

---

# Appendix B

---

## Computer Number Systems

Internally, computers do all their calculations with binary numbers which may only contain the binary digits (bits) 0 and 1.

Imagine that you had to count pages in a book, but were told that you were not allowed to use any number containing the digits 2 through 9. In other words, your numbers would only be allowed to contain zeroes and ones. You would then count like this:

1  
 (skip 2 through 9 because they contain forbidden digits)  
 10  
 11  
 (skip 12 through 99)  
 100  
 101  
 (skip 102 through 109)  
 110  
 111  
 (skip 112 through 999)  
 1000  
 1001  
 (skip 1002 through 1009)  
 and so on.

Thus the first page would be numbered 1, the second would have the number 10, the third would be numbered 11, and so on. In other words, we would get a table like this:

Page	Binary Count
1	1
2	10

Page	Binary Count
3	11
4	100
5	101
6	110

You may think of the left column as being a normal decimal number, while the right column is its binary equivalent. When you extend the table to include zero and also some larger numbers, you get a table like the one below (we've added a few extra zeroes, but that doesn't make a difference. After all, 1, 01, 0001, or even 00000001 are still just 'one'.)

Decimal	Binary	Decimal	Binary
0	0000	9	1001
1	0001	10	1010
2	0010	11	1011
3	0011	12	1100
4	0100	13	1101
5	0101	14	1110
6	0110	15	1111
7	0111	16	10000
8	1000	17	10001

Suppose you were limited to numbers consisting of just four bits. Then the smallest number would be 0000 (decimal zero), while the largest number would be 1111, or the decimal number fifteen. But when you want to go to sixteen, you need one more bit. So we see that you need more and more bits as you count to higher and higher numbers. In fact, binary numbers are roughly three times as long as their decimal equivalents - the decimal number 999 translates to a binary 1111100111, while the decimal 9999 translates to 10011100001111.

Which brings us to the binary numbers that travel on the data bus and address bus of the SK68K computer. With a 24-line address bus, the smallest address in the SK68K computer consists of 24 zeroes, while the largest address consists of 24 ones. All the other addresses have some intermediate combination of zeroes and ones such as, for example, 11010101110000111101010. Although such an address is fairly easy for the computer to handle, we humans find it hard to read or write such long numbers without making a mistake. Hence we use a mathematical shorthand called hexadecimal notation instead.

To use hexadecimal numbers, we first split the binary number into groups of four bits like this:

1101 0101 1100 0011 1110 1010

Now we want to convert each group of four bits into a single digit, but right away run into a problem: the very first 1101 is equal to decimal 13, which is two digits, not one. In fact this is going to happen with 10, 11, 12, 14, and 15 as well. And so we extend the decimal number system with six additional digits (called *hexadecimal digits*) A through F. We write A instead of 10 (decimal ten), B instead of 11 (decimal eleven), and so on, up through F for 15 (decimal fifteen). The number is now written as

```
1101 0101 1100 0011 1110 1010
 13   5   12   3   14   10
  D       C       E   A
```

or just plain D5C3EA. That gives us this conversion table to use between binary and hexadecimal:

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

Returning to the address bus ... the smallest address is 24 zeroes; splitting into groups of four bits we get six groups of 0000, which translates into the hexadecimal (also called just hex) 000000. The largest address is 24 ones, which separates into six groups of 1111, or a hex address of FFFFFFFF.

One last item - when you see a number such as 3152, it is sometimes hard to tell whether this is supposed to be a decimal number or hexadecimal. Whenever there is any doubt, users of Motorola processors always put a \$ before hex numbers. Thus \$3152 would be hexadecimal, whereas a plain 3152 would usually be decimal (except if someone forgot to write the dollar sign, or else if it is obvious from the context.)