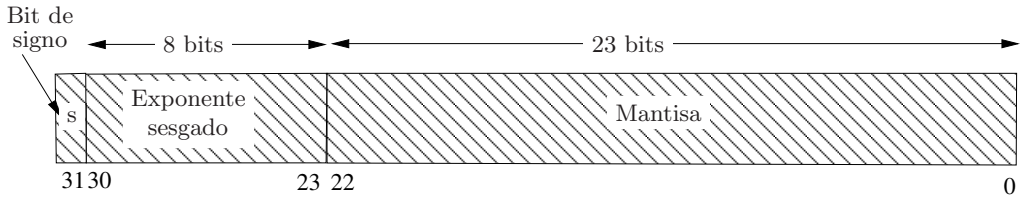
Determinando el formato de precisión simple de un número real.

Pablo Santamaría

v0.1 (Septiembre 2009)

A continuación presentamos un programa escrito en Fortran 90 que permite exponer el formato de precisión simple de un número real tal como es representado en la computadora. Recordemos que el formato de precisión simple utiliza 32 bits para representar el conjunto de números de punto flotante que tiene por base $\beta = 2$, $t = 24$ dígitos (binarios) y un exponente que varía entre los límites $L = -125$ y $U = 128$. Los $N = 32$ bits disponibles en la palabra, numerados de 0 a 31 de derecha a izquierda, son utilizados como sigue:



- El bit más significativo (el bit más a la izquierda de la palabra, el 31) es utilizado como bit de signo, esto es, su valor es 0 si el número es positivo o 1 si es el número es negativo.
- Los siguientes 8 bits (bits 30 a 23) son utilizados para almacenar la representación binaria del exponente que varía en el rango $L = -125 \leq e \leq U = 128$. La representación utilizada para el exponente se conoce como "sesgada", ya que se introduce un nuevo exponente sumando 126 al original: $E = e + 126$. El exponente sesgado varía entonces en el rango $1 \leq E \leq 254$ que puede ser representado por un binario entero de 8 bits. Más aún, podemos incluir los valores del exponente para $L - 1 = -126$ ($E = 0$) y $U + 1 = 129$ ($E = 255$), ya que todos los enteros en el rango $0 \leq E \leq 255$ pueden ser representados como un binario entero sin signo de 8 bits. En efecto, con 8 bits tenemos $2^8 = 256$ combinaciones distintas, una para cada uno de los 256 números enteros del intervalo $[0, 255]$. El límite inferior, el 0, corresponde a todos los bits igual a 0, mientras que el límite superior, el 255, corresponde a todos los dígitos igual a 1. Estos dos valores del exponente *no* representan números de punto flotante normalizados del sistema, sino que son utilizados para almacenar números especiales.
- Los restantes 23 bits (bits 22 a 0) de la palabra son utilizados para almacenar la representación binaria de la mantisa. En principio, esto parecería implicar que sólo pueden almacenarse $t = 23$ dígitos binarios de la mantisa y no 24. Sin embargo, para la base $\beta = 2$, el primer dígito de la mantisa en la representación normalizada siempre es igual a 1 y por lo tanto no necesita ser almacenado (tal bit se denomina *bit implícito*). Así pues, un campo de 23 bits permite albergar una mantisa efectiva de 24 bits.

Como ejemplo, consideremos la representación de formato simple del número cuya representación decimal es -118.625. Su representación binaria es

$$(-1)^1(1110110.101)_2,$$

con lo que, su representación binaria normalizada es

$$(-1)^1\, 0.1110110101 \times 2^7.$$

El bit de signo es 1, y el exponente sesgado es $E = 7 + 126 = 133$ cuya representación binaria de 8 bits es 10000101. El bit implícito de la mantisa no es almacenado, por lo que tenemos 9 dígitos binarios provenientes de la representación normalizada, el resto de los $23 - 9 = 14$ dígitos binarios son ceros. Así pues, la representación de formato simple del número propuesto es:

1	1	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Más interesante es la situación de un número real que no puede ser representado exactamente por un número de punto flotante de simple precisión, sino que tiene que ser aproximado por el más cercano (*redondondeo al más cercano*). Por ejemplo, consideremos el número cuya representación decimal es 0.1. Su representación binaria es, en este caso, infinita:

$$(-1)^0(0.0001100110011001100110011001100110011\cdots)_2,$$

con lo que su representación normalizada (infinita) es

$$(-1)^0\, 0.11001100110011001100110011001100110011\cdots \times 2^{-3}.$$

La representación de punto flotante de precisión simple se obtiene redondeando a 24 dígitos binarios:

$$(-1)^0\, 0.11001100110011001100\mathbf{1}101 \times 2^{-3}.$$

El bit de signo es 0 y el exponente sesgado es $E = e + 126 = 123$, cuya representación binaria de 8 bits es 01111011. Luego, la representación de simple precisión es:

0	0	1	1	1	1	0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1

Nuestro programa, cuyo código es expuesto a continuación, determina la representación de precisión simple asumiendo que los enteros son representados en 32 bits en complemento a dos (una hipótesis válido en todas las computadoras personales actuales). La ejecucción del programa para nuestros ejemplos conduce a

```
Enter a REAL number: -118.625

   1 10000101 11011010100000000000000
   . ........ .......................
   1 09876543 21098765432109876543210
     3          2         1
```

y

```
Enter a REAL number: 0.1

   0 01111011 10011001100110011001101
   . ........ .......................
   1 09876543 21098765432109876543210
     3          2         1
```

tal como esperábamos.

```
PROGRAM representacion_reales
! ------------------------------------------------------------------
  IMPLICIT NONE
  INTERFACE
    SUBROUTINE binrep(x)
        IMPLICIT NONE
        REAL :: x
    END SUBROUTINE binrep
  END INTERFACE
  REAL :: x
!  ------------------------------------------------------------------
  WRITE(*,'(A)',ADVANCE='NO') ' Enter a REAL number: '
  READ(*,*) x
  WRITE(*,*)
  CALL binrep(x)
! ------------------------------------------------------------------
END PROGRAM representacion_reales

SUBROUTINE binrep(x)
  ! ------------------------------------------------------------------
  IMPLICIT NONE
  REAL, INTENT(IN) :: x
  ! ------------------------------------------------------------------
  INTEGER :: i, int
  CHARACTER(32) :: b
  ! ------------------------------------------------------------------
  int = TRANSFER(x,int)
  IF (int >= 0) THEN
     b(1:1) = '0'
     DO I = 32, 2, -1
        IF (MOD(int,2) == 0) THEN
           b(i:i) = '0'
        ELSE
            b(i:i) = '1'
         ENDIF
         int = int / 2
      ENDDO
   ELSE
      b(1:1) = '1'
      int = ABS(int + 1)
      DO i = 32, 2, -1
         IF (MOD(int,2) == 0) THEN
            b(i:i) = '1'
         ELSE
            b(i:i) = '0'
         ENDIF
         int = int / 2
      ENDDO
   ENDIF
   ! ------------------------------------------------------------------
   WRITE(*,*) '   ', b(1:1),' ', b(2:9), ' ', b(10:32)
   WRITE(*,*) '   . ........ ....................... '
   WRITE(*,*) '   1 09876543 21098765432109876543210 '
   WRITE(*,*) '     3          2         1           '
   WRITE(*,*)
   ! ------------------------------------------------------------------
   RETURN
END SUBROUTINE binrep
```