Estimación de Errores en Aplicaciones Numéricamente Intensivas e Implementación de Funciones Elementales de Dos Variables

Javier Díaz Bruguera

Departmento de Electrónica y Computación
Universidad de Santiago de Compostela

Universidad Complutense de Madrid. Junio 2012

Índice

Índice

Procesadores de Propósito General

Procesadores Superescalares

- Procesamiento simultáneo de varias instrucciones
- Búsqueda/decodificación de varias instrucciones por ciclo
- Desacoplamiento de la emisión y el chequeo de dependencias
  - Ejecución determinada por la dependencia entre instrucciones
  - Ejecución fuera de orden
  - Planificación de instrucciones
- Un gran número de instrucciones en el procesador

Número elevado de unidades funcionales

- Unidades de punto fijo: Operaciones sobre enteros, cálculo de direcciones
- Unidades de punto flotante: operaciones sobre reales

Los números y su representación

Representación punto fijo: números enteros

- Rango dinámico limitado
- Overflows y underflows frecuentes
- No es posible representar números muy grandes o muy pequeños
  - 32 bits $\Rightarrow [-2^{-16}, 2^{16} - 1]$

Representación punto flotante: números reales

- Mayor rango dinámico
  - 32 bits $\Rightarrow [-2^{-126}, 2^{127}]$
  - Como en punto fijo, ¡solo podemos representar 2^{32} números!
- Adecuado para aplicaciones científicas y de ingeniería

Operadores punto flotante

Operaciones

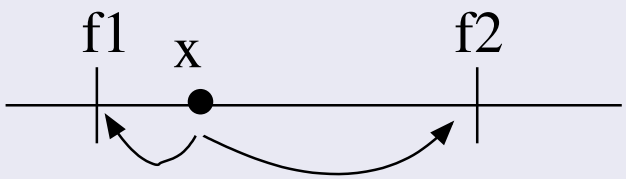
- Básicas: suma/resta y multiplicación
- Iterativas: recíproco/división, inverso raíz cuadrada/raíz cuadrada
- Nuevas operaciones: $A + B \times C$
  - punto fijo: *MAC (Multiply–and–ACcumulate)*
  - punto flotante: *MAF (Multiply–Add–Fused)*, *FMA (Fused–Multiply–Add)*
- Otras operaciones: e^x, 2^x, $log_2(x)$, $ln(x)$, $sin(x)$, $cos(x)$, $tan(x)$, ...

Índice

Representación en punto flotante

Representación estándar de números reales

- Elevado rango dinámico
- Número finito de bits: Representación de un conjunto finito del conjunto infinito de números reales
  - Número real representable exactamente en punto flotante: *Número punto flotante*
  - Resto de número reales: Aproximados por un número punto flotante *(Redondeo)*
    - Produce un error de redondeo



Representación en punto flotante

Formato

- Tres componentes: signo, exponente y significando

$$x = (-1)^{S_x} \times M_x \times 2^{E_x}$$

- Significando: 1 bit entero, $M_x = 1.F_x$, $1 \leq M_x < 2$

$$x = (-1)^{S_x} \times 1.F_x \times 2^{E_x}$$

  - Normalizado
  - Bit entero: *Hidden bit*, no se representa
- Exponente: *Bias*: $B = 2^{e-1} - 1$, e num. bits exponente
  - $E = exponente + B$
  - $E \geq 0$

Representación en punto flotante

Rango del significando y *ulp*

- f bits fraccionales

$$M = 1 + \sum_{i=1}^{f} d_i \times 2^{-i}$$

$$1 \leq M \leq 2 - 2^{-f}$$

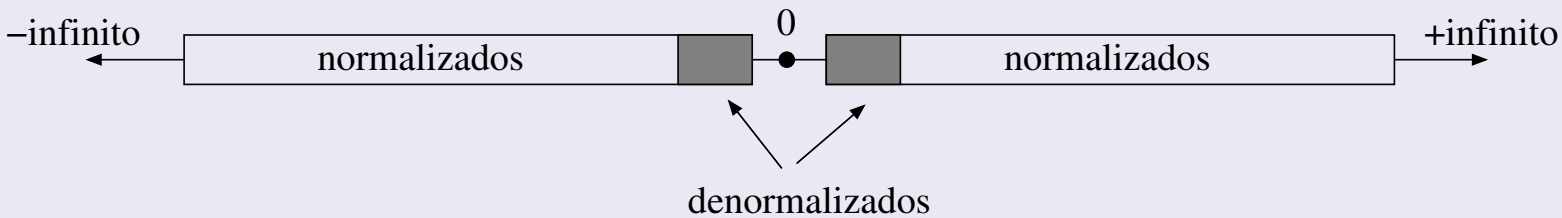
- ulp *(unit in the last place)*: La diferencia entre dos valores consecutivos del significando

$$ulp = 2^{-f}$$

Representación en punto flotante

Representación normalizada y no normalizada

- Normalización:
  - elimina la redundancia en la representación y mejora la exactitud al eliminar los ceros más significativos
  - reduce el rango : $M_{min} = 1 \Rightarrow x_{min} = 1 \times 2^{E_{min}}$
- Se admiten significandos no normalizados *solo* con el exponente mínimo: *Números Denormales*
  - al disminuir la magnitud, incluyen ceros en la parte más significativa



Estándar IEEE 754 – 1985

Parámetros

- Precisión simple (32 bits): *S(1), E(8), F(23)*. *Bias* $= +127$
- Doble precisión (64 bits): *S(1), E(11), F(52)*. *Bias* $= +1023$
- Precisiones extendidas *(Bias sin especificar)*:
  - Simple extendida (≤ 43): *S(1), E($\leq$ 11), F($\leq$ 32)*
  - Doble extendida (≤ 79): *S(1), E($\leq$ 15), F($\leq$ 64)*

Valores especiales

- Cero: $E = 0$, $F = 0 \Rightarrow$ $1{,}0 \times 2^{-B}$ no se representa
- Denormales: $E = 0$, $F \neq 0 \Rightarrow$ $(-1)^S \times 0.F \times 2^{-(B-1)}$, no normalizados
- Not–a–number (NAN): $E = 2B + 1$ *(max. expon.)*, $F \neq 0$
- Infinito (pos. y neg.): $E = 2B + 1$ *(max. expon.)*, $F = 0$

Operaciones con resultado redondeado

- Aritméticas: suma, resta, multiplicación, división y cálculo del resto

$$r = x \; REM \; y \Rightarrow r = x - y \times n$$

  con $y \neq 0$, y n el entero más próximo al valor exacto x/y
- Raíz cuadrada
- Conversiones de formato

Estándar IEEE 754 – 2008

Principales características

- Formatos
  - *Básicos*
    - Tres formatos binarios: 32, 64, 128 bits
    - Dos formatos decimales: 64 y 128 bits
  - *Extendidos*: extienden la precisión y el rango de los formatos básicos
  - *De intercambio* entre implementaciones
- Operaciones con redondeo: más que el estándar IEEE754-1985

Estándar IEEE 754 – 2008

Formatos básicos

	Formatos binarios			Formatos decimales	
Parámetro	binary32	binary64	binary128	decimal64	decimal128
p, dígitos	24	53	113	16	34
emax	127	1023	16383	384	6144

Estándar IEEE 754 – 2008

Parámetros del formato de intercambio binario

Parámetro	binary16	binary32	binary64	binary128	binary(k) $k \geq 128$
N^o bits	16	32	64	128	múltiplo de 32
precisión	11	24	53	113	$k - round(4 \times \log_2(k)) + 13$
signif.	10	23	52	112	$k - round(4 \times \log_2(k)) + 13$
emax	15	127	1023	16383	$2^{k-p-1} - 1$
exponente	5	8	11	15	$round(4 \times \log_2(k)) + 12$

- Ejemplo: binary256 $\Rightarrow$ precision=237, emax=262143

Estándar IEEE 754 – 2008

Operaciones con resultado redondeado

- Suma, resta, multiplicación, división, raíz cuadrada
- Fused Multiply Add: $fma(x, y, x) = (x \times y) + z$
- Conversiones de formato

Estándar IEEE 754 – 2008

Operaciones con *recomendación* de resultado redondeado

Tipo	Función
Exponenciales	e^x, $e^x - 1$, 2^x, $2^x - 1$, 10^x, $10^x - 1$
Logaritmos	$\log_e(x)$, $\log_2(x)$, $\log_{10}(x)$, $\log_e(1+x)$, $\log_2(1+x)$, $\log_{10}(1+x)$
Raíces	$\sqrt{x^2+y^2}$, $1/\sqrt{x}$, $x^{1/n}$
Potencias	x^n, x^y, $(1+x)^n$
Trigonométricas	$\sin(x)$, $\cos(x)$, $\tan(x)$, $\sin(\pi \times x)$, $\cos(\pi \times x)$
	$atan(x)$, $asin(x)$, $acos(x)$, $atan(x)/\pi$
Hiperbólicas	$\sinh(x)$, $\cosh(x)$, $\tanh(x)$, $asinh(x)$, $acosh(x)$, $atanh(x)$

Índice

Algunas operaciones punto flotante

Operaciones

- Básicas: suma/resta y multiplicación
  - Nueva operación: $A + B \times C$
  - *MAF (Multiply–Add–Fused)*, *FMA (Fused–Multiply–Add)*
- Iterativas: recíproco/división, inverso raíz cuadrada/raíz cuadrada
- Otras operaciones: e^x, 2^x, $log_2(x)$, $ln(x)$, $sin(x)$, $cos(x)$, $tan(x)$, ...

Suma punto flotante ($z = x + y$)

Algoritmo básico

1. Restar exponentes. $d = E_x - E_y$
2. Alinear significandos
   - Desplazar operando con menor exponente
   - Exponente resultado
3. Sumar (restar) significandos
   - Operación efectiva
4. Normalización del resultado
   - Suma: desplaz. dcha.
   - Resta: desplaz. izqd.
   - Ajustar exponente
5. Redondeo: Suma

x: 1.0001110011 exp=–20

y: 1.1001010101 exp=–23

↓ alineam.

x: 1.0001110011 exp=–20

y: 0.0011001010 *101* exp=–20

↓ resta

z: 0.1110101000 *011* exp=–20

↓ normaliz.

z: 1.1101010000 *110* exp=–21

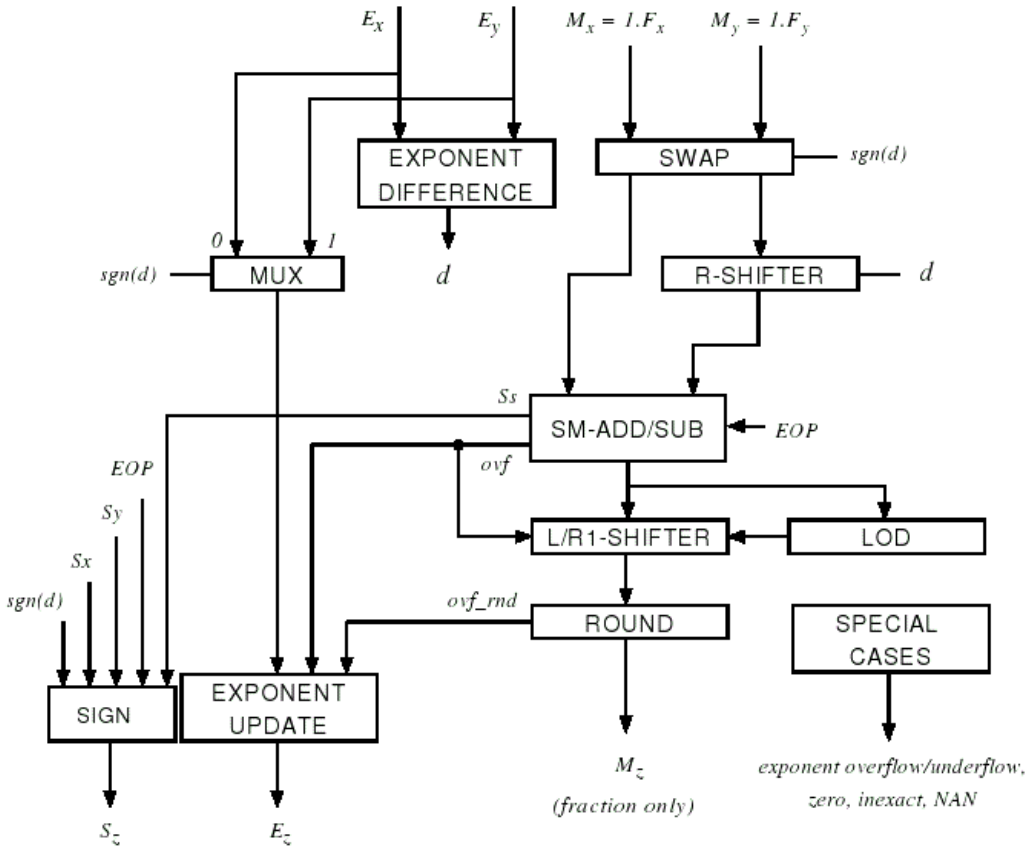
↓ redondeo

z: 1.1101010001 exp=–21

Suma punto flotante: implementación

Algoritmo básico

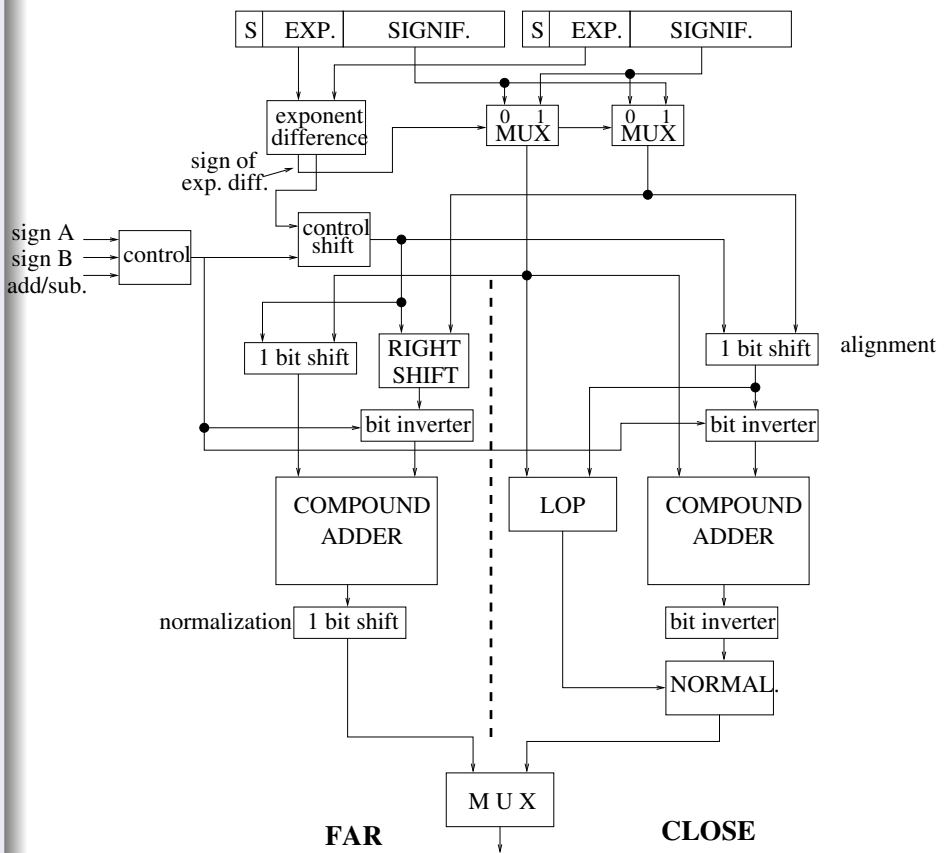
1. Diferencia de exponentes e intercambio
2. Alineamiento
3. Suma signo–magnitud de significandos
4. Normalización: LOD para determinar la posición del 1 más significativo
5. Redondeo. Puede producir *overflow*



Suma punto flotante: Implementación mejorada

Algoritmo mejorado

1. Suma complemento a 2
2. LOP *(Leading-one predic.)*. En paralelo con la suma
3. Doble vía de datos
   - CLOSE: Restas con diferencia de exponentes 0, 1
     - Alineamiento de 1 bit
     - Normalización masiva
   - FAR: Restas con diferencia de exponentes mayor que 1 y sumas
     - Alineamiento masivo
     - Normalización de 1 bit
4. Combinacion de suma y redondeo



Multiplicación punto flotante ($z = x \times y$)

Algoritmo básico

1. $M_z = M_x \times M_y$, $E_z = E_x + E_y$
2. Normalización de M_z
3. Actualizar exponente
4. Redondeo. Puede producir *overflow*

x: 1.0001110011 exp=−20

y: 1.1001010101 exp=−23

↓ multiplicacion y exponente

z: 1.1100001011 0000011111 exp=−43

↓ normaliz.

z: 1.1100001011 0000011111 exp=−43

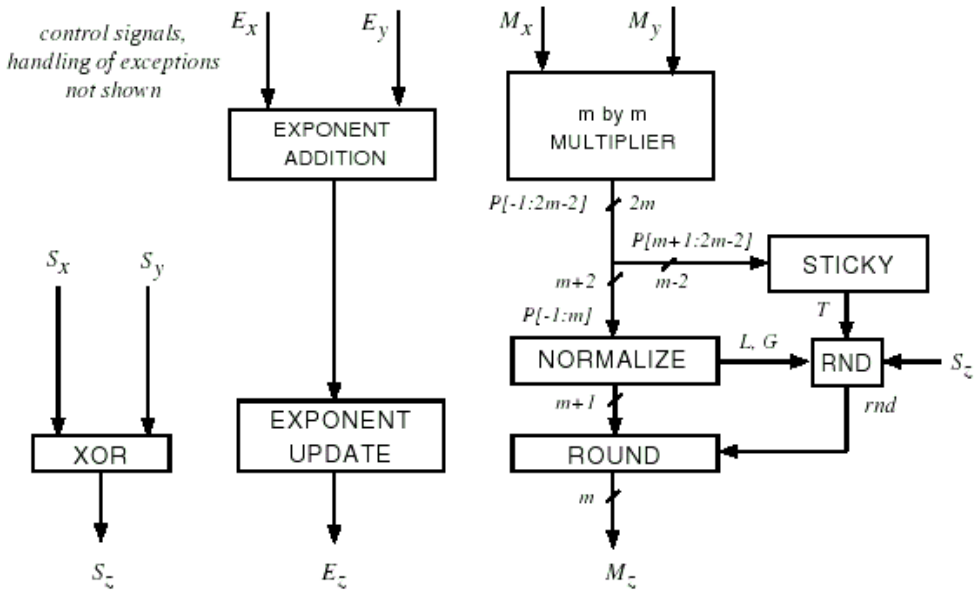
↓ redondeo

z: 1.1100001011 exp=−43

Multiplicación punto flotante: implementación

Algoritmo básico

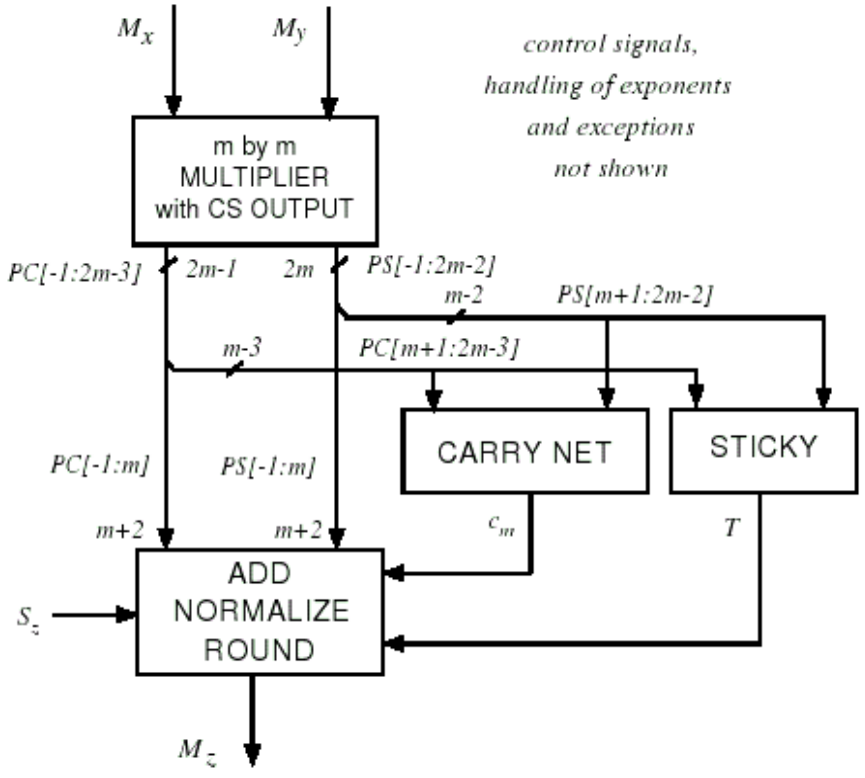
1. Multiplicación de significandos ($2m$ bits)
2. Suma exponentes (con *bias*)
3. Signo del resultado
4. Normalización ($M_z \in [1, 4]$). Desp. 1 bit
5. Redondeo.



Multiplicación punto flotante: Implementación mejorada

Algoritmo mejorado

1. Cálculo de los $m+1$ bits más significativos del resultado
   - Representación *Carry-Save*
   - Conversión de los $m+1$ bits más signif.
   - Resto de bits: sticky y carry
2. Combinación de suma, normalización y redondeo

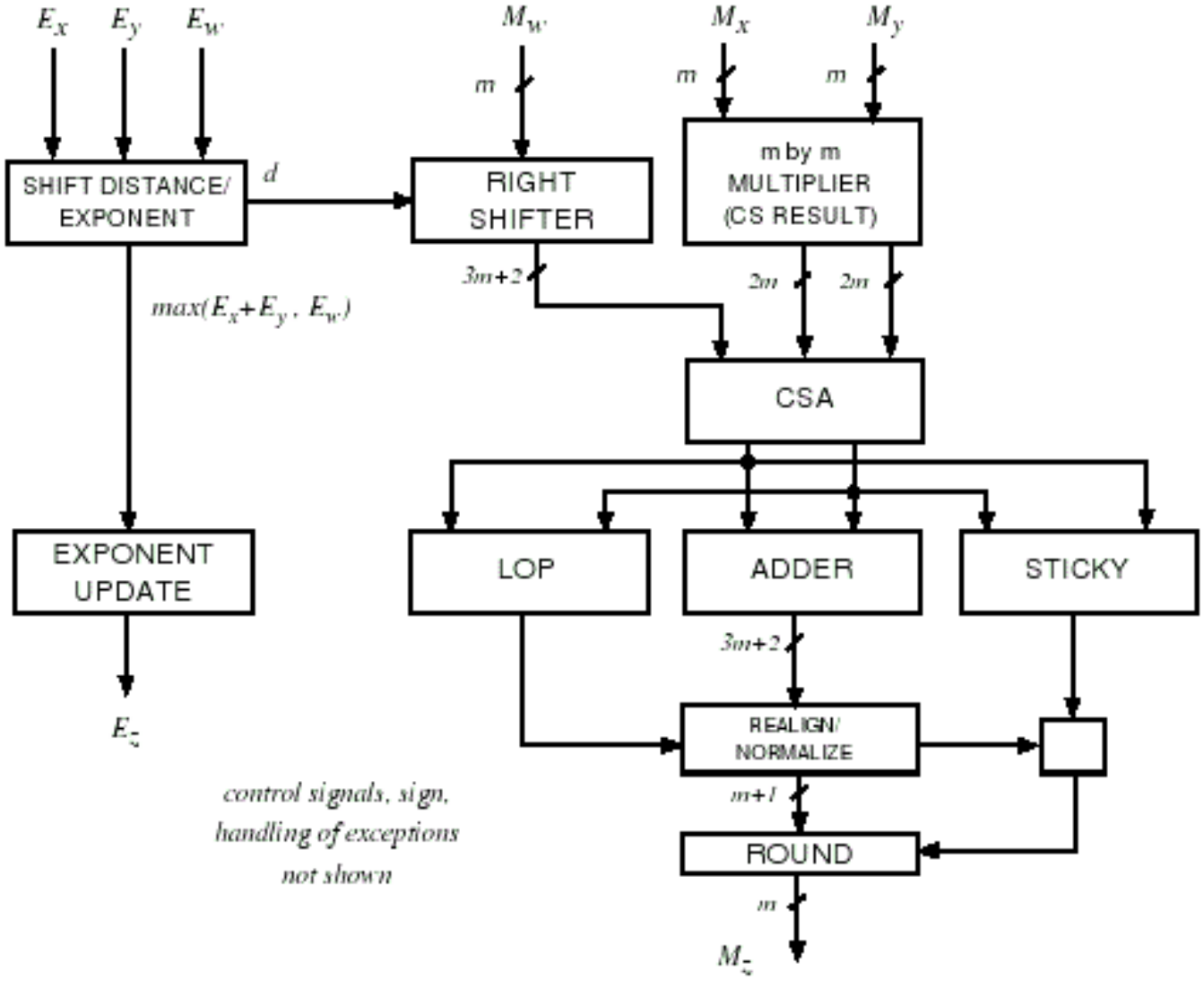


MAF punto flotante

MAF: Multiply-Add Fused

- Operación: $z = x \times y + w$
  - Incluye suma ($y = 1$) y multiplicación ($w = 0$)
- Ventajas:
  - Reduce el número de componentes (sumadores, desplazadores, ...) e interconexiones
  - Mejora la exactitud de la operación MAF: Una *normalización/redondeo* en lugar de dos
  - Ayuda al compilador a obtener código más eficiente
- Desventajas:
  - Misma latencia para multiplicación y suma
  - Mayor número de bits en el sumador
  - Normalizador más complejo
- Procesadores con MAF: Intel Itanium, IBM Power, ...

MAF punto flotante: Estructura básica



MAF punto flotante: Mejoras sobre la estructura básica

Objetivos

- Reducir la complejidad hardware
- Reducir el retraso (ciclos)

¿Cómo?

- Simplificación del sumador (en la parte más significativa sólo hay un operando)
- Doble vía de datos

Técnica que no puede ser aplicada al MAF

- Combinar suma y redondeo intercambiando el orden del redondeo y la normalización
- No se conoce en que bit hay que redondear

División y raíz cuadrada punto flotante

Operación $q = x/d$

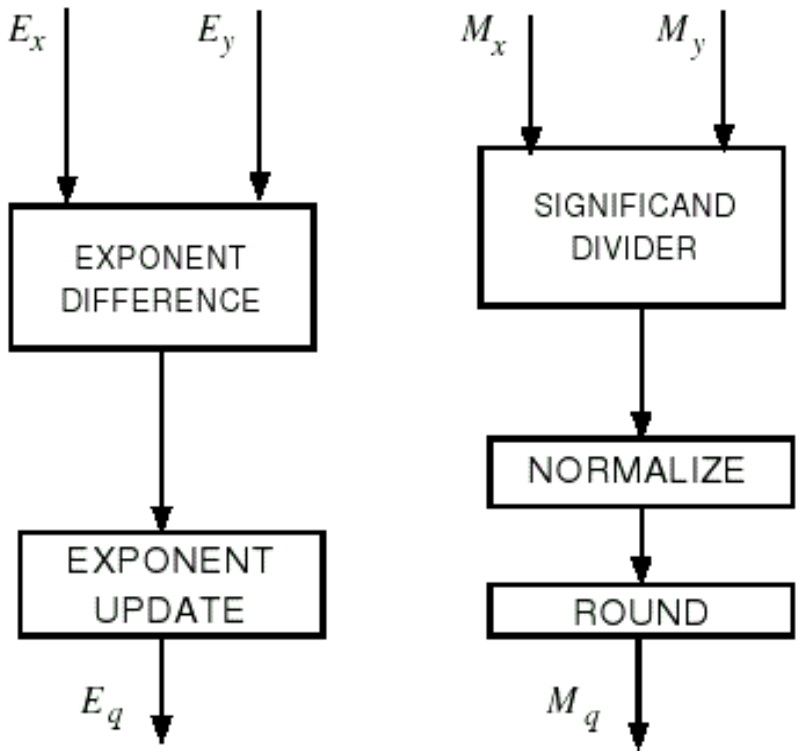
1. Dividir significandos y restar exponentes

$$\begin{aligned} M_q &= M_x / M_d \\ E_q &= E_x - E_d \end{aligned}$$

2. Normalizar: desplaz. 1 bit
3. Redondear

Division de significandos

- Algoritmos multiplicativos
- Algoritmos basados en recurrencias



Algoritmos multiplicativos

Método

- Cálculo del recíproco o inverso de la raíz cuadrada y multiplicación adicional

$$q = x/d = (1/d) \times x$$

$$r = \sqrt{x} = (1/\sqrt{x}) \times x$$

- Mejora iterativa de una aproximación inicial
- Convergencia cuadrática
  - El número de bits de la aproximación se dobla en cada iteración
- Basado en multiplicaciones y sumas
  - Reutilización del multiplicador punto flotante existente
  - Requiere poco hardware adicional
- Redondeo: iteración extra (cálculo del resto)

Cálculo del recíproco: Método Newton–Raphson

Obtención del cero de una función

- Valor de x tal que $f(x) = 0$
- Si $x[j]$ es una aproximación al cero, $x[j+1]$ es un aproximación mejor

$$x[j+1] = x[j] - \frac{f(x[j])}{f'(x[j])}$$

Aplicación al recíproco, $x = 1/d$

- Función $f(R) = 1/R - d$. El cero es $R = 1/d$
- Como $f'(R) = -1/R^2$, se obtiene

$$R[j+1] = R[j] \times (2 - R[j] \times d)$$

- Aproximación inicial: $R[0]$

Cálculo del recíproco: Método Newton–Raphson

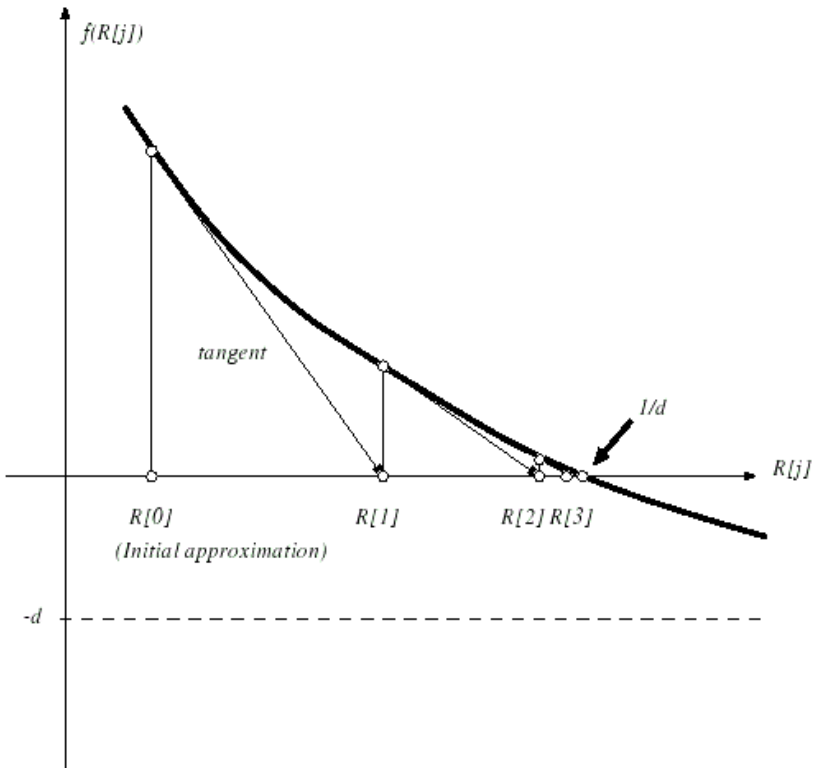
Convergencia cuadrática

- Si $\epsilon[j]$ error en la iteración j, entonces $\epsilon[j+1] = \epsilon[j]^2$
- Número de iteraciones (m)

$$m = \lceil log_2(\frac{n}{k}) \rceil$$

n: precisión final,
k: precisión aprox. inicial

- Aproximación inicial crítica para la latencia del algoritmo



Cálculo del recíproco: Método de Goldschmidt

Multiplicaciones independientes

- Dos recurrencias: convergen a 1 y a la función deseada

$$R = \frac{1}{d} = \frac{1}{d} \times \frac{P[0]}{P[0]} \times \frac{P[1]}{P[1]} \cdots \times \frac{P[m]}{P[m]} = \frac{R[m]}{d[m]}$$

- Dos recurrencias $P[j]$ tal que $R[j] \to 1/d$, $d[j] \to 1$:

Algoritmo ($R[m] = 1/d$)

1. Aprox. inicial $P[0]$, $d[0] = d \times P[0]$, $R[0] = P[0]$
2. For $j = 0 \ldots m-1$ do

$$\begin{aligned} P[j+1] &= 2 - d[j] \\ d[j+1] &= d[j] \times P[j+1] \\ R[j+1] &= R[j] \times P[j+1] \end{aligned}$$

Inverso de la raíz cuadrada $r = 1/\sqrt{(x)}$

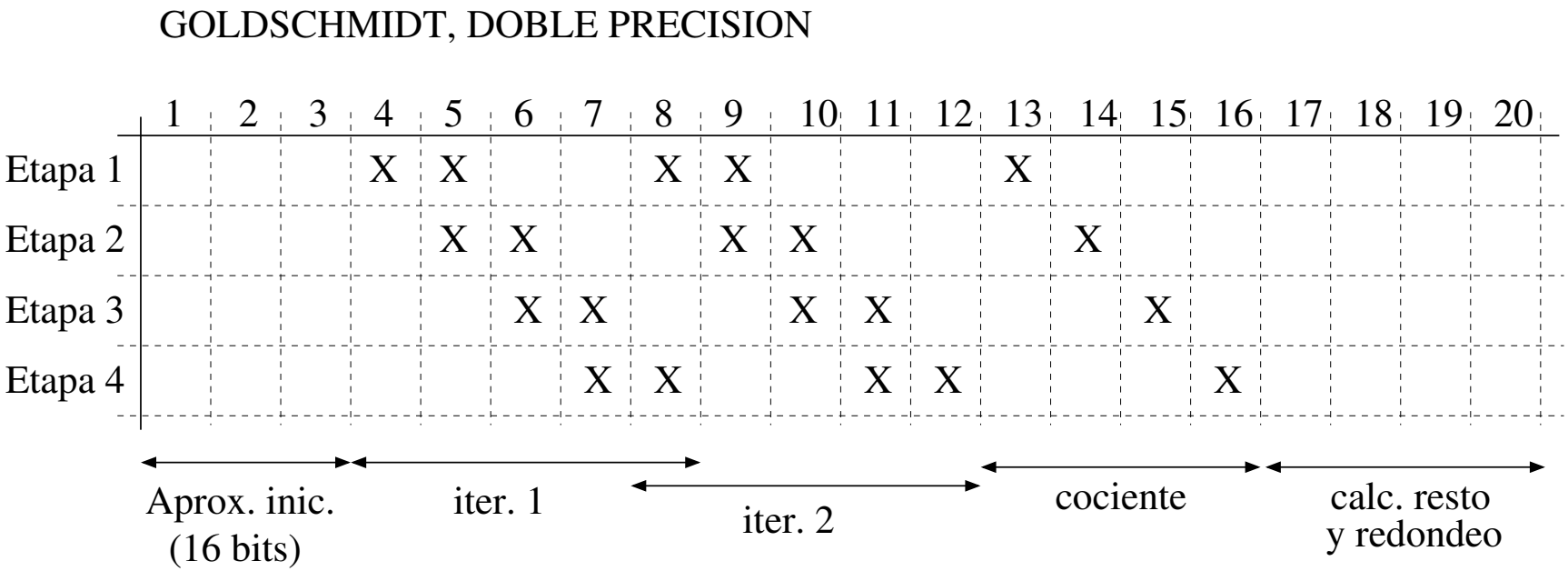
Newton-Raphson

$$R[j+1] = \frac{1}{2} \times R[j] \times (3 - R[j]^2)$$

Goldschmidt

$$\begin{aligned} P[j+1] &= \frac{1}{2} \times (3 - d[j]) \\ d[j+1] &= d[j] \times P[j+1]^2 \\ R[j+1] &= R[j] \times P[j+1] \end{aligned}$$

Ejemplo de división: AMD K7 (K8, K10)



Algoritmos Basados en Dígito–Recurrencias

Características

- Algoritmo iterativo: un dígito por iteración (dígitos radix r)
- Basado en la actualización de una *recurrencia*
  - División

$$\begin{aligned} w[j+1] &= rw[j] - dq_{j+1} \\ q_{j+1} &= SEL(rw[j], d) \end{aligned}$$

  - Raíz cuadrada

$$\begin{aligned} w[j+1] &= rw[j] - 2S[j]s_{j+1} - s_{j+1}^2 r^{-(j+1)} \\ s_{j+1} &= SEL(rw[j], S[j]) \end{aligned}$$

- Mayor latencia que los algoritmos multiplicativos
  - Convergencia lineal vs. cuadrática
  - Radix más elevado: menos iteraciones, más hardware

Algoritmos Basados en Dígito–Recurrencias

Representación redundante

- $q[j]$ cociente después de j iteraciones:

$$q[j] = q[0] + \sum_{i=1}^{j} q_i r^{-i}$$

- Representación redundante: $q_i \in \{-a, \ldots, 0, \ldots a\}$, $a \geq \lceil r/2 \rceil$
  - Ejemplo ($r = 4, a = 2$), tres dígitos: $0, 1, 2, -1, -2$
- Representación redundante para $w[j]$: Selección de los dígitos con una estimación de $w[j]$

Ventajas

- Implementación hardware sencilla (si radix moderado).
- No reutiliza las unidades del procesador

Evaluación de funciones

- Esencial en aplicaciones numéricas
  - Logaritmos, exponenciales, funciones trigonométricas, hiperbólicas
- Cálculo:
  - En software. Librerías con operaciones punto flotante
  - En Hardware. Particular para una función o conjunto de funciones
  - Algoritmos diferentes para software y hardware
- Funciones aproximadas: No pueden calcularse *exactamente* con un número finito de operaciones
- Sin redondeo: Es difícil obtener resultados correctamente redondeados

Evaluación de funciones

Métodos

- Tablas: valores de la función almacenados en una tabla
  - Precisiones bajas ($\leq$ 12 *bits*)
  - Precisiones mayores: tabla demasiado grande
- Aproximación polinómica: válido para funciones contínuas
  - Operaciones: Suma y multiplicación
  - Reducción del grado del polinómio: Aproximación inicial
- Dígito–recurrencias: recurrencia que converge a la función
  - Operaciones: Suma, multiplicación y desplazamientos
  - La recurrencia depende de la función
- Aproximaciones racionales: Cociente de dos polinomios
  - Desventaja: División
  - Útil para funciones con un polo
- Métodos válidos para rangos limitados: *reducción de rango* del argumento

Aproximaciones polinómicas

Polinomios directos

- Series de Taylor (u otros) truncadas

$$f(x) = f(x_0) + \sum_{i=1}^{\infty} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i$$

- Efectivos para aproximar la función en un punto
  - Error de aproximación elevado lejos de ese punto
  - El error depende del número de términos del polinomio
- Útiles para rangos pequeños (como parte de aproximación por tramos o con reducción de rango)
- Ejemplo: $y = sin(x/2)$, $0 \leq x \leq 1/2$, con $\epsilon_{abs} < 2^{-32}$

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

  - Número de términos: 4 $\Rightarrow$ ¡Muchas operaciones!

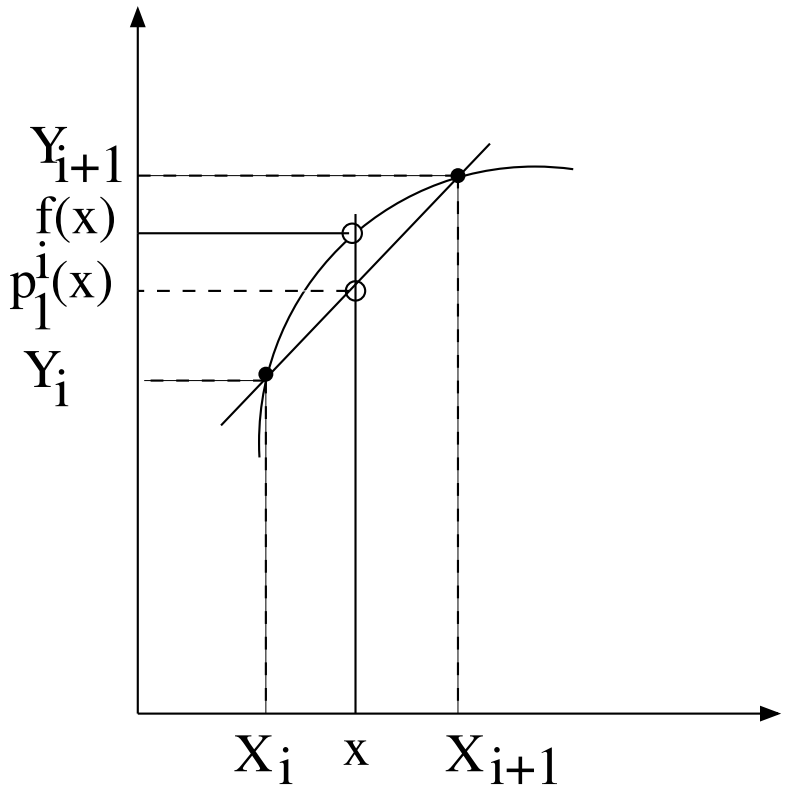
Aproximaciones polinómicas

Interpolación por tramos *(piecewise interpolation)*

- Polinomios diferentes para tramos diferentes
  - Reducción del grado del polinomio
- Interpolación lineal en cada tramo $(i, i+1)$:

$$p_1^{(i)}(x) = \frac{Y_{i+1} - Y_i}{X_{i+1} - X_i}(x - X_i)$$

- Para cada tramo se necesita: Y_i, $(Y_{i+1} - Y_i)/(X_{i+1} - X_i)$



Aproximaciones polinómicas

Ejemplo: $f(x) = x^{1/3},\ 1/2 \leq x < 1$

- Cuatro intervalos: $X_i = 1/2 + (1/8)i,\ i = 0, 1, 2, 3$

i	Y_i	$(Y_{i+1} - Y_i)/(X_{i+1} - X_i)$
0	0,7937	0,4904
1	0,8550	0,4288
2	0,9086	0,3824
3	0,9564	0,3488

- Ejemplo: $x = 0{,}788 \Rightarrow i = 2$

$$p(x) = 0{,}9086 + 0{,}3824(0{,}788 - 0{,}75) = 0{,}9231$$

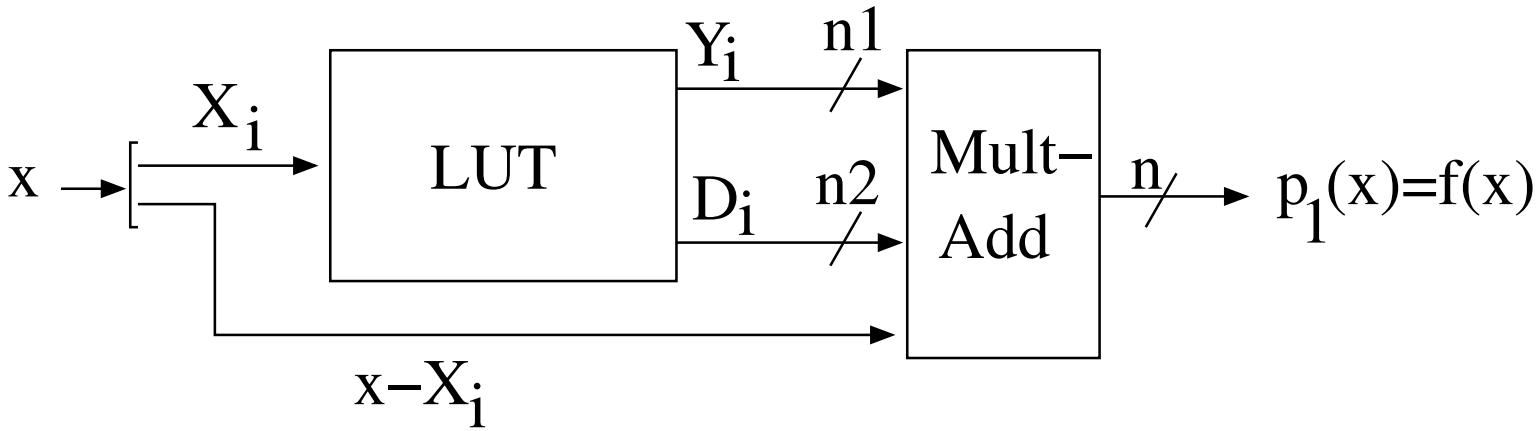
Aproximaciones polinómicas

Implementación

- X_i equiespaciados, múltiplos de 2^{-k}

$$x = X_i + x_r 2^{-k},\ x_r entero$$

- X_i: k bits mas significativos de x
- $(x - X_i)$ resto de los bits de x



Índice

Estimación de errores en aplicaciones punto flotante

Fuente de errores

- Error de redondeo
- Acumulación de errores de redondeo
- Amplificación del error relativo

- *¡El redondeo de la representación punto flotante es el origen de los problemas de errores en las aplicaciones científicas!*

Redondeo en punto flotante

- Resultado de una operación punto flotante: número real
  - Representación exacta: número infinito de bits
  - Punto flotante: f bits fraccionales
  - La representación debe estar próximo al resultado exacto
  - Condición: $x \leq y$ entonces $round(x) \leq round(y)$
- Redondeo:
  - $f1$, $f2$ números punto flotante consecutivos, si $f1 \leq x \leq f2$ entonces $round(x) = f1$ *o* $f2$
  - x calculado con más bits que los utilizados para representar $f1$ y $f2$

Redondeo en punto flotante

Modos de redondeo

- Al más próximo - par
  $(|x - f1(f2)| = d1(d2))$

$$R_{prox}(x) = \begin{cases} f1 & \text{si } d1 < d2 \\ f2 & \text{si } d1 > d2 \\ par & \text{si } d1 = d2 \end{cases}$$

- A cero

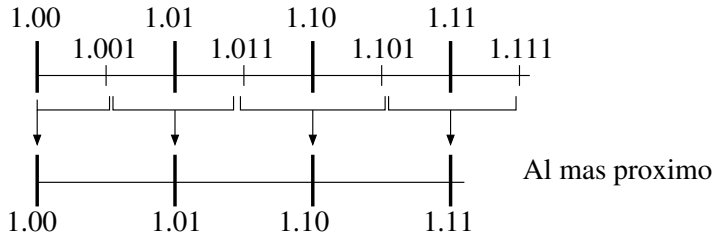
$$R_{cero}(x) = \begin{cases} f1 & \text{si } x \geq 0 \\ f2 & \text{si } x < 0 \end{cases}$$

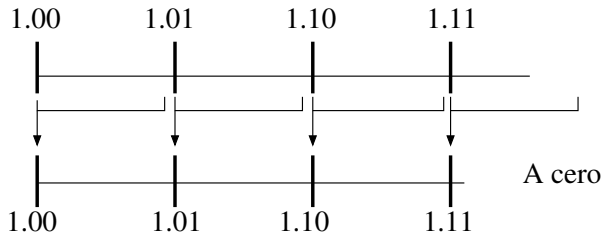
- A más infinito $(x \geq 0)$

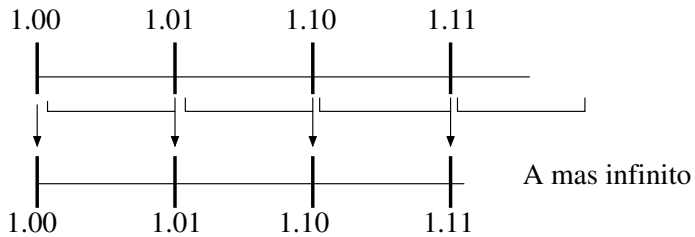
$$R_{+\infty}(x) = f2$$

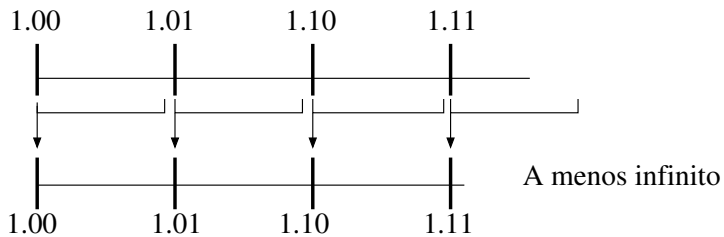
- A menos infinito $(x \geq 0)$

$$R_{-\infty}(x) = f1$$





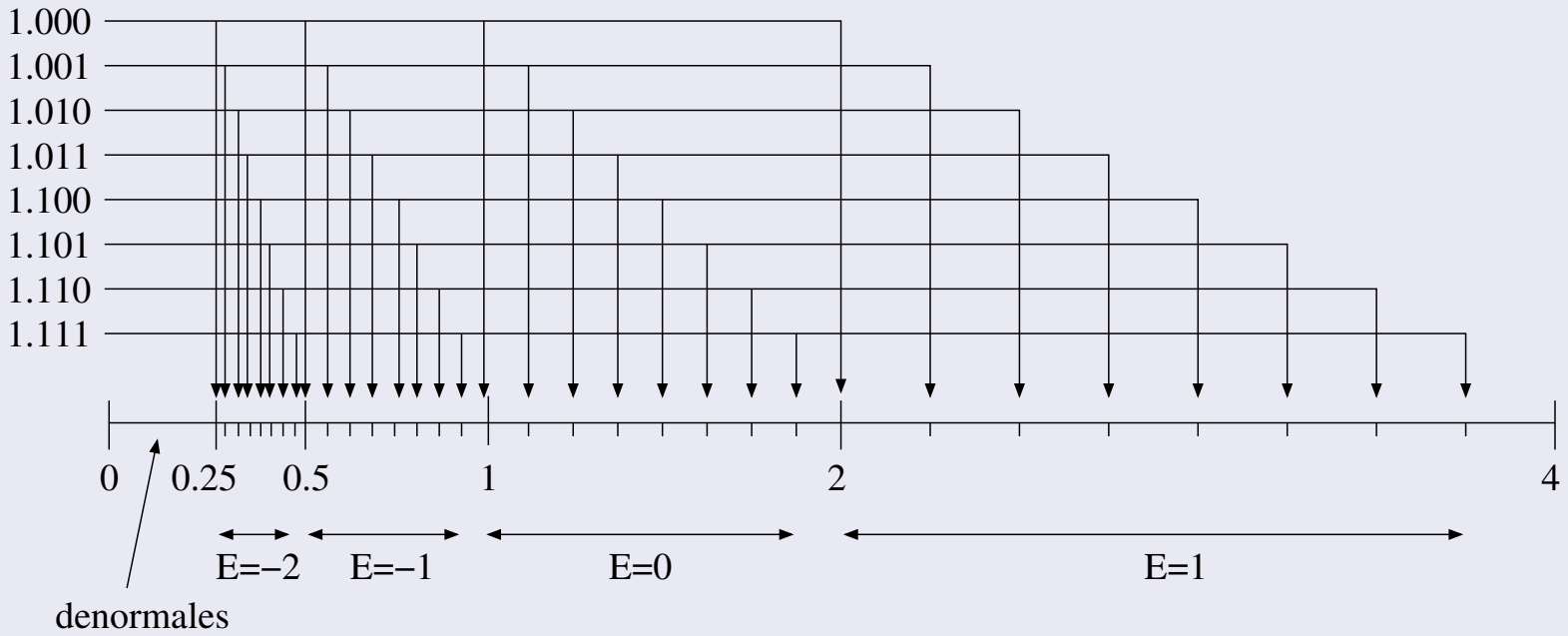




Error de redondeo (redondeo al más próximo – par)

Regiones en la representación punto flotante. Ejemplo

- 3 bits de significando, exponente $\in [-2, 1]$ (2 bits), región positiva
- Binades con diferente densidad de puntos



Error de redondeo (redondeo al más próximo – par)

El error depende la *binade*

- *ulp* (*unit in the last place*): diferencia entre dos valores consecutivos del significando
- El valor de la *ulp* depende de la binade
- Significando de n bits: $1\,ulp = 2^{-n} \times 2^{E_x}$
- Error de redondeo: $\frac{1}{2}\ ulp$

$$\begin{aligned} \varepsilon_{abs} &\leq 2^{-(n+1)} \times 2^{E_x} \\ \varepsilon_{rel} &\leq 2^{-(n+1)} \end{aligned}$$

Ejemplo: $n = 4$

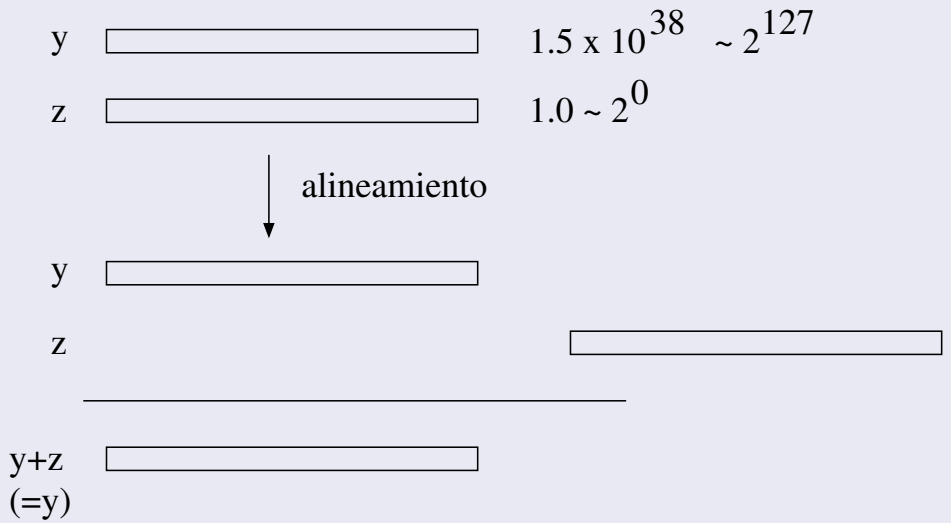
$$\begin{aligned} x &= 1{,}0001 \mid 1100 \times 2^{-7} \\ fp(x) &= 1{,}0010 \times 2^{-7} \\ \\ x - fp(x) &= 0{,}0000\ \ 0100 \times 2^{-7} \\ \varepsilon_{abs} &\leq 2^{-4} \times 2^{-7} \\ \varepsilon_{rel} &\leq 2^{-4} \end{aligned}$$

Problemas de la representación punto flotante

No siempre se verifica la propiedad asociativa

$$x = -1{,}5 \times 10^{38}, y = 1{,}5 \times 10^{38}, z = 1{,}0$$

$$\begin{aligned} x + (y + z) &= -1{,}5 \times 10^{38} + (1{,}5 \times 10^{38} + 1{,}0) \\ &= -1{,}5 \times 10^{38} + (1{,}5 \times 10^{38}) = 0{,}0 \end{aligned}$$



Algunos ejemplos: Argumentos y/o resultados próximos a 1

$acos(x), x \to 1, x = \cos(\theta)$ (SP FP)

$$\begin{aligned} \theta &= 1.00000000000000000000000 \times 2^{-5} \\ \cos(\theta) &= 1.11111111110000000000001 \times 2^{-1} \\ acos(\cos(\theta)) &= 1.11111111111110101010 11 \times 2^{-6} \end{aligned}$$

$$\begin{aligned} \theta &= 1.00000000000000000000000 \times 2^{-50} \\ \cos(\theta) &= 1.00000000000000000000000 \times 2^{0} \\ acos(\cos(\theta)) &= 0 \end{aligned}$$

Algunos ejemplos: Argumentos y/o resultados próximos a 1

$\ln(x), x \to 1, x = 1 + 2^{-23} + 2^{-26}$

$$\begin{aligned} \ln(x) &= 1.11111111111111111111111 \times 2^{-24} \quad \text{(SP FP)} \\ \ln(x) &= 1.00011111111111111111111 \times 2^{-23} \quad \text{(Maple)} \end{aligned}$$

Algunos ejemplos: Cancelaciones

$1 - \cos(x)$, $x \to 0$

$$
\begin{aligned}
x &= 1.00000000000000000000000 \times 2^{-5} \\
\cos(x) &= 1.11111111110000000000000 \times 2^{-1} \quad \text{(SP FP)} \\
1 - \cos(x) &= 1.11111111111100000000000 \times 2^{-12} \quad \text{(SP FP)}
\end{aligned}
$$

$$
1 - \cos(x) = 1.11111111111101010101010 \times 2^{-12} \quad \text{(Maple)}
$$

Algunos ejemplos: Cancelaciones

$(e^x - 1)/x$, $|x| << 1$

Algoritmo 1

if $x = 0$
    $f = 1$
else
    $f = (e^x - 1)/x$
end

Algoritmo 2

$y = e^x$
if $y = 1$
    $f = 1$
else
    $f = (y - 1)/\ln y$
end

Algunos ejemplos: Los errores pueden cancelarse

$(e^x - 1)/x$, $|x| << 1 \Rightarrow$ cancelación

x	Algoritmo 1	Algoritmo 2
10^{-5}	1,000005000006965	1,000005000016667
10^{-6}	1,000000499962184	1,000000500000167
10^{-7}	1,000000049433680	1,000000050000002
10^{-8}	0,999999993922529	1,000000005000000
10^{-9}	1,000000082740371	1,000000000500000
10^{-10}	1,000000082740371	1,000000000050000
10^{-11}	1,000000082740371	1,000000000005000
10^{-12}	1,000088900582341	1,000000000005000
10^{-13}	0,999200722162640	1,000000000000500
10^{-14}	0,999200722162640	1,000000000000050
10^{-15}	1,110223024625156	1,000000000000005
10^{-16}	0	1

Motivo: Incremento del error relativo en FP

Ejemplos

- Cancelaciones

$$R_{x-y} = \frac{x\ R_x + y\ R_y}{|x-y|} = \frac{x\ (R_x + R_y)}{|x-y|}$$

- Funciones con $xf'(x)/f(x)$ grande

$$R_{f(x)} = |x\ \frac{f'(x)}{f(x)}| \times R_x$$

- El error relativo de los operandos se amplifica

Estimación del error

Las operaciones FP son susceptibles de producir errores

- Operaciones FP: acumulación de errores de redondeo
- Resultados inexactos en algunos cálculos
  - Errores grandes solo para algunos valores de los datos, resultados exáctos en la mayoría de las ejecuciones
- Existen varias alternativas para determinar el error
  - Hardware
  - Software

Alternativas para la estimación del error

Métodos en hardware

- Aritmética de significancia
  - Especificación número dígitos correctos del resultado
  - Límite errores relativos sin signo: sobre-estimación del error
- Aritmética de intervalos
  - Acumulación de errores máximos: intervalos muy anchos.
  - Implementación costosa
- Aritmética FP con precisiones doble-doble y quad–doble.
  - Números como suma no evaluada de dos o cuatro DP-FP
  - Errores de redondeo se acumulan en parte menos significativa
  - Lento, varias operaciones para determinar los errores
- Estimación del error
  - Estimación del error concurrentemente con la ejecución del programa

Alternativas para la estimación del error

Métodos software: ejecución mucho más lenta y modificación del programa

- Aritmética de intervalos
  - Algoritmos especiales para evitar intervalos demasiado anchos
- Running error: límite del error durante la ejecución de un programa
  - Sobre-estimación del error
- Cálculo del error:
  - Lento. Basado en diferenciación parcial automática
- Aritmética estocástica: estimación de la exactitud
  - el programa se ejecuta varias veces con diferentes esquemas de error de redondeo

Estimación del error en computaciones punto–flotante

Estimación hardware

- Estimación del error almacenada con cada variable
- Estimación del error del resultado concurrentemente con la ejecución del programa
  - Errores de los datos de entrada
  - Errores de redondeo en el cáculo

Características

- Redondeo al más próximo: errores con signo
  - Los errores pueden cancelarse
- Estimación de la exactitud:
  - Idealmente: Grande si error grande, pequeña si error pequeño
  - Estimación no exacta: primera propiedad siempre, segunda *casi* siempre
  - Indicación cualitativa de errores grandes

Implementación

Sobrecoste adicional

- Implementación simple y poco efecto sobre la ejecución del programa
- Almacenaniento necesario para la estimación del error
  - Registros adicionales
  - Almacenanmiento en memoria adicional
- Aumento de los requisitos de ancho de banda procesador–memoria

Tipo de error

- El más apropiado parece el *error relativo*, pero
  - Difícil de implementar (debido a algunas anomalías)
- Proponemos *Scaled absolute error*
  - Valor próximo al del error relativo
  - Sin sus anomalías

Estimación del error: error absoluto escalado

Errores en cálculos FP

- $z = x\ op\ y$. En FP: $\widehat{z} = fp(\widehat{x}\ op\ \widehat{y})$ $\hat{x}$, $\hat{y}$, representación FP de x and y, $\hat{z}$ redondeo de $\widehat{x}\ op\ \widehat{y}$
- Error: propagado (P_z) + generado (redondeo, G_z).

	Error ($P_z + G_z$)	P_z	G_z
relative	$(z - \widehat{z})/z$	$(z - \widehat{x}\ op\ \widehat{y})/z$	$(\widehat{x}\ op\ \widehat{y} - \widehat{z})/z$
absolute	$z - \widehat{z}$	$z - \widehat{x}\ op\ \widehat{y}$	$\widehat{x}\ op\ \widehat{y} - \widehat{z}$
scaled abs.	$(z - \widehat{z}) \times 2^{-e_{\hat{z}}}$	$(z - \widehat{x}\ op\ \widehat{y}) \times 2^{-e_{\hat{z}}}$	$(\widehat{x}\ op\ \widehat{y} - \widehat{z}) \times 2^{-e_{\hat{z}}}$

- Error relativo (r_z): anomalías
  - $z = 0 \Rightarrow rel\ error = \infty$,
  - $\widehat{z} = 0 \Rightarrow rel\ error = 1$, independientemente de z
- Error absoluto (a_z): No refleja la exactitud de la variable
- Error absoluto escalado (b_z): similar al error relativo cuando éste es pequeño

Estimación del error: error absoluto escalado

	Scaled absolute error ($P_z + G_z$)
Addition	$b_z = (b_x \times 2^{e_{\hat{x}}} + b_y \times 2^{e_{\hat{y}}})2^{-e_{\hat{z}}} + d \times 2^{-e_{\hat{z}}}$
Multiplication	$b_z = 2^{-ovf}(m_{\hat{x}} \times b_y + m_{\hat{y}} \times b_x) + b_x b_y + d \times 2^{-e_{\hat{z}}}$
Division	$b_z = \frac{m_{\hat{y}} \times b_x - m_{\hat{x}} \times b_y}{m_{\hat{y}} \times (m_{\hat{y}} + b_y)} + d \times 2^{-e_{\hat{z}}}$

d es el error de redondeo absoluto (error generado), $d = \widehat{x}\ op\ \widehat{y} - \widehat{z}$

Otras operaciones: sqrt, exp, ln, ...

- Operandos pequeños: derivada $a_z = f'(\widehat{x}) * a_x$
- Operandos grandes: definición $a_z = f(\widehat{x} + a_x) - f(\widehat{x})$

Ejemplo: Relación de áreas de dos triángulos

Cálculo FP $r = \frac{area\ triangle(x,y,z)}{area\ triangle(x,y,2z)}$

- Ecuación

$$r = \sqrt{\frac{(x+y+z)(z-(x-y))(z+(x-y))(x+(y-z))}{(x+y+2z)(2z-(x-y))(2z+(x-y))(x+(y-2z))}}$$

- Los errores de redondeo degradan la exactitud.
- Ejemplo: $x = 1{,}234568 \times 10^6$, $y = 1{,}234567 \times 10^6$, $z = 1{,}00000000023$
  - resultado exacto: $r = 1{,}2382783744254 \times 10^{-5}$.
  - SP FP: $r = 0$
  - DP FP: $r = 1{,}238278425653352 \times 10^{-5}$.
    solo 7 dígitos decimales correctos (23 bits)
    $rel.\ error \approx 2^{-25}$.

Ejemplo: Relación de áreas de dos triángulos

	error abs.	error rel.
SP FP	$1{,}1001 \times 2^{-17}$	1
DP FP	$-1{,}0010 \times 2^{-41}$	$-1{,}0110 \times 2^{-25}$

(a) Resultado con Maple

	$N = 10$	$N = 20$	$N = 23$ (*for SP*) or 52 (*for DP*)
SP FP	$1{,}1001 \times 2^{238}$ *	$1{,}1001 \times 2^{238}$ *	$1{,}1001 \times 2^{238}$ *
DP FP	$-1{,}0010 \times 2^{-24}$	$-1{,}0010 \times 2^{-24}$	$-1{,}0010 \times 2^{-24}$

(b) Estimación

	intervalo escalado	running-error escalado
SP FP	$1{,}0010 \times 2^{243}$ *	$1{,}0111 \times 2^{253}$ *
DP FP	$1{,}1000 \times 2^{-21}$	$1{,}1110 \times 2^{-1}$

(c) Aritmética de intervalos y running-error

Exponencial ($f(x) = (e^x - 1)/x$)

	error abs.	error rel.
Alg. 1	$1{,}0100 \times 2^{-17}$	$1{,}0100 \times 2^{-17}$
Alg. 2	$1{,}1001 \times 2^{-55}$	$1{,}1001 \times 2^{-55}$

(a) Resultado con Maple

	$N = 10$	$N = 20$	$N = 52$
Alg. 1	$1{,}0100 \times 2^{-17}$	$1{,}0100 \times 2^{-17}$	$1{,}0100 \times 2^{-16}$
Alg. 2	$1{,}0101 \times 2^{-55}$	$1{,}0101 \times 2^{-55}$	$1{,}1001 \times 2^{-55}$

(b) Estimación

	intervalo escalado	running–error escalado
Alg. 1	$1{,}1001 \times 2^{-15}$	$1{,}1001 \times 2^{-15}$
Alg. 2	$1{,}1001 \times 2^{-14}$	$1{,}0011 \times 2^{-15}$

(c)Aritmética de intervalos y running-error

Ejemplo: Eliminación Gaussiana (GE)

Resolución FP de un sistema lineal de ecuaciones, $A \times x = b$

- Método de Eliminación Gaussiana con pivoting parcial de columnas
- Resultados inexactos debido a la acumulación de errores de redondeo
- Matriz 10×10 y $b = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$

  Vector resultado exacto $x = (0, 1, -2, 3, -4, 5, -6, 7, -8, 9)$

  Resultado DP FP:
  $(9{,}8407492921 \times 10^{-9}, 0{,}9999999055, -1{,}9999994966,$
  $2{,}9999980387, -3{,}9999937636, 4{,}9999828578,$
  $-5{,}9999578189, 6{,}9999049037, -7{,}9998003422,$
  $8{,}9996049158)$

Ejemplo: Eliminación Gaussiana (GE)

	Res. exact	Maple error abs.	Maple error rel.	Error absoluto escalado $N = 10$	$N = 20$	$N = 52$
x_0	0	-2^{-27}	∞	-2^{9}	-2^{0}	-2^{0}
x_1	1	2^{-24}	2^{-24}	-2^{-17}	2^{-23}	2^{-23}
x_2	-2	-2^{-21}	2^{-22}	2^{-19}	-2^{-21}	-2^{-21}
x_3	3	2^{-19}	2^{-21}	2^{-20}	2^{-20}	2^{-20}
x_4	-4	-2^{-18}	2^{-20}	-2^{-19}	-2^{-19}	-2^{-19}
x_5	5	2^{-16}	2^{-19}	2^{-18}	2^{-18}	2^{-18}
x_6	-6	-2^{-15}	2^{-18}	-2^{-17}	-2^{-17}	-2^{-17}
x_7	7	2^{-14}	2^{-17}	2^{-16}	2^{-16}	2^{-16}
x_8	-8	-2^{-13}	2^{-16}	-2^{-15}	-2^{-15}	-2^{-15}
x_9	9	2^{-12}	2^{-15}	2^{-15}	2^{-15}	2^{-15}

Aritmética de intervalos: El intervalo del divisor incluye el 0
Running error: Error infinito

Índice

Implementación de funciones de dos variables

Funciones de dos variables: X^p, $X^{1/q}$, $log_a X$

- Argumentos punto fijo y/o punto flotante
- Incluyen un gran número de funciones útiles en computación científica
  - $X^{1/2}$, $X^{-1/2}$, $X^{1/3}$, $X^{-1/3}$, X^2, X^{-2}, X^{-1}, $log_2 X$, $log_{10} X$, lnX, 2^X, e^X
- Pueden ser parte de sistemas heterogéneos
  - Aceleradores multifunción
- Computadas con rutinas software
  - Lentas para aplicaciones numericamente intensivas y/o en tiempo real
- Los métodos habituales no son generales
  - Cálculo de la potencia o raíz con un exponente o base particular
  - Logaritmos con una base particular

Funciones elementales en el estándar IEEE 754-2008

Funciones correctamente redondeadas

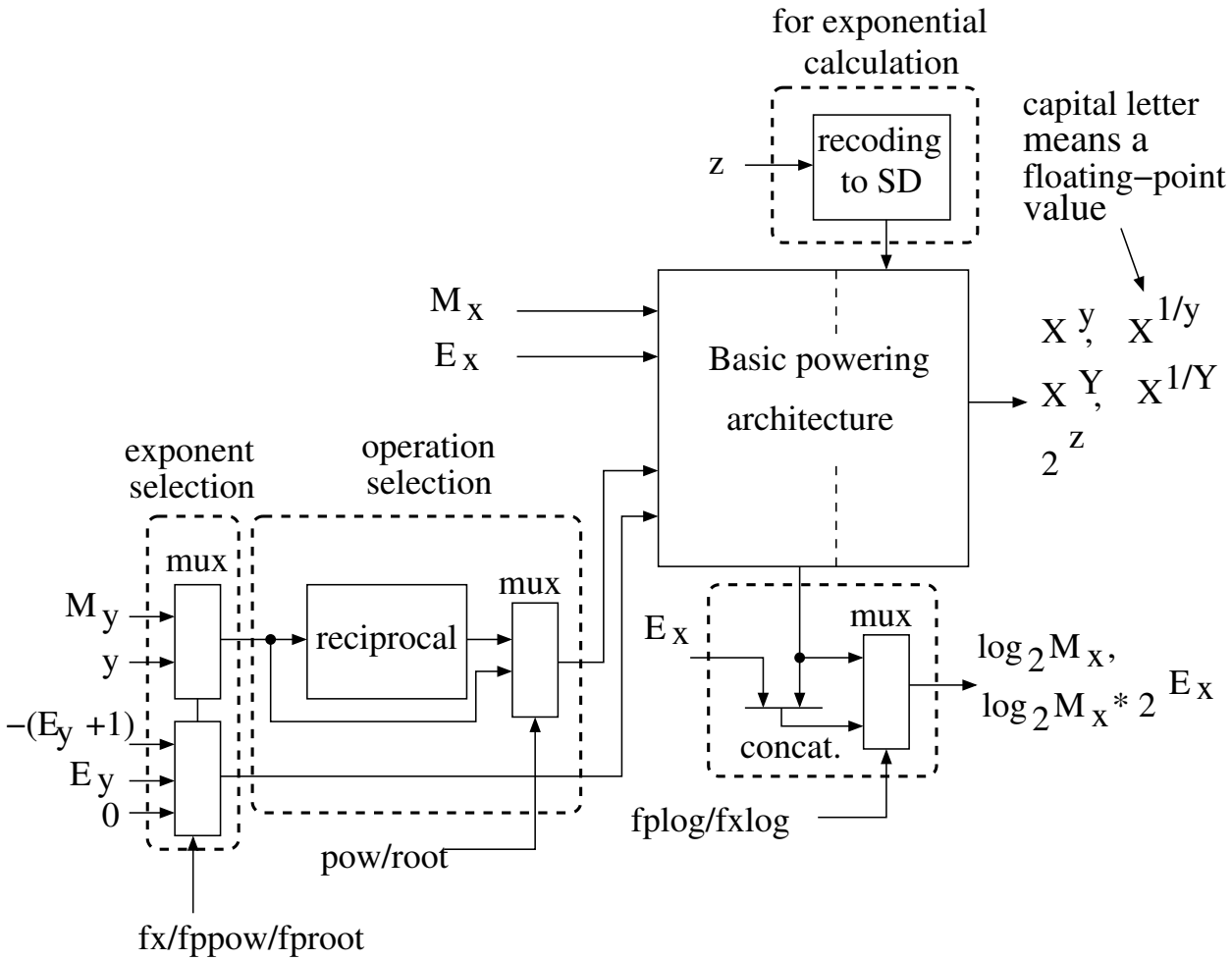
- Potencia
  - **pown(X, y)**, X^y con y entero y X un número punto flotante
  - **powr(X, Y)**, X^Y con Y y $X \geq 0$ números punto flotante
  - **pow(X, Y)**, X^Y con Y y X números punto flotante
- Raíz
  - **rootn(X, y)**, $X^{1/y}$ con y entero y X un número punto flotante

Implementación de funciones de dos variables

Alternativa

- Algoritmo iterativo y arquitectura para el cálculo de la función X^Y
  - Y punto fijo: $Y = p$, $Y = 1/q$, con p y q enteros
  - Y punto flotante: potencia si exponente positivo, raíz si negativo
- Secuencia de operaciones solapadas: logaritmo, multiplicación y exponencial
  - Cambios mínimos para permitir el cálculo independiente del logaritmo y la exponencial

Arquitectura top level



Exponente punto fijo

Algoritmo

- Basado en la identidad,

$$X^Y = 2^{\log_2(X^Y)} = 2^{Y \log_2(X)}$$

- Como X es punto flotante,

$$X^Y = 2^{Y \log_2(M_x \times 2^{E_x})} = 2^{Y \times (\log_2 M_x + E_x)}$$

- Operaciones: logaritmo, concatenación, multiplicación, exponencial
- Problema: el argumento de la exponencial está fuera del rango $(-1, 1)$
  - Separar parte entera y fraccional => exponente y mantisa del resultado punto flotante

$$X^Y = 2^{int(Y \times (\log_2 M_x + E_x))} \times 2^{frac(Y \times (\log_2 M_x + E_x))}$$

Exponente punto fijo

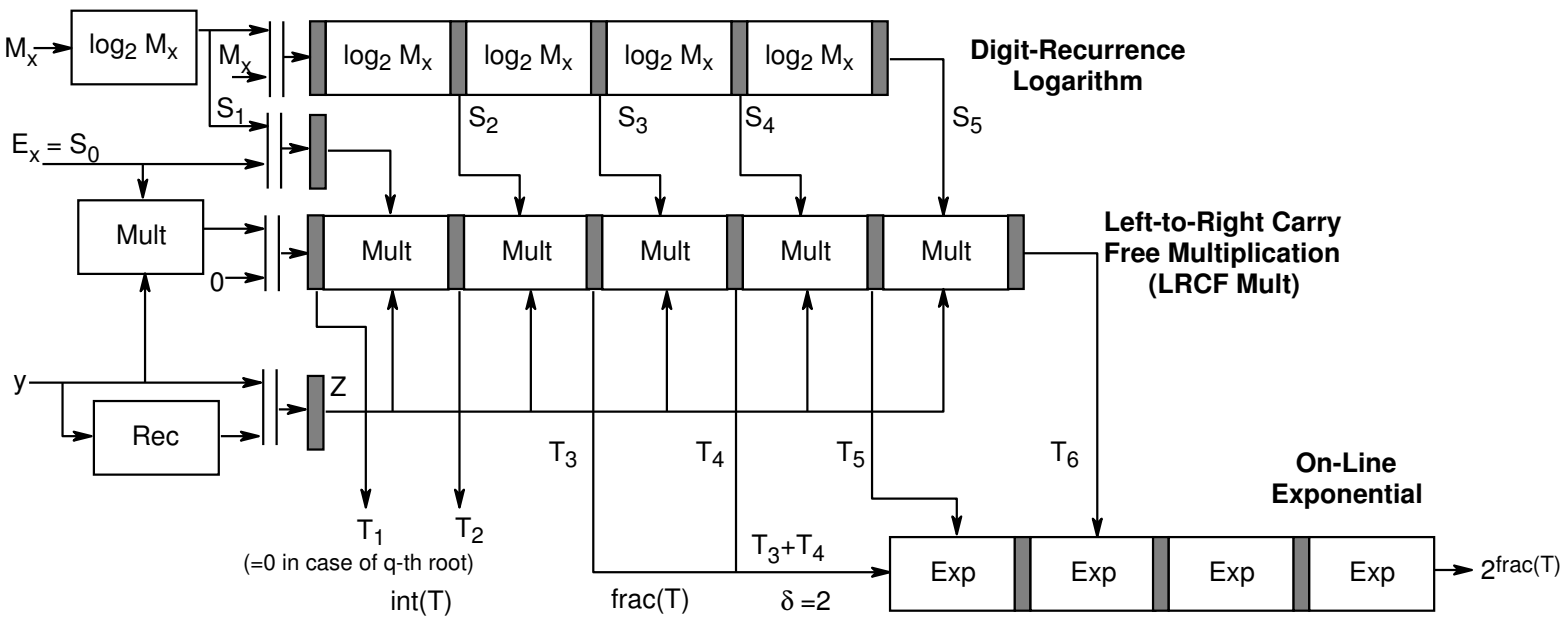
Implementación

$$X^Y = 2^{int(Y \times (\log_2 M_x + E_x))} \times 2^{frac(Y \times (\log_2 M_x + E_x))}$$

- El argumento de la exponencial en $(-1, 1)$
- El número de bits enteros es diferente para $Y = p$ o $Y = 1/q$
- Operaciones:
  - High–radix digit–recurrence logarithm
  - Left–to–right carry–free multiplication
  - On–line high–radix exponential
- Análisis detallado de la precisión y la exactitud (*accuracy*) para obtener el número de etapas (latencia) de cada operación

Exponente punto fijo

Ejemplo: Precisión simple (o *binary32*), radix $r = 128$ ($r = 2^7$), exponente de 8 bits



Exponente punto flotante

Operaciones (X^Y)

- La operación depende también del signo del exponente de Y

$$X^Y = \begin{cases} \text{potencia} & \text{si } E_y \geq 0 \ (|Y| \geq 1) \\ \text{raíz} & \text{si } E_y < 0 \ (|Y| < 1) \end{cases}$$

$$X^{1/Y} = \begin{cases} \text{raíz} & \text{si } E_y \geq 0 \ (|Y| \geq 1) \\ \text{potencia} & \text{si } E_y < 0 \ (|Y| < 1) \end{cases}$$

Algoritmo

$$|X|^Y = 2^{(-1)^{s_y} \times M_y \times (E_x + \log_2 M_x) \times 2^{E_y}}$$

$$|X|^{1/Y} = 2^{(-1)^{s_y} \times (2/M_y)(E_x \times \log_2 M_x) \times 2^{-(E_y+1)}}$$

- Se ha normalizado $1/M_y$ para usar el mismo multiplicador

Exponente punto flotante

Diferencias con exponente punto fijo

$$
\begin{aligned}
|X|^{Y} &= 2^{(-1)^{sy} \times M_y \times (E_x + \log_2 M_x) \times 2^{Ey}} \\
|X|^{1/Y} &= 2^{(-1)^{sy} \times (2/M_y)(E_x \times \log_2 M_x) \times 2^{-(Ey+1)}}
\end{aligned}
$$

- Menos dígitos en la parte entera
- Recíproco de alta precisión
- Desplazamiento 2^{E_y} o $2^{-(E_y+1)}$
  - Derecha /izquierda
  - Desplaza un operando redundante y MSDF
  - Limita el rango de valores útiles de Y

$$
\begin{aligned}
-(n_{E_x} + n_m) &\leq E_y \leq n + n_{E_x} - 2 \quad \textit{potencia} \\
-(n + n_{E_x} - 1) &\leq E_y \leq n_{E_x} + n_m + 1 \quad \textit{raíz}
\end{aligned}
$$

Exponente punto flotante

Ejemplo: Precisión simple (o *binary32*), radix $r = 128$ ($r = 2^7$)