CHAPTER

9

Data Representation

Representation of Numeric Variables

The way in which numbers are represented in your SAS programs affects both the magnitude and the precision of your numeric calculations. The factors that affect your calculations are discussed in detail in *SAS Language Reference: Dictionary*. When processing in an OS/390 environment, the most important factor you should be aware of is the way in which floating-point numbers and integers are stored.

Representation of Floating-Point Numbers

All numeric values are stored in SAS data sets, which are maintained internally in floating-point (real binary) representation, in which values are represented as numbers between 0 and 1 times a power of 10. For example, 1234 (base 10) is .1234 times 10 to the 4th power, which is the same as the following expression:

$$\underbrace{.1234}_{\text{mantissa}} \; * \; \underbrace{10}_{\text{base}} \; ** \; \underbrace{4}_{\text{exponent}}$$

Under OS/390, floating-point representation (unlike scientific notation) uses a base of 16 rather than base 10. IBM mainframe systems all use the same floating-point representation, which is made up of 4, 8, or 16 bytes. SAS always uses 8 bytes, as follows:

```
SEEEEEEE MMMMMMMM MMMMMMMM MMMMMMMM
 byte 1   byte 2   byte 3   byte 4
MMMMMMMM MMMMMMMM MMMMMMMM MMMMMMMM
 byte 5   byte 6   byte 7   byte 8
```

This representation corresponds to bytes of data, with each character occupying 1 bit. The S in byte 1 is the sign bit of the number. If the sign bit is 0, the number is positive. Conversely, if the sign bit is one, the number is negative. The seven E characters in byte 1 represent a binary integer that is known as the characteristic. The *characteristic* represents a signed exponent and is obtained by adding the bias to the actual exponent. The *bias* is defined as an offset that is used to allow for both negative and positive exponents with the bias representing 0. If a bias was not used, an additional sign bit for the exponent would have to be allocated. For example, IBM mainframes employ a bias of 40 (base 16). A characteristic with the value 42 (base 16) represents an exponent of +2 (base 16), whereas a characteristic of 3D (base 16)represents an exponent of –3 (base 16).

The remaining M characters in bytes 2 through 8 represent the bits of the mantissa. There is an implied radix point before the most significant bit of the mantissa, which also implies that the mantissa is always strictly less than 1. The term *radix point* is used instead of decimal point because decimal point implies that we are working with decimal (base 10) numbers, which may not be the case.

The exponent has a base associated with it. Do not confuse this with the base in which the exponent is represented. The exponent is always represented in binary, but the exponent is used to determine what power of the exponent's base should be multiplied by the mantissa. In the case of the IBM mainframes, the exponent's base is 16.

Each bit in the mantissa represents a fraction whose numerator is 1 and whose denominator is a power of 2. For example, the most significant bit in byte 2 represents 1/2**1, the next most significant bit represents 1/2**2, and so on. In other words the mantissa is the sum of a series of fractions such as 1/2, 1/4, 1/8, and so on. Therefore, in order for any floating-point number to be represented exactly, you must be able to express it as the previously mentioned sum. For example, 100 is represented as the following expression:

```
(1/4 + 1/8 + 1/64) * (16**2)
```

The following two examples illustrate how the above expression is obtained. The first example is in base 10. In decimal notation, the value 100 is represented as follows:

```
100.
```

The period in this number is the radix point. The mantissa must be less than 1; therefore, you normalize this value by shifting the number three places to the right, which produces the following value:

```
.100
```

Because the number was shifted three places to the right, 3 is the exponent, which results in the following expression:

```
.100*(10**3)=100
```

The second example is in base 16. In hexadecimal notation, 100 (base 10) is written as follows:

```
64.
```

The period in this number is the radix point. Shifting the number two places to the left produces the following value:

```
.64
```

Shifting the number two places also indicates an exponent of 2. Rewriting the number in binary produces the following value:

```
.01100100
```

Finally, the value .01100100 can be represented in the following expression:

```
(1/2)**2 + (1/2)**3 + (1/2)**6 = 1/4 + 1/8 + 1/64
```

In this example, the exponent is 2 (base 10). To represent the exponent, you add the bias of 64 (base 16) to the exponent. The hexadecimal representation of the resulting value, 66, is 42. The binary representation is as follows:

```
01000010 01100100 00000000 00000000
00000000 00000000 00000000 00000000
```

Floating-point numbers that have negative exponents are represented with characteristics that are less than 40 (base 16). When you subtract 40 from a number that is less than 40, the difference is a negative value that represents the exponent. An example of such a number is the floating-point representation of .03125 (base 10), which is

```
'3F80000000000000'x
```

Subtracting 40 (base 16) from the characteristic 3F (base 16) gives an exponent of –1 (base 16). This exponent is applied to the 14-digit fraction, 80000000000000 (base 16), giving a value of .08 (base 16), which is equal to .03125 (base 10).

Representation of Integers

Like other numeric values, SAS maintains integer variables in 8-byte floating-point (real binary) representation. But under OS/390, outside of SAS, integer values are typically represented as 4-byte (fixed point) binary values using two's complement notation. SAS can read and write these values using informats and formats, but it does not process them internally in this form. SAS uses floating-point representation internally.

You can use the IB*w.d* informat and format to read and write the binary integer values used under OS/390. Each integer uses 4 bytes (32 bits) of storage space; thus, the range of values that can be represented is from -2,147,483,648 to 2,147,483,647.

Using the LENGTH Statement to Save Storage Space

By default, when SAS writes a numeric variable to a SAS data set, it writes the number in IBM double-wide floating-point format (as described in "Representation of Floating-Point Numbers" on page 143). In this format, 8 bytes are required for storing a number in a SAS data set with full precision. However, you can use the LENGTH statement in the DATA step to specify that you want to store a particular numeric variable in fewer bytes.

Using the LENGTH statement can greatly reduce the amount of space required for storing your data. For example, if you were storing a series of test scores whose values could range from 0 to 100, you could use numeric variables with a length of 2 bytes. This would save 6 bytes of storage per variable for each observation in your data set.

However, you must use the LENGTH statement cautiously in order to avoid losing significant data. One byte is always used to store the exponent and the sign. The remaining bytes are used for the mantissa. When you store a numeric variable in fewer than 8 bytes, the least significant digits of the mantissa are truncated. If the part of the mantissa that is truncated contains any nonzero digits, then precision is lost.

Use the LENGTH statement only for variables whose values are always integers. Fractional numbers lose precision if they are truncated. In addition, you must ensure that the values of your variable will always be represented exactly in the number of bytes that you specify. Use the following table to determine the largest integer that can be stored in numeric variables of various lengths:

Table 9.1 Variable Length and Largest Exact Integer

Length in Bytes	Significant Digits Retained	Largest Integer Represented Exactly
2	2	256
3	4	65,536
4	7	16,777,216
5	9	4,294,967,296
6	12	1,099,511,627,776
7	14	281,474,946,710,656
8	16	72,057,594,037,927,936

Note: No warning is issued when the length that you specify in the LENGTH statement results in truncated data. △

How Character Values Are Stored

Alphanumeric characters are stored in a computer using a character-encoding system known as a *collating sequence*, where one or two bytes represents a given character. The two single-byte character-encoding systems that are most widely used in data processing are ASCII and EBCDIC. IBM mainframe computers use the EBCDIC system. The EBCDIC system can be used to represent 256 different characters. Each character is assigned a unique hexadecimal value between 00 and FF.

Table 9.2 on page 146 shows the EBCDIC code for commonly used characters.

Table 9.2 EBCDIC Code: Commonly Used Characters

Hex	Character	Hex	Character	Hex	Character	Hex	Character
'40'x	space	'95'	n	'C4'	D	'E3'	T
'4B'	.	'96'	o	'C5'	E	'E4'	U
'4E'	+	'97'	p	'C6'	F	'E5'	V
'60'	-	'98'	q	'C7'	G	'E6'	W
'81'	a	'99'	r	'C8'	H	'E7'	X
'82'	b	'A2'	s	'C9'	I	'E8'	Y
'83'	c	'A3'	t	'D0'	}	'E9'	Z
'84'	d	'A4'	u	'D1'	J	'F0'	0
'85'	e	'A5'	v	'D2'	K	'F1'	1
'86'	f	'A6'	w	'D3'	L	'F2'	2
'87'	g	'A7'	x	'D4'	M	'F3'	3
'88'	h	'A8'	y	'D5'	N	'F4'	4
'89'	i	'A9'	z	'D6'	O	'F5'	5

'91'	j	'C0'	{	'D7'	P	'F6'	6
'92'	k	'C1'	A	'D8'	Q	'F7'	7
'93'	l	'C2'	B	'D9'	R	'F8'	8
'94'	m	'C3'	C	'E2'	S	'F9'	9

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Companion for the OS/390 Environment, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS® Companion for the OS/390® Environment, Verison 8

1–58025–523–X

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, November 1999