

Sello II nota matemática #3

Funciones: la raíz cuadrada, arccos y arctan, logaritmo e interpolación

(c) 1998 , 2001 EME Systems, Berkeley CA U.S.A.

Tracy Allen

[<índice de sellos>](#) [<casa>](#)

Contenidos (actualizado 3/26/2004)

- [Funciones de sello indocumentado, ATN y HYP](#)
- [funciones trig. y las inversas, arcCos y arcTan](#)
- [Interpolación de la raíz cuadrada](#)
- [La función bitlog](#)
- [Logaritmo por interpolación en una tabla](#)
- [Logaritmo por cálculo](#)
- [Tablas e interpolación](#)
- [Lectura del termistor por interpolación](#)

funciones de sello no documentadas, ATN y HYP

[Arriba](#)

La función **ATN** devuelve el ángulo que tienen los componentes X e Y dados. La función **HYP** da la longitud de la hipotenusa del triángulo rectángulo formado por X e Y (la magnitud del vector). En conjunto, estos operadores le permiten convertir de coordenadas rectangulares a polares. Estos no fueron documentados en las primeras versiones del manual de sellos, pero recientemente se han hecho oficiales y se puede encontrar más información en la ayuda en línea dentro del Editor de sellos, versión 2.0 beta 1.

La entrada es de -127 a +127 para X e Y, y para la salida, los ángulos alrededor del círculo van de 0 a 256 brads (en lugar de 0-360 grados o radianes de 0 a 2pi). Puede convertir fácilmente brads a grados con una operación $*/360$ como se muestra a continuación. El sello mide los ángulos en sentido contrario a las agujas del reloj desde el eje X positivo. La magnitud de salida está en el rango de 0 a 179. Tenga en cuenta que la resolución es de solo 8 bits, si es que eso.

```
'{$STAMP BS2}
' Calculate inverse tangent (arctangent) and magnitude of a vector.
X          con 100      ' in the range -127 to +127
Y          con -100     ' in the range -127 to +127
angle      var word
magnitude  var word

angle = (X ATN Y) */ 360 ' order of X and Y matters. */360 converts from brads to degrees
debug ? angle           ' gives 315 degrees, 45 degrees less than a full circle.
magnitude = X HYP Y     ' the order of X and Y does not matter for HYP
debug ? magnitude       ' gives 141 (looks like the square root of 2, huh? SQR(100^2 +100^2))
end
```

Por cierto, las características no documentadas también pueden aparecer como "errores". Puede intentar usar "ATN" o "HYP" como nombres para variables en uno de sus propios programas y sorprenderse al recibir un mensaje de error, "nombre ya definido". Miras en la lista de "palabras reservadas", y esos nombres no están allí. Pero, de hecho, están reservados por el IDE.

funciones trig. y las inversas, arcCos y arcTan

[Arriba](#)

El BS2 tiene funciones SIN y COS, pero debe comprender cómo funcionan para que funcionen para usted. La nota de Tnis muestra algunos ejemplos, incluida una forma de calcular las funciones INVERSAs SIN y COS.

```
X = COS A
      ' in normal trig: 0<=A<360 degrees, -1<=X<=+1
      ' or
      '      0<=A<=2*pi radians
      ' but...
      ' in stamp trig: 0<=A<255 brads, -127<=X<=+127
```

Tenga en cuenta la escala que utiliza el sello. Un círculo completo es de 0-256 brads, en lugar de los familiares 0-360 grados o radianes de 0 a $2 * \pi$. El término "brad" significa "radian binario". Y el coseno del ángulo va de -127 a +127, para representar los valores de coseno de -1 a +1. Se puede pensar en el rango de X como una fracción que va desde -127/128 hasta 127/128, siempre con un denominador de 128. Aunque X cubre solo un pequeño rango en matemáticas de sellos, siempre debe definirse como una variable de palabra, para representar correctamente los números negativos. Podría preguntarse por qué el denominador es 128 en lugar de 127. La respuesta es que no hace mucha diferencia, dada la precisión de la función en el sello.

Esta no es una función de alta precisión. Cuando el ángulo va de 0 a 64 (un cuarto de círculo), el coseno pasa de 127 a cero. Por lo tanto, la resolución es solo una parte en 64. En ángulos pequeños, cerca de $A = 0$, la función del coseno cambia muy lentamente. En el sello BASIC, los ángulos $A=0$, $A=1$, $A=2$ y $A=3$ producen el mismo resultado, $X=127$.

conversión de brads a grados y viceversa, y normalización.

Para convertir de brads a grados, multiplique por 45 y luego divida por 32 ($45/32 = 360/256$ simplificado). O simplemente use el operador `*/` con 360 en el lado derecho. (El operador `*/` hace una división implícita por 256.) Ejemplo, para convertir 256 brads a grados:

```
degrees = 256 * 45 / 32      ' gives 360 degrees
```

O...

```
degrees = 256 */ 360 = 360    ' also gives 360 degrees
```

Los dos tienen la misma precisión, pero el formulario `*/` se ejecuta más rápido y requiere menos memoria. Para convertir de la otra manera, de grados a brads, multiplique grados por 32 y divida por 45, o use `*/` o `**` con una aproximación de la fracción $256/360$. Detalles sobre `*/` y `**` [en otros lugares](#). Por ejemplo, para convertir 157 brads a grados:

```
brads = 157 * 32/45          ' gives 111 brads
```

O...

```
brads = 157 ** 46603         ' gives 111 brads (46603/65536~32/45)
```

O...

```
brads = 157 */ 182           ' gives 111 brads (182/256~32/45)
```

Y a veces desea que el valor de X abarque un rango decimal más conveniente, digamos de -100 a 100 en lugar de -127 a 127. Utilice las conversiones después de las conversiones. X1 es un valor entre -100 y +100, que representa la fracción $-100/100$ hasta $+100/100$. X es un valor entre -127 y +127 que representa la fracción $-127/128$ hasta $+127/128$. El proceso de cambiar un denominador como ese se llama **renormalización**.

```
X1 = abs X * 25 / 32 * (-2*X.bit15+1)      ' renormalization X/128 to X1/100
```

O...

```
X1 = abs X */ 200 * (-2*X.bit15+1)          ' 25/32=200/256 using */ operator
```

y para ir por el otro lado, desde ...

```
X = abs X1 * 2 ** 41943 * (-2*X.bit15+1)    ' renormalization X1/100 to X/128
' 32/25~2*41939/65536
```

O...

```
X = abs X1 */ 983 / 3 * (-2*X.bit15+1)      ' 32/25 ~ (983/256)/3
```

El truco aquí es que los valores de X y X1 pueden ser negativos; el sello BASIC no puede realizar la división en números negativos. El procedimiento consiste en tomar el valor absoluto de X o X1, luego multiplicar y dividir, y luego restaurar el signo basado en el bit de signo del número original.

Aquí hay un programa de demostración simple que ilustra el uso del SIN y COS en el sello BASIC. Pasa por todos los ángulos de 0 a 360. El ángulo se imprime en grados y el coseno y el seno del ángulo se imprimen como números decimales en el rango de -1.00 a +1.00. Luego, para una buena medida, calcula la tangente del ángulo e imprime eso también, como un número de 0.00 hasta 65535 (infinito). Recordemos que la tangente es igual al seno dividido por el coseno.

```
for A1=0 to 360
  A=A1 */ 182      ' <--convert degrees to brads
  X=COS A
  Y=SIN A
  X1 = abs X */ 200 * (-2*X.bit15+1)  ' <-- renormalize.
  Y1 = abs Y */ 200 * (-2*Y.bit15+1)
  ' now have the values in +/- decimal form to display in -0.xx form
  debug "COS ",DEC A1,"= ",X1.bit15*13+32,"0.",DEC2 X1," "
  debug "SIN ",DEC A1,"= ",Y1.bit15*13+32,"0.",DEC2 Y1," "
  Z=abs Y * 100 / abs X * (X.bit15^Y.bit15*-2+1) ' calculate the tangent.
  if X>0 then okay
    Z=65535
okay:
  debug "TAN ",DEC A1,"= ",Z.bit15*13+32,DEC abs Z/100,".",DEC2 Z,cr
  pause 100      ' slow it down so I can see it
next
```

El cálculo de la tangente implica tomar el valor absoluto de ambos números y luego restaurar el signo al final. El signo del resultado es positivo si tanto X como Y son positivos o si ambos son negativos. Observe cómo se divide el valor decimal para mostrarlo con el punto decimal. Se necesitan algunos trucos adicionales para mostrar el signo y el punto decimal correctamente, como se detalla [en otra parte](#).

Funciones trig inversas \cos^{-1} y \sin^{-1} .

A veces se necesitan los trig funcionones inversos, que surgen mucho en el trabajo por computadora con control de movimiento y análisis de datos y en astronomía y cálculos de tiempo. El COS inverso a veces se escribe ARCCOS, o \cos^{-1} . ARCCOS X es el ángulo A que daría el valor $X=\cos A$.

La función del coseno es periódica, por lo que muchos valores de A darán $X = \cos A$. Entonces, por conveniencia, el valor de A está restringido a $0 < A < 180$ grados (radianes π , 128 brads).

```
A = COS-1 X      ' in normal trig, -1<X<=1, 0<=A<=180 degrees or 0<=A<=pi
                  ' but...
                  ' in stamp trig -127<=X<=127, 0<=A<=128 brads
```

Del mismo modo, $\sin^{-1} Y$ es el ángulo A que daría $Y = \sin A$. El ángulo A está restringido a los valores de -90 a $+90$ grados ($\pm\pi/2$ radianes, 64 a 192 brad). Al sello BASIC no le gustan los ángulos negativos, por lo que usará de 64 a 192 brads.

Por ejemplo, al promediar la dirección del viento vectorial, los componentes seno y coseno del vector viento se separan y se acumulan. Luego, al final, los totales para las direcciones x e y se dividen por el número de muestras. El valor $(X^2 + Y^2)^{1/2}$ es la magnitud vectorial promedio de la velocidad del viento, y $\arctan Y/X$ es la dirección vectorial promedio.

El truco rápido para calcular \cos^{-1} es usar la función \cos incorporada del sello y el bucle para encontrar el ángulo que se acercará más al valor dado de X . El siguiente cálculo restringe aún más el dominio, a $0 \leq X \leq 1$ ($0 \leq X \leq 127$ para la escala de sellos), y el rango a $0 \leq Y \leq 90$ grados ($0 \leq Y \leq 64$ brads para la escala de sellos). Estos son ángulos en el primer cuadrante. Las identidades trigonométricas se pueden utilizar para calcular la respuesta para el segundo cuadrante.

```
X var byte      ' input value 0 to 99 (0.00 to 0.99)
A var byte      ' angle in brads, 0<=A<=64.
Z var byte      ' helper variable

for X = 0 to 99  ' demo, just step through the values of X
  ' calculate the arctan here:
  Z = X * 983 / 3 ' renormalize to 127
  A=63-(Z/2)      ' first approx, to minimize iterations
loop:
  if COS A <= Z then done
  A = A+1
  goto loop
done:
A = A * 360      ' convert brads to degrees using */ operator
' now have the angle in decimal degrees, 0->90
debug "arccos 0.", dec2 X, " = ", dec A
pause 100       ' slow it down so I can see it
next            ' do some more
```

El núcleo de la rutina para calcular $\arctan X$ se muestra en rojo. Incluye un paso para renormalizar la fracción decimal en una fracción /127, y luego al final un paso para renormalizar de brads a grados. Esto no es cegadoramente rápido ni preciso ni NIST: el ángulo inicial = $63 - (X/2)$ es una aproximación inicial para mantener el número de iteraciones en un mínimo (13 iteraciones en el peor de los casos).

Lo anterior cubre $X = 100$ a $X = 0$, para dar de cero a 90 grados. ¿Cómo encontrar la otra mitad, de $X=0$ a $X=-100$, para dar de 90 a 180 grados? Utilice la identidad:

$$\arccos(-X) = 180 - \arccos(X)$$

Entonces, si tuviera que encontrar $\cos^{-1}(-37)$, usaría el algoritmo anterior para encontrar $A = \cos^{-1}(+37)$, y luego calcular $\cos^{-1}(-37) = 180 - A$ grados.

Para encontrar $\sin^{-1} Y$, es posible desarrollar una rutina muy similar a la anterior, y utilizar la relación

$$\arcsin(-X) = -\arcsin(X)$$

o, si necesita encontrar el $\sec^{-1} Y$, entonces busque $\cos^{-1} Y$ en su lugar, y use la identidad

$$\sin^{-1} Y = 90 - \cos^{-1} Y$$

Esto es simplemente una reafirmación del hecho bien conocido de que los dos ángulos opuestos en un triángulo rectángulo suman 90 grados.

El BS2 no ofrece una función TAN. Pero recuerde de la escuela secundaria trig que si bronce $A = a/b$, entonces $\cos A = b/(a^2 + b^2)^{1/2}$. Si necesita encontrar $A = \tan^{-1} Z$, primero puede calcular cuál sería el coseno de ese ángulo y luego calcular el ángulo en sí. Por ejemplo, para encontrar $\tan^{-1} 3$, puede encontrar $\cos^{-1} \{1/(3^2+1)^{1/2}\}$...

$$\tan^{-1} 3 = \cos^{-1} \{1/(3^2+1)^{1/2}\} = \cos^{-1} (1/10^{1/2})$$

En estampesa con ángulos en brads y normalización a 128, esto se convierte:

$$\cos^{-1} 40/128$$

que da un ángulo cercano al valor correcto, 71 grados, del algoritmo anterior.

Interpolación de la raíz cuadrada

[Arriba](#)

La función $Y = \text{SQR } X$ del BS2 toma un argumento X de 16 bits y devuelve un resultado Y de 8 bits. Esto da como resultado una baja resolución y baja precisión. Por ejemplo, el resultado es $Y = 10$ para todo $X = 100$ a $X = 120$, y luego es 11 para todos los valores de X de 121 a 143. Supongamos que quieres encontrar la raíz cuadrada de 115. El sello le dirá que la raíz cuadrada es 10. Pero si 115 es 3/4 del camino entre 110 y 120. Así que la raíz cuadrada es aproximadamente 3/4 del camino entre 10 y 11, es decir, alrededor de 10.75. Esto es interpolación

lineal. Por supuesto, el sello no puede usar el punto decimal, pero puede darle una respuesta de 1075, donde se entiende que el punto decimal está dos lugares por encima. La raíz cuadrada de 65536 es 256, pero podemos mover con seguridad el decimal dos para convertirnos en 25600, que representa 256.00. La fórmula de interpolación solo toma una línea de código BS2:

```
' To calculate square root on BS2
' using interpolation for greater resolution
' best for larger values of X
X var word
Y0 var byte
Y var word
for x=0 to 65535
  Y0 = sqr X
  Y = X-(Y0*Y0)*100+Y0/(2*Y0+1)+(Y0*100)
  debug dec X,tab,dec Y0,tab,dec Y/100,".",dec2 Y,CR
  pause 50
next
```

El programa de demostración recorre todos los valores posibles de X y muestra tanto el valor simple Y0 que proviene de la función SQR incorporada del BS2 como el valor interpolado con dos decimales. La interpolación es más precisa a valores más altos de X, donde la función es "más plana", es decir, naturalmente más de una línea recta.

Aquí hay otra forma de extender la precisión de la raíz cuadrada, por continuación. Esto utiliza un algoritmo llamado "algoritmo de la escuela secundaria". El método es similar a la división larga. Cuando el BS2 toma la raíz cuadrada de un número dado, encuentra el número entero más grande que, cuando está al cuadrado, sigue siendo menor que el número dado.

```
y = sqr x      ' this is what the BS2 calculates
'y*y < x      ' which squared is less than x
'R = x - (y * y)  This is the "remainder"      '(y+1)*(y+1)>x      but y+1 squared is greater than x
```

Lo siguiente: comienza con este resto, así como el propio resultado SQR del sello, para extender el resultado a 14 bits.

```
sqr64:
  y=sqr x      ' BASIC Stamp does its thing to 8 bit result
  R=x-(y*y)    ' remainder to start
  y=y*2
  for ix=5 to 0 ' continuation, using the "high school" algorithm.
    R=(R<<2)
    y=y&~1<<1|1 ' how do you like this for a trick formula?!
    if y>R then sqr64b
      R=R-y
      y=y|2
  sqr64b
  next
  y=y*/200      ' calculate to two extra decimals
  debug dec y/100,".",dec2 y,cr
  return
```

Referencia para el algoritmo High School:

Jack Crenshaw', [Math Toolkit for Real-Time Programming](#), (c) 2000, [CMP Books](#), ISBN: 1-929629-09-5

El logaritmo a través de NCD y la función bitlog

[Arriba](#)

El logaritmo es importante para los cálculos que involucran decibelios, niveles de potencia y cifras de ruido. También es importante para algunos tipos de procesamiento de datos donde la señal varía en un rango muy amplio de niveles. Por ejemplo, la intensidad de la luz y el nivel de sonido pueden variar en muchos órdenes de magnitud. Nuestro aparato sensorial, nuestros ojos y oídos, responden intrínsecamente al logaritmo de la señal. Algunos tipos de sensores, como los termistores, tienen una respuesta logarítmica, por lo que la mejor ecuación para derivar la temperatura de la resistencia requiere la función logarítmica. Muchos problemas que involucran ecuaciones diferenciales de primer orden, como la desintegración radiactiva, involucran registros y exponenciales.

¿Qué hacer en el BS2? No tiene ni un log ni una función exp, ¿o sí?

El BS2 tiene una especie de registro incorporado, la función NCD. El manual basic stamp explica NCD x en términos del conjunto de bits más alto en la variable x. Por ejemplo, el número 4097 en binario es, %1000000000001, y el decimotercer bit es el conjunto más alto, por lo que $NCD\ 4097 = 13$.

Además, $2^{13}=8192$ es la potencia exacta de dos que es mayor que ($x=4097$), y $2^{12}=4096$ es la potencia de dos que es menor que x. La función ($NCD\ x - 1$) es la parte entera, o *característica*, del logaritmo base 2 de X.

Supongamos que necesita hacer un medidor de VU para el nivel de sonido que se mostrará como un gráfico de barras en una pantalla de 8 segmentos, y que el nivel de sonido de un convertidor analógico a digital varía de 0 a 256. Entonces el valor para poner en la pantalla es simplemente

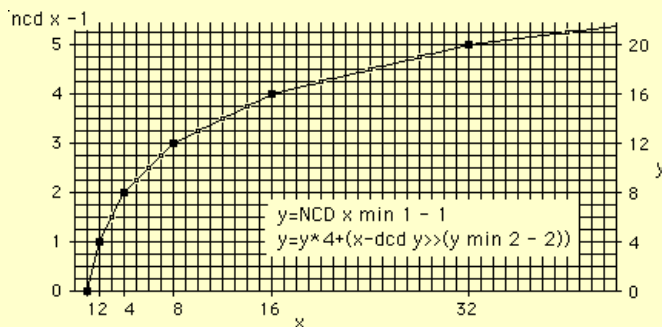
```
y = NCD x      ' value that ranges from 0 to 8
                ' as x ranges from 0 to 255.
```

El primer segmento viene para un nivel de sonido de 1, el segundo segmento para niveles de sonido de 2->3, y así sucesivamente hasta el segmento de eighth para niveles de sonido de 128 ->255

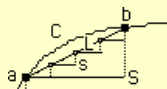
El inverso del operador NCD es el operador DCD. En efecto, eleva 2 a un cierto poder. Por ejemplo, $y = DCD\ 7$ devuelve 128, porque 128 es la séptima potencia de dos. Esta es una especie de función exponencial cruda. El sello acepta valores de $x=0$ a $x=15$, y devuelve los 16 valores correspondientes $y=1$, $y=2$, $y=4$, $y=8$ y así sucesivamente hasta $y=32768$. Bastante crudo, cuando se ve como un exponencial.

¿Qué pasa si en lugar de solo 8 segmentos, desea hacer un medidor de VU para mostrar el nivel de sonido en un gráfico de barras de 100 segmentos en una **pantalla LCD**? La función bitlog es una forma sencilla de expandir la escala.

El siguiente gráfico muestra los valores de $x=\{1,2,4,8,16,32\}$, con los valores correspondientes de $NCD\ x - 1 = \{0,1,2,3,4,5\}$ en el eje y de la izquierda. El gráfico muestra líneas rectas que conectan las potencias binarias exactas. El eje y de la mano derecha muestra valores que son cuatro veces los valores del eje y de la izquierda. Hay pequeños puntos en las líneas rectas que conectan los puntos negros principales. Esos van a ser nuestros puntos de interpolación, dado por la función bitlog.



De acuerdo, es una forma de interpolación lineal. El logaritmo real conecta los puntos negros principales (que se llaman la característica, o parte entera del logaritmo) con una curva (llamada mantissa del logaritmo). La curva "C" se muestra en el recuadro a la derecha, y la interpolación lineal son los puntos a lo largo de la línea recta que conecta **a** y **b**. No es nada perfecto, pero la aproximación lineal es una mejora para cosas como un medidor de VU. La pequeña figura de la derecha muestra dos de las potencias binarias, **a** y **b** conectadas por la curva **C**. Si no usamos la interpolación, la curva resultante es solo un gran paso, **S**.



Digamos que queremos interpolar 4 puntos. La fórmula para calcular los valores se muestra como en el recuadro en el gráfico anterior. La primera línea de la fórmula calcula la característica NCD-1, limitando el valor a >0 . La segunda línea multiplica ese resultado preliminar por 4, y luego agrega la interpolación: un valor de 0,1,2 o 3 tomado de los dos bits siguientes desde el bit más significativo en el valor original. De ahí el nombre, "bitlog". La fórmula es solo una forma complicada de extraer el valor de esos dos bits. Todo lo que está haciendo es esa interpolación. El negocio con el **min** está ahí simplemente para manejar valores de x menores que cuatro, para los cuales no podemos tener cuatro puntos interpolados, y para manejar el valor $x=0$.

Aquí está la fórmula para 8 (2^3) puntos interpolados:

```
y=NCD x min 1 - 1
y=y*8+(x- dcd y >>(y min 3 - 3))
```

Si el valor de entrada de x puede variar de 1 a 4095, entonces el valor del bitlog definido por esta fórmula varía de 0 a 96, justo para mostrar en un gráfico de barras LCD de 100 segmentos. Aquí está el cálculo cuando se sustituye $x = 4096$...

```

y=NCD 4096      ' results in y=12, then...
y=12 min 1 - 1  ' results in y=11, then...
y=y*8 + (4096-dcd 11 >> (11 min 3 - 3))
    ^^^          ^^^^^^^^^^^^^^^^^^
          2048
    ^^^^^^^^^^^
          2048    >>      8
    ^^^^^^^^^^^^^^^^^^^
88      +      8
= 96

```

Aquí está la fórmula general, donde debe haber 2^n puntos interpolados:

```
y=ncd x min 1 -1 : y=y*dcd n+((x-dcd y)>>(y min n-n))
```

Por ejemplo, con $n=8$, habría $256=2^8$ puntos interpolados entre cada potencia exacta de dos.

```
y=NCD x min 1 - 1
y= y*256+((x-dcd y)>>(y min 8 - 8))
```

Entrando con $x=10000$, la fórmula viene con $y=3384$. Divida eso por 256 en una calculadora para obtener 13.218. Compare eso con el valor real, $\lg 10000 = 13.276$. (Lg significa logaritmo base 2) La función bitlog es una interpolación lineal cruda, pero da una mejor aproximación que solo 13.000.

Para convertir a base 10, si es necesario, multiplique el valor lg por un factor constante, utilizando el operador **. (factor = $100/256 * 0.30103 * 65536 = 7706$).

$$y_{10} = y^{**} 7706$$

Esto devuelve un valor de $y_{10}=397$ cuando $x=10000$. Compare eso con el valor verdadero, $100 * \log 10000 = 400$.

El inverso del operador lg es la potencia de 2, ya que en el sello el inverso de NCD es DCD. Aquí está el inverso de la función bitlog. El valor n en la fórmula es, como se mencionó anteriormente, el número de pasos de interpolación:

```
x=dcd (y/dcd n)
x=(x>>n)*(y-x)+x
```

Hay una buena discusión de la función bitlog y su inversa en

Jack Crenshaw'
Math Toolkit for Real-Time Programming,
 (c) 2000, *CMP Books*, ISBN: 1-929629-09-5

Logaritmo por interpolación en una tabla

[Arriba](#)

La siguiente tarea es encontrar una mejor aproximación a la curva entre las potencias exactas de dos. El operador (NCD X - 1) devuelve la parte entera del logaritmo base 2 de X. Esta parte entera generalmente se llama la "característica". La parte fraccionaria se llama "mantissa". Por ejemplo $2^{5.75} = 57.8368$, y la otra forma de escribir esto es, $\log_2 57.8368 = 5.75$. La característica es 5, la mantissa es 0.75. En el sello con matemáticas enteras, tenemos que mover el punto decimal para llegar a un valor de $575 = 100 * \log_2 58$.

El siguiente programa utiliza la función NCD integrada en el BS2 para encontrar la característica y, a continuación, busca la mantissa en una tabla con interpolación. La tabla aquí consta de cinco constantes de tamaño de palabra, pero el programa facilita el cambio del número de entradas de tabla donde hay más memoria disponible, como en el BS2SX o en eeprom externo. Los valores que se van a utilizar en una tabla de 66 bytes se enumeran más adelante.

Es importante entender que no necesitamos una tabla separada para cada octava entre potencias de dos. la misma tabla funciona para cada octava. Es importante entender que el logaritmo tiene una propiedad de periodicidad. La forma de la mantissa entre 1 y 2 es en todos los aspectos congruente con la forma que tiene entre 2 y 4, o entre 4 y 8, o entre dos números cualesquiera que sean múltiplos de dos separados. La diferencia es sólo una cuestión de escala. De esta manera las funciones exponenciales se relacionan con las funciones trigonométricas, una de las nociones profundas de las matemáticas.

```
' LOG2T.BS2
' Tracy Allen
' Example of how to determine log N
' Integer part (characteristic) comes from BS2 function, NCD.
' Fractional part (mantissa) from interpolation in a table in eeprom
' Could use external memory for the table if available.
' Conversion to other bases using constant basex:
' 65536*Log(2)= 65536*0.30103=19728. (19728.3)
' 65536*Ln(2)=65536*0.69315=45426 (45426.2784)
Ld con 2 'number of table entries is (2^Ld+1)
basex con 65535 ' 65535 for base 2, 19728 for base 10, 45426 for base e
Ldata data word 0, word 322, word 585, word 807, word 1000
' x 0.000 1.250 1.500 1.750 2.000
' log2(x) 0.000 .3219 .5850 .8074 1.000
N var word ' input to log2 routine
M var word ' output and intermediate variable
Q var word ' lower estimate, (and log base e)
Q0 var Q.byte0 ' for table lookup, implied array
P var word ' upper estimate, (and log base 10)
L var nib ' integer part of log2(N)
K var nib ' adjusted for interpolation
J var nib ' fractional category of log2(N)
I var nib ' index into table

demo: ' just pick a number N and find its log
N=48756
debug "N=",dec N,CR ' show the number
gosub log2 ' find lg2 base 2 and display:
debug "lg N=",dec M/1000,".",dec3 M,CR
P = M ** 19728 ' convert to log base 10 & display:
debug "log N=",dec P/1000,".",dec3 P,CR
Q = M ** 45426 ' convert to ln base e and display:
debug "ln N=",dec Q/1000,".",dec3 Q,CR
end

log2: ' input N, returns M = log2(N)
L = NCD N - 1 ' L=integer part of log2(N)
M = DCD L ' M < N < 2M between powers of two
K = L min Ld - Ld ' restrict value for interpolation
J = N - M >> K ' fractional category, 2^Ld categories
for I=0 to 3 ' read values from table
read J*2+I,Q0(I) ' Q < frac. part of log2(N) < P
next
' now interpolate
' note 32 bit math, divide by M is >> K
M = N-M-(M>>Ld*J)**(P-Q)<<(16-K)+(N-M-(M>>Ld*J)*(P-Q)>>K)+Q+(L*1000)
return
```

Valores para Ld=5, 32 categorías, tabla de 66 bytes para mantissa: 0,444, 875, 1293, 1699, 2095, 2479, 2854, 3219, 3576, 3923, 4262, 4594, 4919, 5236, 5546, 5850, 6147, 6439, 6724, 7004, 7279, 7549, 7814, 8074, 8329, 8580, 8826, 9069, 9307, 9542, 9773, 10000. Las entradas de la tabla Ld=2 que se implementa en el programa de demostración anterior están subrayadas. La tabla se puede ampliar aún más si tiene suficiente memoria disponible, por ejemplo, en un BS2SX o BS2e.

Tenga en cuenta que las matemáticas se pueden simplificar si $Ld > 6$, una tabla de 258 bytes.

$$M = N - (M > Ld * J) * (P - Q) > K + Q + (L * 1000) \quad \text{if } Ld > 6$$

Esto se debe a consideraciones de desbordamiento de enteros. Con una tabla más grande, las diferencias entre los valores vecinos en la tabla se vuelven más pequeñas, y en $Ld > 6$ se vuelven lo suficientemente pequeñas como para que las matemáticas de doble precisión ya no sean necesarias. Por supuesto, cuantos más elementos haya en la tabla, más preciso será el resultado.

¿Qué pasa con la base 10 o la base e? Una vez que tenga la base 2, los demás se pueden calcular simplemente multiplicando el valor de la base 2 por una constante, 0.30103 para la base 10 o 0.69315 para la base e. Esto se logra en el programa anterior mediante el uso del operador ** en su papel de multiplicador fraccionario.

Logaritmo por cálculo

[Arriba](#)

La búsqueda de la tabla anterior es rápida. Sin embargo, la tabla ocupa una gran cantidad de eeprom para la mejor precisión. La siguiente es una implementación del ejercicio 24 en el capítulo 1.2.2 de Donald Knuth, "[Art of Computer Programming](#)". Calcula el logaritmo por aproximación sucesiva. El código de sello BASIC para el cálculo real tiene solo 9 o 10 líneas de largo. ¡Se tarda más en explicarlo que en hacerlo!

El primer programa que se da a continuación muestra cómo calcular la manitssa. El algoritmo asume que el número a convertir se encuentra entre 1 y 2, por lo que el logaritmo (base 2) se situará entre 0 y 1. Dado que el sello BASIC no puede hacer partes fraccionarias de 1 a 2 directamente, tenemos que usar un truco. Usaremos enteros de 32768 a 65535 para representar las fracciones de $1 = 32768/32768$ a $2 = 65536/32768$. Es decir, 32768 partes fraccionarias. Los logaritmos de los números del 1 al 2 van desde $\lg 1 = 0$ hasta $\lg 2 = 1$. En nuestro sistema con 32768 como "unidad", tenemos $0 = 0/32768$ a $1 = 32768/32768$. El siguiente algoritmo funciona examinando el cuadrado del número en cada paso, y siempre ajustándolo por una división por 2 para que permanezca en el intervalo [1,2]. El núcleo del código basic stamp está resaltado en color rojo.

```
' LOGRTHM.BS2
' logarithm base 2 by calculation
' y = lg n
' for 1<=n<2    0<=y<1
' represented on the Stamp as fractions
' 32768/32768<=n<=65535/32768
' 0/32768<=y<=32767/32768
' Tracy Allen
' implements Knuth, ArtComProg ch. 1.2.2, exercise 24.
'
x      var word      ' input, a number between 1(32768) and 2(65536)
xf     var x.bit15   ' the most significant bit of x
x2     var word      ' auxiliary variable, high word of x^2
x2f    var x2.bit15  ' msb of x2
lgx    var word      ' result, log base 2 of x
lgx0   var lgx.bit0  ' lowest bit of lgx, for bit addressing
bitk   var bit       ' auxiliary bit for calculation
k      var nib       ' index for steps of approximation

x=39457      ' example x=39457/32768 = 1.204132
debug dec x**20000,cr ' prints 12041 (decimal conversion)
  lgx=0      ' initialize logarithm
for k=14 to 0 ' 15 bit result
  x2=x*x     ' square of x, high word
  x=x*x      ' low word
  lgx0(k)=x2f ' this bit is 1 if x^2>=2
  bitk=~x2f   ' complement it for calculation
  x=x2<<(bitk+(bitk&xf)) ' adjusted value of x
next         ' next bit
  debug dec lgx,32,cr, lgx**20000
  ' print the result
  ' as fraction e.g. 8781/32768
  ' as decimal value e.g.
  ' log2 x = 2680/10000=0.2680

debug dec lgx**60206,cr,dec lgx**13863
  ' log10 x = 8067/100000 =0.08067
  ' ln x = 1857/10000=0.1857
```

Notas:

- El algoritmo tiene una precisión de entre 3 o 4 decimales.
- El paso de ajuste puede necesitar una pequeña explicación. El cuadrado de una cantidad de 16 bits es una cantidad de 32 bits y en el sello BASIC que sale como una palabra superior ($x2 = x * x$) y una palabra inferior ($x = x * x$). El resultado de 32 bits es un número entre $2^{30} <= x^2 < 2^{32}$, porque el valor inicial antes de cuadrar estaba entre 2^{15} y 2^{16} , representando números del 1 al 2 cuando se normaliza con un denominador de 2^{15} . Pero el valor cuadrado bruto también debe normalizarse, al deslizar 15 bits a la derecha. El valor al cuadrado se convierte en un número entre 2^{15} y 2^{17} , representando números del 1 al 4. El algoritmo realiza la normalización no desplazando 15 bits a la derecha, sino desplazando un bit a la izquierda y dejando caer 16 bits de la derecha. Pero la división condicional por dos se enrolla en eso, por lo que se desplaza a la izquierda o no dependiendo del valor del bit "2".
 - si $x^2 < 2$, el siguiente paso del algoritmo operará sobre el valor actual de x^2 sin dividir por dos. La normalización se realiza desplazando un bit hacia la izquierda. $x2 << bitk$ lo desplaza un bit a la izquierda, y el $+(bitk \& xf)$ mueve el bit más significativo de la palabra inferior al bit menos significativo de la palabra superior. Los 16 bits inferiores se truncan.

- o si $\text{bitk}=0$, esto significa que $x^2 \geq 2$. El siguiente paso del algoritmo debe tener el valor actual de x^2 dividido por 2. La división se combina con la normalización simplemente tomando los 16 bits superiores del resultado intermedio, sin normalizar. Cuando $\text{bitk}=0$, no hay desplazamiento, y no se agrega ningún bit desde la palabra inferior. Los 16 bits inferiores se truncan.
- Truncar los 16 bits inferiores en cada paso provoca errores que se propagan a pasos posteriores del algoritmo. Esto limita la precisión final de este algoritmo. Consulte la referencia citada para obtener más detalles. Otro algoritmo potencialmente más preciso (misma referencia, problema 25) también se puede implementar en el sello. Implica una mesa auxiliar y un enfoque unilateral intencional al límite.
- Los pasos de conversión de base 2 a base 10 y logaritmos de base natural e combinan los factores de conversión, $\log 2=0.30103$ y $\ln 2 = .69315$ con la multiplicación por 20000. Tenga en cuenta que eso da 5 decimales para el registro base 10, aunque el último dígito no es significativo.

La situación comúnmente encontrada con el sello BASIC es que se necesita un logaritmo para un entero, N, que se encuentra entre 1 y 65535. Estos números oscilan entre 16 potencias de 2. Simplemente estamos interesados en $\lg N$. (\lg es la abreviatura de log base 2, y podemos convertir a cualquier otra base por medio de una simple multiplicación **). La característica (base 2) se encuentra utilizando el operador NCD. Luego se aplica el algoritmo anterior para resolver la mantissa. Un ejemplo puede ayudar a entender lo que está pasando. Digamos que necesita el logaritmo de 3456, tomado como una lectura en bruto de un convertidor analógico a digital. La característica es 11, es decir, $2^{11}=2048$ es la potencia más alta de 2 que es menor que 3456. De hecho, desde los fundamentos,

$$3456 = 2^{11+x} = 2048 * 2^x$$

o escribir esto de una manera diferente,

```
lg 3456 = 11+x
characteristic=11
mantissa=x    0<=x<1, value to be determined
```

Para encontrar la mantissa, reescriba la ecuación como,

$$2^x = 3456/2048, \text{ or } x = \lg 3456/2048$$

Ahora 3456/2048 es nuestro número entre 1 y 2. El numerador y el denominador se pueden multiplicar por 16, lo que da la fracción, 55296/32768. Esta es la condición del algoritmo justo arriba, donde 32768 representa la unidad. El siguiente algoritmo primero calcula la característica utilizando el operador NCD, y luego normaliza el denominador a 32768, y luego aplica el algoritmo iterativo anterior para calcular la mantissa. El núcleo del código basic stamp está resaltado en color rojo.

```
' LOGRTHM2.BS2
' logarithm base 2 by calculation
' Tracy Allen
' implements Knuth, ArtComProg ch. 1.2.2, exercise 24.
' characteristic and mantissa in base 2, e and 10

y   var word      ' to hold an input number from 1 to 65535
x   var word      ' for processing the number
xf  var x.bit15    ' high bit of x, note alias
x2  var word      ' for squaring the number
x2f var x2.bit15   ' high bit of x2, note alias
lgx var word      ' will be the lg (base 2) of y, the mantissa
lgx0 var lgx.bit0  ' lowest bit of lgx, for bit addressing
bitk var bit      ' temporary bit
k   var nib       ' loop and array index
cc  var nib       ' characteristic of the lg
lg  var word      ' to hold the log base 2
log var word      ' to hold the log base 10
ln  var word      ' to hold the log base e

pick:
debug cr,"Enter a number from 1 to 65536:"
serin 16,$54,[DEC y] ' get the number

cc=ncd y - 1      ' find the characteristic
x=y << (15-cc)    ' adjust for a denominator of 32768
'debug cr,"y=",dec dcd cc," * 1.",dec4 x**20000
' optionally, show the decomposition
lgx=0             ' initialize accumulator
for k=14 to 0     ' 15 steps of precision
  x2=x**x         ' high byte of x squared
  lgx0(k)=x2f     ' high bit of x squared is this bit of log.
  bitk=~x2f       ' complement of that bit
  x=x2<<bitk+(bitk&xf) ' adjust x
next              ' repeat

' lgx now holds the mantissa, log base 2 of x
' cc holds the characteristic

' show it:
debug cr,"lg x=",dec cc,".",dec4 lgx**20000,cr ' 0 to 15.9999

' combine it into one 16 bit word (but lose one digit!):
lg=cc*1000+(lgx**20000/10)
debug "lg x=",dec lg/1000,".",dec3 lg ' lg from 0 to 15.999

' convert it to log base 10:
log=lg**19728
```



```
debug "log x=",dec log/1000,".",dec3 log ' log from 0 to 4.816

' or to get one more digit resolution in log base 10:
' still 0.30103 * log2 x
' cc*10000*0.30103, in stampese is cc*10000**19728, or cc*4000**49321
' to avoid overflow in the first cc*4000, and to get best resolution in the 49321.
' First term **6021 because lgx as it comes out of the loop has an implied denominator
' of 32768. Multiply x2 to get denominator of 216 for **, with 0.30103 * 10000
log=lgx**6021+(cc*4000**49321)
debug "log x=",dec log/1000,".",dec4 log ' log from 0 to 4.8164

' convert log base 2 to log base e:
ln=lg**45426
debug "ln x=",dec ln/1000,".",dec3 ln ' ln from 0 to 11.089

pause 1000
goto pick          ' get another number to try
```

Al final, la demostración del programa combina la característica y la mantissa en una cantidad de 16 bits para su visualización, y se convierte a otras bases como ejemplos de cómo manipular el logaritmo.

Lectura de datos de palabras de una tabla, interpolación

Arriba

A veces, la mejor manera de condicionar la salida de un sensor o proceso no lineal es mediante una tabla de búsqueda. Por lo general, surge la pregunta de cuántas entradas se necesitan en la tabla. Por ejemplo, si un convertidor analógico a digital de 12 bits proporciona los datos, es posible que necesite una tabla con 4096 elementos para traducir cada lectura en sus correspondientes unidades científicas o de ingeniería de salida. Sin embargo, esto rara vez es necesario. La curva puede ser lo suficientemente suave como para que se puedan elegir algunos puntos para representar toda la curva, y los puntos intermedios se construyen sobre la marcha mediante interpolación. Esta podría ser la curva de calibración para un termistor, o una curva de altitud vs presión, o la curva de control para una variable de proceso, etc. La tabla contiene los valores de salida correspondientes a un conjunto de valores de entrada.

Hay varios tipos de interpolación que tienen en cuenta más o menos la curvatura local de los segmentos de línea que representa la tabla. La interpolación lineal es la más simple. Toma los pares de entradas de la tabla como puntos finales de los segmentos de línea recta. Un valor de salida calculado será una proporción igual a lo largo del segemnt de salida, ya que el valor de entrada está a lo largo de los puntos finales de entrada. De eso es de lo que voy a hablar aquí.

Por ejemplo, si el valor de entrada x se encuentra a 1/3 de la distancia entre dos de las entradas de la tabla de entrada, entonces el valor de salida calculado se situará 1/3 del camino entre las dos entradas de la tabla de salida correspondientes.

En la siguiente tabla, los valores de entrada comienzan en cero y aumentan uniformemente en saltos de 128. En el Sello BASIC es conveniente utilizar intervalos binarios, para facilitar los cálculos. Los valores de salida también comienzan en cero, pero no aumentan en saltos pares. En general, una tabla de interpolación no tiene que comenzar en cero, ni los valores tienen que estar espaciados uniformemente. Pero echemos un vistazo más de cerca a esta tabla en particular y la interpolación lineal.

entrada, X _i	salida, Y _i	Los valores de entrada son múltiplos de 128 para mayor comodidad en el cálculo en el sello BASIC. Aquí hay una interpolación lineal. Considere el valor de entrada, x = 427, que se encuentra entre los valores de entrada 384 y 512. Los valores y correspondientes a x=427 deben estar entre 1293 y 1699, que son los valores y correspondientes a 384 y 512. Calculamos la proporción a lo largo del eje x, (427-384)/(512-384)=43/128. La proporción a lo largo del eje y debe ser la misma. (43/128)*(1699-1293) = (43/128)*408 = 136. Sumando eso a 1293 se obtiene el valor y, 1429, correspondiente a x=427. Esto es interpolación lineal. x=427 y = 1429
0	0	
128	444	
256	875	
384	1293	
512	1699	
640	2095	
668	2479	
796	2854	

En el código stamp, un ejemplo. El resultado de una conversión analógica a digital se mantendrá en una variable, RESULT, y su valor puede ser un entero de 0 a 4095 recuentos. Asigne a la tabla 33 entradas y 32 intervalos. El intervalo cero es de 0 a 127, el primero de 128 a 255, ... , hasta el 31 de 3968 a 4095. El ancho de cada categoría es 128. La lectura del convertidor A/D encajará en una de las 32 catagorias (4096/128=32). La categoría que se debe usar se puede encontrar dividiendo la lectura actual de A / D por 128. El resto después de dividir es la posición dentro de la categoría, para la interpolación. Por ejemplo, si la lectura A/D resulta ser 427, la dividida por 128 da 3 con un resto de 43. El número está en la tercera categoría, y se encuentra a 43/128 del camino entre los puntos finales. El algoritmo selecciona las entradas de punto final y correspondientes de la tabla (1293 y 1699 en este caso), y luego calcula la diferencia entre ellas (1699-1293 = 406), toma 43/128 de eso (136) y agrega al valor inicial de la categoría (1293 + 136 = 1429). Ese es el resultado devuelto por la subrutina.

```
' Interpol.BS2
' Tracy Allen
' Example of how to interpolate in a table
table data word 0,word 444,word 875,word 1293,word 1699,word 2095
      data word 2479,word 2854,word 3219,word 3576,word 3923
      data word 4262,word 4594,word 4919,word 5236,word 5546
      data word 5850,word 6147,word 6439,word 6724,word 7004
      data word 7279,word 7549,word 7814,word 8074,word 8329
      data word 8580,word 8826,word 9069,word 9307,word 9542
      data word 9773,word 10000.

X var word      ' this will be the input variable
Z con 4096      ' the maximum value of X, 0<=X<Z
```

```

' make this an exact power of 2.
L con 32      ' the number of intervals in the table
K con Z/L     ' the width of catagories of X
M con 65536/K ' for the interpolation formula
J var byte    ' Jth table entry, 0<=J<=32
F var word    ' the remainder of X in the catagory X=J*K+F
Q var word    ' for lower bound from table read, Q<=X
Q0 var Q.byte0 ' for table lookup, array of bytes
P var word    ' for upper bound from table read, P>X
I var nib     ' index into table for lowbyte highbyte
Y var word    ' this will be the output variable

X=427         ' for example, an input datum
J=X/K         ' choose the catagory within the table
F=X//K        ' remainder
for I=0 to 3  ' read values that bracket the catagory.
  read J*2+I+table,Q0(I) ' Q and P are contiguous 4 bytes
next
Y=P-Q*F/K+Q   ' simple interpolation, works only when P-Q and F are small.
' Y=(P-Q)**(M*F) +Q ' better alternative
' Y=(P-Q)**(F<<(17-NCD K))+Q ' another alternative, using fast shifts
debug DEC J,tab,DEC Q,tab,DEC P,tab,dec Y,CR
next

```

La fórmula de interpolación necesita algunos comentarios. La fórmula básica de interpolación es

$$Y = Q + ((P-Q) * F/K) \quad \text{' e.g. } 1293 + ((1699-1293)*43/128)$$

esto es lo mismo que la fórmula en el programa, solo reorganizada.

$$Y = P - Q * F / K + Q$$

Esta fórmula simple funcionará directamente cuando tanto P-Q como F sean pequeños, siempre y cuando su producto sea inferior a 65536. En el ejemplo, 408*43 es solo 17544. Resulta que en esta tabla, el valor de F nunca será mayor que 127, y P-Q nunca será mayor que 444, por lo que el producto nunca será mayor que 56388. Así que la fórmula simple estará bien. Pero es algo que hay que tener en cuenta. La fórmula no funcionaría, por ejemplo, si el valor P-Q fuera 4444 en lugar de 444.

La fórmula alternativa aprovecha el operador **: F/K se aproxima por una nueva fracción que tiene un denominador de 65536. El numerador proviene de resolver la ecuación:

$$\begin{aligned} \text{Numerator}/65536 &= F/K \\ \text{Numerator} &= (65536/K)*F \end{aligned}$$

El factor 65536/K se puede calcular de antemano, y es 512 en este ejemplo, donde K =128. Tenga en cuenta que el producto de M * F nunca puede ser mayor que 65536, porque F siempre es menor que K. (Otra fórmula alternativa logra la tarea implementando el factor (65536 / K) como un desplazamiento binario. Eso funciona solo cuando las categorías son anchos binarios, pero es rápido y fácil de transferir a un PIC. Cuando las categorías son de 128 de ancho, entonces 17- NCD K es igual a 9. Un turno 9 lugares a la izquierda es lo mismo que multiplicar por 512.

Otra advertencia es que el valor de P-Q no puede ser negativo, porque la división no funciona correctamente en el sello para números negativos. El programa anterior utiliza valores de datos que aumentan a medida que aumenta la entrada. Si los valores sucesivos van a ser decrecientes, o si la curva a aproximar tiene segmentos tanto crecientes como decrecientes, la fórmula debe ser adaptada o generalizada. Aquí hay una fórmula de interpolación que puede manejar valores positivos y negativos. El valor absoluto se separa para el tratamiento y luego se restaura el signo.

```

sign var Y.bit15
Y=P-Q
Y= -sign^(abs Y**(M*F))+sign +Q ' can handle both + and -

```

Si las entradas de la tabla son bytes en lugar de palabras, leerlas de la tabla se vuelve más fácil: en lugar del bucle for-next, es simplemente,

```

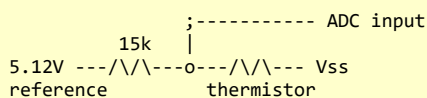
read J+table,Q
read J+1+table,P

```

Interpolación de lecturas del termistor

[Arriba](#)

Aquí hay un ejemplo de una linealización del termistor, donde el termistor es parte de un divisor de voltaje simple conectado a un convertidor A / D:



Este termistor en particular obedece a la ecuación simplificada de Steinhart-Hart

$$T = (B / (\ln(R) - A)) - C$$

donde $A = -6.913$, $B = 5078$, $C = 42.24$ (de la hoja de datos del fabricante), y R es la resistencia del sensor en ohmios y T es la temperatura en grados Celsius. El código producido por el ADC está directamente relacionado con la resistencia del termistor. (5.12 voltios da un código de 4096)

$$\text{code} = 4096 * R / (15000 + R) \quad \text{or} \quad R = 15000 * \text{code} / (4096 - \text{code})$$

Esto puede ser sustituido por R en la ecuación anterior:

$$T = (B / (\ln(15000 / (4096 / \text{code} - 1)) - A)) - C$$

Esto se puede conectar a un programa de hoja de cálculo como Excel™ para desarrollar una tabla de valores de temperatura correspondiente a diferentes códigos igualmente espaciados.

código de ADC	grados C	Kelvin*100
1	176.8	44999
128	71.67	34482
256	51.36	32451
384	40.05	31320
512	32.18	30532
640	26.09	29924
768	21.09	29424
896	16.80	28995
1024	13.03	28618
1152	9.63	28278
1280	6.51	27966
1408	3.61	27676
1536	0.87	27402
1664	-1.73	27142
1792	-4.24	26891
1920	-6.67	26648
2048	-9.06	26409

La columna final contiene los valores de Kelvin, multiplicados por 100, que es lo que usaré para las entradas de la tabla. Kelvin tiene la ventaja de ser siempre positivo. Tenga en cuenta que este programa utiliza adapta la interpolación simple para que pueda hacer frente a una función estrictamente decreciente del código de entrada.


Aquí está el programa:

```
' {$stamp bs2e}
' Tracy Allen, http://www.emesystems.com

table data word 44999,word 34482,word 32450,word 31320,word 30532,word 29923
      data word 29423,word 28995,word 28617,word 28277,word 27966
      data word 27675,word 27402,word 27141,word 26891,word 26647
      data word 26409
X var word ' this will be the input variable
Z con 2048 ' the maximum value of X, 0<=X<Z ' make this an exact power of 2.
L con 16 ' the number of intervals in the table
K con 128 ' Z/L the width of catagories of X
M con 512 ' 65536/K for the interpolation formula
J var byte ' Jth table entry, 0<=J<=32
F var word ' the remainder of X in the catagory X=J*K+F
Q var word ' for lower bound from table read, Q<=X
Q0 var Q.byte0 ' for table lookup, array of bytes
P var word ' for upper bound from table read, P>X
I var nib ' index into table for lowbyte highbyte
Y var word ' this will be the output variable
degC var word

' Interpol.BS2e
' Tracy Allen info@emesystems.com
' Unidata Red thermistor, 15kohm termination to 5.120 volts
thermistor:
  gosub ADread ' not shown, returns wx
  debug dec wx,tab
  x=wx max 2047
  J=X/K ' choose the catagory within the table
  F=X//K ' remainder
  for I=0 to 3 ' read values that bracket the catagory.
    read J*2+I+table,Q0(I) ' Q and P are contiguous 4 bytes
  next
  Y = Q - ((Q-P)**(M/F)) ' interpolate
  degC = Y-27315 ' convert Kelvin*100 to degrees C *100
  debug DEC J,tab,DEC Q,tab,DEC Y,tab, rep "-"degC.bit15,dec abs degC/100, ".",dec2 abs degC,CR
  pause 1000 ' show values including degrees C, then pause
  goto thermistor
```

El programa muestra el código ADC devuelto, y los puntos finales del intervalo de temperatura, y el resultado de la interpolación en Kelvin*100 y en grados Celsius. Incluso con solo las 16 catagorias, la interpolación es precisa (en comparación con la fórmula de Steinhart-Hart) dentro de 0.1 grados Celsius.

[<top>](#) [<index>](#) [<home>](#)  [<mailto:info@emesystems.com>](mailto:info@emesystems.com)