Representación de la información: números en punto fijo y números en punto flotante

Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna

Febrero, 2016

Esquema de la lección

1. Los números con parte fraccionaria
2. Fracciones binarias
   - Rep. en binario natural de fracciones binarias
   - Rep. en complemento a 2 de fracciones binarias.
3. El formato de punto fijo
4. El formato de punto flotante
   - Características generales: normalización
   - Reglas básicas del estándar IEEE754
   - Valores especiales en IEEE754
   - Reglas para los números normalizados
   - El rango de los números normalizados
   - Los números denormalizados

Números con parte fraccionaria

- La información numérica necesita representar y operar números con parte fraccionaria.
- Los lenguajes de alto nivel como C suelen disponer de varios tipos de datos con estas características.
- Por ejemplo:
  - Float: flotante de precisión simple.
  - Double: flotante de precisión doble.
  - Long double: flotante de precisión extendida.

Las fracciones binarias en binario natural (I)

- Como ya hemos visto, las fracciones binarias separan la parte entera de la parte fraccionaria mediante un punto.
- Por ejemplo:
  - $(101{,}011)_2$
  - Interpretemos este número en binario natural:
    - La parte entera ya sabemos como convertirla a decimal $(101)_2 \rightarrow (5)_{10}$.
    - La parte fraccionaria se interpreta asignando a cada bit a partir del punto, la mitad del valor del peso correspondiente al bit situado más a la izquierda, por lo que la fracción binaria tiene su valor decimal:

$$y = \sum_{i=0}^{N_e} (2^i) \times b_i + \sum_{i=1}^{N_f} \frac{1}{2^i} b_{-i}$$

  - Así,
$(101{,}011)_2 = 5 + \frac{1}{2} \times 0 + \frac{1}{2^2} \times 1 + \frac{1}{2^3} \times 1 = 5 + 0{,}375 = 5{,}375$

Las fracciones binarias en binario natural (II)

- Conversión de decimal fraccionario a binario fraccionario:
  - Sea el número: $(6{,}5859375)_{10}$
  - Parte entera: $6 \rightarrow 110$
  - Parte fraccionaria, seguimos el proceso de la siguiente tabla basado en multiplicar por 2 y quedarnos con la parte entera para obtener 1001011.
  - Resultado final: $(110{,}1001011)_2$

	$0{,}5859375 \times 2$
1	$,171875 \times 2$
0	$,34375 \times 2$
0	$,6875 \times 2$
1	$,375 \times 2$
0	$,75 \times 2$
1	$,5 \times 2$
1	0

El problema de la precisión finita

- Es muy posible que el número decimal con parte fraccionaria no pueda ser representado exactamente. Por ejemplo:
  - Sea el número: $(0{,}2)_{10}$
  - Parte entera: $0 \rightarrow 0$
  - Parte fraccionaria, seguimos el proceso de la siguiente tabla. Como vemos el proceso continuaría de forma indefinida
  - Resultado final: $(0{,}0011001100\ldots)_2$
  - Hay un error debido a **la precisión finita del computador**.

	$0{,}2 \times 2$
0	$,4 \times 2$
0	$,8 \times 2$
1	$,6 \times 2$
1	$,2 \times 2$
0	$,4 \times 2$
1	$,8 \times 2$
1	6
...	...

Números con signo y fracciones binarias (I)

- La representación en complemento a 2 de fracciones binarias es análogo al de números enteros, pero hay una diferencia importante en la definición del operador NEG.
- Sea k el número positivo decimal fraccionario. Supongamos que quiero obtener el binario cuya interpretación en complemento a 2 es $-k$.
  - Obtenemos el binario fraccionario x, que representa a k
  - Aplicamos sobre x el operador $NEG(x)$, definido como:

$$NEG(x) = NOT(x) + 0{,}00\ldots01$$

  - Como vemos, ahora el valor añadido en la definición de NEG no es 1, sino el menor valor positivo representable diferente de 0.

Números con signo y fracciones binaria (II)

- NEG(x) también se puede calcular mediante el método de copiar los bits de derecha a izquierda hasta encontrar el primer 1, y entonces ir aplicando el operador NOT a cada bit. Este método es válido aquí también.
- Veamos un ejemplo:
  - Sea +69,75 en una celda de 10 bits, con 2 bits para la parte fraccionaria. La representación es: $(01000101{,}11)_2$
  - Aplicamos el operador NOT, cambiando 1s por 0s: $(10111010{,}00)_2$
  - Aplicamos la suma binaria con: $(00000000{,}01)_2$
  - El resultado es: $(10111010{,}01)_2$. Observen que también se podía haber obtenido por el método de copia + NOT.
  - Observen que $(10111010{,}01)_2$ en binario natural es $(186{,}25)_{10}$, que se corresponde con $2^8 - 69{,}75$ (definición de complemento a 2).

El formato de punto fijo (I)

- La utilización de fracciones binarias en el computador se denomina formato de punto fijo cuando el número de bits destinados a la parte entera y a la parte fraccionaria es siempre el mismo.
- La principal ventaja es que los algoritmos que permiten operar esta clase de representaciones son similares a los de los números enteros.
- El principal inconveniente, es que es más fácil que las operaciones den lugar a desbordamientos al obtenerse un resultado fuera de rango.
- Otro problema importante, es que no se aprovecha la capacidad real de representación. Por ejemplo, con cuatro bits en un punto fijo con dos bits para la parte fraccionaria, el menor positivo representable mayor que 0 es 00,01. Sin embargo, si no se usara punto fijo, sería 0,001.

El formato de punto fijo (II)

- Este formato se utiliza en procesadores especializados:
  - Procesadores para el tratamiento de señales digitales (DSPs).
  - Microcontralores.
- Así, lo encontramos en toda clase de sistemas empotrados formando parte de electrodomésticos, automóviles, sistemas de comunicación, etcétera.

El formato de punto flotante

- La notación científica en base decimal es utilizada para magnitudes que tienen un amplio rango, y al mismo tiempo se desea una gran precisión.
- En esta notación, un número diferente de 0 está normalizado, si el primer dígito diferente de cero se encuentra justo a la izquierda del punto decimal:

$$-245{,}34 \rightarrow -2{,}4534 \times 10^2$$

- La potencia de 10 se suele abreviar, de modo que el resultado anterior se expresa como: $-2{,}4534E2$

Normalización en binario

- Consideremos el siguiente número en binario:
  - $(10101{,}101)_2$
  - La normalización sería: $(1{,}0101101)_2 \times 2^4$
  - El efecto del exponente positivo es mover el "punto" hacia los bits menos significativos.
- Otro ejemplo:
  - $(0{,}01110101)_2$
  - La normalización es: $(1{,}110101)_2 \times 2^{-2}$
  - El efecto del exponente negativo es mover el "punto" hacia los bits más significativos.

Representación binaria en punto flotante

- Esta representación tiene como objetivo convertir un binario fraccionario en un número en punto flotante donde el punto fraccionario se puede ubicar multiplicando por una potencia de 2.
- Utiliza tres campos:
  - Un bit para el signo: 1 significa negativo y 0 positivo.
  - Un campo para la magnitud, denominado parte significativa o significante.
  - Un campo para el exponente de la potencia 2.

El estándar IEEE754

- El estándar IEEE754 establece normas precisas para la representación en punto flotante así como sobre la aritmética. En este curso sólo nos ocuparemos de la representación.
- Admite diferentes precisiones básicas en binario:
  - Precisión mitad (16 bits): exponente 5 bits, significante 10 bits.
  - Precisión simple (32 bits): exponente 8 bits, significante 23 bits.
  - Doble precisión (64 bits): exponente 11 bits, significante 52 bits.
  - Cuádruple precisión (128 bits): exponente 15 bits, significante 112 bits.
- Las más usadas actualmente son, precisión simple (single format) y precisión doble (double format).

Tipos de valores en IEEE754

El estándar IEEE754 asocia cadenas de bits a los siguientes tipos de valores:

- **Valores especiales**. Son el infinito, el -infinito, el NaN (not a number) y el número 0.
- **Números denormalizados**. En torno al valor 0 se establece un intervalo pequeño donde las reglas de codifición / decodificación del estándar cambian para mejorar la precisión.
- **Números normalizados**. Es el resto de números reales, donde se usan las normas "básicas" del estándar.

Valores especiales en IEEE 754: el valor 0

- El 0 requiere de una representación especial. El binario IEEE 754 se interpreta como 0 siempre que tanto el exponente como el significante tengan todos sus bits a 0.
- Obsérvese que hay dos códigos para el 0, uno con el bit de signo a 0 y otro con el bit de signo a 1.

Valores especiales en IEEE 754: $+\infty$, $-\infty$ y *NaN*

- $+\infty$ y $-\infty$. Se usan para representar las regiones de overflow o para indicar el resultado de operaciones como la división por 0. La regla para representarlos es: establecer a 1 todos los bits del campo del exponente y a 0 todos los bits del significante. El bit de signo, indica si es positivo o negativo.
- Not a Number o *NaN*. Se usa para representar el resultado de operaciones no permitidas en el estándar, como por ejemplo, la raíz cuadrada de un número negativo. La codificación implica poner a 1 todos los bits del exponente y cualquier combinación de bits en el significante que sea diferente de 0.

Determinación del juego de reglas a utilizar: números normalizados o denormalizados

- Hemos visto que al margen de los valores especiales hay dos rangos en los números reales que usan reglas de codificación ligeramente diferentes: los números denormalizados, cercanos al valor 0, y el resto de los números.
- Detectamos un número denormalizado si:
  - El código no es un valor especial, y además:
  - Todos los bits del exponentes son 0, y además:
  - Hay bits diferentes de 0 en el significante.
- En cualquier otro caso se trata de un número normalizado y se aplican las reglas básicas del estándar.

El problema de la codificación

El primer paso en la codificación de un número en el estándar IEEE754 es determinar si es un número en el rango de los normalizados o en el rango de los denormalizados. Esto se puede hacer de dos formas:

- Previamente hemos calculado los límites del intervalo de los denormalizados en torno al cero. Si el número está en el intervalo será denormalizado y en caso contrario normalizado.
- Hacemos la hipótesis de que el número es normalizado y obtenemos el binario fraccionario normalizado que le corresponde en representación módulo / signo. Si aplicamos ahora las normas del estándar podremos comprobar durante el proceso si la hipótesis es correcta o no. En caso de que sea incorrecta volvemos a empezar usando el procedimiento específico para los números denormalizados.

Reglas básicas del IEEE 754 para números normalizados

- Estas son las reglas particulares para los números normalizados
- Por lo general se asume que el número debe estar normalizado. Posteriormente veremos una excepción importante a esta regla.
- El exponente de n bits se representa en exceso $2^{n-1} - 1$.
- La cifra a la izquierda del punto, siempre será un 1 (número normalizado), por lo que no es necesario representarla.
- Los campos se sitúan en el siguiente orden: signo, exponente, significante.
- Existen códigos especiales para el exponente que obligan a una interpretación diferente del número.

Reglas para la representación del exponente en números normalizados (I)

- Se utiliza un formato especial denominado exceso Z.
- Transformación de decimal a exceso Z en una celda de n bits:
  - Se suma Z al número. El resultado debe quedar comprendido entre 1 y $2^n - 2$ (si no es así estamos fuera del rango).
  - **Si el resultado fuese inferior a 1, significa que el módulo es demasiado pequeño para ser normalizado debemos tratar de codificar el número considerándolo "denormalizado".**
  - **Si el resultado es superior a $2^n - 2$ tenemos un desbordamiento. El número es demasiado grande.**
  - El resultado de la suma se transforma a binario natural.
- Continúa en la siguiente transparencia.

Reglas para la representación del exponente en números normalizados (II)

Ejemplo:

- Para una celda de $n = 5$ bits el exceso a considerar en los números normalizados es $Z = 2^{n-1} - 1 = 15$.
- Sea el binario fraccionario normalizado $1.m \times 2^x$.
- El menor valor para x posible sería -14. Si fuera -15 al sumarle el exceso de 15 obtendríamos 0 y el exponente no sería propio de los números normalizados. Si x fuera menor que -14 habría que usar las reglas de los números denormalizados.
- El mayor valor posible para x sería 15. Si fuera por ejemplo 16 y sumáramos el exceso se obtendría 31. Este valor del exponente en el formato IEEE754 implica que todos sus bits son 1 y entonces habría que interpretarlo como un símbolo especial.

Ejemplo de uso para un número normalizado

- Asumamos 4 bits para el significante y 3 para el exponente. Calculemos la representación bajo las reglas básicas del $(0{,}34)_{10}$.
  - Construimos la representación binaria fraccionaria: $0{,}0101011\ldots$.
  - El bit de signo es 0 (número positivo).
  - Normalizamos: $0{,}0101011\cdots = 1{,}01011\cdots \times 2^{-2}$
  - El significante queda representado por los 4 bits 0101. El 1 de la parte entera, no se representa porque siempre tendrá ese valor en los números normalizados.
  - Calculamos el exceso $2^{n-1} - 1 = 2^2 - 1 = 3$: $-2 + 3 = 1 = (001)_2$.
  - El resultado es: $0|001|0101$ conforme a las reglas.

El rango de los números normalizados en el IEEE 754 (I)

- Supongamos un exponente de 3 bits y un significante de 4 bits.
- El **menor número mayor que cero** representable es:

$$0|000|0001 = (1{,}0001)_2 \times 2^{-3} = (0{,}0010001)_2 = (0{,}1321825)_{10}$$

- El **mayor número positivo diferente de** $+\infty$ es:

$$0|110|1111 = (1{,}1111)_2 \times 2^3 = (1111{,}1) = (15{,}5)_{10}$$

- Para los número negativos el cálculo es similar. De tal manera que ....

El rango de los números normalizados en el IEEE 754 (II)

- Pueden darse las siguientes circunstancias que impiden la representación de un número normalizado:
  - Undeflow en el rango positivo.
  - Overflow en el rango positivo.
  - Underflow en el rango negativo.
  - Overflow en el rango negativo.
- El underflow implica que el número en valor absoluto es demasiado pequeño para ser representado.
- El overflow implica que el número en valor absoluto es demasiado grande para ser representado.

Overflow > 0
15.5
...
0.1321825
Underflow > 0
0
Underflow < 0
-0.1321825
...
-15.5
Overflow < 0

Los números denormalizados (I)

- Con el objetivo de reducir la región de underflow el estándar IEEE 754 establece excepciones a las reglas básicas: es lo que se denomina número denormalizado.
- Para interpretar el número como denormalizado es necesario que todos los bits del exponente sean 0 y que el significante no sea 0.
- Ahora, el exponente debe interpretarse en exceso $2^{(n-1)} - 2$. Se consigue así distribuir de manera más uniforma los números en la región de undeflow de los normalizados.
- El bit oculto a la izquierda del punto se asume que es 0 en lugar de 1.

Los números denormalizados (II)

- Ejemplo (exponente 3 bits, significante 4 bits): 0|000|0100
  - El patrón nos lleva a considerarlo como denormalizado.
  - El número es positivo porque el bit de signo es 0.
  - El exponente se interpreta en exceso $2^{(n-1)} - 2 = 2^2 - 2 = 2$. Luego en este caso el exponente es -2.
  - El significante es (bit oculto es 0) $(0{,}0100)_2$.
  - El resultado es $(0{,}0100_2) \times 2^{-2} = (0{,}0001)_2 = (0{,}0625)_{10}$.

El rango bajo las reglas del IEEE754

- Overflow (positivo y negativo).
- Rango de los normalizados (positivo y negativo).
- Rango de los denormalizados (positivo y negativo).
- Underflow (positivo y negativo)

Con 3 bits para el exponente y 4 para el significante tenemos:

Overflow	
Mayor pos. norm.	0|110|1111
	. . .
Menor pos. norm.	0|001|0000
Mayor pos. denorm.	0|000|1111
	. . .
Menor pos. denorm.	0|000|0001
Underflow	
0	0|000|0000
Underflow	
Menor abs neg. denorm.	1|000|0001
	. . .
Mayor abs neg. denorm.	1|000|1111
Menor abs. neg. norm.	1|001|0000
	. . .
Mayor abs. neg. norm.	1|110|1111
Overflow	