the **base 2 numeral system**. Another name for **base** of a numeral system is **radix**. Lets start with a set of four bits and see which decimal numerals match the patterns they can be arranged into." I began by writing the words 'Binary' and 'Decimal' on the whiteboard. Then, I wrote 0000 under the word 'Binary' and matched it with the decimal digit 0. Underneath this I wrote 0001 and matched it with the decimal digit 1. (see Fig. 3)

Figure 3

Binary		Decimal
0000	-	0
0001	-	1
	-	2

I then wrote the decimal numeral 2 on the next line and started to write the bit pattern that went with it, but then I stopped and said "I am counting the binary numerals up from 0000 to match the decimal numerals that are being counted up from 0. What binary pattern do you think comes next in the sequence to match the decimal numeral 2?"

Pat thought about this question for a while then said "I am not quite sure because we have not counted far enough for me to see if there is a pattern to the sequence."

"There is indeed a pattern to the sequence that the binary numerals go through as they count upwards. This pattern follows the same counting rules that the decimal system does, so lets study these rules with the decimal system first and then apply them to the binary system." I then wrote the decimal digits 0 through 9 on another area of the whiteboard and said the name of each digit as I wrote it "zero, one, two, three, four, five, six, seven, eight, nine, ?... ummm, I have a problem Pat. I have run out of decimal digits and I do not know what to do next!" (see Fig. 4)

Figure 4

0
1
2
3
4
5
6
7
8
9
?

Pat gave me a strange look then said "Of course, the next numeral in the sequence is 10. Why didn't you just write a 10 and continue?"

"Tell me what rule you followed to go from 9 to 10 and I will." I replied.

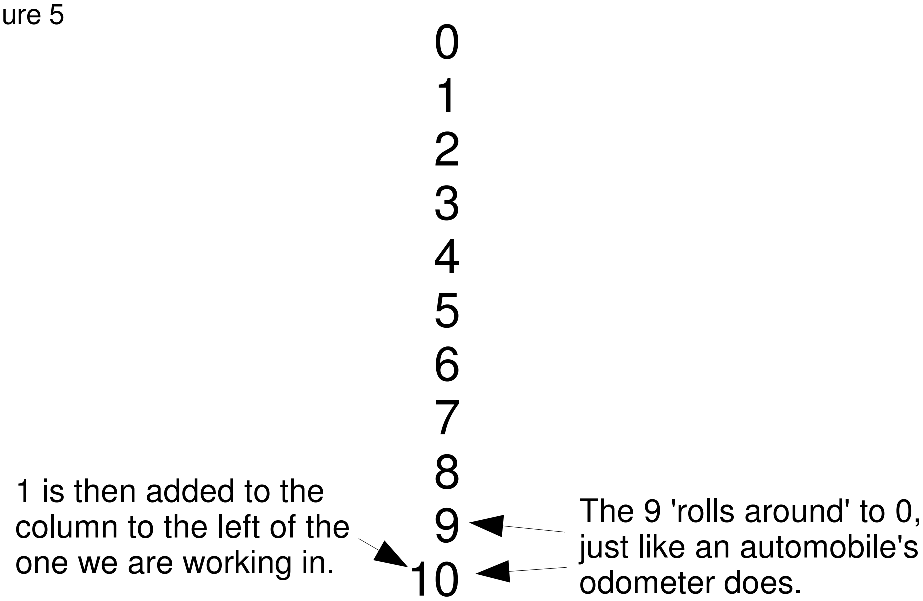
"Well," said Pat "when you need to add one more to the column you are in, but you have run out of digits in that column, you have the column go back to 0 and then add one to the digit in the column which is to the immediate left of the one you are working on."

"And what are the names that are given to these columns in the decimal system?" I asked.

"The initial column is the ones column, to its left is the tens column and to the tens column's left is the hundreds column. Each time you move left a column, that column represents a place value that is ten times larger than the column you are in."

"Very good Pat." I said. "Lets see if your counting rule solves my problem. Seven, eight, nine, I have run out of decimal digits in the ones column so I roll from 9 around to 0 in the ones column, just like an odometer in an automobile does, and then I add one to the column immediately to the left, which is the tens column." (see Fig 5)

Figure 5



"Your counting rules seems to work with the decimal system, lets see what happens when we apply them to the binary system. 0000, 0001, I just ran out of binary digits in the first column, so I 'roll' it around to 0 and add one to the column which is immediately to its left. The new pattern is 0010 and this is correct. Now, what pattern do you think comes next to match the decimal numeral 3?" I asked. (see Fig. 6)

Figure 6

Binary		Decimal
0000	-	0
0001	-	1
0010	-	2
	-	3

Pat studied the diagram then said "In the decimal system we would continue to the next digit in the sequence in the ones column and it looks like we would do something similar here. I think the next pattern of bits is 0011."

"Correct." I replied. Then I wrote the pattern 0011 on the whiteboard and also added the numeral 4 in the next row in the decimal column. (see Fig. 7)

Figure 7

Binary		Decimal
0000	-	0
0001	-	1
0010	-	2
0011	-	3
	-	4

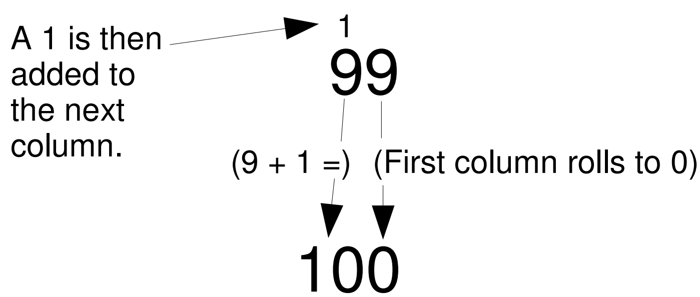
"What pattern comes next to match the decimal numeral 4, Pat?" I asked.

Pat stared at the digram for a long while. Eventually I said "Perhaps it would be helpful if we see how a similar pattern is handled in the decimal system." I then wrote the number 99 in another section of the whiteboard and said "what number comes next? Follow the counting rules and say them out loud as you do so."

Pat said "The 9 in the ones column rolls around to 0 and one is added to the 9 that is in the tens column. This makes the tens column also roll around to 0 and a 1 is added to the 100s column resulting in the next decimal numeral in the sequence being 100."

"Very good Pat" I said as I wrote the numeral 100 under the 99 on the whiteboard. (see Fig. 8) Now apply the same thinking to the binary numeral 0011."

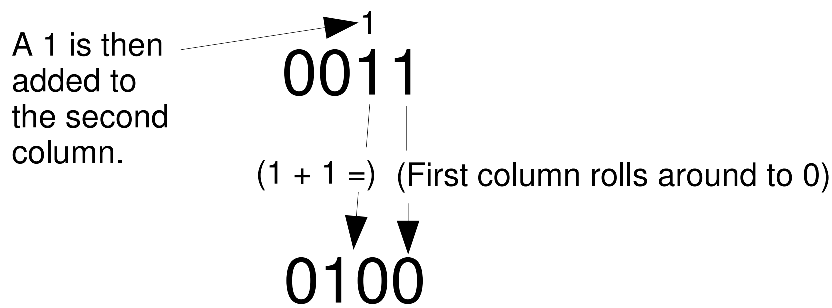
Figure 8



Pat studied the binary numeral 0011 again and then said "The 1 in the first column rolls around to 0 and then 1 is added to the column to its left. But I don't know what 1 + 1 is in binary."

"1 + 1 in binary is 10 binary." I said. "After we perform this addition in the second column the result is 0100 (see Fig. 9).

Figure 9



I then filled in the rest of the binary numerals up through 1111 using these counting rules and matched them with their decimal equivalents (see Fig 10)

Figure 10

Binary		Decimal
0000	-	0
0001	-	1
0010	-	2
0011	-	3
0100	-	4
0101	-	5
0110	-	6
0111	-	7
1000	-	8
1001	-	9
1010	-	10
1011	-	11
1100	-	12
1101	-	13
1110	-	14
1111	-	15

After Pat studied the binary numerals I had written for a while, I raised my left hand, put all 4 of my fingers down and said "Watch as I count from 1 to 15 in binary." I raised my index finger, said 'one' and then continued counting on my fingers in binary until I reached 15.

Then I said "I would like you to periodically practice counting in binary like this on your fingers because it will help you to become more comfortable with the binary system. Okay?"

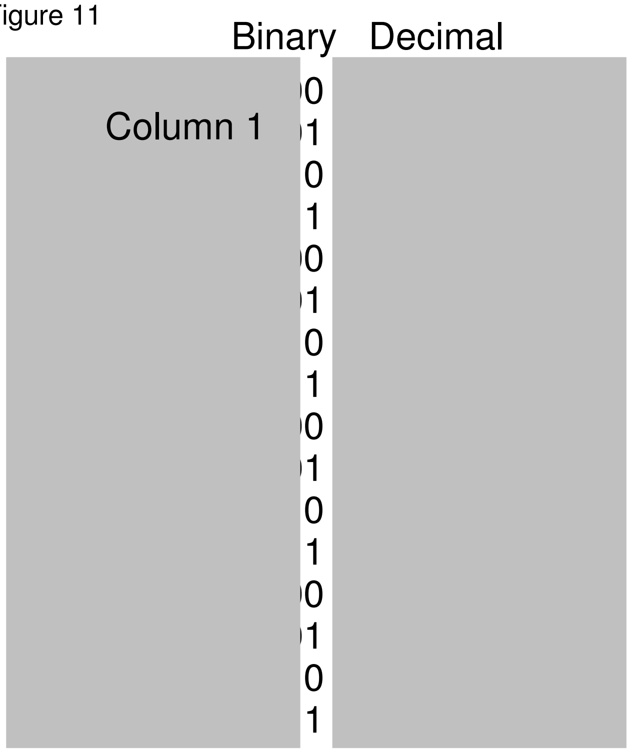
"Okay, said Pat"

"Now, I would like to show you some interesting patterns that binary numerals exhibit." I said. Then, I took two pieces of scrap paper and covered all the numerals on the whiteboard except the rightmost column of the binary numerals." (see Fig 11)

"What do the 0's and 1's do in the rightmost column, Pat?" I asked.

"They alternate between 0 and 1!" said Pat with excitement.

Figure 11



"Yes they do." I said. "Now, lets see what patterns appear when we look at the rest of the columns in the binary sequence." I then moved the pieces of paper so each of the remaining columns was shown in turn." (see Figs. 12, 13, and 14)

Figure 12

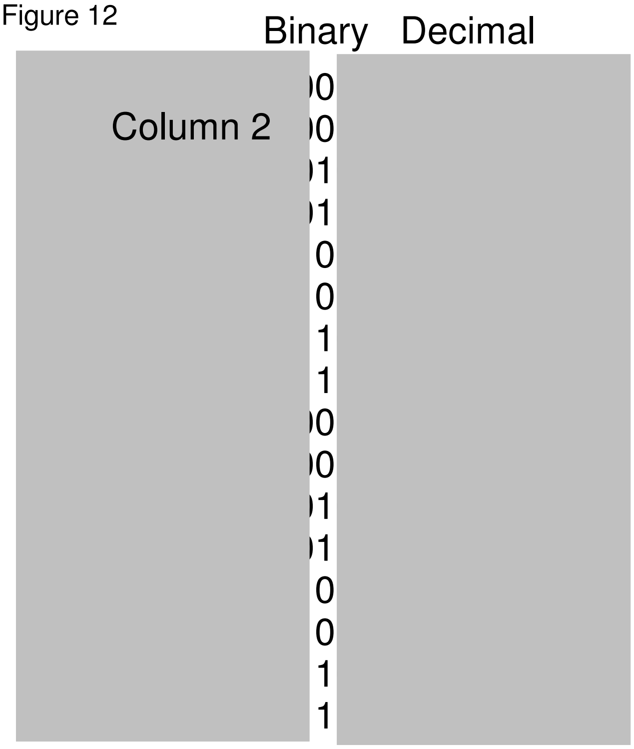
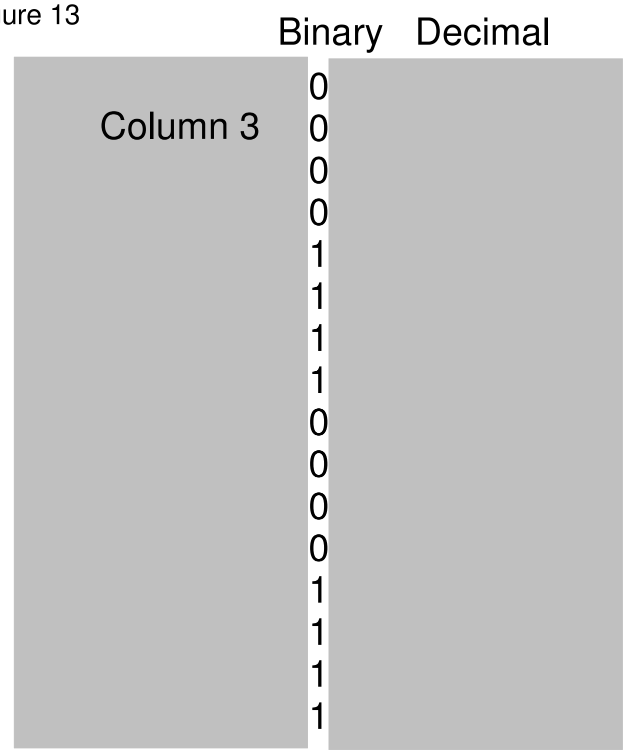
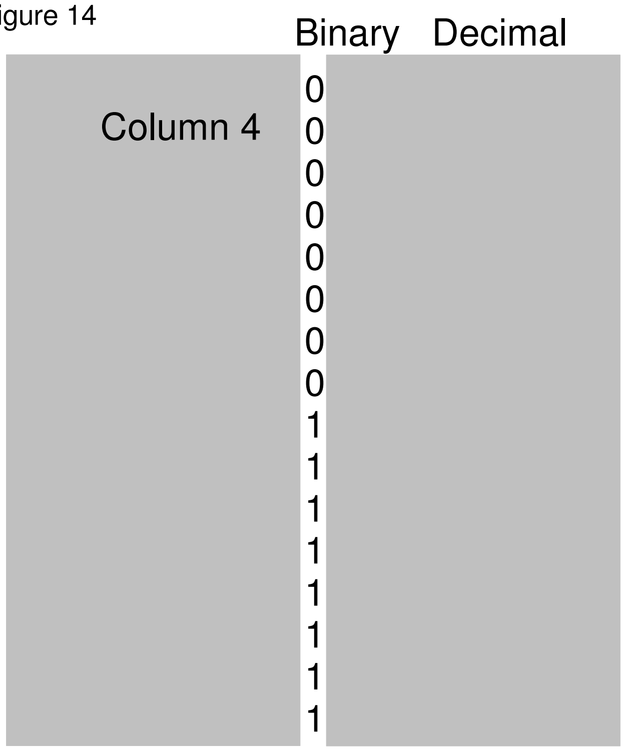


Figure 13



Pat looked at the pattern of 0's and 1's that each column contained then said "The second column contains a repeating pattern of two 0's followed by two 1's, the third column contains a repeating pattern of four 0's followed by four 1's and the fourth column contains a pattern of eight 0's followed by eight 1's. Thats cool, I didn't notice these patterns before."

Figure 14



The Number Of Patterns That Can Be Formed By N Bits

"Another aspect of sets of bits we can look at is how many patterns a given set of bits can be formed into. For example, how many patterns can be formed by one bit?" I said.

Pat thought about this for a while then said "I am not quite sure what you mean."

I held up my left hand and started moving my index finger up and down. "How many states can one bit be placed into?"

Pat studied my moving finger then said "Two states, with one state being the 0 state and the other state being the 1 state."

"That is right." I replied. I then wrote the sentence '1 bit can form 2 patterns' on the whiteboard and drew the 2 patterns next to it.

"Now, how many states or patterns can 2 bits be placed into?" I asked.

After a few moments of thought Pat began using two fingers to form the patterns 00, 01, 10, and 11. "2 bits can be formed into 4 patterns!" said Pat.

I nodded my head then wrote '2 bits can form 4 patterns' on the whiteboard and drew the 4 patterns next to it.

"The next question you are probably going to ask me is how many patterns can be formed by 3 bits." said Pat.

"You are right," I replied "can you figure it out?"

Pat used 3 fingers and counted the following patterns: 000, 001, 010, 011, 100, 101, 110, and 111.

Pat looked at me and said "3 bits can be formed into 8 patterns. This is fun!"

"I agree, working with binary patterns is fun!" I said, then I added a section related to the patterns that 3 bits could be formed into to the whiteboard. "Can you figure out how many patterns 4 bits can be formed into?"

"I think so." replied Pat. Using 4 fingers, Pat slowly formed them into the following patterns: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111. "4 bits can be formed into 16 patterns!"

I recorded the information for 4 bits on the whiteboard and then we looked at what had been written so far. (see Fig. 15)

Figure 15

1 bit can be formed
into 2 patterns:
0
1

2 bits can be formed
into 4 patterns:
00
01
10
11

3 bits can be formed
into 8 patterns:
000
001
010
011
100
101
110
111

4 bits can be formed
into 16 patterns:
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

"Do you notice anything interesting about what happens to the number of patterns that can be formed by a set of bits when one more bit is added to the set?" I asked Pat.

Pat looked at each of the sets then said "when we went from 1 bit to 2 bits, the number of patterns that could be formed increased from **2** to **4**. When we went from 2 bits to 3 bits, the number of patterns that could be formed increased to **8** and when we went from 3 bits to 4 bits, the number of patterns that could be formed increased to **16**..."

I then said "2, 4, 8, 16..."

"The number of patterns that can be formed doubles each time one bit is added to a set of bits!" cried Pat.

"Very good Pat!" I said "You are starting to see how these patterns work. Now, how many patterns can a set of 7 bits be formed into?"

"7 bits?" asked Pat. "Wow, I am going to have to think about that one. I suppose we could use the pattern doubling method we just discussed, but it would be nice if there were an easier way to figure it out."

"Would you like me to show you how to determine the answer using a calculator?" I asked.

"Yes." said Pat, so I opened a drawer, pulled out a calculator and handed it to Pat.

"1 single bit is able to produce 2 patterns. Enter a single number 2 into the calculator and press the equals button. What is the answer?"

"2." said Pat.

"We also know that 2 bits are able to produce 4 patterns. Enter 2 x 2 into the calculator and tell me what the answer is."

"4." said Pat "I think I see how to do the calculation now!"

"Okay, then use this method to calculate the number of patterns each set of bits from 1 to 7 can produce and write them on the whiteboard." Pat did so and produced this table:

1 bit can generate 2 patterns.
2 bits can generate 2 x 2 = 4 patterns.
3 bits can generate 2 x 2 x 2 = 8 patterns.
4 bits can generate 2 x 2 x 2 x 2 = 16 patterns.
5 bits can generate 2 x 2 x 2 x 2 x 2 = 32 patterns.
6 bits can generate 2 x 2 x 2 x 2 x 2 x 2 = 64 patterns.
7 bits can generate 2 x 2 x 2 x 2 x 2 x 2 x 2 = 128 patterns.

"Notice that 2 x 2 is the same as 2 to the 2nd power 2^2 ,2 x 2 x 2 is the same as 2^3 , and so on." I then created the following exponents version of Pat's patterns calculations."

1 bit can generate 2^1 = 2 patterns.
2 bits can generate 2^2 = 4 patterns.
3 bits can generate 2^3 = 8 patterns.

4 bits can generate 2^4 = 16 patterns.
5 bits can generate 2^5 = 32 patterns.
6 bits can generate 2^6 = 64 patterns.
7 bits can generate 2^7 = 128 patterns.

Pat looked at the exponents version of the patterns calculations then said "It looks like an easy way to calculate the number of patterns that a given number of bits can produce is to **raise 2 to that power**."

"Yes, this is an easy way to perform the calculation. Try using the exponent key on the calculator to determine how many patterns can be produced by 10 bits, 16 bits, and 20 bits."

Pat entered 2^{10} , 2^{16} , and 2^{20} and came up with the answers, 1024, 65535, and 1048576.

I have one more calculation I would like you to perform before we move on, Pat. How many patterns can be generated by 8 bits?"

Pat entered 2^8 into the calculator and received an answer of 256.

"256," said Pat "that number seems familiar to me."

"It should," I replied "what is the range of numbers that can be held in a typical computer's memory location?"

"0 through 255." said Pat. "255 is close to 256, but it is off by 1."

"How many patterns can be represented by the decimal numerals 1 through 255?" I asked.

Pat gave me a questioning look and then said "255."

"And if you add to these 255 patterns the pattern that is represented by the numeral 0, how many patterns does this make?"

"256 patterns!" cried Pat. "Do you mean that the reason that a memory location in a computer can only represent a number between 0 and 255 is because a memory location contains 8 bits?"

"Yes!" I replied. "The number of patterns that n bits can produce is 2^n

but the highest number that can be represented by n bits is $2^n - 1$ because 0 is always counted as one of the patterns. Memory locations can represent numbers between 0 and 255 because 8 bits, which are also called a **byte**, are capable of producing 256 patterns.

Hexadecimal Numerals

"Computers use binary numerals instead of decimal numerals Pat," I said "because binary numerals are easier to represent with transistors. There is a problem with binary numerals, however, when humans use them."

"What is the problem with them?" asked Pat.

"Let me illustrate the problem with an example." I said. "My telephone number is 740-555-3308, which is easy to remember, but this number encoded in binary is 0111010000000101010101010011001100001000. If you asked me for my telephone number and I said 'zero one one one zero one zero zero zero zero zero zero zero one zero one zero one zero one zero one zero one zero zero one one zero zero one one zero zero zero zero one zero zero zero', do you think you would be able to remember it?

"No way!" cried Pat. "I don't even think I could remember more than a few 1's and 0's!"

"Most humans are not very good at working with binary numerals, Pat, but humans are able to work with decimal numerals very well. It would be nice if decimal numerals were easy to convert into binary numerals and binary numerals were easy to convert into decimal numerals but this is not so."

Figure 16

Binary		Decimal
0000	-	0
0001	-	1
0010	-	2
0011	-	3
0100	-	4
0101	-	5
0110	-	6
0111	-	7
1000	-	8
1001	-	9
1010	-	10
1011	-	11
1100	-	12
1101	-	13
1110	-	14
1111	-	15

"No?" said Pat. "What is the problem?"

"Lets look again at the list of 4-bit binary numerals that are matched with the decimal numerals 0 - 15. The binary numerals start with 4 bits and they maintain a regular 4 bits through the end of the pattern. Something irregular happens, however, with the decimal numerals. Do you see what the irregularity is? (see Fig. 16)

Pat looked at the decimal digits then said "The decimal pattern starts with single digits up through 9 and then it switches to 2 digits after that. Is this the irregularity you are talking about?"

"Yes, this is the irregularity. It is caused by the fact that the base 10 numeral system is not a power of 2 and therefore base 2 numerals and base 10 numerals can not be converted into each other very easily. This caused a significant problem for the early computer programmers. What they needed was a numeral system that was similar enough to the decimal system so that humans could work with it easily, but it had to be a power of 2 so that it could be easily converted into binary numerals."

"How did they solve the problem?" asked Pat.

"There are 2 numeral systems that are a power of 2 that were likely candidates." I said. "The first one was the **base 8 or octal** numeral system and the second one was the **base 16 or hexadecimal** system. Both numeral systems were satisfactory for use with computers, but the hexadecimal numerals system became more widely used than the octal numeral system over time." I then added a column for the hexadecimal numeral system next to the binary and decimal columns and filled its numerals in as far as 9. I then added question marks for the remainder of the digits. (see Fig. 17)

Figure 17

Binary		Decimal		Hexadecimal
0000	-	0	-	0
0001	-	1	-	1
0010	-	2	-	2
0011	-	3	-	3
0100	-	4	-	4
0101	-	5	-	5
0110	-	6	-	6
0111	-	7	-	7
1000	-	8	-	8
1001	-	9	-	9
1010	-	10	-	?
1011	-	11	-	?
1100	-	12	-	?
1101	-	13	-	?
1110	-	14	-	?
1111	-	15	-	?

"What symbols come after 9 in the hexadecimal numeral system?" asked Pat.

"When the hexadecimal numeral system was first created, almost any symbols could have been chosen. It didn't take long, however, for the creators to realize that the number of symbols they had available to them was limited." I said.

"Why were they limited?" asked Pat.

I responded "What is the most widely used method for humans to input information into a computer?"

"A keyboard," answered Pat "oh, I see, they were limited by the symbols that are on a keyboard."

"Yes, they were." I replied. "They could have used special symbols like #, %, & and * but these would have looked confusing. Eventually they decided that it made the most sense to use the alphabetic characters A, B, C, D, E, and F because they were part of a sequence that people were already comfortable with." I then replaced the question marks on the whiteboard with these alphabetic characters. (see Fig. 18)

Figure 18

Binary		Decimal		Hexadecimal
0000	-	0	-	0
0001	-	1	-	1
0010	-	2	-	2
0011	-	3	-	3
0100	-	4	-	4
0101	-	5	-	5
0110	-	6	-	6
0111	-	7	-	7
1000	-	8	-	8
1001	-	9	-	9
1010	-	10	-	A
1011	-	11	-	B
1100	-	12	-	C
1101	-	13	-	D
1110	-	14	-	E
1111	-	15	-	F
10000	-	16	-	10
10001	-	17	-	11
10010	-	18	-	12
10011	-	19	-	13
10100	-	20	-	14
10101	-	21	-	15
10110	-	22	-	16
10111	-	23	-	17
11000	-	24	-	18
11001	-	25	-	19
11010	-	26	-	1A
11011	-	27	-	1B
11100	-	28	-	1C
11101	-	29	-	1D
11110	-	30	-	1E
11111	-	31	-	1F
100000	-	32	-	20
100001	-	33	-	21

What numeral comes after 'F' asked Pat.

"10." I replied. I then extended the table to 21 hexadecimal so Pat could better see how counting in binary and hexadecimal worked.

"So, it is easy to convert binary numerals to hexadecimal and hexadecimal numerals to binary?" asked Pat.

"Very easy." I said. "All you need to do is to use this table to either replace each hexadecimal digit with its 4-bit binary equivalent or to replace each set of 4-bit binary numerals with its hexadecimal equivalent. For example, to convert the binary numeral 111100101010010101111111010111 into hexadecimal, you start at the right side of the numeral then move to the left and break it into sets of 4 bits. If the leftmost group does not have enough bits to form a set of 4, add

0's to it from the left until a set of 4 is obtained. After the sets of 4 bits have been formed, just use the table to determine what hexadecimal digit is equivalent to each 4 bit set." I had wrote the binary number on the whiteboard as I said this and also performed the conversion process. (see Fig. 19)

Figure 19

0011	1100	1010	1001	0101	1111	1101	0111
3	C	A	9	5	F	D	7

"That was easy!" said Pat.

"The resulting hexadecimal number is also reasonably easy to remember. Say the number to yourself a few times and then close your eyes and try saying it aloud." I suggested.

Pat looked at the number for about 10 seconds then closed both eyes and said "3 C A 9 5 F D 7 hey, I did it!"

"It is also easy to convert from hexadecimal to binary. Lets do this with the hexadecimal number B 2 D 7 8 E 0 6. The conversion process is similar, except you replace each hexadecimal digit with its 4-bit binary equivalent." I then showed Pat how to do this on the whiteboard. (see Fig. 20)

Figure 20

B	2	D	7	8	E	0	6
1011	0010	1101	0111	1000	1110	0000	0110

"Now I know what those strange looking numbers were when we used the Dump command to look at the emulator's memory locations." said Pat. "They were the hexadecimal numeral equivalents of the binary numerals that were in each memory location."

"That is correct, Pat." I said. "The next time we meet, we will go back to the memory location dump and study it further."

Exercises

1) Convert the following binary numerals into hexadecimal:

a) 110010101100100101100101
b) 111000101
c) 00110011010100101
d) 0101101010101101010101

Note: you can check your answers in MathRider's MathPiper console using code similar to this:

ToBase(16,FromBase(2,"10101010"));

See the *MathRider For Newbies* book for instructions on how to use the MathPiper concole.

2) Convert the following hexadecimal numerals into binary:

a) BC2FD
b) C3
c) 16
d) F2C9

3) How may patterns can the following number of bits be formed into?

a) 6 bits
b) 11 bits
c) 16 bits
d) 32 bits

4) Recreate Figure 18 and extend it so that it runs to 11111111 binary, 255 decimal, and FF hexadecimal.

Note: you can test your answer by copying the following fold, pasting it into a .mrw MathRider worksheet file, and then executing it:

```
%mathpiper,description="Generate numerals between 0 and 255."
    nums := 0 .. 255;
```

```
    ForEach(dec,nums)
    [

       hex := ToBase(16,dec);

       bin := ToBase(2,dec);

       If(Length(bin) = 1, pre := "0000000");
       If(Length(bin) = 2, pre := "000000");
       If(Length(bin) = 3, pre := "00000");
       If(Length(bin) = 4, pre := "0000");
       If(Length(bin) = 5, pre := "000");
       If(Length(bin) = 6, pre := "00");
       If(Length(bin) = 7, pre := "0");
       If(Length(bin) = 8, pre := "");

       binWithLeadingZeros := ConcatStrings(pre,bin);

       Echo(binWithLeadingZeros, dec, hex);
       NewLine();
    ];

%/mathpiper
```