# The Binary and Related Number Systems

Robert B. Heckendorn
University of Idaho

August 26, 2019

Numbers are said to be represented by a **place-value system**, where the value of a symbol depends on where it is... its place. For instance, in the decimal number system, an 8 in right most position in a number means 8 but in the third place from the right means 800. That is, each position from the right side of the number has an associated value. In the **decimal** number system each position to the left is worth 10 more than the next position to the right. For example: in the number 7654, the rightmost place is worth 1, the next to the right most position is worth 10, next position is worth 100 and the left most is worth 1000. The number 7654 is therefore: $7 * 1000 + 6 * 100 + 5 * 10 + 4 * 1$ which is 7654 in decimal. Because each place is worth 10 times as much as the place to the right it is called a **base 10** number system.

It is important to see that 7654 is a representation for a number; a way to write it. It is not the number itself. For example Romans might use MMMMMMMDCLIV to represent the same number. **Roman numerals** are not a place value system: a V is always 5 regardless of if it is VIII for 8 or IV for 4. (Technically, 4 can be written as IIII or IV in a version of roman numerals that is additive only. In an additive only system one could write 2019 as MMXVIIII or IXMIIVIM, but no one ever does, including the Romans. :-) )

In the base 10 number system, numbers are represented by a list of symbols of which there are 10 kinds of symbols: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. The position or place-values are powers of 10: $10^0, 10^1, 10^2, 10^3, ...$ or $1, 10, 100, 1000, ....$ Remember that any positive number to the 0 is 1, so $10^0 = 1$.

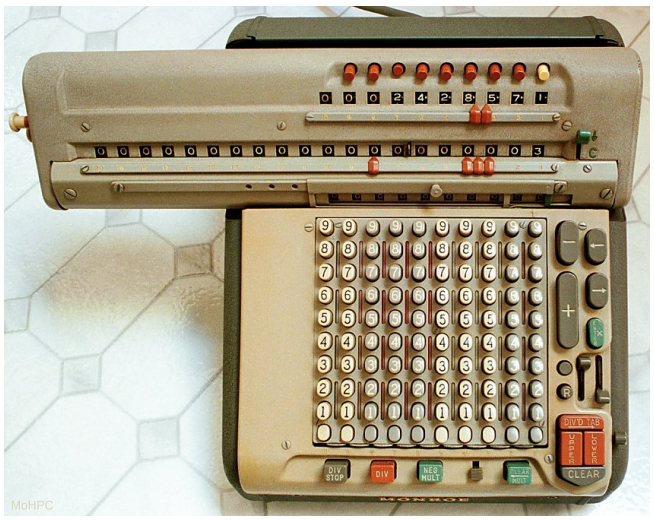



Figure 1: Monroe CSA-10 Mechanical Calculator using decimal arithmetic.

Long ago computing machines used mechanical mechanisms and decimal representation (See Figure 1, but now with fast electricity based computing devices need a different way to encode numbers and data in general is needed; a way that uses only on and off, positive and negative. This number system is a base 2 system called binary.

# 1 Binary

In **binary**, or base 2, each position to the left is worth 2 times the next position to the right. For example: in the number 1101, the rightmost position is worth 1, the position next to the right most position is worth 2, next position is worth 4 and the left most is worth 8. The number 1101 is therefore: $1*8+1*4+0*2+1*1$ which is 13 in decimal.

## 1.1 Converting Binary to Decimal

Converting from binary to decimal is as easy as just adding up the binary digits times their place-values. The term **bit** is short for **binary digit**.

The base 2 number system, numbers are represented by a list of symbols of which there are 2 kinds: $0, 1$. The position values are powers of 2: $2^0, 2^1, 2^2, 2^3, 2^4, ...$ or $1, 2, 4, 8, 16, ...$

Let's try converting 110010 from binary to decimal. List out the bits and next to each write its power of 2. Only sum up the powers of 2 that are next to 1 bits.

| bit | | power of 2 | | decimal value |
|---|---|---|---|---|
| 1 | $\times$ | $2^5$ | $\rightarrow$ | 32 |
| 1 | $\times$ | $2^4$ | $\rightarrow$ | 16 |
| 0 | $\times$ | $2^3$ | $\rightarrow$ | 0 |
| 0 | $\times$ | $2^2$ | $\rightarrow$ | 0 |
| 1 | $\times$ | $2^1$ | $\rightarrow$ | 1 |
| 0 | $\times$ | $2^0$ | $\rightarrow$ | 0 |
| | | | Total | 49 |

We find and 110010 in binary is equal to 49 in decimal. If we had exactly 8 bits to represent 49 we would pad on the left with zeroes: 00110010.

Counting from 0 to 10 in binary is: 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010. See how the numbers from 0 to 7 can be represented by 3 binary digits and at 8 you have to go to 4 binary digits. This is part of the general rule that the largest number you can represent with $k$ bits is $2^k - 1$. The smallest number is, of course, 0. If $k = 3$ then $2^3 - 1 = 7$. What is the largest number that can be represented by 8 bits? The answer is 255 which is $2^8 - 1$. This would be represented by eight 1 bits in a row: 11111111.

Because any nonnegative integer can be represented in binary, those numbers can be represented as a string of 1's/0's, electricity on/electricity off, positive current/negative current, north magnetic field/south magnetic field, etc. So this is how numbers are represented inside modern computers which use electric components and magnetic fields on magnetic disks. Music on CDs is stored the same way, as 1's/0's on the CD surface with tiny flat reflective areas as 1's.

## 1.2 Converting Decimal to Binary

Converting decimal to binary is not as easy. Let's look at a 4 bit example. Here is how to convert a number between 0 and 15 inclusive into a 4 bit binary number:

```
is it >=8?
      if yes write 1 and subtract 8
```

```
      if no write 0

is it >=4?
      if yes write 1 and subtract 4
      if no write 0

is it >=2?
      if yes write 1 and subtract 2
      if no write 0

is it >=1?
      if yes write 1 and subtract 1
      if no write 0
```

Pretty easy, eh? This version of the conversion algorithm requires that you know the largest power of 2 that is smaller than the number you converting. Then you try each progressively smaller power of 2 subtracting away each power of 2 you find that will fit in the remainder. Using this scheme binary digits are generated from left to right.

How would you convert 23 to binary? You would start by subtracting $2^4$ or 16 from the number because starting with $2^5$ or 32 is larger than 23. The 16 would count as a 1 in the $5^{\text{th}}$ place of the binary number. So... we start with a test if the number is $>= 16$ and that gives us the first bit. Then we just do the four tests of the 4 bit case above.

It is important to see that it is not completely trivial to convert decimal to binary. It requires knowing the powers of 2, asking a question for each digit and subtracting off the power of two (starting with the largest that is still smaller than the number) if the answer is yes that counts as a 1 bit. In short, it requires answering a yes/no question for every power of 2 up to the size of the number you want to convert. Let's run through the above algorithm with the decimal number 13:

```
is 13 >= 8?
      yes: write 1 and subtract off 8 so we look at the number 5.

is 5 >= 4?
      yes: write 1 and subtract off 4 so we look at the number 1.

is 1 >= 2?
      no: write 0

is 1 >= 1?
      yes: write 1 and subtract off 1 so we look at the number 0.
```

So the number 13 in decimal is 1101 in binary. What is the decimal number 23 in binary[1]? Hint: you have to add a test for 16 to the above approach.

[1]The answer is 10111

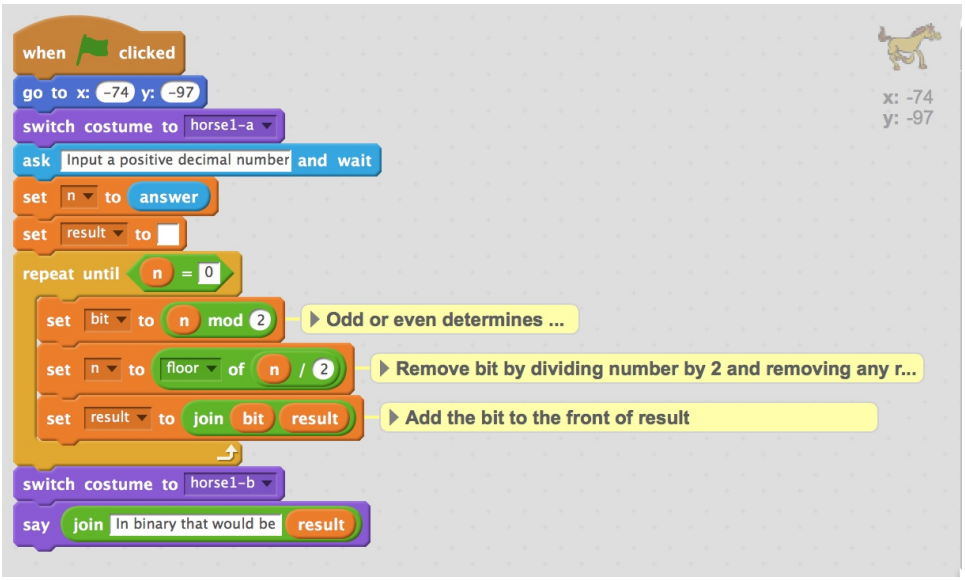
Let's try another one. Let's try converting 146 to binary. This time let's take notes in a table:

| remainder | bits | power of 2 | new remainder |
|---|---|---|---|
| $146 -$ | $1 \times 128$ | | $\rightarrow 18$ |
| $18 -$ | $0 \times 64$ | | $\rightarrow 18$ |
| $18 -$ | $0 \times 32$ | | $\rightarrow 18$ |
| $18 -$ | $1 \times 16$ | | $\rightarrow 2$ |
| $2 -$ | $0 \times 8$ | | $\rightarrow 2$ |
| $2 -$ | $0 \times 4$ | | $\rightarrow 2$ |
| $2 -$ | $1 \times 2$ | | $\rightarrow 0$ |
| $0 -$ | $0 \times 1$ | | $\rightarrow 0$ |

The answer is 10010010!

## 1.3 An Alternate Decimal to Binary Algorithm

In this algorithm we notice that all odd numbers end in a 1 in binary. This gives us an idea how to get binary by producing the bits from right to left rather than left to right as we did above. We find out if the number is odd or even giving us 1 or 0. Then we divide the number by 2 without saving the remainder (nothing after the decimal). This removes the last binary digit of the number. Then we repeat with the new smaller number to let the next bit etc. Here is the algorithm in the Scratch programming language. Note the use of the floor function to remove anything after the decimal.




## 1.4 Bits and Information

Each **binary digit**, or **bit** for short, represents the quantity of information that can be determined by answering a yes or no question. You can see this in the 4 bit conversion routine above. Four questions are asked. The four answers were then encoded as binary digits. **Bits are the fundamental unit of information!**

A **byte** is 8 bits. Memory is often divided up into blocks of 8 bits called bytes. A byte can have one of 256 different values. As integers this is usually the numbers 0 to $2^8 - 1$ or 255. A byte is also enough to contain a simple encoding of a character. **ASCII** is one such encoding standard. We will discuss ASCII later.

It takes about $3\frac{1}{3}$ bits to represent each decimal digit. That means a 10 bit number is about 3 decimal digits. The number $1,000,000$ in decimal is about 20 bits long! In fact: $1,000,000$ in decimal is 11110100001001000000 in binary. That is a lot of writing to express a number, but if you store the number $1,000,000$ in a computer that is exactly how it will store it. It may use current, magnetics, or capacitance to store the `1`'s and `0`'s but the number will be 11110100001001000000 in binary all the same.

## 2 Octal and Hex

Computer scientists use two shortcut bases to make writing binary easier for humans. The first is base 8 or **octal**. The second is base 16 or **hexadecimal**. The reason they use these is it is insanely easy to convert from binary to octal and back! Same for hexadecimal.

Let's do octal to binary and back. What makes octal so easy is, unlike a decimal digit, there are **exactly** 3 binary bits in each octal digit! So for every octal digit I can translate that into 3 bits. This is because $2^3 = 8$. For example: the octal number: 3705 is 4 octal digits. This should become 12 binary digits: 3 is 011 in binary, 7 is 111 in binary, 0 is 000 in binary, 5 is 101 in binary. So $3705_8 = 011111000101_2$. **Note the use of subscripts to denote the base of the number!!** Yes, it is that straight forward. If you know the 3 bit values for each of the 8 octal digits you are practically done.

What is $11110100001001000000_2$ in octal? First divide the number in groups of 3 starting on the right: 11 110 100 001 001 000 000. Then simply read off the octal digits: $3641100_8$.

Break for a joke: Why are computer scientists always confusing Halloween and Christmas? (pause) Because 31 Oct = 25 Dec (if Oct means octal and Dec means decimal).

Hexadecimal numbers are base 16. Hexadecimal is sometimes simply referred to as **hex**. Each position to the left is worth 16 more than the next position to the right. The base 16 number system, numbers are represented by a list of symbols of which there are 16 kinds: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f. (When they ran out of digits they used letters.) The position values are powers of 16: $16^0, 16^1, 16^2, 16^3, ...$ or $1, 16, 256, 4096, ...$

For example: in the hex number 1fab, the rightmost position is worth 1, the next to the right most position is worth 16, next position is worth 256 and the left most is worth 4096. The number 1fab is therefore: $1 * 4096 + 15 * 256 + 10 * 16 + 11 * 1$ which is 8107 in decimal. But it is super easy to convert to binary one hex digit at a time because like octal each hex digit is worth **exactly** 4 bits: $1fab_{16} = 0001\ 1111\ 1010\ 1011_2 = 0001111110101011_2$. This is because the $1 \rightarrow 0001$, $f \rightarrow 1111$, $a \rightarrow 1010$, $b \rightarrow 1011$.

When computer scientists wants to write a binary number they usually use either octal or hex because it is so easy to write.

Table 1: A table of counting in different bases. The 8-bit case shows what it might be like if you had a number in hardware that could store 8 bits.

| Hexadecimal | Decimal | Octal | Binary | 8-bit Binary |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 00000000 |
| 1 | 1 | 1 | 1 | 00000001 |
| 2 | 2 | 2 | 10 | 00000010 |
| 3 | 3 | 3 | 11 | 00000011 |
| 4 | 4 | 4 | 100 | 00000100 |
| 5 | 5 | 5 | 101 | 00000101 |
| 6 | 6 | 6 | 110 | 00000110 |
| 7 | 7 | 7 | 111 | 00000111 |
| 8 | 8 | 10 | 1000 | 00001000 |
| 9 | 9 | 11 | 1001 | 00001001 |
| a | 10 | 12 | 1010 | 00001010 |
| b | 11 | 13 | 1011 | 00001011 |
| c | 12 | 14 | 1100 | 00001100 |
| d | 13 | 15 | 1101 | 00001101 |
| e | 14 | 16 | 1110 | 00001110 |
| f | 15 | 17 | 1111 | 00001111 |
| 10 | 16 | 20 | 10000 | 00010000 |
| 11 | 17 | 21 | 10001 | 00010001 |
| 12 | 18 | 22 | 10010 | 00010010 |
| 13 | 19 | 23 | 10011 | 00010011 |
| 14 | 20 | 24 | 10100 | 00010100 |
| 15 | 21 | 25 | 10101 | 00010101 |
| 16 | 22 | 26 | 10110 | 00010110 |
| 17 | 23 | 27 | 10111 | 00010111 |
| 18 | 24 | 30 | 11000 | 00011000 |
| 19 | 25 | 31 | 11001 | 00011001 |
| 1a | 26 | 32 | 11010 | 00011010 |
| 1b | 27 | 33 | 11011 | 00011011 |
| 1c | 28 | 34 | 11100 | 00011100 |
| 1d | 29 | 35 | 11101 | 00011101 |
| 1e | 30 | 36 | 11110 | 00011110 |
| 1f | 31 | 37 | 11111 | 00011111 |

The key ideas I want you to know:

- Current day computers use electricity and so the binary number system has become a convenient way to represent information.
- As an example, binary is a great way to represent integers.
- If a Computer Scientist needs to talk about the detail of what the representation of a number they may use binary, octal, or hexadecimal.
- The **bit** is the fundamental unit to measure the quantity of information. It represents the answer to a yes or no question.

## 3 A Binary Card Trick

Below are the cards for a **Binary Card Trick**. Cut them out. Ask person to think of a number between 1 and 31 inclusive but don't tell you the number. Now hand the cards to the person and tell them to select all the cards that have their secret number on them. When they hand you the selected cards simply add the numbers in the upper left corner and that is the number they are thinking of. Amazing? Not really, it is simply the binary number system.

Why does this work? On the card with a 1 in the upper left are all the numbers that have a 1 in right most place in the binary representation of the number. That is the 1's place. The card with a 2 in the upper left are all the numbers that have a 1 in next to last position in the binary representation of the number. That is the 2's place. The 4 card is for the 4's place, etc. By handing you that card they are answering a simple yes/no question and giving you one bits worth of information. When they don't hand you a card say the 8 card, that means means there is a 0 in the 8's place. Another way to say this is the question for the card with a 1 in the upper left is "What is the right most digit in the binary representation of the number". When they hand you all the selected cards you have the binary for the number and you simply add the place-value of each of the 1 bits. That place value is the number in the upper left. That is because that is always a number that looks like a 1 followed by some number of 0's. Note also that it must be the case that each secret number will cause a different set of cards to be chosen? Why? Because each number has a unique binary number. What would the cards look like if the secret number was from 1 to 63 inclusive? Do the cards below have to change to handle the numbers between 0 and 31 inclusive instead of between 1 and 31 inclusive?

| | | | |
|---|---|---|---|
| 1 | 3 | 5 | 7 |
| 9 | 11 | 13 | 15 |
| 17 | 19 | 21 | 23 |
| 25 | 27 | 29 | 31 |

| | | | |
|---|---|---|---|
| 2 | 3 | 6 | 7 |
| 10 | 11 | 14 | 15 |
| 18 | 19 | 22 | 23 |
| 26 | 27 | 30 | 31 |

| | | | |
|---|---|---|---|
| 4 | 5 | 6 | 7 |
| 12 | 13 | 14 | 15 |
| 20 | 21 | 22 | 23 |
| 28 | 29 | 30 | 31 |

| | | | |
|---|---|---|---|
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |
| 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 |

| | | | |
|---|---|---|---|
| 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 |

# 4 Converting decimal directly to octal and hexadecimal

Converting to octal is based on powers of 8: 1, 8, 64, 512, 4096, 32768, ... Here is an algorithm for the first 4 octal digits.

```
is it >=512?
      if yes write integer portion of number/512 and subtract that many 512's
      if no write 0

is it >=64?
      if yes write integer portion of number/64 and subtract that many 64's
      if no write 0

is it >=8?
      if yes write integer portion of number/8 and subtract that many 8's
      if no write 0

is it >=1?
      if yes write integer portion of number/1 and subtract that many 1's
      if no write 0
```

For example: What is 666 in octal?

- 512 goes into 666 1 time with a remainder of 154 so the first octal digit is 1.
- 64 goes into 154 2 times with a remainder of 26 so the second octal digit is 2.
- 8 goes into 26 3 times with a remainder of 2 so the third octal digit is 3.
- 1 goes into 2 2 times with a remainder of 0 so the third octal digit is 2.
  the answer is $1232_8$.

Similarly the algorithm for converting a number into as many as 4 hex digits is:

```
is it >=4096?
      if yes write integer portion of number/4096 and subtract that many 4096's
      if no write 0

is it >=256?
      if yes write integer portion of number/256 and subtract that many 256's
      if no write 0

is it >=16?
      if yes write integer portion of number/16 and subtract that many 16's
      if no write 0
```

```
is it >=1?
      if yes write integer portion of number/1 and subtract that many 1's
      if no write 0
```

What is 43785 in hex?

- 4096 goes into 43785 10 times with a remainder of 2825 so the first hex digit is 10. In hex this is represented as an a. See the table for counting in different bases.

- 256 goes into 2825 11 times with a remainder of 9 so the second hex digit is 11 or b.

- 16 goes into 9 0 times with a remainder of 9 so the third hex digit is 0.

- 1 goes into 9 9 times with a remainder of 0 so the third hex digit is 9.

The answer is $ab09_{16}$. Notice that because base 16 is larger that 10 the number of digits to represent a number is less than or equal to the number of digits to represent a number in base 10.

## 5 Binary can also Encode Characters

ASCII is a 7-bit encoding of characters, the upper bit is zero in 8 bit bytes. It is given in the table below. UTF-8 subsumes ASCII and is the most popular on the web.

```
Binary:
00000000 nul 00000001 soh 00000010 stx 00000011 etx 00000100 eot 00000101 enq 00000110 ack 00000111 bel
00001000 bs  00001001 ht  00001010 nl  00001011 vt  00001100 np  00001101 cr  00001110 so  00001111 si
00010000 dle 00010001 dc1 00010010 dc2 00010011 dc3 00010100 dc4 00010101 nak 00010110 syn 00010111 etb
00011000 can 00011001 em  00011010 sub 00011011 esc 00011100 fs  00011101 gs  00011110 rs  00011111 us
00100000 sp  00100001 !   00100010 "   00100011 #   00100100 $   00100101 %   00100110 &   00100111 '
00101000 (   00101001 )   00101010 *   00101011 +   00101100 ,   00101101 -   00101110 .   00101111 /
00110000 0   00110001 1   00110010 2   00110011 3   00110100 4   00110101 5   00110110 6   00110111 7
00111000 8   00111001 9   00111010 :   00111011 ;   00111100 <   00111101 =   00111110 >   00111111 ?
01000000 @   01000001 A   01000010 B   01000011 C   01000100 D   01000101 E   01000110 F   01000111 G
01001000 H   01001001 I   01001010 J   01001011 K   01001100 L   01001101 M   01001110 N   01001111 O
01010000 P   01010001 Q   01010010 R   01010011 S   01010100 T   01010101 U   01010110 V   01010111 W
01011000 X   01011001 Y   01011010 Z   01011011 [   01011100 \   01011101 ]   01011110 ^   01011111 _
01100000 `   01100001 a   01100010 b   01100011 c   01100100 d   01100101 e   01100110 f   01100111 g
01101000 h   01101001 i   01101010 j   01101011 k   01101100 l   01101101 m   01101110 n   01101111 o
01110000 p   01110001 q   01110010 r   01110011 s   01110100 t   01110101 u   01110110 v   01110111 w
01111000 x   01111001 y   01111010 z   01111011 {   01111100 |   01111101 }   01111110 ~   01111111 del

Octal:
000 nul  001 soh  002 stx  003 etx  004 eot  005 enq  006 ack  007 bel
010 bs   011 ht   012 nl   013 vt   014 np   015 cr   016 so   017 si
020 dle  021 dc1  022 dc2  023 dc3  024 dc4  025 nak  026 syn  027 etb
030 can  031 em   032 sub  033 esc  034 fs   035 gs   036 rs   037 us
```

```
040 sp   041  !   042  "   043  #   044  $   045  %   046  &   047  ’
050  (   051  )   052  *   053  +   054  ,   055  -   056  .   057  /
060  0   061  1   062  2   063  3   064  4   065  5   066  6   067  7
070  8   071  9   072  :   073  ;   074  <   075  =   076  >   077  ?
100  @   101  A   102  B   103  C   104  D   105  E   106  F   107  G
110  H   111  I   112  J   113  K   114  L   115  M   116  N   117  O
120  P   121  Q   122  R   123  S   124  T   125  U   126  V   127  W
130  X   131  Y   132  Z   133  [   134  \   135  ]   136  ^   137  _
140  ‘   141  a   142  b   143  c   144  d   145  e   146  f   147  g
150  h   151  i   152  j   153  k   154  l   155  m   156  n   157  o
160  p   161  q   162  r   163  s   164  t   165  u   166  v   167  w
170  x   171  y   172  z   173  {   174  |   175  }   176  ~   177 del
```

```
Decimal:
000 nul  001 soh  002 stx  003 etx  004 eot  005 enq  006 ack  007 bel
008 bs   009 ht   010 nl   011 vt   012 np   013 cr   014 so   015 si
016 dle  017 dc1  018 dc2  019 dc3  020 dc4  021 nak  022 syn  023 etb
024 can  025 em   026 sub  027 esc  028 fs   029 gs   030 rs   031 us
032 sp   033  !   034  "   035  #   036  $   037  %   038  &   039  ’
040  (   041  )   042  *   043  +   044  ,   045  -   046  .   047  /
048  0   049  1   050  2   051  3   052  4   053  5   054  6   055  7
056  8   057  9   058  :   059  ;   060  <   061  =   062  >   063  ?
064  @   065  A   066  B   067  C   068  D   069  E   070  F   071  G
072  H   073  I   074  J   075  K   076  L   077  M   078  N   079  O
080  P   081  Q   082  R   083  S   084  T   085  U   086  V   087  W
088  X   089  Y   090  Z   091  [   092  \   093  ]   094  ^   095  _
096  ‘   097  a   098  b   099  c   100  d   101  e   102  f   103  g
104  h   105  i   106  j   107  k   108  l   109  m   110  n   111  o
112  p   113  q   114  r   115  s   116  t   117  u   118  v   119  w
120  x   121  y   122  z   123  {   124  |   125  }   126  ~   127 del
```

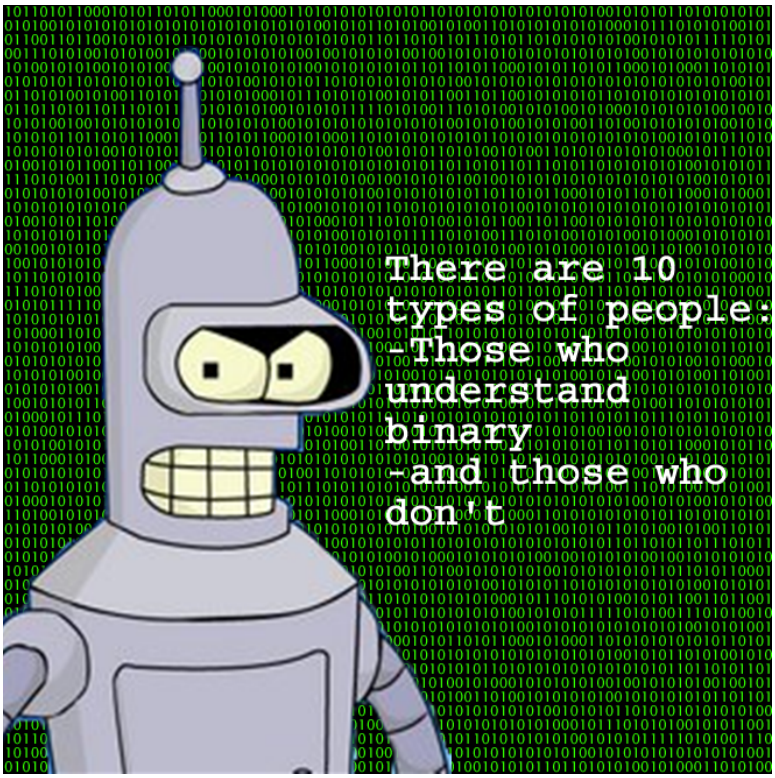
```
Hexadecimal:
00 nul   01 soh   02 stx   03 etx   04 eot   05 enq   06 ack   07 bel
08 bs    09 ht    0a nl    0b vt    0c np    0d cr    0e so    0f si
10 dle   11 dc1   12 dc2   13 dc3   14 dc4   15 nak   16 syn   17 etb
18 can   19 em    1a sub   1b esc   1c fs    1d gs    1e rs    1f us
20 sp    21  !    22  "    23  #    24  $    25  %    26  &    27  ’
28  (    29  )    2a  *    2b  +    2c  ,    2d  -    2e  .    2f  /
30  0    31  1    32  2    33  3    34  4    35  5    36  6    37  7
38  8    39  9    3a  :    3b  ;    3c  <    3d  =    3e  >    3f  ?
40  @    41  A    42  B    43  C    44  D    45  E    46  F    47  G
48  H    49  I    4a  J    4b  K    4c  L    4d  M    4e  N    4f  O
50  P    51  Q    52  R    53  S    54  T    55  U    56  V    57  W
58  X    59  Y    5a  Z    5b  [    5c  \    5d  ]    5e  ^    5f  _
60  ‘    61  a    62  b    63  c    64  d    65  e    66  f    67  g
68  h    69  i    6a  j    6b  k    6c  l    6d  m    6e  n    6f  o
70  p    71  q    72  r    73  s    74  t    75  u    76  v    77  w
78  x    79  y    7a  z    7b  {    7c  |    7d  }    7e  ~    7f del
```

## 6 Futurama and Binary

It turns out the writers of the TV show Futurama have degrees in Mathematics and have included some insider humor in their show. Can you tell why the numbers were chosen or what the joke is in the frames from the show below?

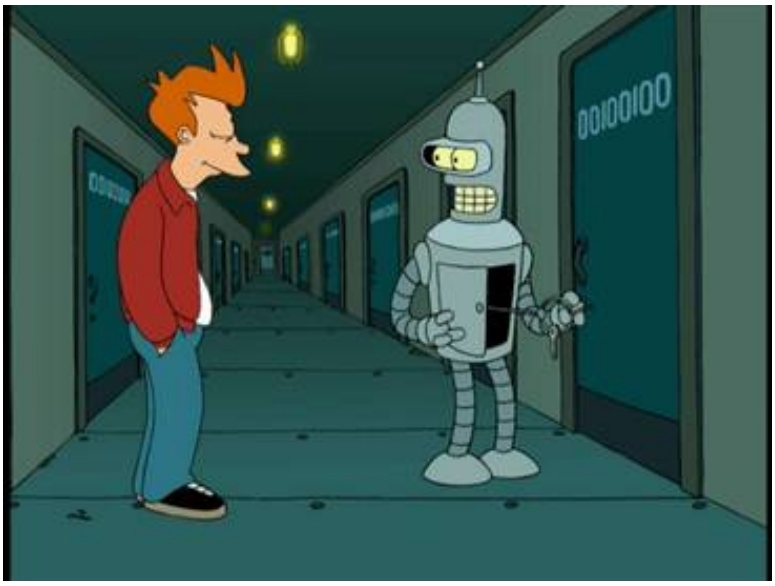
---

What is funny about this:




---

Why is this Bender's apartment number? (Hint: what is this is ASCII?)

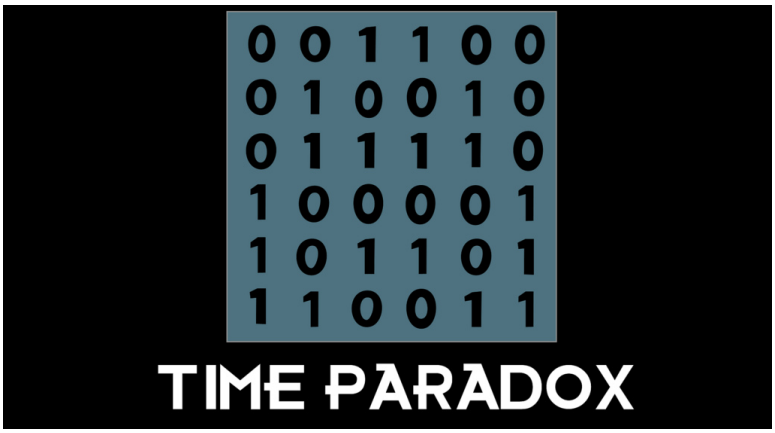



---

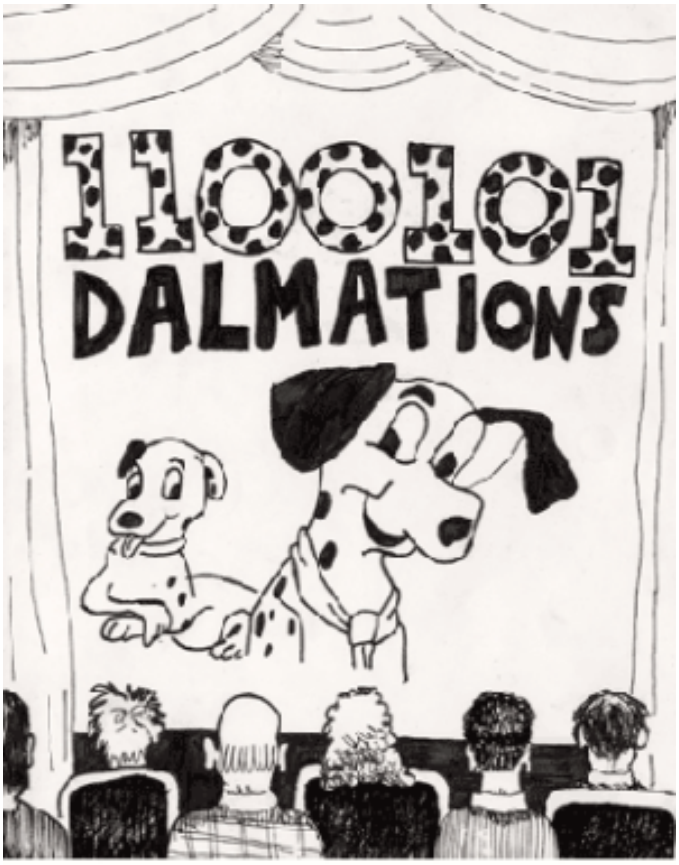
Why is this exact binary number written in blood on the mirror?


---

How was this simple pattern made? (Hint: The story line required that the pattern be able to be read the same in a mirror.)

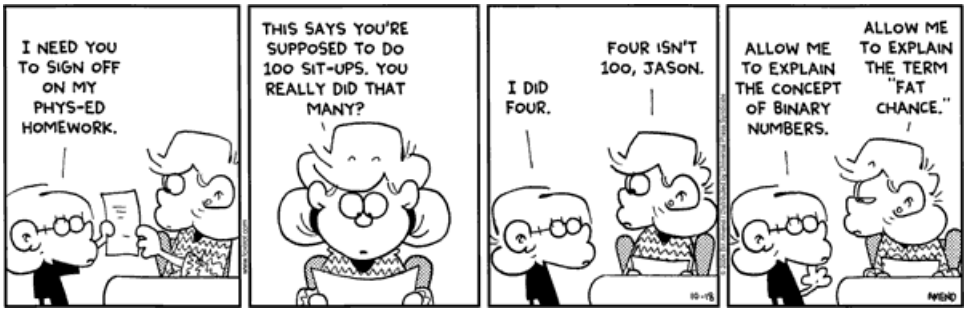


Remember this classic Disney movie?




Film Night at the Binary Society

Using binary to get out of exercise. How does that work?




OK, so this isn't Futurama but it is related to the topic of number bases. Why did being a horse suggest that the horse was correct to have $2 + 2 \to 10$?


GUY WALKS INTO A BAR
I'LL BET $50 MY HORSE CAN DO ARITHMETIC!
OKAY... WHAT'S 2+2 ?
10.
WELL, THINK ABOUT IT. WHY WOULD A HORSE USE BASE 10?
BAR

Answers:

- 10 is 2 in binary not 10 in base 10.
- Bender's apartment number is the ASCII code for '$' and Bender is all about money.
- This is the biblical number of the beast: 666 but in binary.
- This is made by counting in 3 bit binary and mirroring it right to left.
- 101 Dalamations, of course.
- 100 in binary is 4 in decimal.
- It assumes that the modern horse has only 1 finger on each "hand" (essentially a horse gallops on its "finger nail" that we call a hoof). And it assumes that the horse might count up to 4 in the same way we count up to 10 because we have 10 fingers on our hands. (Some civilizations such as the Maya counted up to 20, presumably because it included hands and feet.) If it counts 4 as all its fingers then it might count in base 4 so 10 base 4 would be 4 in base 10.