CHAPTER 1

How are Numbers Represented?

Most programming languages have data types for representing integers and numbers that have decimal parts. For example, Fortran has two byte and four byte integers and four byte and eight byte real numbers (also called single and double precision). In C, these four types are called `int`'s, `long`'s, `float`'s, and `double`'s. In this section we describe how these numbers are actually stored in the computer. We describe the situation of Sun workstations and IBM PC's and compatibles.

Integers

Suppose that i is a two byte integer and let $b_{15}, \ldots, b_1, b_0$ represent the 16 bits in the representation of i, that is, b_{15} is the most significant bit and b_0 is the least significant bit. Then:

- If $i \geq 0$, then $b_{15} = 0$ and the remaining bits are the binary representation of i, that is, b_i is the coefficient of 2^i.
- If $i < 0$, then to get the values of the b's:
  1. Write the binary representation of $-i$.
  2. Complement the bits, that is, change the 0's to 1's and the 1's to 0's.
  3. Add 1 to the result of step 2.

For example, if $i = -149$ we have (writing b_{15} through b_0 from left to right):

```
149          = 128 + 16 + 4 +1
             = 0000 0000 1001 0101
complement:    1111 1111 0110 1010
add 1:         1111 1111 0110 1011
bytes:            255       107
```

Four byte integers are done the same way except they have 32 bits. Note that two byte integers can range from $-32,768$ to 32,767, while four byte integers range from -2×10^9 to 2×10^9.

Real (Floating Point) Numbers

For a four byte real number, the most significant bit (b_{31}) for a number x is a sign bit, that is, it is 0 if $x \geq 0$ and 1 if $x < 0$. The remaining 31 bits are divided into a set b_{30} through b_{23} of eight bits to represent the exponent of x while the remaining 23 bits represent the fractional part of x, which is also called the mantissa or significand of x. To determine the representation of x, the following steps are followed (consider $x = -197.625$ for example):

1. Write the binary representation of $|x|$, for example,

```
197.625 D = 128 + 64 + 4 + 1 + 1/2 + 1/8
          = 11000101.101 B
          = 1.1000101101 x 2\^{\ }7 B
```

2. The bits b_{30} through b_{23} are the binary representation of the exponent plus 127 (the 127 is called a bias):

```
134 = 128 + 4 + 2
    = 10000110
```

3. The bits b_{22} through b_0 are the bits to the right of the binary point in step 1 (note that we don't need to store the fact that there is a 1 to the left of the binary point):

```
10001011010000000000000
```

Thus, the representation of -197.625 is:

```
-197.625 = 1100 0011  0100 0101  1010 0000  0000 0000
         =    195        69          160         0
```

This makes it so that

$$8.43 \times 10^{-37} \leq |x| \leq 3.37 \times 10^{38},$$

with about 6 or 7 decimal place accuracy.

For an eight byte real, b_{63} is a sign bit, the next 11 bits are the binary representation of the exponent plus 1023 (which is $2^{10} - 1$), while the last 52 bits are the significand. This makes it so that

$$4.19 \times 10^{-307} \leq |x| \leq 1.67 \times 10^{308},$$

with about 15 or 16 decimal place accuracy.

Character Variables

In Fortran, each character in a character variable occupies one byte of memory so that there can be at most 256 such characters (each character has a numeric code associated with it called its ASCII code and these codes range from 0 to 255). On most computers, the first 32 characters are reserved for special purposes (such as signals to printers), while the second 128 characters are reserved for special purposes. Thus characters numbered 32 through 127 are the ones commonly used in creating text. A file that only has these characters is called an "ASCII file," and any file that has characters other than these cannot be guaranteed to be handled properly in going from one type of computer to another (one of the compelling reasons to use TeX is that it uses only ASCII files which are perfectly portable from one computer to another). In the table below we give the ASCII characters having codes 32 through 127. Note that the capital letters have ASCII codes 65 through 90 while the lower case letters are from 97 through 122.

```
       0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
-----------------------------------------------------
 32       !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
 48    0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
 64    @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
 80    P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _
 96    `  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
112    p  q  r  s  t  u  v  w  x  y  z  {  |  }  ~
```

An Example

To illustrate what we have been discussing, consider the following Fortran program

```
      integer*2 i
      real*4 x
      character*1 ci(2),cx(4)

      equivalence (i,ci),(x,cx)

      i=-149
      write(*,10) i,(ichar(ci(j)),j=1,2)
10    format(1x,3i10)

      x=-197.625
      write(*,20) x,(ichar(cx(j)),j=1,4)
20    format(1x,f10.4,4i10)

      stop
      end
```

which uses two uncommon features of Fortran: 1) the intrinsic function `ichar` which returns the ASCII code of a character, and 2) the `equivalence` statement which tells the compiler to store the two variables it names in the same location. This allows us to see how the numbers we have used as examples (-149 and -197.625) are represented. On a PC, the following results are obtained from the program:

```
      -149       107       255
 -197.6250         0       160        69       195
```

whereas on a Sun Sparcstation we get

```
      -149       255       107
 -197.6250         0       160        69       195
 -197.6250       195        69       160         0
```

This shows that the bytes are stored in reverse order on the two platforms.