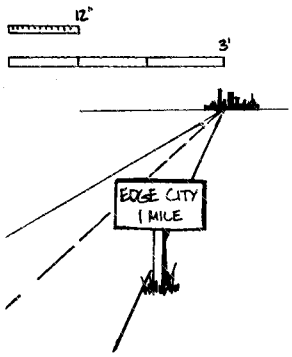
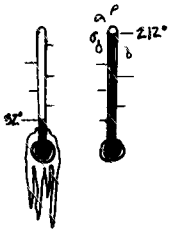
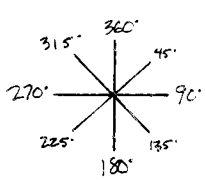
2.







Welcome to the subject that strikes unreasoning terror in the heart of every programming novice — numbers and number systems. There are many opinions about computer number systems. Here's mine: if you don't learn them, you'll end up hacking your way through assembly language programming. You'll never feel comfortable or competent doing it.

That said, I'll start by telling you that I'm no mathematician. Numbers make me cringe. Yet binary and hexadecimal computer representation are really easy. Partly that's because I found that, when ordinary sheep failed me, counting backwards hexadecimal sheep jumping a fence put me to sleep before I reached zero. I'm not making it up.

Seriously, computer number systems have been made frightening by obscure use of language and knot-headed programmers. In truth, we live and live well in a world of non-decimal number systems. Here's a short list:

12 inches to a foot, 3 feet to a yard
5280 feet to a mile
32 degrees is freezing, 212 degrees is boiling
60 minutes in a hour, 24 hours in a day
7 days in a week, 52 weeks in a year
365 days in a year, but 360 degrees in a circle
3 teaspoons to a tablespoon, 4 quarts to a gallon
30 days hath September, April, June, and so forth

There are dozens, acres, ounces and hundreds of other examples. All are the daily measurements of our bread and butter, our life and time. All are irregular ways of numbering, but few confuse us. Chances are you can identify every one of these groups of numbers:

727, 737, and 747	98.6
33, 45 and 78	3.1416

You have finished the first lesson. The programmed learning section of that lesson was simple and repetitive; all of the programmed learning is somewhat repetitive, but as you go, the pace will begin to quicken. Also, the questions in this lesson will assume you know the material in the first lesson. Much of the groundwork in assembly language is rote learning, just like memorizing times tables, so keep up with the programmed learning questions.

* Different number systems are our heritage. How many cards in a poker deck? How many weeks in a year?

52.

* How many cards in a suit? How many weeks in a quarter?

4.

* How many spots in a card deck? How many days in a year?

365.

* How many face cards in a deck? How many months in a year?

12.

* Is the decimal system used for computer operations?

No.

* Why aren't decimal numbers used for computer number systems?

Because the activities the numbers represent would be clumsy or make little sense in decimal.

* What number system is used for computers?

The binary system.

* How many numbers can be represented by the binary system?

2 numbers.

* What are the names of the two numbers represented by the binary system.

The binary numbers are 0 and 1.

* Name another pair of conditions that can be represented by the binary system.

On and off.

* Name some other opposite pairs of conditions that might be represented by the binary system (there are many correct answers to this question).

High and low; in and out; forward and backward; red and green; and so forth.

* Was Dr. Watson a medical doctor?

Yes. He was a general practitioner. This question in no way relates to the discussion of number systems.

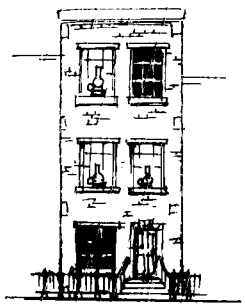
What I guess came to mind were airplanes, records, normal body temperature, and pi. My point is that this is a conceptual issue. These are not numbers, they're representations of something useful in real life. And computer numbers are conceptual, too. The metric system is official in the U.S., but how much use does it get? Perhaps it too is a conceptual issue. I know how long a centimeter is, but can't convert from feet to meters. Same with a liter. Now that soft drinks come in liter bottles, I finally know what one is. Never could make a mathematical conversion of it, though. I can tell an acre, even though I don't know its actual measurement. In other words, once a number is represented by something in "real life", so to speak, I can make sense of it.

Basically that's what computer number systems are all about — they represent activities that are clumsy or make little sense when described by "regular" decimal numbers.

Keep that in mind. By now you're probably familiar with that old standard, the light-switch analogy. Computers, it's been said, operate using electronic switches that are either on or off, just like light switches. That's two conditions — the binary system, it's called. On or off.

Such a description is true as far as it goes, but it leaves out a lot. To cast a different light on the binary system, I've enlisted the help of two old friends, Sherlock Holmes and Doctor Watson, who will discover some clues in this slightly rewritten scene . . .

Watson: . . . but it's just someone turning the lamps on, Holmes. It's past dusk, after all.



Holmes: Ah, yes, Watson, but why would someone light a lamp and extinguish it so quickly? And move from room to room? Eh, Watson?

Watson: Perhaps they're looking for something they can't find.

Holmes: Or perhaps they're signaling someone. A cipher of some kind, I would say.

Watson: With lamps in five windows? Nonsense, Holmes!

Holmes: Just copy this down, Watson. I'll read starting from the uppermost window. Lamp on, lamp off, lamp on, lamp on. . .

Watson: Slow down, Holmes.

Holmes: Keep at it, Watson, they won't stop for your fingers. Lamp off. They're changing now. Again, from the

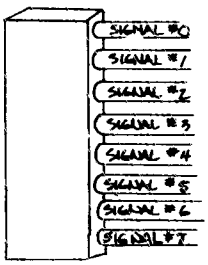
uppermost. Off, on, off, off, on... I've got it, Watson! These are letters of the alphabet. Five windows create 32 combinations, enough for all the letters of the alphabet.

Watson: Amazing, Holmes!

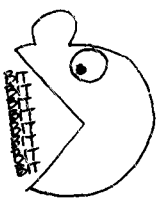
Holmes: You don't need to write down the lamps now, Watson. I think I can read the message. S – E – E M – E A – T . . . See me at the August Lion Tavern at 6 o'clock. That's it! We have just five minutes. Come on, Watson!. . . .

What Holmes discovered, of course, was that by using the most basic information — a simple pattern of lamps lit or extinguished — a complete message might be sent and received. Morse formalized that with his telegraph code. In this case, Holmes perceived quickly and correctly that with five lamps, 32 combinations were possible by rearranging the pattern of lamps lit and darkened.

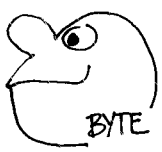
You will find that computers are really quite simple-minded devices. You're dealing with nothing more than a vast but microscopic nest of electronic switches. There's no intelligence involved — just an impulse here, an impulse there, all moving very fast. For reasons that have more to do with manufacturing economy than anything else, the decision to use the on-off switch was chosen over something more familiar like a decimal type counter. Programming might have been much easier otherwise. But, cheap as it was to manufacture, the on-off idea limited each meaningful computer signal to those two conditions alone. For more conditions — larger numbers, that is — more on-off signals are needed. Groups of signals, all working simultaneously, like Holmes's five lamps.



Everything in computers began to take on the color of two choices, base 2, the binary system. Data was parceled out in base 2, and grouped in powers of 2. The first microprocessor device used four simultaneous signals for transmission of data. The 6809 uses eight signal lines. Newer, more sophisticated computers use 16 or more concurrent on-off signals.



You can probably guess I'm taking you easy into this. But stay with me. If you think back to Holmes's discovery, you'll remember that the operant concept was not the number, but rather the pattern of lamps. The patterns represented codes for letters — an inspired idea from the time Morse developed his telegraph code to the present day American Standard Code for Information Interchange (ASCII).



In computers, these are patterns of binary signals, thought of as binary numbers or binary digits. Binary digit is conveniently abbreviated "bit". So when the 6809 is called an 8-bit processor, that means that all its information is created from the combinations of eight binary digits. Here's the grabber: no matter what the information

* How many combinations did Holmes figure could be made from five lamps?

32 combinations.

* How many lamps would produce only 16 different patterns?

Four lamps.

* How many different combinations would Holmes have discovered if there were six lamps?

64 combinations.

* How many different combinations would Holmes have calculated from eight lamps?

256 combinations.

* Write down the powers of 2 from 2^1 to 2^8.

2, 4, 8, 16, 32, 64, 128, 256

* What number system is used in computers?

The binary system.

* How many different numbers can be represented by the binary system, and what are they?

2 numbers; they are 0 and 1.

* How many different combinations can be formed from eight on-or-off lamps? From eight one-or-zero binary digits?

Both answers are the same: 256 combinations.

* What does bit mean?

Bit means binary digit.

* How many binary digits (bits) are used by the 6809 processor to represent information?

Eight bits.

* How many different combinations can be formed from eight bits?

256 combinations.

* How many different combinations of binary digits can the 6809 processor produce from its 8 bits?

256 combinations.

* How does the 6809 processor distinguish letters, commands, display, sound and other purposes of the 256 combinations of 8 binary digits?

By the context in which those digits are presented.

* What number system is used in computers?

The binary number system is used in computers.

* What counting system is used for clarity in discussing computer numbers?

The hexadecimal counting system is used for clarity.

* How many numbers are represented by the hexadecimal counting system?

16 numbers are represented by the hexadecimal counting system.

represents — letters, numbers, commands, display, sound, whatever — it is formed by some pattern of those eight binary digits, formed from those eight bits. The microprocessor, the computer's heart, can know the difference only by the context in which those digits are presented.

If that seems far-fetched, consider that there are only 26 letters in the alphabet, 10 numerical symbols, and a dozen or so punctuation marks. Those letters, in specific combinations and contexts, make up the half-million or so words in the English language. Those same letters, combined into words and melded through punctuation into sentences and paragraphs, can describe the entire known history of humanity with multiple levels of nuance, politics, or poetry. Quite a bit from a simple 26-letter code.

At last it's time to get down to specifics, and deal with those numerical symbols. The trick is for you to gain an appreciation of the computer number system that's used exclusively for clarity. It's called hexadecimal. Base 16. Don't run for the Maalox. Keep in mind that we're not talking about counting-type numbers here, but simply representations, symbolic abstractions.

First, there's a program to get up and running.

Program #4, a BASIC program. Turn on the power of your Extended Color BASIC computer. When the cursor appears, type CLOAD and press ENTER. The computer will search (S) and find(F). When the cursor reappears, LIST this program. If the program is not similar to the listing, or if an I/O error occurs, rewind to the start of the program and try again. For severe loading problems, see the Appendix.

```
10 CLS:L=164
20 PRINT"CONVERSION:":PRINT"FROM DECIMAL TO BINARY"
30 FORB=1TO5
40 PRINT:INPUT"NUMBER 0 TO 255":X
50 IFX(0ORX)255THENPRINT"OUT OF RANGE":GOSUB290:GOTO40
60 GOSUB320
70 PRINTC:E:G:I:K:M:O:Q:
80 NEXT
90 PRINT:INPUT"(ENTER) TO CONTINUE":A$
100 CLS
110 PRINT:PRINT"DECIMAL AND BINARY DO NOT BEAR  A DISTINCT VISUA
L RELATIONSHIP:"
120 FORX=0TO255
130 GOSUB320
140 PRINT@256,X;:PRINT@266,"(----DECIMAL"
150 PRINTC:E:G:I:K:M:O:Q:
160 NEXT
170 CLS
180 PRINT:PRINT"HEXADECIMAL AND BINARY SHOW A   CLEAR VISUAL REL
ATIONSHIP.  THE FOUR BINARY DIGITS CREATE 16    PATTERNS.  EACH
BINARY PATTERN  IS IDENTIFIED BY A UNIQUE       HEXADECIMAL SYMB
OL FROM 0 TO F."
190 FORX=0TO255
200 X$=HEX$(X)
210 IFLEN(X$)=1THENX$="0"+X$
220 PRINT@260,"...."LEFT$(X$,1)"....  ...."RIGHT$(X$,1)"...."
230 GOSUB320
240 PRINT"   "C;E;G;I;K;M;O;Q
250 PRINT"    ........  ........"
```

```
260 NEXT
270 FORX=1TO300:NEXT
280 STOP
290 FORN=1TO1000:NEXT:RETURN
300 X=C*128+E*64+G*32+I*16+K*8+M*4+O*2+Q
310 RETURN
320 C=INT(X/128):D=C*128
330 E=INT((X-D)/64):F=E*64
340 G=INT((X-D-F)/32):H=G*32
350 I=INT((X-D-F-H)/16):J=I*16
360 K=INT((X-D-F-H-J)/8):L=K*8
370 M=INT((X-D-F-H-J-L)/4):N=M*4
380 O=INT((X-D-F-H-J-L-N)/2):P=O*2
390 Q=INT(X-D-F-H-J-L-N-P)
400 RETURN
```

I've been talking about symbols, relationships and legibility. I'm also talking about memorizing patterns for instant recognition. You're about to run a program which will show all 256 rearrangements of eight binary digits, represented as a string of ones and zeros. Run this program now. Enter a number from 0 to 255, and check out the binary equivalent printed below. Enter another number, and look. Enter three more numbers, and examine the binary equivalents. Chances are, what you see is not very useful. Hit <ENTER>. The decimal values from 0 to 255 will be displayed in order, together with their binary equivalents.

The decimal numbers count up nicely from 0 to 255, and the binary numbers also follow a regular pattern. That binary counting-up pattern probably isn't familiar to you yet, but there are lots of ways of understanding it. For example, as you watch, notice that the right-hand digit alternates quickly between 1 and 0. Its neighbor's alternation takes twice as long, and its neighbor's alternation in turn takes twice as long as that. There's that binary, base 2 system working again. The binary counting is useful to watch; try to get familiar with it.

When the decimal counting is finished, the program will show you a much easier system. Mentally break that eight-bit binary group into two halves. Remembering Sherlock Holmes' discovery, you can see that the four binary digits in each group can be rearranged 16 different ways. Instead of trying to recall lines of ones and zeros, though, each arrangement can be identified with a single, unique character. The arrangement **0000** can be identified as 0. **1** can be identified as 1. **0010** becomes 2, **0011** becomes 3, **0100** becomes 4, on up to **1001**, which is called 9. **1010** is labeled A, **1011** is labeled B, **1100** is labeled C, **1101** is D, **1110** is E, and **1111** is F.

```
0000  0
0001  1
0010  2
0011  3
0100  4
0101  5
0110  6
0111  7
1000  8
1001  9
1010  A
1011  B
1100  C
1101  D
1110  E
1111  F
```

As you watch the screen, you will notice that a separate symbol — a hexadecimal symbol — is used for each half of the 8-bit group. That gives you an easy-to-handle two-digit reference for each long binary number from **00000000** to **11111111**.

The advantage of this method is very real. By knowing a binary number, you can almost instantaneously know the hexadecimal equivalent. By knowing the hexadecimal

```
* If you are working with eight binary digits, what is the binary equivalent of decimal number 1?

00000001 is the 8-bit binary equivalent of the decimal number 1.

* If you are working with eight binary digits, what is the binary equivalent of decimal number 255?

11111111 is the 8-bit equivalent of decimal number 255.

* What is the hexadecimal symbol for decimal number 1?

1 is the hexadecimal symbol for decimal number 1.

* What decimal number is represented by binary number 00001111?

00001111 is the decimal number 15.

* What hexadecimal number represents binary number 0000?

Hexadecimal number 0 represents binary 0000.

* What hexadecimal number represents binary number 1111?

Hexadecimal number F represents binary 1111.

* What hexadecimal number represents binary number 00001111?

Hexadecimal number 0F represents binary 00001111.

* Count the binary numbers from 0000 to 1111.

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
```

Reading Hex

* Count the hexadecimal numbers from 0 to F.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

* Count backwards in hexadecimal numbers from F to 0.

F, E, D, C, B, A, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

* What binary number is represented by the hexadecimal number C?

Hexadecimal C is binary 1100.

* What is the shorthand word for hexadecimal?

The shorthand word for hexadecimal is hex.

* Count aloud quickly from hex 20 to hex 30.

Two-zero, two-one, two-two, two-three, two-four, two-five, two-six, two-seven, two-eight, two-nine, two-A, two-B, two-C, two-D, two-E, two-F, three-zero.

* What symbol is used to indicate a hex number?

The dollar sign.

* What is the hexadecimal number for binary 1100?

The hexadecimal number is $C.

* Count aloud quickly, backwards in hex from $FF to $E8.

FF, FE, FD, FC, FB, FA, F9, F8, F7, F6, F5, F4, F3, F2, F1, F0, EF, EE, ED, EC, EB, EA, E9, E8.

* What is the binary number for hexadecimal $AA?

Hexadecimal A is binary 1010, so hex $AA must be 10101010.

representation, you can get at the binary equivalent at any time. Remember, these microprocessors work in a binary world. Knowing that world is essential for you as the programmer to call the shots.

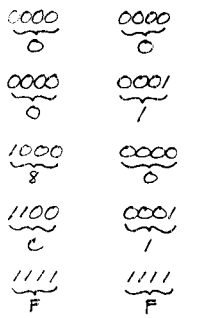
How can you learn the hexadecimal numbers? Memorize them. Just like the times tables in elementary school. Count sheep, forwards and backwards. Go back to this program and RUN170. Read the numbers aloud. By the way, hexadecimal numbers — you'll just call them hex after a while — are read a little different from decimal numbers. If there's a leading zero, for example, you hang onto it. Like this:

00 01 (not just "one") **02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10** (not "ten") **11** (not "eleven") **12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20** (not "twenty") and so on. Remember than when you get up to **9A 9B 9C 9D 9E 9F**, the next hex number is **A0**. It goes all the way up to **FF**.

Another convention for hexadecimal numbers is their written form. The letters are always uppercase, and in order to distinguish hex from decimal, it's common practice to put a dollar sign in front of a hexadecimal number.

You should also try to learn your hex numbers backwards. Assembly language has certain kinds of program activities that move backwards, and being able to make an accurate backward count — **FF FE FD FC FB FA F9 F8** etc. — will ease this process.

As far as converting from decimal to hex and vice versa, you'll do it occasionally. Use a chart, a special calculator like the Texas Instruments programmer, or a formula. When you learn to use the editor/assembler/debugger programs, much of this conversion is done by the assembler itself. For the moment, learn to recognize the four-bit binary patterns and their hex equivalents. In fact, you might take a break from this tape right now to practice binary and hexadecimal patterns.



HEX 76 =
0111 0110

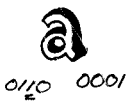
100
FF
FE
FD
FC
FB
FA
F9
F8
F7
F6
F5

Hexadecimal numbers will be used for the remainder of this series. Please practice the hexadecimal numbers patterns and return to the tape when you can recognize the four-bit binary patterns and their hexadecimal equivalents.

Since this is a lesson about numbers and codes, I'd like to introduce another essential preliminary to diving into assembly language programming, the ASCII codes. ASCII — the American Standard Code for Information Interchange — is a set of 128 numerical codes to represent letters, numbers, symbols, punctuation, and special control functions.

I'll talk hex. Punctuation marks start at **$20**, numbers at **$30. $40** points to uppercase letters, **$60** starts lowercase. Simple? Only in hex. Ever try to convert from uppercase to lowercase in BASIC? It can be tricky. But in binary, it's a cinch. Grab paper and pencil.

Write down hexadecimal **41**, and across from it write its binary equivalent, **0100 0001**. This is the uppercase letter A. On the next line, write down hex **61**, and across from it the binary, **0110 0001**. This is lowercase a. Now write hex **5A**, and its binary, **0101 1010**; this is uppercase Z. Lowercase z is **7A**, binary **0111 1010**.



Sit back and look at these numbers. The hex numbers seem related enough, but the real clue lies in the binary. In referring to binary numbers, the rightmost digit is called bit zero. Find bit five in both upper and lowercase A; it's third from the left. Notice that bit five is the only digit that's different in upper and lower case. Same with letter Z. Bit five clearly distinguishes uppercase from lowercase. In decimal, upper and lowercase Z are 90 and 122 respectively. There's no visible relationship there. But bit five! Just one digit makes all the difference. ASCII looks illogical in decimal, not binary.

I'll talk more about ASCII codes, especially those from **$00** to **$1F** — the control codes that ring bells, backspace, line and form feed, carriage return, and perform special activities like clearing the screen. In the meantime, there's work for you to do.

For your assignment: learn to count in hexadecimal, explore all the ASCII codes in binary, and learn to read the ASCII bit table in the back of your documentation package. Review this lesson until you are familiar and comfortable with binary and hexadecimal. Please continue with these lessons only when you have reviewed the number systems thoroughly.

* What does ASCII stand for?

ASCII stands for American Standard Code for Information Interchange.

* Where are uppercase letters found in the ASCII code? Give the answer in hexadecimal.

The uppercase ASCII codes are $40 to $5F.

* Where are the lowercase letters found in the ASCII code? Give the answer in hexadecimal.

The lowercase ASCII codes are $60 to $7F.

* How are the rightmost and leftmost bits numbered in a group of eight binary digits?

The rightmost is bit 0, the leftmost is bit 7.

* What binary digit distinguishes uppercase from lowercase characters in the ASCII code?

Bit 5 distinguishes uppercase ASCII from lowercase ASCII.

* How is ASCII pronounced?

ASCII is pronounced ASSkey.

* The ASCII code for the uppercase letter E is hexadecimal $45. What are the binary and hexadecimal values for both uppercase and lowercase letter E?

Uppercase E is $45, binary 0100 0101. Lowercase E is $65, binary 0110 0101.

* What ASCII codes are located from hexadecimal $00 to $1F?

The machine control codes are found from $00 to $1F.