UMASS

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Digital Computer Arithmetic
ECE 666

Part 9b
Evaluation of Elementary Functions - II

Israel Koren

Inverse Trigonometric Functions

- **Multiplicative normalization**

$$(i) \quad x_{i+1} = x_i \cdot b_i,$$
$$(ii) \quad y_{i+1} = y_i + g(b_i)$$

- **$b_i=(1+j\ s_i\ 2^{-i})$**

$$x_m = x_0 \cdot \prod_{l=0}^{m-1} b_l = x_0 \cdot \prod_{l=0}^{m-1} (1 + j s_l 2^{-l}) = x_0 \cdot K e^{j\theta},$$

- **where**

$$\theta = \sum_{l=0}^{m-1} \theta_l, \theta_l = \tan^{-1}(s_l 2^{-l}) \qquad K = \prod_{l=0}^{m-1} \sqrt{1 + (s_l 2^{-l})^2}.$$

- **If $g(b_i)=\theta_i=\arctan(s_i\ 2^{-i})$ & $y_0=0$**

$$y_m = \sum_{l=0}^{m-1} \theta_l = \theta$$

- **x_i complex variable = $U_i + jV_i$**
- **Recursive formulas for real and imaginary parts:**
- **(i) $U_{i+1} + j\ V_{i+1} = (U_i+jV_i)\ (1+j\ s_i\ 2^{-i})$**
- **(i) $U_{i+1} = U_i - s_i\ 2^{-i}\ V_i$**
- **(i)' $V_{i+1} = V_i + s_i\ 2^{-i}\ U_i$**

Final Values of Um and Vm

- Relationship between U_m, V_m and y_m :

$$x_{i+1} \cdot e^{-jy_{i+1}} = x_i \cdot b_i \cdot e^{-jy_i} e^{-jg(b_i)}$$

- Substituting b_i and $g(b_i)$

$$x_{i+1} \cdot e^{-jy_{i+1}} = x_i \cdot e^{-jy_i} \cdot \sqrt{1 + (s_i 2^{-i})^2}$$

$$K = \prod_{l=0}^{m-1} \sqrt{1 + (s_l 2^{-l})^2} \qquad x_m \cdot e^{-jy_m} = x_m \cdot e^{-j\theta} = x_0 \cdot K$$

- Replacing x_m and x_0 by real and imaginary parts
- $(U_m + jV_m)(\cos\theta - j\sin\theta) = K(U_0 + jV_0)$
- $U_m \cos\theta + V_m \sin\theta = K U_0$
- $-U_m \sin\theta + V_m \cos\theta = K V_0$
- Set $V_0=0$ & $U_0=1/K$: $U_m=\cos\theta$ and $V_m=\sin\theta$
- K constant if $s_i \in \{-1, 1\}$

Calculate φ =arccos C for given C

- Select s_i's - $U_m \to C \Rightarrow \theta \to \varphi$
- θ obtained from $y_m=\theta$, $\theta = \sum_{l=0}^{m-1} \tan^{-1}(s_l 2^{-l}) = \sum_{l=0}^{m-1} s_l \tan^{-1}(2^{-l})$
- Table of $\arctan(2^{-i})$ needed as for sine and cosine
- To calculate φ =arcsin D select s_i's so that $V_m \to D$ and $\theta \to \varphi$
- arctan C: $V_m \to 0$ - $\tan\theta = -V_0/U_0$ & $\theta = -\arctan(V_0/U_0)$
- Set $V_0=C$, $U_0=1$: arctan $C=-\theta$
- In summary

$$\begin{aligned} &(i) && U_{i+1} = U_i - s_i 2^{-i} \cdot V_i \; ; && U_0 = 1, \\ &(i)' && V_{i+1} = V_i + s_i 2^{-i} \cdot U_i \; ; && V_0 = C, \\ &(ii) && y_{i+1} = y_i - s_i \tan^{-1}(2^{-i}) \; ; && y_0 = 0, \end{aligned}$$

- s_i in {-1, 1} according to signs of V_i and U_i so that V_{i+1} is closer to 0

Example: arctan(1.0) in 10-bit Precision

i	U_i	V_i	y_i	s_i
1	1.0000000000	1.0000000000	0.0000000000	-1
2	1.1000000000	0.1000000000	0.0111011011	-1
3	1.1010000000	0.0010000000	0.1011010110	-1
4	1.1010010000	-0.0001010000	0.1101010101	1
5	1.1010010101	0.0000011001	0.1100010101	-1
6	1.1010010110	-0.0000011100	0.1100110101	1
7	1.1010010110	-0.0000000010	0.1100100101	1
8	1.1010010110	0.0000001011	0.1100011101	-1
9	1.1010010110	0.0000000100	0.1100100001	-1
10	1.1010010110	0.0000000001	0.1100100011	-1
11	1.1010010110	-0.0000000001	0.1100100100	

- Final result: $y_{11}=0.1100100100_2$
- Equal to "exact" result in 10-bit precision

Approximation Error: Additive normalization

- x_0 n-bit fraction - $\sum_{l=0}^{m-1} g(b_l)$ approaches x_0
- Error $\epsilon = x_0 - \sum_{l=0}^{m-1} g(b_l)$ - $|\varepsilon| \le 2^{-n}$
- Evaluate $y=F(x_0)$ by calculating $y = F(\sum_{l=0}^{m-1} g(b_l)) = F(x_0 - \epsilon)$
- Taylor series expansion:

$$y = F(x_0 - \epsilon) = F(x_0) - \epsilon \cdot \frac{dF}{dx}|_{x_0} + \frac{\epsilon^2}{2} \cdot \frac{d^2F}{dx^2}|_{x_0} + 0(\epsilon^3)$$

- ε is of order 2^{-n} - last two terms negligible
- Error δ in y of order $\varepsilon\, dF/dx|_{x_0}$
  - Magnitude of error depends on function $F(x_0)$
- Example - exponential function $F(x_0)=e^{x_0}$
  $dF/dx|_{x_0}=e^{x_0}$ - thus $|\delta| \le 2^{-n} e^{\ln 2} = 2^{-(n-1)}$ for $x_0 \le \ln 2$
  - Maximum error double size of error in argument
  - Can be reduced by increasing number of bits in x_i

Approx Error: Multiplicative Normalization

$x_0 \prod_{l=0}^{m-1} b_l \to 1$ **Error is** $\epsilon = 1 - x_0 \prod_{l=0}^{m-1} b_l$ or $x_0 \prod_{l=0}^{m-1} b_l = 1 - \epsilon$

- **Instead of $F(x_0)$ we calculate**

$$y = F\left(\frac{1}{\prod_{l=0}^{m-1} b_l}\right) = F\left(\frac{x_0}{1-\epsilon}\right) \approx F(x_0\left[1 + \epsilon - \epsilon^2 + \cdots\right]) \approx F(x_0 + x_0\epsilon)$$

- **Taylor series expansion**

$$F(x_0 + x_0\epsilon) = F(x_0) + x_0\epsilon \cdot \frac{dF}{dx}|_{x_0} + \frac{1}{2}(x_0\epsilon)^2 \cdot \frac{d^2F}{dx^2}|_{x_0} + 0(\epsilon^3)$$

- **Natural logarithm $F(x_0) = \ln x_0$**

$$\frac{dF}{dx}|_{x_0} = \frac{1}{x_0} \qquad \frac{d^2F}{dx^2}|_{x_0} = -\frac{1}{x_0^2}$$

- **$y = F(x_0) + \varepsilon + O(\varepsilon^2)$**
- **Error in y: $|\delta| \approx |\varepsilon| \le 2^{-n}$ - satisfactory precision**

Speed-up Techniques: Exponential Function

$$\ln(1 + s_i 2^{-i}) = s_i 2^{-i} - (s_i)^2 2^{-2i-1} + \frac{1}{3} s_i\, 2^{-3i} - \cdots$$

- **For $i > n/2$ - approximately $s_i\, 2^{-i}$**
  - Smaller ROM & Replace last $n/2$ steps by a single operation
- **Previously canceled at least first $(i-1)$ bits in x_i:**

$$x_i = 0.\underbrace{0\cdots00}_{i-1} z_i z_{i+1} \cdots$$

  - z_k $(k = i, i+1, \ldots)$ - single bit
- **To cancel remaining nonzero bits in x_i $(i \ge n/2)$ select $s_k = z_k$ $(k > i)$**
  - All remaining s_k's can be predicted ahead of time
- **In last steps calculate**

$$y_m = y_i \prod_{k=i}^{m-1} (1 + s_k 2^{-k}) \approx y_i \left(1 + s_i 2^{-i} + s_{i+1} 2^{-(i+1)} + \cdots\right)$$

- **Since $s_k = z_k$ - last term is $(1 + x_i)$ $\Rightarrow$ $y_m = y_i\,(1 + x_i)$**
  - Called Termination Algorithm

Termination Algorithm

- Required - fast multiplier
  - If not available - can still benefit - perform all additions of products in carry-save manner
- Also for $F(x_0)=\ln x_0$ $\quad 1 - x_i = 0.\underbrace{0\cdots 00}_{i-1} z_i z_{i+1}\cdots$
- x_i has the form:
- or $\quad x_i = 1 - z_i 2^{-i} - z_{i+1}2^{-(i+1)} - \cdots = 1 - \sum_{k=i}^{m-1} z_k 2^{-k}$
- Formula for x_{i+1}

$$x_{i+1} = x_i \cdot (1 + s_i 2^{-i}) = 1 + s_i 2^{-i} - z_i 2^{-i} - \cdots$$

- For $x_{i+1} \to 1$ select $s_k = z_k$ for $k \geq i$
- In parallel calculate $\quad y_m = y_i - \sum_{k=i}^{m-1} \ln(1 + s_k 2^{-k})$
- Based on Taylor series for $\ln(1+s_k 2^{-k})$ - terminal approximation for $i \geq n/2$: $\quad y_m \approx y_i - \sum_{k=i}^{m-1} s_k 2^{-k} = y_i - (1 - x_i)$

Accelerate Trigonometric Functions

- Predict s_i's based on $\quad \tan^{-1}(s_i 2^{-i}) = s_i 2^{-i} - s_i \frac{2^{-3i}}{3} + \cdots$
- To obtain constant K - s_i restricted to {-1,1}
- If $K = \prod_{l=0}^{m-1} \sqrt{1 + 2^{-2l}}$ replaced by $K = \prod_{l=0}^{n/2} \sqrt{1 + 2^{-2l}}$
- Deviation from exact value < 2^{-n}
- For $i > n/2$, may select s_k ($k \geq i$) from {0,1} and predict $s_k = z_k$, where: $\quad x_i = 0.\underbrace{0\cdots 00}_{i-1} z_i z_{i+1}\cdots$
  - Can perform additions for x_i, Z_i and W_i in carry-save manner
- Reexamining series expansion of arctan - may predict s_l's for $i \leq l \leq 3i$ at once
- Corresponding terms added in carry-save adder
- For $l < n/2$ - still use {-1,1}

Speeding-Up First steps

- $0 \le i \le n/2$
- Use radix > 2
- Example: $g(bi)=1+s_i\, r^i$ (r - power of 2, say $r=2^q$)
  $s_i \in \{-(r-1),-(r-2),\ldots,-1,0,1,\ldots,r-1\}$
- q bits of x in a single step - but increased complexity of step - s_i has larger range
- Some improvement in speed is possible
- Another method: allow s_i to assume values in {-1, 0,1} - modify selection rule so that probability of $s_i=0$ maximized
- Similar to variations of SRT

Other Techniques for Evaluating Elementary Functions

- (1) Use rational approximations (commonly used in software)
- When fast adder and multiplier available and high precision is required; e.g., extended double-precision IEEE floating-point
  - Hardware implementation of rational approximations can compete with continued summations and multiplications when execution time and chip area are considered
- (2) Use polynomial approximations with look-up table

Polynomial Approximations with look-up table

- Domain of argument x of f(x) divided into smaller intervals (usually of equal size) - $f(x_i)$ kept in look-up table
- Value of f(x) at x calculated based on $f(x_i)$, for x_i closest to x, and a polynomial approximation $p(x-x_i)$ for $f(x-x_i)$
- Distance $(x-x_i)$ is small - simple polynomial requiring less time than a rational approximation
- Overall algorithm has three steps:
  - (1) Find closest boundary point x_i and calculate "reduction transformation" - usually distance $d=x-x_i$
  - (2) Calculate p(d)
  - (3) Combine $f(x_i)$ with p(d) to calculate f(x)

Example - Calculating 2^x in [-1,1]

- 32 boundary points $x_i=i/32$, $i=0,1,\ldots,31$
  - 2^{x_i} precalculated and stored in look-up table
- Step 1: search for x_i such that $|x-m-x_i| \le 1/64$, where $m=-1,0,1$; calculate $d=(x-m-x_i)\ln 2$
- d satisfies $e^d = 2^{x-m-x_i}$, and $|d| \le (\ln 2)/64$
- Step 2: approximation for e^d-1 using a polynomial

$$p(d) = d + p_2 d^2 + p_3 d^3 + \cdots p_k d^k$$

  - p_2, p_3 … p_k precalculated coefficients of polynomial approximation of e^d-1 on $[-(\ln 2)/64, (\ln 2)/64]$
- Step 3: reconstruct 2^x using

$$2^x = 2^{m+x_i} \cdot e^d = 2^m \left(2^{x_i} + 2^{x_i} \cdot (e^d - 1)\right) \approx 2^m \left(2^{x_i} + 2^{x_i} \cdot p(d)\right)$$

- Error bounded by 0.556 ulp (ulp=2^{-52})