

6502 Emulator, Binary, And Hexadecimal

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/>

Table of Contents

MathRider and the U6502 Emulator.....	3
Using The U6502 Emulator.....	5
Decimal and Binary Numerals.....	10
The Number Of Patterns That Can Be Formed By N Bits.....	20
Hexadecimal Numerals.....	24
Exercises.....	29

1 MathRider and the U6502 Emulator

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

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

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

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

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

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

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

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

33 The file finished downloading and I had Pat create a directory on the hard

34 drive called pats_files and then unzip the file into it.

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

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

43 I replied "Java™ consists of a simulated CPU and a language which has been
44 specifically designed to run on this CPU. Another name for a simulated
45 CPU is a **virtual machine** and a CPU simulator that runs Java code is
46 called a **Java Virtual Machine** or **JVM**. Java was created in the early
47 1990s and it was one of the first software technologies that was specifically
48 designed for use in a globally-networked environment. One of Java's core
49 capabilities is to allow programs to be compiled into the JVM's machine
50 language (which are called Java bytecodes) and then these programs can be
51 executed on any devices that has a JVM on it."

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

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

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

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

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

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

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

79 I laughed too!

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

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

85 **Using The U6502 Emulator**

86 After MathRider was installed, I ran it.

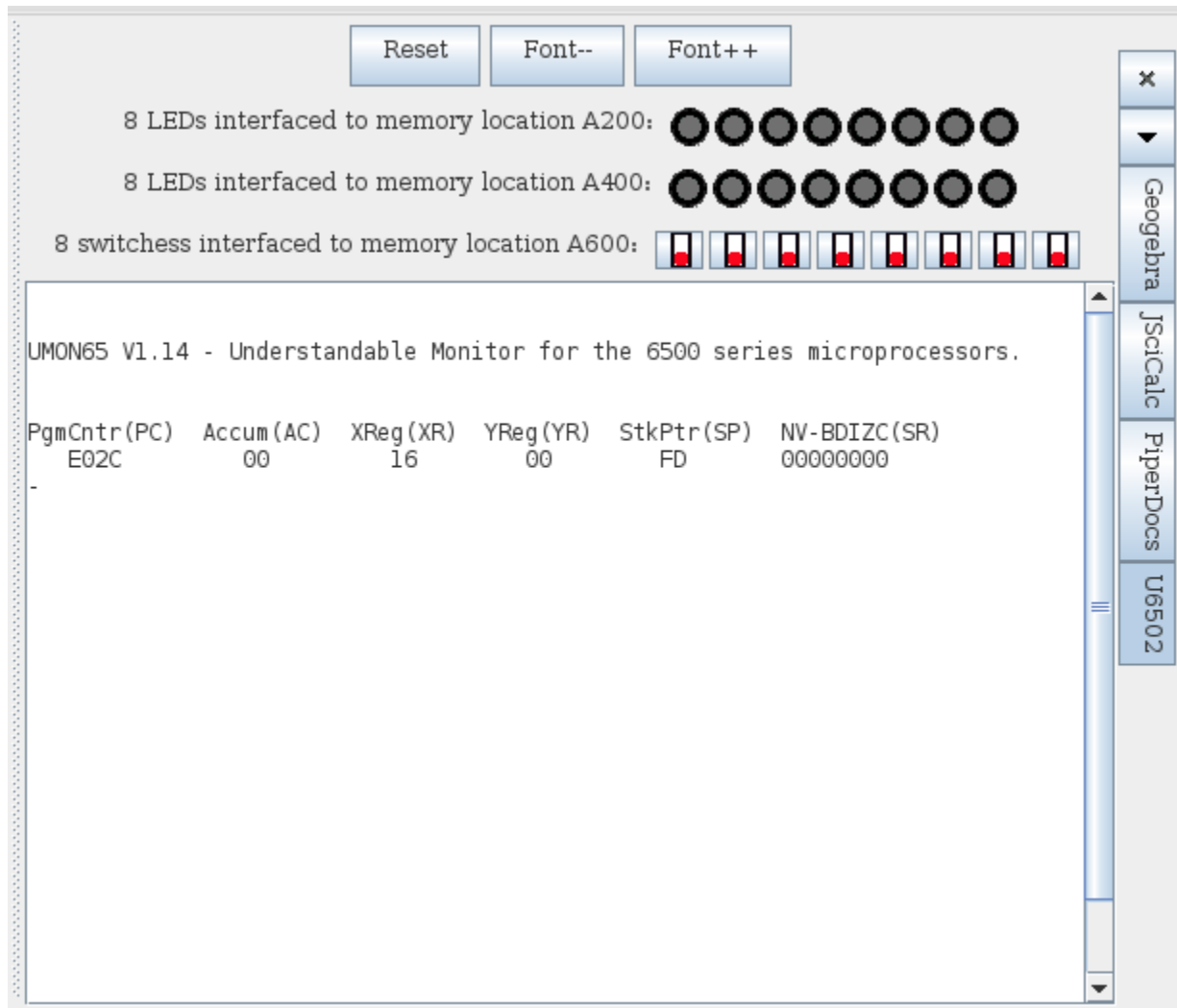
87 "Wow!" said Pat "MathRider looks cool! What can it do?"

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

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

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

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



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

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

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

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

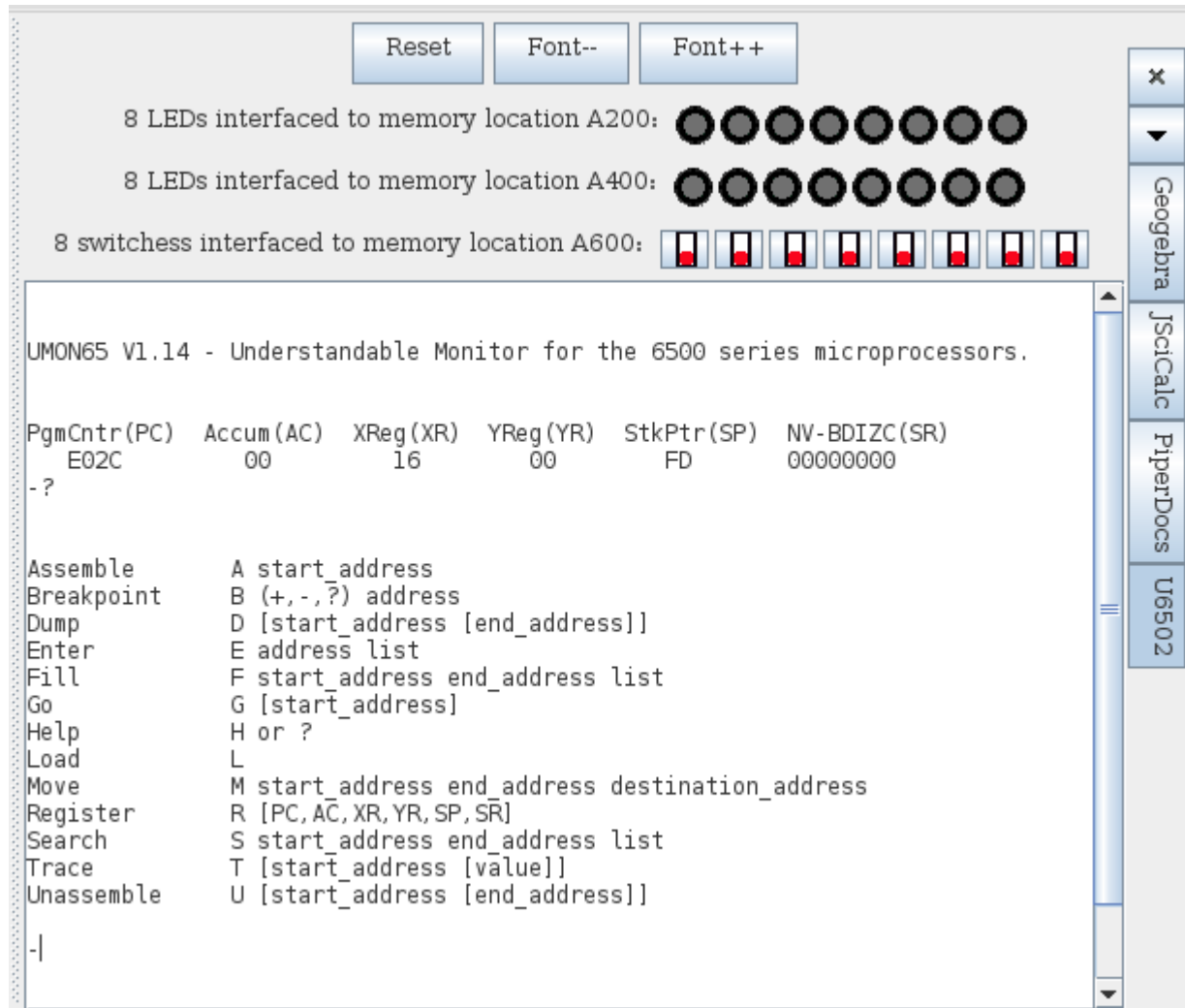
115 Pat studied the emulator's GUI for a while then said "The first device
116 contains 8 switches, the second device contains 8 LEDs and the third device
117 also contains 8 LEDs. Is there a reason why these devices all have 8 lights
118 or switches in them?"

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

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

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

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



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

133 ?

```
134 Assemble      A start_address
135 Breakpoint     B (+,-,?) address
136 Dump           D [start_address [end_address]]
137 Enter          E address list
138 Fill           F start_address end_address list
139 Go             G [start_address]
140 Help           H or ?
141 Load          L
142 Move           M start_address end_address destination_address
143 Register       R [PC,AC,XR,YR,SP,SR]
144 Search         S start_address end_address list
145 Trace         T [start_address [value]]
146 Unassemble    U [start_address [end_address]]
```

147 -

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

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

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

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

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

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

167 -d e000 e090

```
168 E000 4C 1B E0 4C B5 F3 4C 74 - F3 4C 97 F3 4C 57 F3 4C L..L..Lt.L..LW.L
169 E010 A7 F2 4C 03 F3 4C 41 EA - 4C 70 E0 A2 FF 9A D8 20 ..L..LA.Lp.....
170 E020 37 F3 4C 25 E0 A2 16 A0 - F4 20 57 F3 00 C9 02 D0 7.L%..... W.....
171 E030 13 29 F0 4A 4A 4A 18 69 - C0 8D 09 00 A9 00 8D 08 .).JJJ.i.....
172 E040 00 6C 08 00 4C 47 E0 A0 - 00 8C 44 00 A2 64 C8 D0 .l..LG....D..d..
173 E050 FD CA D0 FA EE 44 00 AD - 44 00 4C 4C E0 20 69 E0 .....D..D.LL. i.
174 E060 20 DB E0 20 33 E1 4C 5D - E0 A2 66 A0 F4 20 57 F3 .. 3.L]..f.. W.
175 E070 A0 00 8C 5E 00 8C CD 00 - 8C 5F 00 8C 73 00 8C 87 ...^....._...s...
176 E080 00 A0 00 A9 00 99 A5 00 - C8 C0 27 D0 F8 A0 00 20 ....._'.....
177 E090 97
178 -
```

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

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

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

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

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

188 Decimal and Binary Numerals

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

192 "Hmmm..." said Pat. "I know that a number is used to indicate the amount of
193 something and that numerals have something to do with digits like 1, 2 and
194 3, but I am not sure exactly what the difference is between a number and a
195 numeral."

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

202 **mathematics** and the deeper one goes into the field of computing, the
203 more mathematics one needs to know."

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

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

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

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

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

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

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

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

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

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

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

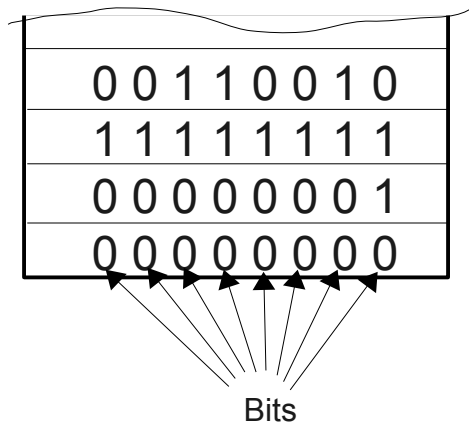
251 Figure 1

252	
253	
254	off off on on off off on off
255	on on on on on on on on
256	off off off off off off off on
257	off off off off off off off off
258	
259	
260	
261	

"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

262 what the memory locations I just drew look like when the words 'on' and
263 '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?"

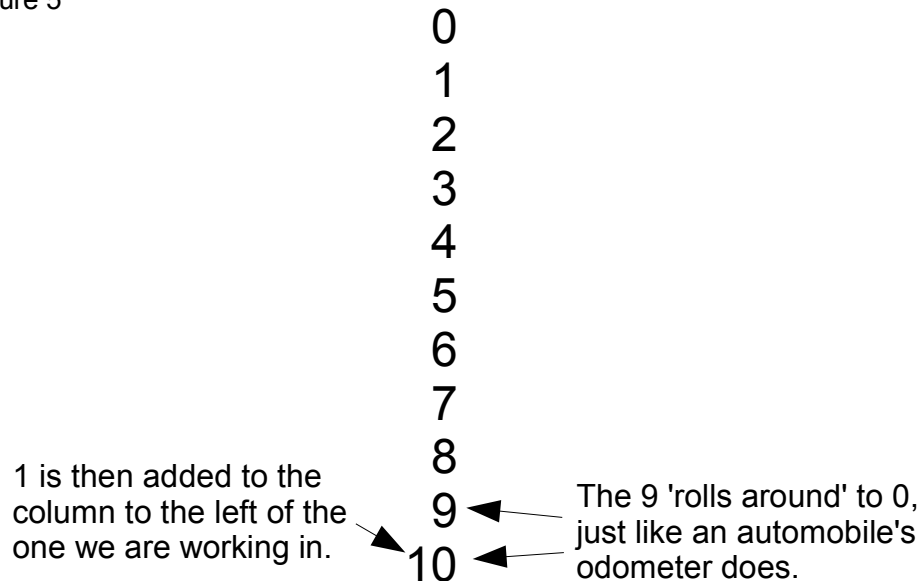
Pat thought about this question for a while then said "I am not quite sure because we have not counted far enough for me to see if there is a pattern to the sequence."

"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

302 wrote the decimal digits 0 through 9 on another area of the whiteboard and
303 said the name of each digit as I wrote it "zero, one, two, three, four, five,
304 six, seven, eight, nine, ?... ummm, I have a problem Pat. I have run out of
305 decimal digits and I do not know what to do next!" (see Fig. 4)

306 Figure 4 0 Pat gave me a strange look then said "Of course, the
307 1 next numeral in the sequence is 10. Why didn't you
308 2 just write a 10 and continue?"
309 3 "Tell me what rule you followed to go from 9 to 10
310 4 and I will." I replied.
311 5 "Well," said Pat "when you need to add one more to
312 6 the column you are in, but you have run out of digits
313 7 in that column, you have the column go back to 0 and
314 8 then add one to the digit in the column which is to the
315 9 immediate left of the one you are working on."
316 ? "And what are the names that are given to these
317 columns in the decimal system?" I asked.
318 "The initial column is the ones column, to its left is
319 the tens column and to the tens column's left is the hundreds column. Each
320 time you move left a column, that column represents a place value that is
321 ten times larger than the column you are in."
322 "Very good Pat." I said. "Lets see if your counting rule solves my problem.
323 Seven, eight, nine, I have run out of decimal digits in the ones column so I
324 roll from 9 around to 0 in the ones column, just like an odometer in an
325 automobile does, and then I add one to the column immediately to the left,
326 which is the tens column." (see Fig 5)

Figure 5



327 "Your counting rules seems to work with the decimal system, lets see what
 328 happens when we apply them to the binary system. 0000, 0001, I just ran
 329 out of binary digits in the first column, so I 'roll' it around to 0 and add one
 330 to the column which is immediately to its left. The new pattern is 0010 and
 331 this is correct. Now, what pattern do you think comes next to match the
 332 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

342 and also added the numeral 4 in the next row in the decimal column. (see
 343 Fig. 7)

344
345

Figure 7

Binary Decimal

346
347
348
349
350
351
352
353
354

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

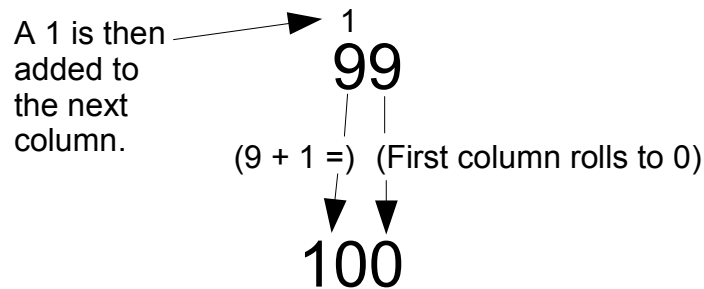
"What pattern comes next to match the decimal numeral 4, Pat?" I asked.

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."

355 Pat said "The 9 in the ones column rolls around to 0 and one is added to the
356 9 that is in the tens column. This makes the tens column also roll around to
357 0 and a 1 is added to the 100s column resulting in the next decimal numeral
358 in the sequence being 100."

359 "Very good Pat" I said as I wrote the numeral 100 under the 99 on the
360 whiteboard. (see Fig. 8) Now apply the same thinking to the binary numeral
361 0011."

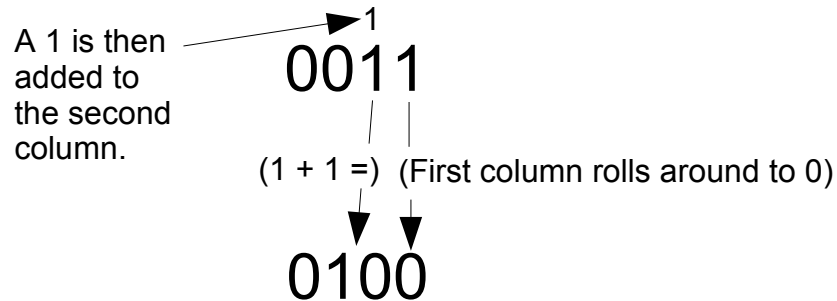
Figure 8



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

365 "1 + 1 in binary is 10 binary." I said. "After we perform this addition in the
366 second column the result is 0100 (see Fig. 9).

Figure 9



367 I then filled in the rest of the binary numerals up through 1111 using these
 368 counting rules and matched them with their decimal equivalents (see Fig 10
 369)

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.

391
392
393
394
395
396
397
398

[illegible]

"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 12

	Binary	Decimal
	00	
Column 2	00	
	01	
	01	
	00	
	00	
	11	
	11	
	00	
	00	
	01	
	01	
	00	
	00	
	11	
	11	

Figure 13

	Binary	Decimal
	0	
Column 3	0	
399	0	
400	0	
401	1	
402	1	
403	1	
404	1	
405	1	
406	0	
407	0	
408	0	
409	0	
410	1	
	1	
	1	
	1	
	1	

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

	Binary	Decimal
	0	
Column 4	0	
	0	
	0	
	0	
	0	
	0	
	0	
	0	
	0	
	1	
	1	
	1	
	1	
	1	
	1	
	1	
	1	
	1	

411 The Number Of Patterns That Can Be Formed By N Bits

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

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

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

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

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

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

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

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

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

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

432 Pat used 3 fingers and counted the following patterns: 000, 001, 010, 011,
433 100, 101, 110, and 111.

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

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

439 "I think so." replied Pat. Using 4 fingers, Pat slowly formed them into the
440 following patterns: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000,
441 1001, 1010, 1011, 1100, 1101, 1110, and 1111. "4 bits can be formed into
442 16 patterns!"

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

Figure 15

		4 bits can be formed into 16 patterns:
1 bit can be formed into 2 patterns:	0 1	0000
		0001
2 bits can be formed into 4 patterns:	00 01 10 11	0010
		0011
		0100
		0101
		0110
		0111
		1000
		1001
3 bits can be formed into 8 patterns:	000 001 010 011 100 101 110 111	1010
		1011
		1100
		1101
		1110
		1111

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

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

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

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

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

458 "7 bits?" asked Pat. "Wow, I am going to have to think about that one. I
459 suppose we could use the pattern doubling method we just discussed, but it
460 would be nice if there were an easier way to figure it out."

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

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

465 "1 single bit is able to produce 2 patterns. Enter a single number 2 into the
466 calculator and press the equals button. What is the answer?"

467 "2." said Pat.

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

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

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

474 1 bit can generate 2 patterns.

475 2 bits can generate $2 \times 2 = 4$ patterns.

476 3 bits can generate $2 \times 2 \times 2 = 8$ patterns.

477 4 bits can generate $2 \times 2 \times 2 \times 2 = 16$ patterns.

478 5 bits can generate $2 \times 2 \times 2 \times 2 \times 2 = 32$ patterns.

479 6 bits can generate $2 \times 2 \times 2 \times 2 \times 2 \times 2 = 64$ patterns.

480 7 bits can generate $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 128$ patterns.

481 "Notice that 2×2 is the same as 2 to the 2nd power 2^2 , $2 \times 2 \times 2$ is the
482 same as 2^3 , and so on." I then created the following exponents version of
483 Pat's patterns calculations."

484 1 bit can generate $2^1 = 2$ patterns.

485 2 bits can generate $2^2 = 4$ patterns.

486 3 bits can generate $2^3 = 8$ patterns.

487 4 bits can generate $2^4 = 16$ patterns.

488 5 bits can generate $2^5 = 32$ patterns.

489 6 bits can generate $2^6 = 64$ patterns.

490 7 bits can generate $2^7 = 128$ patterns.

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

494 "Yes, this is an easy way to perform the calculation. Try using the exponent
495 key on the calculator to determine how many patterns can be produced by
496 10 bits, 16 bits, and 20 bits."

497 Pat entered 2^{10} , 2^{16} , and 2^{20} and came up with the answers, 1024,
498 65535, and 1048576.

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

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

502 "256," said Pat "that number seems familiar to me."

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

505 "0 through 255." said Pat. "255 is close to 256, but it is off by 1."

506 "How many patterns can be represented by the decimal numerals 1 through
507 255?" I asked.

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

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

511 "256 patterns!" cried Pat. "Do you mean that the reason that a memory
512 location in a computer can only represent a number between 0 and 255 is
513 because a memory location contains 8 bits?"

514 "Yes!" I replied. "The number of patterns that n bits can produce is 2^n
515 but the highest number that can be represented by n bits is $2^n - 1$ because

516 0 is always counted as one of the patterns. Memory locations can represent
517 numbers between 0 and 255 because 8 bits, which are also called a **byte**,
518 are capable of producing 256 patterns.

519 **Hexadecimal Numerals**

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

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

524 "Let me illustrate the problem with an example." I said. "My telephone
525 number is 740-555-3308, which is easy to remember, but this number
526 encoded in binary is 01110100000001010101010011001100001000. If
527 you asked me for my telephone number and I said 'zero one one one zero
528 one zero zero zero zero zero zero zero one zero one zero one zero one zero
529 one zero one zero zero one one zero zero one one zero zero zero zero one
530 zero zero zero', do you think you would be able to remember it?

531 "No way!" cried Pat. "I don't even think I could remember more than a few
532 1's and 0's!"

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

Figure 16 Binary Decimal

537 0000 - 0 "No?" said Pat. "What is the problem?"
538 0001 - 1
539 0010 - 2 "Lets look again at the list of 4-bit binary
540 0011 - 3 numerals that are matched with the decimal
541 0100 - 4 numerals 0 - 15. The binary numerals start with 4
542 0101 - 5 bits and they maintain a regular 4 bits through
543 0110 - 6 the end of the pattern. Something irregular
544 0111 - 7 happens, however, with the decimal numerals. Do
 1000 - 8 you see what the irregularity is? (see Fig. 16)
545 1001 - 9
546 1010 - 10 Pat looked at the decimal digits then said "The
547 1011 - 11 decimal pattern starts with single digits up
548 1100 - 12 through 9 and then it switches to 2 digits after
549 1101 - 13 that. Is this the irregularity you are talking
 1110 - 14 about?"
550 1111 - 15
551 " Yes, this is the irregularity. It is caused by the
552 fact that the base 10 numeral system is not a
553 power of 2 and therefore base 2 numerals and
554 base 10 numerals can not be converted into each other very easily. This
555 caused a significant problem for the early computer programmers. What
556 they needed was a numeral system that was similar enough to the decimal
557 system so that humans could work with it easily, but it had to be a power of
558 2 so that it could be easily converted into binary numerals."

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

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

594	0000	-	0	-	0	What numeral comes after 'F' asked Pat.
595	0001	-	1	-	1	
	0010	-	2	-	2	"10." I replied. I then extended the table to 21 hexadecimal so Pat could better see how counting in binary and hexadecimal worked.
596	0011	-	3	-	3	
597	0100	-	4	-	4	
598	0101	-	5	-	5	
599	0110	-	6	-	6	
	0111	-	7	-	7	"So, it is easy to convert binary numerals to hexadecimal and hexadecimal numerals to binary?" asked Pat.
600	1000	-	8	-	8	
601	1001	-	9	-	9	
602	1010	-	10	-	A	
603	1011	-	11	-	B	"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
604	1100	-	12	-	C	
605	1101	-	13	-	D	
606	1110	-	14	-	E	
607	1111	-	15	-	F	
608	10000	-	16	-	10	
609	10001	-	17	-	11	
610	10010	-	18	-	12	
611	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	

612 11110010101001010111111010111 into hexadecimal, you start at the
 613 right side of the numeral then move to the left and break it into sets of 4
 614 bits. If the leftmost group does not have enough bits to form a set of 4, add

615 0's to it from the left until a set of 4 is obtained. After the sets of 4 bits have
616 been formed, just use the table to determine what hexadecimal digit is
617 equivalent to each 4 bit set." I had wrote the binary number on the
618 whiteboard as I said this and also performed the conversion process. (see
619 Fig. 19)

Figure 19

00	11	11	00	10	10	10	01	01	01	11	11	11	01	01	11
3	C	A	9	5	F	D	7								

620 "That was easy!" said Pat.

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

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

626 "It is also easy to convert from hexadecimal to binary. Lets do this with the
627 hexadecimal number B 2 D 7 8 E 0 6. The conversion process is similar,
628 except you replace each hexadecimal digit with its 4-bit binary equivalent."
629 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

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

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

636 **Exercises**

637 1) Convert the following binary numerals into hexadecimal:

638 a) 110010101100100101100101

639 b) 111000101

640 c) 00110011010100101

641 d) 0101101010101101010101

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

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

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

647 2) Convert the following hexadecimal numerals into binary:

648 a) BC2FD

649 b) C3

650 c) 16

651 d) F2C9

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

653 a) 6 bits

654 b) 11 bits

655 c) 16 bits

656 d) 32 bits

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

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

```
661 %mathpiper,description="Generate numerals between 0 and 255."  
662     nums := 0 .. 255;
```

```
663   ForEach(dec,nums)
664   [
665
666       hex := ToBase(16,dec);
667
668       bin := ToBase(2,dec);
669
670       If(Length(bin) = 1, pre := "0000000");
671       If(Length(bin) = 2, pre := "000000");
672       If(Length(bin) = 3, pre := "00000");
673       If(Length(bin) = 4, pre := "0000");
674       If(Length(bin) = 5, pre := "000");
675       If(Length(bin) = 6, pre := "00");
676       If(Length(bin) = 7, pre := "0");
677       If(Length(bin) = 8, pre := "");
678
679       binWithLeadingZeros := ConcatStrings(pre,bin);
680
681       Echo(binWithLeadingZeros, dec, hex);
682       NewLine();
683   ];
684   %/mathpiper
```