PharmaSUG China 2021 - Paper CC-073

Float-point number and encode in SAS

Weineng Zhou, Parexel International, Shanghai, China

ABSTRACT

SAS can process two type of data, numeric and character. IEEE 754-2008 double-precision binary floating-point number are used by SAS on Windows, Linux, and UNIX platforms. The calculation of numerical variables and decimal place retention are based on floating-point data in SAS, it is necessary for SAS programmer to understand floating-point data. Moreover, we often run into data encoding issue when processing character clinical data globally and locally. Due to variety languages in the world, we may involves variety encode version of SAS, such as ASCII, GB2312 (简体中文), wlatin1 and UTF-8, etc. We often get confused about how to process the special characters, so it is helpful for us to know about encoding principle of character. This article may help you understand floating-point data and character encoding through several examples, especially for SAS user.

INTRODUCTION

As we all know, SAS plays an important role in the clinical trial data statistics process. Creating ADaM and TFL is almost our daily work. When generating a table, we often use *put* and *round* function to keep decimal places. But you many noticed that sometimes you get different results. In addition, we also often run into data encoding issue when manipulating character variables in SDTM and ADaM data set. With the clinical trials go to global, and data collection goes to electronic data capture (EDC), the clinical trial data are entered directly by the investigational sites no matter whether the sites are in English-speaking countries or the non-English speaking countries. So the data my coded in different encoding systems. In view of this, this article aims to dissect the nature of floating-point data and data encoding issue.

COMPUTER AND BINARY

1.COMPUTER STRUCTURE

Before we describe the topic in this paper, let's know about computer and binary. As you know, we still use the computer of the von Neumann structural system, which consist of Input device, memory, arithmetic unit, controller and output device. (Fig. 1)

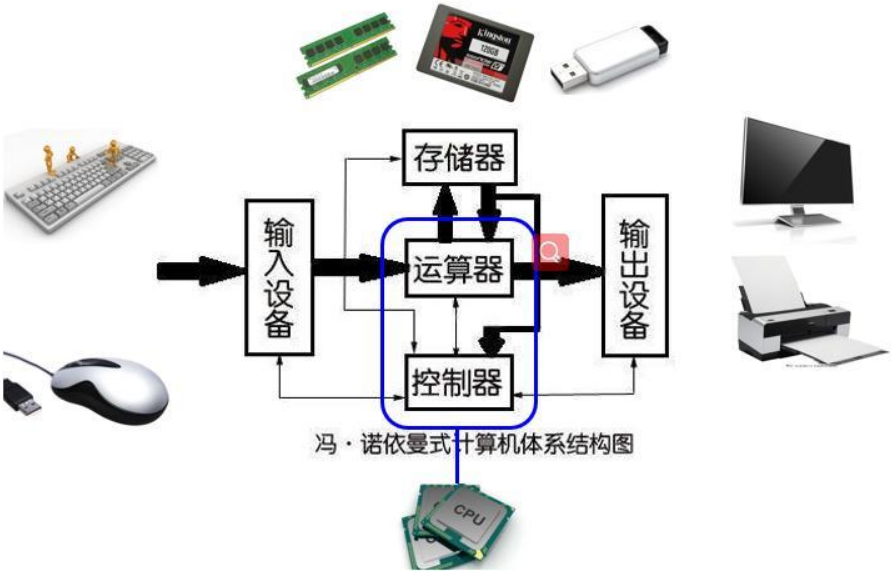


Fig. 1 the computer of the von Neumann structural system

2.BINARY

Actually, the computer only can recognize the "high" and "low" of the physical logic circuit (logic level), the "on" and "off" of the switch, the "bright" and "off" of the bulb, etc. The numbers in mathematics are encoded as 1 and 0, so the computer only can recognize 1 and 0. We can use eight light bulbs and one power supply to simulate a computer with 8-bit operating system (Fig. 2). The binary number 00000001 or decimal 1 could be encoded when only the first one is on; The binary number 00000010 or decimal 2 could be encoded when only the second one is on; The binary number 00000011 or decimal 3 could be encoded when both the first and second bulb is on. So, the max positive integer 01111111 could be encoded when the highest bit as a sign, which corresponding decimal is $2^7 - 1$=127.

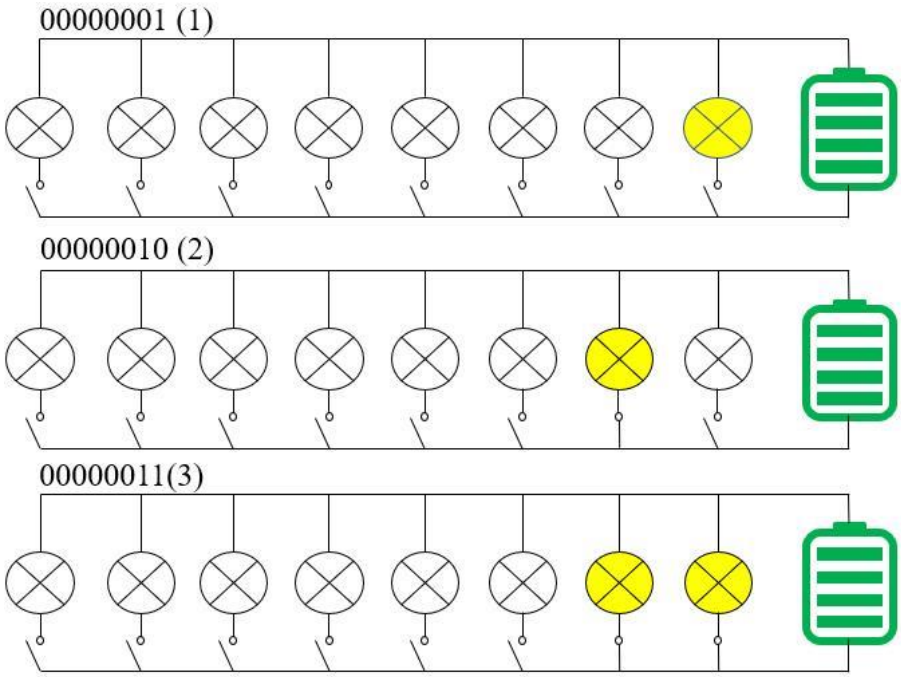


Fig. 2 simulate a computer with 8-bit operating system

3.CALCULATION IN BINARY

Actually, the bottom layer of the computer is binary operation, that is, every two in one. In mathematics, we use decimal arithmetic. Fortunately, binary and decimal operations is easy to convert. Let's see some simple calculation, herein, some calculation is executed in SAS.

Example 1:

Herein, we could map decimal to binary using proc format:

```
proc format;
value dec2bin
    0=0
    1=1
    2=10
    3=11
    4=100
    5=101
    6=110
```

```
        7=111
        8=1000
        9=1001
        10=1010
        11=1011
        12=1100
        13=1101
        14=1110
        15=1111
        16=10000
        ...
        64=1000000
        ;
quit;
```

In decimal, the calculation is as follows:

```
data dec;
        x1=1+1;
        x2=1+2;
        x3=2+3;
        x4=3+4;
        x5=4+5;
        x6=5+6;
        x7=8+8;
        x8=16+16;
        x9=32+32;
        proc print;
run;
```

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9
1	2	3	5	7	9	11	16	32	64

In binary, the calculation is as follows:

```
data ten2bin;
        x1=1+1;
        x2=1+2;
        x3=2+3;
        x4=3+4;
        x5=4+5;
        x6=5+6;
        x7=8+8;
        x8=16+16;
```

```
    x9=32+32;
    format x1-x9 dec2bin.;
    proc print;
run;
```

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9
1	10	11	101	111	1001	1011	10000	100000	1000000

Binary to decimal can be converted by the formula:

$$b_n b_{n-1} \cdots b_2 b_1 b_0 = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

For example, binary x6 could be converted to decimal by the formula:

$$1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11$$

FLOATING-POINT DATA

As the accuracy requirements for numerical calculations, floating-point data begins to be generated. Floating-point number is consist of **single-precision** and **double-precision**. In computing, floating-point arithmetic (FP) is arithmetic using formulaic representation of real numbers as an approximation so as to support a trade-off between range and precision. The IEEE (Institute of Electrical and Electronics Engineers) standardized the computer representation for binary floating-point numbers in IEEE 754. It is a scientific notation, expressed by a **sign** (positive or negative), **exponent** and **mantissa**, and the base number is determined to be 2. That is to say, the floating-point number is represented as the mantissa multiplied by the exponential power of 2 followed by the symbol. The specific format is as follows (Table 1):

Table 1 The IEEE 754 double-precision binary floating-point number

	Sign	Exponent domain	Mantissa domain	Exponential offset
single	1bit [31]	8bit [30-23]	23bit [22-00]	127
double	1bit [63]	8bit [62-52]	23bit [51-00]	1023

The IEEE 754 double-precision binary floating-point format used by SAS on Windows, Linux, and UNIX platforms can be represented by the formula:

$$N = (sign) * 2^{exp-1023} \times (1 + \frac{b1}{2^1} + \frac{b2}{2^2} + \frac{b3}{2^3} + \cdots + \frac{b52}{2^{52}})$$

For example, we could look at the binary number using put function in SAS as follow:

Example 2:

```
data _null_;
    x=-9.625;
    bin=put(x,binary64.);
    put bin;
run;
```

```
1     data _null_;
2         x=-9.625;
3         bin=put(x,binary64.);
4         put bin;
5     run;

1100000000100011010000000000000000000000000000000000000000000000
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds
```

Decimal: -9.625

Binary: 1 10000000010 0011010000000000000000000000000000000000000000000000

Where the highest bit is as a sign -1;

The exponent domain is exp=2**10+2**1-1023=3

The decimal is: N = (-1) ×2**3×(1+1/2**3+1/2**4+1/2**6)=(-1) ×[2**3+2+2**(-1)+2**(-3)]

=-9.625

The max floating-point numbers in IEEE 754 is

0 11111111110 1111111111111111111111111111111111111111111111111111,

Which could be converted to decimal by the formula:

Max = 2**(2046-1023) *(1 + 1/2**1 + 1/2**2+…+ 1/2**52) =

2**1023*($\frac{1-0.5^{53}}{1-0.5}$) =2**1023*1.999999999999998 ≈1.7976931348623157E308< 2**1024

In SAS, the max floating-point number is less than 2**1024:

```
data _null_;
    max=2**1024;
    put max;
run;
```

```
1     data _null_;
2         max=2**1024;
NOTE: Invalid argument(s) to the exponential operator "**".
3         put max;
4     run;

.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

In the following example, some floating-point numbers are presented by decimal and binary.

Example 3:

```
data nums;
    length n 8 sign $1 exponent11 $11 exponent 4. mantissa52 $52 bits64 $64;
    input n @@;
    bits64 = put(n,binary64.);
    sign = substr(bits64,1,1);
    exponent11 = substr(bits64,2,11);
    exponent=input(substr(bits64,2,11),binary12.)-1023;
    mantissa52 = substr(bits64,13,52);
    put n '= ' @;
```

```
        e = -exponent;
        put '1/2^' e @;
        do i=1 to 52;
        e = -exponent + i;
        if (substr(mantissa52,i,1)='1') then put '+ 1/2^' e @;
        end;
        put;
        format n best19.;
        cards;
        1 2 3 .5 .75 1.5 .1
        ;
    run;
```

```
1    data nums;
2        length n 8 sign $1 exponent11 $11 exponent 4. mantissa52 $52 bits64 $64;
3        input n @@;
4        bits64 = put(n,binary64.);
5        sign = substr(bits64,1,1);
6        exponent11 = substr(bits64,2,11);
7        exponent=input(substr(bits64,2,11),binary12.)-1023;
8        mantissa52 = substr(bits64,13,52);
9        put n '= ' @;
10       e = -exponent;
11       put '1/2^' e @;
12       do i=1 to 52;
13       e = -exponent + i;
14       if (substr(mantissa52,i,1)='1') then put '+ 1/2^' e @;
15       end;
16       put;
17       format n best19.;
18       cards;

1 = 1/2^0
2 = 1/2^-1
3 = 1/2^-1 + 1/2^0
0.5 = 1/2^1
0.75 = 1/2^1 + 1/2^2
1.5 = 1/2^0 + 1/2^1
0.1 = 1/2^4 + 1/2^5 + 1/2^8 + 1/2^9 + 1/2^12 + 1/2^13 + 1/2^16 + 1/2^17 + 1/2^20 + 1/2^21 + 1/2^24 + 1/2^25 + 1/2^28
+ 1/2^29 + 1/2^32 + 1/2^33 + 1/2^36 + 1/2^37 + 1/2^40 + 1/2^41 + 1/2^44 + 1/2^45 + 1/2^48 + 1/2^49 + 1/2^52 + 1/2^53
+ 1/2^55
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.NUMS has 7 observations and 8 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds


20       ;
21   run;
```

Looking at the result, the decimal floating-point number could be expressed as polynomial fractions.

	n	sign	exponent11	exp	mantissa52	bits64
1	1	0	01111111111	0	0000000000000000000000000000000000000000000000000000	0011111111110000000000000000000000000000000000000000000000000000
2	2	0	10000000000	1	0000000000000000000000000000000000000000000000000000	0100000000000000000000000000000000000000000000000000000000000000
3	3	0	10000000000	1	1000000000000000000000000000000000000000000000000000	0100000000001000000000000000000000000000000000000000000000000000
4	0.5	0	01111111110	-1	0000000000000000000000000000000000000000000000000000	0011111111100000000000000000000000000000000000000000000000000000
5	0.75	0	01111111110	-1	1000000000000000000000000000000000000000000000000000	0011111111101000000000000000000000000000000000000000000000000000
6	1.5	0	01111111111	0	1000000000000000000000000000000000000000000000000000	0011111111111000000000000000000000000000000000000000000000000000
7	0.1	0	01111111011	-4	1001100110011001100110011001100110011001100110011010	0011111110111001100110011001100110011001100110011001100110011010

In the case of the binary representation of .1, we find that the mantissa bit pattern “0011” repeats. The last bit in the mantissa breaks the pattern because of floating-point rounding rules. In fact, we can never exactly store 1/10 as a binary floating-point number. The polynomial approximation of the decimal value 0.1 can be expressed as:

$$0.1 = \frac{3}{2^5} + \frac{3}{2^9} + \frac{3}{2^{13}} + \frac{3}{2^{17}} + \frac{3}{2^{21}} + \frac{3}{2^{25}} + \frac{3}{2^{29}} + \frac{3}{2^{33}} + \frac{3}{2^{37}} + \frac{3}{2^{41}} + \frac{3}{2^{45}} + \frac{3}{2^{49}} + \frac{3}{2^{53}} + \frac{1}{2^{55}}$$

This simplifies to:

$$\frac{3602879701896397}{36028797018963968}$$

Or (in decimal): 0.10000000000000000555111512312578270211815834045410156 25

```
    data test1;
        x=3602879701896397/36028797018963968;
    run;
```

```
data test2;
    x=0.1;
run;
proc compare b=test1 c=test2; run;
```

the result could be as follow:

```
                    Observation Summary

              Observation      Base  Compare

              First Obs           1        1
              Last  Obs           1        1

Number of Observations in Common: 1.
Total Number of Observations Read from WORK.TEST1: 1.
Total Number of Observations Read from WORK.TEST2: 1.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 1.

NOTE: No unequal values were found. All values compared are exactly equal.
```

It can be known from the results that 3602879701896397/36028797018963968 is equal 0.1 because of that computers use binary arithmetic with finite precision. As we know, addition follows the law of association in mathematics, but floating-point number may not follow the law of association, such as following calculation:

Example 4:

```
data a;
    x=0.1+0.2+0.3;
run;

data b;
    x=0.1+(0.2+0.3);
run;

proc compare b=a c=b criterion=1e-17; run;
```

the result could be reported as follow:

```
          Value Comparison Results for Variables

                  ||       Base    Compare
       Obs        ||          x          x      Diff.     % Diff
 ________         ||  _________  _________  _________  _________
                  ||
        1         ||     0.6000     0.6000  -1.11E-16  -1.85E-14
```

The essential cause of unequalness is calculation with floating-point rounding rules which lead to precision loss

In the following example, we could explain why 0.3 is equal 3/10, but different from 0.1*3.

```
data a;
    x=0.3;
    y=3/10;
    z=0.1*3;
    dif1=x-y;
    dif2=z-x;
    dif3=z-x;
    put x=binary64.;
    put y=binary64.;
    put z=binary64.;
    put dif1=;
    put dif2=;
    put dif3=;
run;
```

```
1    data a;
2        x=0.3;
3        y=3/10;
4        z=0.1*3;
5        dif1=x-y;
6        dif2=z-x;
7        dif3=z-y;
8        put x=binary64.;
9        put y=binary64.;
10       put z=binary64.;
11       put dif1=;
12       put dif2=;
13       put dif3=;
14   run;

x=0011111111010011001100110011001100110011001100110011001100110011
y=0011111111010011001100110011001100110011001100110011001100110011
z=0011111111010011001100110011001100110011001100110011001100110100
dif1=0
dif2=5.551115E-17
dif3=5.551115E-17
NOTE: The data set WORK.A has 1 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.03 seconds
```

Here, to get a better view of binary of x, y and z, we output the specify binary number using put function with format binary64., we could also find that the mantissa bit pattern “0011” repeats. In fact, we can never exactly store 3/10 or 0.3 as a binary floating-point number so that 0.3 is less than 0.1*3 because of precision loss. 0.3, 3/10 and 0.1*3 could be calculated by formula:

$$0.3=3/10 = 2^{1021-1023} \times (1+\frac{1}{2^3}+\frac{1}{2^4}+\frac{1}{2^7}+\frac{1}{2^8}+\frac{1}{2^{11}}+\frac{1}{2^{12}}+\frac{1}{2^{15}}+\frac{1}{2^{16}}+\frac{1}{2^{19}}+\frac{1}{2^{20}}+\frac{1}{2^{23}}+\frac{1}{2^{24}}+\frac{1}{2^{27}}+\frac{1}{2^{28}}+\frac{1}{2^{31}}+\frac{1}{2^{32}}+\frac{1}{2^{35}}+\frac{1}{2^{36}}+\frac{1}{2^{39}}+\frac{1}{2^{40}}+\frac{1}{2^{43}}+\frac{1}{2^{44}}+\frac{1}{2^{47}}+\frac{1}{2^{48}}+\frac{1}{2^{51}}+\frac{1}{2^{52}})$$

$$3*0.1 = 2^{1021-1023} \times (1+\frac{1}{2^3}+\frac{1}{2^4}+\frac{1}{2^7}+\frac{1}{2^8}+\frac{1}{2^{11}}+\frac{1}{2^{12}}+\frac{1}{2^{15}}+\frac{1}{2^{16}}+\frac{1}{2^{19}}+\frac{1}{2^{20}}+\frac{1}{2^{23}}+\frac{1}{2^{24}}+\frac{1}{2^{27}}+\frac{1}{2^{28}}+\frac{1}{2^{31}}+\frac{1}{2^{32}}+\frac{1}{2^{35}}+\frac{1}{2^{36}}+\frac{1}{2^{39}}+\frac{1}{2^{40}}+\frac{1}{2^{43}}+\frac{1}{2^{44}}+\frac{1}{2^{47}}+\frac{1}{2^{48}}+\frac{1}{2^{50}})$$

We could find that the difference between 0.3 and 0.1*3 is the polynomial fractions marked in red, the difference could be calculated as follow:

$$2^{1021-1023} \times (\frac{1}{2^{51}} + \frac{1}{2^{52}} - \frac{1}{2^{50}}) = \frac{1}{2^{53}} + \frac{1}{2^{54}} - \frac{1}{2^{52}} = -5.551115123125783e\text{-}17$$

which is equal the dif2 and dif3 in the log calculated by SAS.

When doing some clinical trial project, descriptive statistics is often used by **PROC MEANS** in SAS. Summary statistics will be often presented to the following degree of precision:

Table 2 Degree of precision of statistics

Statistics	Degree of Precision
n	Integer
Mean, Median	One more than the raw data, up to 3 decimal places
SD	Two more than the raw data, up to 3 decimal places
Minimum, Maximum	same as raw data
Percentage	One decimal place. A percentage of 100% will be reported as 100%. Percentages of zero will be reported as 0.

For example, mean would be two decimal places when raw data is one decimal place. One time, 2.025 was calculated by **PROC MEANS** in SAS:

Example 5:

proc means ……

VIEWTABLE: Work.A

	mean
1	2.025

```
data b;
input mean;
cards;
2.025
;
run;
proc compare b=a c=b; run;
```

the result is as follow, actually, the 2.025 calculated by **PROC MEANS** is less than the input number 2.025 because of floating-point rounding rules and precision loss.

Value Comparison Results for Variables

Obs	Base mean	Compare mean	Diff.	% Diff
1	2.0250	2.0250	8.882E-16	4.386E-14

Why 2.025 calculated by **PROC MEANS** is presented in SAS dataset, but not 2.025-8.882E-16≈2.024999999999…, because SAS rounds up the result if the argument is within 5E-11 in this case, for example, when we input x=2.02499999995 and y=2.025, they would be presented the same as 2.025.

```
data test;
    x=2.02499999995;
    y=2.025;
    z=y-x;
    put z;
run;
```

	x	y	z
1	2.025	2.025	5E-11

```
1    data test;
2        x=2.02499999995;
3        y=2.025;
4        z=y-x;
5        put z;
6    run;

5E-11
NOTE: The data set WORK.TEST has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

From binary to decimal:

```
data test;
    bin1='0100000000000000000110011001100110011001100110011001100110011001100110001';
    bin2='0100000000000000000110011001100110011001100110011001100110011001100110011';
    dec1=input(bin1, binary64.);
    dec2=input(bin2, binary64.);
    diff=dec2-dec1;
    put dec1 8.2 / dec2 8.2 / diff;
run;
```

```
1    data test;
2        bin1='0100000000000000000110011001100110011001100110011001100110011001100110001';
3        bin2='0100000000000000000110011001100110011001100110011001100110011001100110011';
4        dec1=input(bin1, binary64.);
5        dec2=input(bin2, binary64.);
6        diff=dec2-dec1;
7        put dec1 8.2 / dec2 8.2 / diff;
8    run;

    2.02
    2.03
8.881784E-16
NOTE: The data set WORK.TEST has 1 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

In fact, the corresponding decimal number dec1 is not exactly 2.025, as can be seen from the penultimate, the difference is Diff=dec2-dec1= $2^1 \times \frac{1}{2^{51}}$=8.881784197001252E-16, which agrees well with the result calculated by SAS, so we can conclude that, the float-point number that calculated or presented in SAS is not always the exact float-point number as it is due to floating-point rounding rules. The float-point number less than 2.025 could be presented by many binary numbers in computer with binary. In the following example, the binary-point number bin0, bin1, bin2, bin3 and bin4 is less than 2.025, but presented with 2.025 because SAS rounds up automatic.

bin5 is the real binary number of 2.025 that is input using SAS data step.

Example 6:

```
data bin2dec;
    bin0='0100000000000000000110011001100110011001100110001011111111111111';/* 2.025 */
    bin1='0100000000000000000110011001100110011001100110010000000000000011';/* 2.025 */
    bin2='0100000000000000000110011001100110011001100110010000000000000111';/* 2.025 */
    bin3='0100000000000000000110011001100110011001100110010000000000001111';/* 2.025 */
    bin4='0100000000000000000110011001100110011001100110011001100110010';/* 2.025 */
    bin5='0100000000000000000110011001100110011001100110011001100110011';/* 2.025 */

    dec0=input(bin0, binary64.);/*2.025*/
    dec1=input(bin1, binary64.);/*2.025*/
    dec2=input(bin2, binary64.);/*2.025*/
    dec3=input(bin3, binary64.);/*2.025*/
    dec4=input(bin4, binary64.);/*2.025*/
    dec5=input(bin5, binary64.);/*2.025*/

    r0=round(dec0,0.01);
    r1=round(dec1,0.01);
    r2=round(dec2,0.01);
    r3=round(dec3,0.01);
    r4=round(dec4,0.01);
    r5=round(dec5,0.01);

    put '-----------------------------------------------------';
    put dec0= / dec1= / dec2= /dec3= /dec4= / dec5=;
```

```
        put 'put function----------------------------------------';
        put dec0= 8.2 / dec1= 8.2 / dec2= 8.2 / dec3= 8.2 / dec4= 8.2 / dec5= 8.2;
        put 'round function----------------------------------------';
        put r0= / r1= / r2= / r3= / r4= / r5=;
            dif=dec5-dec4;
        put dif;
        put '----------------------------------------------------';
run;
```

```
----------------------------------------------------
dec0=2.025
dec1=2.025
dec2=2.025
dec3=2.025
dec4=2.025
dec5=2.025
put function----------------------------------------
dec0=2.02
dec1=2.02
dec2=2.02
dec3=2.02
dec4=2.02
dec5=2.03
round function----------------------------------------
r0=2.02
r1=2.02
r2=2.02
r3=2.02
r4=2.03
r5=2.03
4.440892E-16
----------------------------------------------------
NOTE: The data set WORK.BIN2DEC has 1 observations and 19 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds
```

We could find that the float-point number bin0, bin1, bin2, bin3 and bin4 is 2.02 with put function, but the float-point number 2.025 that is input is 2.03 with put function, which is normal result by rounding. But the difference can be seen from r4 and r5. dec4 is less than 2.025 which can be seen from the last one bit between bin4 and bin5, dec4 become 2.03 but not 2.02 with round function. So that, we could conclude that precision compensation is used in round function, which is different from put function. So it is suggest that we round number first before using put function.

CHARACTER ENCODING

Character encoding is used to represent a repertoire of characters by some kind of encoding system. In the early days of computer technology development, character sets such as ASCII (1963) and EBCDIC (1964) gradually became the standard. But the limitations of these character sets quickly come out, so many methods were developed to extend them. The requirement of support writing systems (including the East Asian CJK character family) is can support a larger number of characters and requires a system rather than a temporary method to implement the encoding of these characters.

There are some common encode method of SAS, such as ASCII (American Standard Code for Information

Interchange), GB2312, wlatin1 (windows 1252) and Unicode. ASCII is a set of computer coding systems based on the Latin alphabet, mainly used to display modern English and other Western European language. Latin-1 is the most used and defines the first 256 codes in Unicode. Unicode (International Code, Unicode, Single Code) is an industry standard in the field of computer science. It organizes and encodes most of the world's text systems so that computers can present and process text in a much simpler way. Unicode characters are encoded in hexadecimal digits. The first 128 characters of Unicode, which correspond one-to-one with ASCII.

The ASCII and latin1 is shown as Fig. 3.

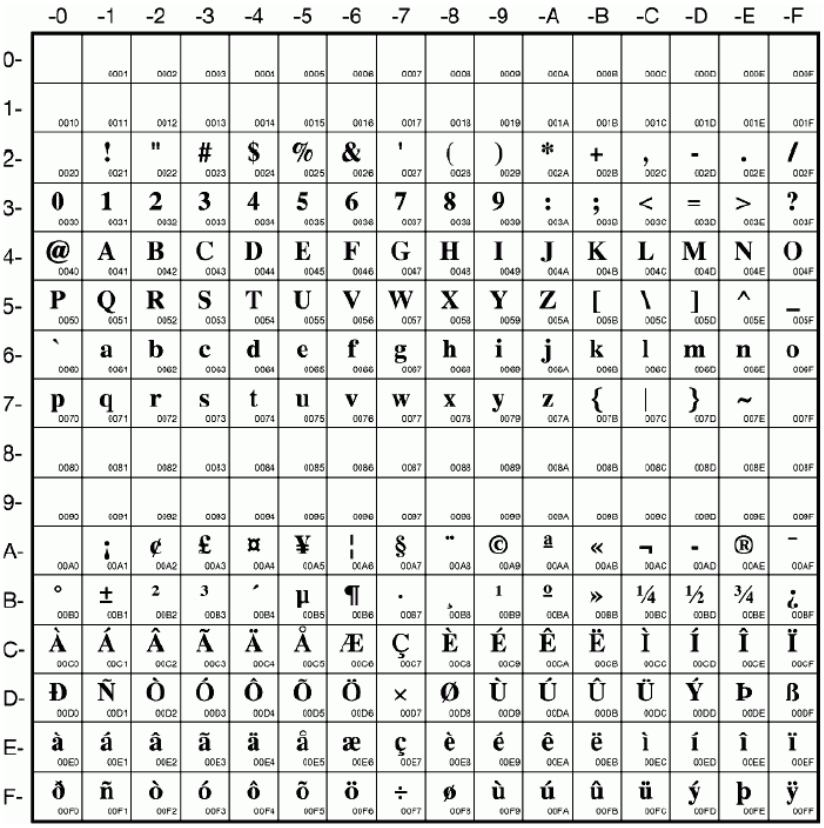


Fig. 3 The encode of ACSII and Latin1

Wlatin1 is different from latin1 in 80-9F with Unicode, which are the control character that could not be output, but they are visible character in wlatin1 that could be output with Unicode. Such as character € in 80 with Unicode, character *ƒ* in 83 with Unicode and etc.

UTF-8 is a variable width character encoding capable of encoding all 1,112,064 valid code points in Unicode using one to four 8-bit bytes. The first 128 characters of Unicode, which correspond one-to-one with ASCII, are encoded using a single octet with the same binary value as ASCII, so that valid ASCII text is valid UTF-8-encoded Unicode as well. UTF-8 is an implementation of encoding of Unicode character set, UTF-8 encode a character with 1-6 bytes which can encode any Unicode character (Table 3).

Table 3

Number of bytes	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
1	U+0000	U+007F	0xxxxxxx					

2	U+0080	U+07FF	110xxxxx	10xxxxxx				
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx			
4	U+10000	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
5	U+200000	U+3FFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
6	U+4000000	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Let's see some character encoded by UTF-8, in the following character, the blank and 'A' is encoded by 1 byte with UTF-8 or 8 bit in binary, the Latin character © and ë are encoded by 2 bytes with UTF-8 or 16 bit in binary. €, '中', '国' is encoded by 3 bytes with UTF-8 or 24 bit in binary. Little Chinese character such as '𠀀' is encoded by 4 bytes with UTF-8 or 32 bit in binary.

Example 7:

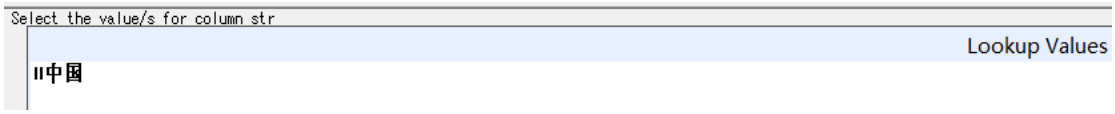
```
data test;
    length char $6;
    char=' ';output;
    char='A';output;
    char='©';output;
    char='ë';output;
    char='€';output;
    char='中';output;
    char='国';output;
    char='𠀀';output;
run;
data test;
    length char $6 utf8 $12 bin $48;
    set test;
    if _N_=1 then utf8=substr(put(char, $hex.),1,2);
    else if _N_>1 then utf8=tranwrd(put(char, $hex.),'20',");
    if char in (' ','A') then bin=put(char, binary8.);
    else if char in ('©','ë') then bin=put(char, binary16.);
    else if char in ('€', '中', '国') then bin=put(char,binary24.);
    else if char in ('𠀀') then bin=put(char,binary32.);
run;
```

	char	utf8	bin
1		20	00100000
2	A	41	01000001
3	©	C2A9	1100001010101001
4	ë	C3AB	1100001110101011
5	€	E282AC	111000101000001010101100
6	中	E4B8AD	111001001011100010101101
7	国	E59BBD	111001011001101110111101
8	𠀀	F0A08081	11110000101000001000000010000001

Let's see another example about character encoding issue, we could see there are two vertical line which is not '|'

before character '中国' using SAS look up. But we could not see any special symbol in dataset. The issue is that these symbols is some invisible control character with Unicode.

Example 8:




```
data a;
    set raw.data;
    put str=hex.;
run;
```

the log is display:
str = 0D0AE4B8ADE59BBD

We could find that the UTF-8 encode 0D0A before the string '中国', look up the ASCII and latin1 Table (Fig 3), 0D and 0A are control character that could not be output with UTF-8, In such cases, the compress function or prxchange function could be used. The methods could be as follow:

```
data b;
  set a;
  str=compress(str,'0D0A'x);
run;
or
data b;
  set a;
  str=prxchange('s/[\x0D\x0A]//', -1, str);
run;
```

Another example, if we want to compress all ASCII character or compress all non-ASCII character, we could use regular expression as follow:

Example 9:

```
data example1;
    var1="ßet ÿöû c@ñ'† get ®id of mÊ";
    var2=prxchange('s/[\x20-\x7F]//',   -1, var1);
    var3=prxchange('s/[^\x20-\x7F]//', -1, var1);
run;
```

	var1	var2	var3
1	ßet ÿöû c@ñ' † get ®id of mÊ	ßÿöûñ†®Ê	et c@' get id of m

This can be extended to character in every country, if we want to keep all Chinese characters, we could look up the Unicode table or UTF-8 table, find the first and last encode of Chinese character The first encode of Chinese character is E39080, and the last encode of Chinese character is EFA8A9. We could also use the regular expression to keep all Chinese character.

```
data example2;
    str1='abcdefghijklmnopqrstuvwxyz~!@#$%^&*,./()<>:"|\中国漢字丷丰噫堺寥兀鴎';
    str2=prxchange('s/[^\xE3\x90\x80-\xEF\xA8\xA9]//', -1, str1);
run;
```

	str1	str2
1	abcdefghijklmnopqrstuvwxyz~!@#$%^&*,./()<>:"|\中国漢字丷丰噫堺寥兀鴎	中国漢字丷丰噫堺寥兀鴎

CONCLUSION

This paper introducing the float-point data and encode in SAS through multiple examples to solve the numeric or character issues when manipulating clinical data in our daily work. It is important to understand the difference between what you see on your display monitor and what are stored on your machine. Understanding the problem and knowing your data, helps one keep the issue of numeric or character representation errors under control, and you will not be confusing with same numeric or character issues.

ACKNOWLEDGMENTS

I would like to acknowledge my previous line manager Yadong Miao and my line manager Fu Wang for their encouragement.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Name: Alan Zhou

Enterprise: PAREXEL International

Address: 9F & Unit A/B/C 10F, No.506, Shangcheng Road, Pudong District, Shanghai, China, 200120

City, State ZIP: Shanghai, 200120

Work Phone: +86 2120505341

E-mail: Alan.Zhou@parexel.com

Web: http://www.parexel.com/