

INTRODUCTION TO COMPUTING

TOPIC 4: DATA REPRESENTATION

PAUL L. BAILEY

1. MEMORY

The *memory* of a computer is where the computer stores temporary information; when you turn off the computer, the contents of the memory are gone. The memory may be viewed as a sequence of slots which may contain either a zero and or a one. The number of slots is the indicates size of that particular computer's memory.

Each slot is called a *bit*. These bits are grouped together into blocks of various sizes. Then each block is capable of holding a natural number, stored in base two. If the block contains n bits, it can contain a number between 0 and $2^n - 1$.

A *byte* is a block of eight bits. Thus a byte can contain a number between 0 and 256.

A *word* is a block of w bits, where w depends on the particular computer; we call w the *word size* of that computer. The word size is typically a multiple of eight by a power of two, but this is not necessarily the case. For example, old DEC PDP computers used 12, 18, and 36 bit words, interpreted as 4, 6, and 12 octal digits, respectively. Intel chips used in DOS and Windows computers have used 16, 32, and more recently, 64 bit words, interpreted as 4, 8, and 16 hexadecimal digits, respectively.

The computer views its memory as broken up into words; the number of bits in the memory is a multiple of the word size.

The computer accesses its memory by enumerating its words; the number (or location) of a word is called its *address*. In the simplest case, the computer stores an address as a word, so the computer's memory contains at most 2^w words, numbered from 0 to $2^w - 1$. The maximum number of words the computer can access is called the *address space* of the computer. We note that, using segmentation, Intel chips do not work this way, and their address spaces are larger then 2^w .

When the computer stores information in its memory, it stores different types of information in different ways. These types are called *data types*. Later, it interprets the information according to its data type; in particular, it needs to know the data type of the information stored in a specific area of its memory, in order to interpret it correctly.

We now discuss the format in which a computer stores various data types. Keep in mind that the memory in actually a sequence of bits, or from a slightly different point of view, the memory is a sequence of words, each of which is a sequence of bits.

2. INTEGERS

Consider a computer with word size w ; there are w bits in each word.

2.1. Unsigned integers. A nonnegative integer is typically called an *unsigned integer* in computer jargon. One unsigned integer is stored in a single word. The bits of the word are interpreted as representing the integer in base two. In this way, a word can store any integer between 0 and $2^w - 1$.

Example 1. The word size is $w = 8$ bits, so a word can hold an integer between 0 and 255.

A given word contains the bits 10010011. Interpret this word as an unsigned integer.

Solution. Since $(10010011)_2 = 128 + 16 + 2 + 1 = 147$, this word is interpreted as the integer 147. \square

If the computer adds two unsigned integers in two words and places the result in another word, it is possible that overflow may occur; that is, the result may be too large to fit in a single word. In this case, the highest order bit is dropped.

Suppose a and b are integer stored in words of length w bits. Since a word can only contain integers between 0 and $2^w - 1$, the sum $a + b$ is actually the remainder when $a + b$ is divided by 2^w .

Example 2. The word size is $w = 8$. Let a , b , and c represent words containing bits. Suppose $a = 11001100$ and $b = 01110111$, and the computer adds these and stores the result in c . Find c . Also find the decimal representation of a , b , and c .

Solution. The binary sum of a and b is 101000011, which is 9 bits long. The highest bit is dropped, so the computer stores $c = 01000011$. In decimal, a represents $128 + 64 + 8 + 4 = 204$, and b represents $64 + 32 + 16 + 4 + 2 + 1 = 119$. Note that $204 + 119 = 323$, which is greater than 255. Now c represents $64 + 2 + 1 = 67$, which is the remainder when 323 is divided by 256. \square

2.2. Signed Integers. A *signed integer* is a word which is interpreted an integer which may be positive or negative. The high bit is reserved to the sign, where 0 means nonnegative and 1 means negative. This cuts in have the number of positive integers which may be stored.

In a word of length w , a signed integer varies between -2^{w-1} and $2^{w-1} - 1$. For example, if $w = 8$, a word is capable of storing numbers between -128 and 127 .

It is interesting to examine how a computer stores negative numbers. Now we realize that subtracting a number means adding its negative. Thus the computer stores a negative number in such a way that, when the numbers are added according to the addition algorithm already outlined, the desired result is obtained.

Computers store negative in a form known as *two's complement*. Since in a word of length w , 2^w is treated as zero (as described above), then $-a$ and $2^w - a$ will be treated identically. The two's complement of a with respect to word length w is $2^w - a$, expressed in binary.

Example 3. Let $w = 8$ and $a = 00110011$. Find the two's complement of a .

Solution. We have $(00110011)_2 = 32 + 16 + 2 + 1 = 51$, and $256 - 51 = 205$. Converting 205 to binary yields $(205)_{10} = (11001101)_2$.

Note that 11001101 may be obtained from 00110011 by flipping every bit of 00110011, and adding 1. \square

A word of length w equals its highest possible value $(2^w - 1)$ if and only if every bit is equal to one. If a is a word and b is obtained from a by flipping every bit (changing zero to one and changing one to zero), then the sum of a and b has every bit set to one. So, it is $2^w - 1$. Adding one to this gives 2^w , which gives zero when stored in a word of length w .

In conclusion, in order to negate a word a , the computer flips every bit of a , and adds one, using binary addition. To subtract a from b , the computer constructs negative a , and adds b to this.

To negate a signed integer:

- (a) Flip every bit
- (b) Add 1

If the original number was negative, this procedure will cause an overflow bit; drop it. If the original number had a 1 as its least significant bit, it may be easier to reverse the order:

- (a) Subtract 1.
- (b) Flip every bit.

This actually produces the same result (try it).

Example 4. A computer stores -5 in a signed byte. Find the bits which are stored.

Solution. Since $(5)_{10} = (101)_2$, 5 is stored as 00000101. Flipping every bit gives 11111010. Adding one gives 11111011. The computer stores 11111011. \square

Example 5. A byte contains the bits 10011001. Interpret this as a signed integer.

Solution. Since the high bit is set, the number is negative. We negate it, to find the positive integer of which it is the negative.

Subtracting one gives 10011000. Flipping every bit gives 01100111. Converting this to decimal gives $64 + 32 + 4 + 2 + 1 = 103$. So, the original stored integer was -103 . \square

Example 6. Let $a = (51)_{10}$ and $b = (100)_{10}$. Show how the computer of word length $w = 8$ would compute $b - a$.

Solution. Note that, in binary, we have $a = 00110011$ and $b = 01100100$. Now let $c = 11001101$; this is the two's complement of a , and is interpreted by the computer as negative a . Now $b - a = b + c$ internally. We have

$$(01100100)_2 + (11001101)_2 = (100110001)_2;$$

however, this result is nine bits long, and cannot fit in a word. The high bit is dropped, and the result is $b - a = 00110001$. Note that $(00110001)_2 = 32 + 16 + 1 = 49$; also, $100 - 51 = 49$. \square

3. FLOATING POINT NUMBERS

We think of rational numbers as a quotient of integers; it is possible to write a computer program to do arithmetic on rational numbers stored as a pair of integers with no common factors; however, typical computer hardware does not do this automatically.

However, most modern computers can, in their hardware, deal with a format known as *floating point*. This is very similar to what is commonly referred to as *scientific notation*.

Consider a computer with word length w . To store a floating point number, the w bits of the word are broken up into three parts; the *sign* (1 bit), the *mantissa* (t bits), and the *exponent* (x bits, where $w = 1 + t + x$).

A simplified version of this could go as follows. Define

$$s = \begin{cases} 1 & \text{if the sign bit is zero;} \\ -1 & \text{if the sign bit is one;} \end{cases}$$

$m =$ the unsigned binary integer given by the t bits of the mantissa;

$e =$ the signed binary integer given by the x bits of the exponent.

Note that m may represent any integer in the range $0 \leq m \leq 2^t - 1$, and e can represent any integer in the range $-2^{x-1} \leq e \leq 2^{x-1} - 1$. Then, the number stored in the word is interpreted as $s \times m \times 2^e$.

For example, suppose $w = 8$ (8 bit words), $t = 3$ (3 bits per mantissa), and $x = 5$ (5 bits per exponent). Note that the mantissa can store unsigned integers up to $2^3 = 8$, and the exponent can store positive or negative numbers of absolute value up to $2^{5-1} = 16$ (the minus one comes from the fact that one of the 3 bits of the exponent is used for the sign of the exponent).

We know that there are only $2^8 = 256$ possible different quantities which can be represented in 8 bits. However, in this floating point arrangement, one word can represent numbers within the range

$$\pm 8 \times 2^{\pm 2^4}.$$

That is, numbers as small as $\pm 1 \times 2^{-16} = \pm \frac{1}{65536}$, and as big as $8 \times 2^{16} = \pm 524288$. However, not every number in this range can be represented, (think about why); we have lost *precision* to increase our range.

Example 7. A byte holds 01011101. Interpret this as a number according to the scheme outlined above.

Solution. We break the byte into three parts:

$$01010101 \Rightarrow 0 \ 101 \ 1101.$$

The first part is the sign bit, which is 0, so $s = 1$ (the number is positive). The second part is the mantissa, and its three bits 101 are interpreted as an unsigned integer, so $m = 5$. The third part is the exponent, and its four bits 1101 are interpreted as a signed integer. Since the high bit is on, this is a negative integer, stored as a two's complement. To find out which integer is its negative, subtract one and then flip every bit to get 0011, which represents 3, so $e = -3$.

Thus, the number is $sm2^e = 1 \times 5 \times 2^{-3} = \frac{5}{8}$. \square

4. TEXT

Computers store text as a sequence of characters. Characters are represented in the computer as numbers. There are various standard way of representing characters as numbers; the standards which is most commonly used today are ASCII and its extension, Unicode.

4.1. ASCII. The acronym ASCII stands for American Standard Code for Information Interchange. This standard uses seven bits; it is a mapping of numbers between 0 and 127 to how they should be interpreted by an output device.

The characters are grouped into four types:

- Alphabetic: upper and lower case letters
- Numeric: the digits 0 through 9
- Punctuation: other printable characters
- Control: nonprintable characters, used to control devices

The following table lists the standard; control characters are in angle brackets. We use <sp> to indicate a space, which is considered a printable character.

0 <NUL>	16 <SO>	32 <sp>	48 0	64 @	80 P	96 ‘	112 p
1 <SOH>	17 <XON>	33 !	49 1	65 A	81 Q	97 a	113 q
2 <STX>	18 <DC2>	34 "	50 2	66 B	82 R	98 b	114 r
3 <ETX>	19 <XOF>	35 #	51 3	67 C	83 S	99 c	115 s
4 <EOT>	20 <DC4>	36 \$	52 4	68 D	84 T	100 d	116 t
5 <ENQ>	21 <NAK>	37 %	53 5	69 E	85 U	101 e	117 u
6 <ACK>	22 <SYN>	38 &	54 6	70 F	86 V	102 f	118 v
7 <BEL>	23 <ETB>	39 ’	55 7	71 G	87 W	103 g	119 w
8 <BS>	24 <CAN>	40 (56 8	72 H	88 X	104 h	120 x
9 <TAB>	25 	41)	57 9	73 I	89 Y	105 i	121 y
10 <LF>	26 <SUB>	42 *	58 :	74 J	90 Z	106 j	122 z
11 <VT>	27 <ESC>	43 +	59 ;	75 K	91 [107 k	123 {
12 <FF>	28 <FS>	44 ,	60 <	76 L	92 \	108 l	124
13 <CR>	29 <GS>	45 -	61 =	77 M	93]	109 m	125 }
14 <SO>	30 <RS>	46 .	62 >	78 N	94 ^	110 n	126 ~
15 <SI>	31 <US>	47 /	63 ?	79 O	95 _	111 o	127 <RUB>

4.2. Printable characters. Alphabetic, numeric, and punctuation characters are called *printable characters*. We list these.

Alphabetic characters: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz

Numeric characters: 0123456789

Punctuation: <sp> !"#%&'()*+,-./:;<=>?@[\\]^_`{|}~

4.3. Control characters. Control characters are used to control the behavior of peripheral devices, or may be sent by peripheral devices to the computer as signals.

Certain control characters are used very frequently, and in a more or less standard way. We list some of these.

0 <NUL>	Null	Used as a string terminator in the C programming language
3 <ETX>	End Transmission	Breaks the program execution
7 <BEL>	Bell	Sounds a bell or beep
8 <BS>	Backspace	Delete previous character
9 <TAB>	Tab	Separates columns; sometimes used as a field separator
10 <LF>	Line Feed	Move the cursor down one line; used as a line separator in UNIX
13 <CR>	Carriage Return	Put the cursor at the beginning of the current line; output of the "Enter" key
17 <XOF>	X-off	Resume transmission
19 <XON>	X-on	Pause transmission
27 <ESC>	Escape	Exits some programs
127 <RUB>	Rubout	Delete current character; sometimes delete previous character

Pressing the control key together with another key toggles bit 6; since $2^6 = 64$, this has the effect of subtracting 64 from the ASCII value of a letter. Thus CTRL-C often stops the program execution, CTRL-S pauses transmission, and CTRL-Q resumes transmission. Also, CTRL-G is a beep.

4.4. Strings. A *string* is a finite sequence of characters; usually the word string refers to a sequence of characters in memory, without any carriage returns. It is one word, or name, or description. In most programming languages, strings are surrounded by quotation marks when they are referred to.

To reference a string, the computer needs to know where the string starts (its beginning position) and where it stops (its ending position). There are two standard ways of doing this, which we refer to as the Pascal and C conventions.

- Pascal: the first byte holds the length of the string, so only strings containing up to 255 characters can be stored;
- C: the first byte holds the first character; the last character in the string is followed by a zero byte, call the string terminator.

For example, consider the string "This is a test". The length of this string is 14 (we include the spaces but not the quotation marks as characters which we count). Pascal and C each allocate 15 bytes to store this string.

Pascal stores the number 14 followed by the numbers which are the ASCII values of the characters:

```
14 84 104 105 115 32 105 115 32 97 32 116 101 115 116
[ len ] T h i s <sp> i s <sp> a <sp> t e s t
```

C stores the ASCII values of the characters, followed by a byte containing zero.

```
84 104 105 115 32 105 115 32 97 32 116 101 115 116 0
T h i s <sp> i s <sp> a <sp> t e s t [end]
```

4.5. **Text files.** Text files contain sequences of bytes containing numbers, interpreted as characters via ASCII. Lines in a text file are separated according to two different standards:

- UNIX: lines are separated by a single <LF> (ASCII 10).
- DOS: lines are separated by a <CR><LF> (ASCII 13 followed by ASCII 10).

This difference is nightmarish for programmers, and is another example of the horrible mistakes made by Microsoft (UNIX preceded DOS by more than a decade).

5. IMAGES

We describe a simplified version of how a computer may store an image.

The atomic unit of storage is a *pixel*. This word may refer to a physical attribute of the computer monitor, or to a logical attribute of the storage protocol. It is the latter meaning which we use.

An image is stored as a grid, with x pixels across and y pixels down. Each pixel is a position in this grid, and represents a specific color.

The *resolution* of the image refers to $x \times y$.

One pixel represents a specific color. This is stored as a number, so the entire image requires xy numbers.

The *depth* of the color refers to the number of bits used to represent a color. In *indexed* mode, the number specifying the color is used as an index to a *palette*. In *direct* mode, the bits of the number are broken up into three blocks, which specify the intensity of the primary colors red, green, and blue. The combined intensities of these colors is often referred to as *RGB*.

We describe the evolution of IBM PC compatible graphics adaptors.

- **CGA** resolution 320×200 , choice of two palettes with 4 colors each, and a choice of low or high intensity for each color.
- **EGA** resolution 640×350 , with a sixteen color palette. The sixteen colors of the palette are selected from a choice of 64 colors ($64 = 2^6$, with 2 bits for each primary color intensity)
- **VGA** resolution 640×400 , with a sixty four color palette, selected from a possible 262144 colors ($262144 = 2^{18}$, with 6 bits per primary color)
- **SVGA** Super VGA allows for variations. It is now common to use 8 bits per primary color, for 256 possible intensities per color, giving $2^{24} = 16777216$ possible colors.

6. SOUND

Sound is the variation in air pressure. Thus, sound can be modeled as a single real valued function of time. Such a function is represented on a computer breaking up time into small time units of equal size, and recording the air pressure at that moment. This is then stored as a sequence of numbers, each representing the air pressure.

7. APPENDIX

We list some of the history of word sizes. Note that IBM microcomputers and compatibles used Intel chips, whereas Macintosh used Motorola chips.

- DEC PDP-8 used a twelve bit word ($2^{12} = 4096$). Introduced in 1965, this was the first successful minicomputer, and was built with integrated circuits.
- Intel 4004 was a four bit microprocessor ($2^4 = 16$). Introduced in 1971, this was the first single chip microprocessor.
- Intel 8008 was an eight bit microprocessor ($2^8 = 256$). Introduced in 1974.
- Intel 8088 was a sixteen bit microprocessor ($2^{16} = 65536$) with an eight bit bus. Introduced in 1979 and used in the first IBM PC.
- Intel 8086 was a sixteen bit microprocessor with a sixteen bit bus. Introduced in 1978.
- Intel 80286 was sixteen bits. Introduced in 1982 and used the IBM AT.
- Intel 80386 was 32 bits ($2^{32} = 4294967296$). Introduced in 1985 and used in the high end IBM PS/2. DOS still used 16-bit compatibility mode. Later, Microsoft developed the Windows 95 operating system, which used the 32-bit instruction set.
- Intel 80486 was 32 bits. Introduced in 1989.
- Intel Pentium was 32 bits with a 64 bit bus. Introduced in 1993.
- Intel Pentium II was 32 bits. Introduced in 1997.
- Intel Pentium III was 32 bits. Introduced in 1999.
- Intel Pentium 4 was 32 bits. Introduced in 2000.
- Intel Itanium was 64 bits ($2^{64} \approx 1.84 \times 10^{19}$). Introduced in 2001 with a completely new (not backwards compatible) instruction set.

8. PROBLEMS

Problem 1. An eight bit computer adds the unsigned integers 200 and 100, and stores the result in a byte. Find the bits that are stored in the byte, and find the value of the byte as an unsigned integer.

Problem 2. An eight bit computer stores -100 as a signed integer in a byte. Find the bits that are stored in the byte.

Problem 3. A byte holds the bits 11001100.

- (a) Interpret this byte as an unsigned integer.
- (b) Interpret this byte as a signed integer.
- (c) Interpret this byte as a floating point number as in Example 7.

Problem 4. A program stores the string “Sept 11, 2001” in memory using the Pascal convention. Determine the sequence of numbers (expressed in decimal) which is stored.

Problem 5. A program finds the sequence

79 83 65 77 65 32 57 47 49 49 0

in memory, and interprets it as a string stored with the C convention. Find the string.