Chapter 1. Architecture and Machines

Problem 1.8

A certain computation results in the following (hexadecimal) representation:

Fraction	`+1.FFFF...`
Exponent (unbiased)	`+F` (radix 2)

Show the floating-point representation for the preceding in:

a. IBM long format.

b. IEEE short format.

c. IEEE long format.

We are given a floating point value `+1.FFFF...` with an unbiased exponent of `+F` which corresponds to the value 2^{+15}. For simplicity (and clarity), we will represent this as `+1.FFFF...`$\times 2^{+15}$. the first thing to note is that any rounding (truncation not being considered rounding) will result in rounding the value up. Taking rounding into account would result in the (finite) value `+2.0000...0`$\times 2^{+15}$. We will use this as the basis for our answers in the following solutions.

Note that the form `1234ABCD` will be used to represent hex values and *1010101* will be used to represent binary values. This notation is being used since the numbers that are being used are, by necessity, fractional and traditional representations (for example, the C language form `0x1234abcd`) don't seem as clear when dealing with fractional values. Also note that since there are no clear specifications of the actual encoding of the sign, mantissa, and characteristic into a machine word for the different representations we will only present the signed mantissa and the characteristic—true to their specified lengths and in the same number system but not packed into a machine representation.

a. IBM long format. IBM long format—hex-based (base 16) with no leading digit, characteristic bias 64, 14 digit mantissa.

   We need to convert the value to use an exponent base of 16 and then adjust it to get a normalized form for this representation. A normalized IBM representation has its most-significant (non-zero) digit immediately to the right of the decimal point. We also need to convert the exponent to correspond to base 16—this will fall out as we perform the conversion. Thus, shifting the mantissa for the value `+2.0000...0`$\times 2^{+15}$ right one power of 2—one bit—(don't forget to adjust the exponent the right direction!) we get `+1.0000...0`$\times 2^{+16}$ which can be rewritten using a hex-based exponent as `+1.0000...0`$\times 16^{+4}$.

   We now have the right base for the exponent but do not have a valid IBM floating-point value. To get this we need to shift the mantissa right by one hex digit—four bits—resulting in `+0.1000...0`$\times 16^{+5}$. This is a valid hex-based mantissa that has a non-zero leading digit. Note that the leading three bits of this representation are zero and represent wasted digits as far as precision is concerned—they could have been much more useful holding information at the other end of the representation.

   Next, in order to get the characteristic for this value we need to add in the bias value to the exponent—which is 64. This results in a characteristic of +69 (in a hex representation we have `45`. Finally, we need to limit the result to 14 hex digits—a total of 56 bits although the loss of

precision in the leading digit makes this comparison less exact—which gives us a mantissa of `+0.10000000000000` with a characteristic value of `45`.

If rounding is not taken into account the (truncated) mantissa would be `+0.FFFFFFFFFFFFF` with a characteristic value of `44`

b. IEEE short format. IEEE short format—binary based, leading (hidden) 1 to the left of the decimal point, characteristic bias 127, 23 bit mantissa (not including the hidden 1).

As before, we need to adjust the value to get it into a normalized form for this representation. A normalized IEEE (short or long) has a leading (hidden) 1 to the left of the decimal point. Thus, shifting the mantissa right one bit we get `+1.0000...0`$\times 2^{+16}$ which, miraculously, is normalized!. The key point to note is that the *representation* of mantissa is all zeros despite the fact that the mantissa itself is non-zero. This is due to the fact that the leading 1 is a hidden value saving one bit of the mantissa and resulting in a greater precision for the physical storage. This may be written as `+(1).0000...0`$\times 2^{+16}$ to make the hidden 1 value explicit.

The question arises, if the mantissa (as stored) is all zeros, how do we distinguish this value from the actual floating point value is 0.00? This is really quite easy—although the mantissa is zero, the representation of the value as a whole is non-zero due to a non-zero characteristic. For simplicity and compatibility we define the representation of all zeros—both mantissa and characteristic—to *be* the value 0.00. This makes sense for two reasons. First, it allows simple testing for zero to be performed—floating point and integer representations for zero are the same value. Second, it maps relatively cleanly into the range of values represented in the floating point system—and, due to the use of the characteristic instead of a normal exponent, is smaller than the smallest representable value. Now, we need to compute the characteristic and convert the value to binary—both fairly straightforward.

Using the IEEE short format values of bias and mantissa size, we get

$$+(1).00000000000000000000000$$

for the mantissa and a value of 143 (or *10001111*) for the characteristic.

If rounding is not taken into account the (truncated) mantissa would be

$$+(1).11111111111111111111111$$

with a characteristic of 142 (or *10001110*).

c. IEEE long format. IEEE long format—binary based, leading (hidden) 1 to the left of the decimal point, characteristic bias 1023, 52 bit mantissa (not including the hidden 1).

As in the previous case, we get a normalized mantissa by shifting the mantissa right one bit resulting in `+1.0000...0`$\times 2^{+16}$. Now, we need to compute the characteristic and convert the value to binary—both fairly straightforward but using different values for both the bias and number of bits in the mantissa from the short form. This was done for the IEEE representation to ensure that both the range and precision of the values representable were scaled comparably as the size of the representation grew. Previously, only the mantissa had grown as the representation size was increased. This did not prove to be an effective use of bits.

Now, using the IEEE long format values of bias and mantissa size, we get

$$+(1).0000000000000000000000000000000000000000000000000000$$

for the mantissa and a value of 1039 (or *10000001111*) for the characteristic.

If rounding is not taken into account the (truncated) mantissa would be

$$+(1).1111111111111111111111111111111111111111111111111111$$

with a characteristic of 1038 (or *10000001110*).

Problem 1.9

Represent the decimal numbers (i) +123, (ii) −4321, and (iii) +00000 (zero is represented as a positive number) as:

a. Packed decimal format (in hex).

b. Unpacked decimal format (in hex).

Assume a length indicator (in bytes) is specified in the instruction. Show lengths for each case.

First, let's break down these numbers into sequences of BCD digits—from there we can easily produce the packed or unpacked forms.

(i) $+123 \rightarrow \{+, 1, 2, 3\}$

(ii) -4321 $\rightarrow \{-, 4, 3, 2, 1\}$

(iii) $+00000 \rightarrow \{+, 0, 0, 0, 0, 0\}$

Second, the answers will be presented in big-endion sequence for simplicity of reading. This is neither an endorsement of big-endion nor a rejection of little-endion as an architectural decision—only as a presentation mechanism.

Third, note that the hex notations for the sign differ between packed and unpacked representations. For the packed representation the 4-bit ("nibble") values are '`A`' for '`+`' and '`B`' for '`-`', while for the unpacked representation the byte values are '`2B`' for '`2D`' and '`B`' for '`-`'. The latter is standard ASCII text encoding (although we would typically write the sign bit first in text).

a. Packed decimal format (in hex).

For packed values we can put two digits in one byte. Recall that there must be an odd number of digits followed by a trailing sign field, and thus we must pad case (ii) which doesn't have this property. Thus, we get:

(i) {`12`, `3A`}, 2 bytes

(ii) {`04`, `32`, `1B`}, 3 bytes

(iii) {`00`, `00`, `0A`}, 3 bytes

b. Unpacked decimal format (in hex).

For unpacked values we put only one digit in a byte so that there are no restrictions with respect to length. Thus we get:

(i) {`31`, `32`, `33`, `2B`}, 4 bytes

(ii) {`34`, `33`, `32`, `31`, `2D`}, 5 bytes

(iii) {`30`, `30`, `30`, `30`, `30`, `2B`}, 6 bytes

Problem 1.10

(a) R/M machine

In this solution, the assumption made is that there are no unnecessary memory addresses used. The necessary memory addresses are `Addr`, `Bddr`, and `Cddr` to hold the coefficients for the quadratic equation and `ROOT1` and `ROOT2` to hold the solutions.

And the `R/M` version:

LD	R1, BASE	; load offset address
LD.D	F5, Bddr[R1]	; F5 $\leftarrow$ B
MPY.D	F5, F5	; F5 $\leftarrow$ B^2
LD.D	F6, #4.0	; F6 $\leftarrow$ 4
MPY.D	F6, Addr[R1]	; F6 $\leftarrow$ 4A
MPY.D	F6, Cddr[R1]	; F6 $\leftarrow$ 4AC
SUB.D	F5, F6	; F5 $\leftarrow$ $B^2 - 4AC$
SQRT.D	F5	; F5 $\leftarrow$ $\sqrt{B^2 - 4AC}$
LD.D	F7, Baddr[R1]	; F7 $\leftarrow$ B
MPY.D	F7, #-1	; F7 $\leftarrow$ $-B$
MOV.D	F8, F7	; F8 $\leftarrow$ $-B$
SUB.D	F7, F5	; F7 $\leftarrow$ $-B - \sqrt{B^2 - 4AC}$
ADD.D	F8, F5	; F8 $\leftarrow$ $-B + \sqrt{B^2 - 4AC}$
LD.D	F9, Addr[R1]	; F9 $\leftarrow$ A
SHL.D	F9, #1	; F9 $\leftarrow$ 2A
DIV.D	F7, F9	; F7 $\leftarrow$ $\frac{-B-\sqrt{B^2-4AC}}{2A}$
DIV.D	F8, F9	; F8 $\leftarrow$ $\frac{-B+\sqrt{B^2-4AC}}{2A}$
ST.D	ROOT1[R1], F7	; ROOT1 $\leftarrow$ $\frac{-B-\sqrt{B^2-4AC}}{2A}$
ST.D	ROOT2[R1], F8	; ROOT1 $\leftarrow$ $\frac{-B+\sqrt{B^2-4AC}}{2A}$

(b) L/S machine

LD	R1,BASE	; load offset address
LD.D	F2,Bddr[R1]	; F2 $\leftarrow$ B
LD.D	F3,Cddr[R1]	; F3 $\leftarrow$ C
LD.D	F4,Addr[R1]	; F4 $\leftarrow$ A
MPY.D	F5,F2,F2	; F5 $\leftarrow$ B^2
MPY.D	F6,F3,F4	; F6 $\leftarrow$ AC
SHL.D	F6,#2	; F6 $\leftarrow$ 4AC
SUB.D	F6,F5,F6	; F6 $\leftarrow$ $B^2 - 4AC$
SQRT.D	F6,F6	; F6 $\leftarrow$ $\sqrt{B^2 - 4AC}$
MPY.D	F2,F2,#-1	; F2 $\leftarrow$ $-B$
SUB.D	F7,F2,F6	; F7 $\leftarrow$ $-B - \sqrt{B^2 - 4AC}$
ADD.D	F8,F2,F6	; F8 $\leftarrow$ $-B + \sqrt{B^2 - 4AC}$
SHL.D	F9,#1	; F4 $\leftarrow$ 2A
DIV.D	F7,F7,F9	; F7 $\leftarrow$ $\frac{-B-\sqrt{B^2-4AC}}{2A}$
DIV.D	F8,F8,F9	; F8 $\leftarrow$ $\frac{-B+\sqrt{B^2-4AC}}{2A}$
ST.D	ROOT1[R1],F7	; ROOT1 $\leftarrow$ F7
ST.D	ROOT2[R1],F8	; ROOT2 $\leftarrow$ F8