1999/2000 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Floating Point Format Conversion

One of the breakthrough introductions of the Intel x86 family of processors was its floating point unit (FPU). Although the concept of performing floating point operations in hardware rather than software was nothing new, the careful design for the representation of values and behavior of such values became the foundation of the IEEE-754 floating point standard, which most modern computer architectures employ.

Given the format definitions in Tables 1–4, you are to write a program that takes various floating point representations and converts them to their IEEE-754 single precision (32-bit) equivalent value. When more bits of precision exist in the original value than in the IEEE conversion, always round the value towards zero. For floating point formats that can represent values outside the [min,max] range of IEEE-754 single precision, convert the values outside [min,max] to the extreme values *-Infinity* or *+Infinity* for negative and positive values, respectively. For floating point formats that can represent values smaller than the smallest normalized value for an IEEE-754 single precision value, round down smaller values to zero, preserving the sign of the original value.

Table 1. IEEE-754 32- and 64-bit — Format Type: **ieee-32, ieee-64**

According to IEEE-754, any binary floating point number other than zero has a normalized representation, which is similar to scientific notation, of 1 and some binary fraction, m, multiplied by some power of 2:

$$1.m \times 2^E$$

For example, the value $2\frac{1}{2}$ (decimal) would have the binary representation 10.1; when normalized, $2\frac{1}{2}$ is expressed as 1.01×2^1. Thus, $m = 01$, and $E = 1$. The IEEE-754 32-bit format has a sign bit, s, in bit number 31; an excess-127 exponent, e, in bits 23–30, and a normalized mantissa, m, in bits 0–22.

```
 3          2          1
10987654321098765432109876543210
seeeeeeeemmmmmmmmmmmmmmmmmmmmmm
```

To construct an IEEE-754 32-bit value, the whole number 1 prior to the radix point is stripped off and *assumed* to be present in the number, leaving only the the fractional part, m. 127 (1111111_2) is added to E, hence the description *excess-127;* the value added to E is often called the *bias*. The sign bit is zero if the number is positive, 1 if negative. The excess-127 2's exponent is *right justified* in the 8-bit exponent, while m with its leading zeros retained is *left justified* in the 23-bit mantissa. $2\frac{1}{2}$ is represented by the binary bit pattern `01000000001000000000000000000000`, or in hexadecimal as `40200000`.

A 64-bit IEEE 754 floating point value is similar in construction to a 32-bit value—but the exponent, e, is 11 bits wide stored as excess-1023, and the mantissa is 52 bits.

```
   6          5          4          3          2          1
3210987654321098765432109876543210987654321098765432109876543210
seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
```

Zero is a word with $e = 0$ and $m = 0$, which allows for both negative ($s = 1$) and positive ($s = 0$) values of zero. The extreme value *Infinity* is represented with every bit in e set to 1 and $m = 0$; positive Infinity has $s = 0$; negative Infinity has $s = 1$. Floating point undefined, *Not-a-Number* or *NaN*, is represented by every bit in e set to 1 and any value $m \neq 0$; positive and negative NaN takes $s = 0$ and 1, respectively. When translating to IEEE-754 NaN, choose $m = 1$ left justified in the mantissa.

Problem 2
Floating Point Format Conversion (continued)

Table 2. Control Data 170-state 60-bit — Format Type: **cdc-60**

A Control Data binary floating point number other than zero is represented by a binary integer coefficient, k, multiplied by some power of 2:

$$k \times 2^E$$

A single precision Control Data 170-state floating point value is contained in a 60-bit word. Bit 59 is the sign bit s. The exponent is right justified in bits 48–58, stored in excess-1024 ($e = E + 1024$) format; the coefficient, k, is 48 bits, filling out bits 0–47. Normalization occurs when the highest order 1 in k is shifted left into bit 47, with e decremented by one for each left shift. Normalization alters k and E, but does not change the formula $k \times 2^E$.

```
          5         4         3         2         1
9876543210987654321098765432109876543210987654321098765432109876543210
seeeeeeeeeeekkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk
```

Positive zero is represented by a word with all 60 bits equal to zero. Negative zero is represented by a word with all 60 bits equal to one. Positive values occur when $s = 0$. To represent negative values, the bit pattern for the absolute value is constructed, then the one's complement is performed on the entire 60 bits. Control Data floating point format can represent extreme values. +Infinity is represented when $s = 0$ and $e = 11111111111_2$ (the maximum biased exponent); -Infinity is the one's complement, $s = 1$ and $e = 0$. +Indefinite (floating point undefined) occurs when $s = 0$ and $e = 01111111111_2$ (the unbiased value $E = -0$ in one's complement form); -Indefinite is the one's complement, $s = 1$, $e = 10000000000_2$.

Table 3. Gould 32- and 64-bit — Format Type: **sel-32, sel-64**

In the Gould format, a floating point number other than zero has a normalized representation with a fraction, f, in the range $[\frac{1}{16},1)$ multiplied by some power of 16:

$$0.f \times 16^E$$

Put another way, the floating point value is represented by a hexadecimal fraction $0.Hhhhhh \times 16^E$, where H and h are hexadecimal digits, with the first digit H being non-zero. The Gould 32-bit format has a sign bit, s, in the high order bit (bit number 0), an excess-64 exponent, $e = E + 64$, in bits 1–7, and a normalized fraction, f, in bits 8–31.

```
          1         2         3
01234567890123456789012345678901
seeeeeeeffffffffffffffffffffffff
```

A 64-bit Gould floating point number differs from the 32-bit representation by having a 56-bit fraction in bits 8–63.

```
          1         2         3         4         5         6
0123456789012345678901234567890123456789012345678901234567890123
seeeeeeeffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Zero is represented by a word with all 32 or 64 bits equal to zero. The sign bit, s, is zero for positive values. For negative values, the bit pattern for the absolute value is formed, then the integer two's complement is performed on the entire 32 or 64 bits to arrive at the negative representation. There are no extreme values in Gould floating point format.

Problem 2
Floating Point Format Conversion (still continued)

Table 4. IBM System 360 32- and 64-bit — Format Type: **ibm-32, ibm-64**

The IBM 360 format for floating point values is similar to the Gould format. A nonzero normalized floating point value is represented by a fraction in the range [$\frac{1}{16}$,1) times a power of 16:

$$0.f \times 16^E$$

Like the Gould format, this may be interpreted as a hexadecimal fraction $0.Hhhhhh \times 16^E$, where H is nonzero. An IBM 32-bit value has a sign bit, s, in the high order bit (bit number 0), an excess-64 exponent, $e = E + 64$, in bits 1–7, and a normalized fraction, f, in bits 8–31.

```
          1         2         3
01234567890123456789012345678901
seeeeeeeffffffffffffffffffffffff
```

An IBM 64-bit floating point value differs from the 32-bit format only by having a longer fraction in bits 8–63.

```
          1         2         3         4         5         6
0123456789012345678901234567890123456789012345678901234567890123
seeeeeeeffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Zero is represented by a word with all 32 or 64 bits equal to zero. The sign bit, s, is zero for positive values. For negative values, s=1. There are no extreme values in IBM floating point format.

Input to your program is a series of whitespace-delimited format/value pairs, one pair per line. The format is the input floating point format identified by the *Format Type* specification in Tables 1–4. The value is a hexadecimal representation of a value in the given format. The value is always a big-endian sequence of hexadecimal digits. Input values will always be normalized in their respective formats. *The third column of data in the printed sample input is the decimal value of the number represented in the second column. It will not appear in the judges' input.*

For each format/value pair, print on a separate line starting in the first column a big-endian 32-bit (8 hexadecimal digit) value that represents the appropriate IEEE-754 single precision conversion.

Sample Input

```
sel-32 bed80000                    -2.5
cdc-60 c00123456789abc             -Indefinite
ibm-32 c1f00000                    -15.0
ieee-64 8010000000000000           -2.2250738585072014e-308
ieee-64 7feffffffffffff            1.7976931348623157e+308
```

Sample Output

```
c0200000
ffc00000
c1700000
80000000
7f800000
```