Speeding Up the Division Process

♦ Unlike multiplication - steps of division are serial
♦ Division step consists of selecting quotient digit and calculating new partial remainder
♦ Two ways to speed up division:
  ♦ (1) Overlapping full-precision calculation of partial remainder in step i with selecting quotient digit in step i+1
    * Possible since not all bits of new partial remainder must be known to select next quotient digit
  ♦ (2) Replacing carry-propagate add/subtract operation for calculating new partial remainder by carry-save operation
First Speed-Up Method

♦ Truncated approximation of new partial remainder calculated in parallel to full-precision calculation of partial remainder - can be done at high speed
  * Quotient digit determined before current step completed

♦ Instead of: (1) Calculate $r_{i-1}$ (with carry propagation) in step $i-1$ & (2) input $N_p$ most significant bits to PLA to determine $q_i$

♦ Use a small adder with inputs: most significant bits of previous partial remainder, $\beta r_{i-2}$, and most significant bits of corresponding multiple of divisor, $\bar{q}_{i-1}D$

♦ Approximate Partial Remainder (APR) adder

Approximate Partial Remainder Adder

♦ Produces approximation of $N_p$ most significant bits of partial remainder, $\bar{r}_{i-1}$, before full-precision add/sub operation ($r_{i-1}=\beta r_{i-2}-q_{i-1}D$) is completed

♦ Allows to perform look-ahead selection of $q_i$ in parallel with calculation of $r_{i-1}$

♦ Size of APR adder determined so that sufficiently accurate $N_p$ bits generated

♦ Uncertainty in result of this adder larger than of truncated previous partial remainder $\beta r_{i-2}$ - may need additional inputs for quotient look-up table

♦ 8-bit APR adder sufficient to generate necessary inputs to PLA for $\beta=4$, $\alpha=2$
Example: 
**P-D Plot**

(β=4, α=2) 

D∈[1,2) 

* Horizontal lines determined to reduce complexity of PLA 
* Only 3 divisor bits needed as PLA inputs - most significant bit of divisor always 1 
* For partial remainder - 5 bits (including sign bit) sufficient in most cases 

--- 

Positive 

Remainder 

♦ 3 cases (a,b,c) when additional bit required 
♦ Case a: D=1.001, P=1.1 - single fractional bit of P insufficient 
♦ Divisor can assume any value from 1.001 to 1.010 
♦ Partial remainder can have a value from 1.1 to 10.0 - range for P/D from 1.1/1.010=1.2 to 10.0/1.001=1.77 
♦ First requires q=1, while second requires q=2 
♦ Add 2nd fractional bit to P 
  * Can select q=1 for P=1.10 and q=2 for P=1.11
Cases b and c

- 8-bit APR adder may introduce additional error - increasing P/D range
- Case b: D=1.100, P=10.0
- No APR adder: P/D range from 10.0/1.101=1.23 to 10.1/1.100=1.66 - q=1 can be selected
- 8-bit APR adder: introduces error of up to $2^{-6}$ in $r_{i-1}$ - increases to $2^{-4}$ after multiplying by 4
- This additional error increases maximum value of P/D from 1.66 to 1.7, requiring q=2
- An extra fractional bit of P solves this problem

Negative Partial Remainder

- Represented in two's complement
- 6 cases - 1 or even 2 additional output bits of APR adder required to guarantee correct selection of quotient digit
Second Speed-Up Method

In first method - time needed for division step determined by add/subtract for remainder - quotient digit selected in previous step.

Second method avoids time-consuming carry propagation when calculating remainder.

Truncated remainder sufficient for selecting next quotient digit - no need to complete calculation of remainder at any intermediate step.

Replace carry-propagate adder by carry-save adder.

Partial remainder in a redundant form using 2 sequences of intermediate sum and carry bits (stored in 2 registers).

Only most significant sum and carry bits must be assimilated using APR adder to generate approximate remainder and allow selection of quotient digit.

SRT Divider with Redundant Remainder

Most time consuming - calculate approximate remainder and select quotient digit.

In each division step: carry-save adder calculates remainder, APR adder accepts most significant sum and carry bits of remainder & generates required inputs to quotient selection PLA.

As in first method - number of PLA inputs and its entries need to be calculated, taking into account uncertainty in sum and carry bits representing truncated remainder.
Example

♦ An algorithm for high-speed division with $\beta=4$, $\alpha=2$, $D \in [1,2)$ has been implemented
♦ Partial remainder calculated in carry-save manner resulting in a somewhat more complex design
♦ 8-bit APR adder used to generate most significant remainder bits - inputs to quotient selection PLA
♦ Inputs to APR adder: 8 most significant sum bits and carry bits in redundant representation of remainder
♦ Outputs of APR adder converted to a sign-magnitude representation - only 4 bits of approximate partial remainder needed in most cases
♦ Additional bit required only in 4 cases - simple PLA

Further Speed-up of SRT Division

♦ Achieved by increasing radix $\beta$ to 8 or higher
♦ Reduces number of steps to $\lceil n/3 \rceil$ or lower
♦ Several radix-8 SRT dividers have been implemented
♦ Main disadvantage: high complexity of PLA - most time-consuming unit of divider
♦ Avoiding complex PLA - implementing radix-$2^m$ SRT unit as a set of $m$ overlapping radix-2 SRT stages
♦ Radix-2 SRT requires very simple quotient selection logic - $q_i \in \{-1,0,1\}$ solely determined by remainder - independent of divisor
♦ Must overlap quotient selections for $m$ bits - all $m$ quotient bits generated in one step
Two Overlapping Radix-2 SRT Stages

- Implementing radix-4 division
- All 3 possible values of $q_{i+1}$ generated using 3 Qsel units - correspond to 3 possible intermediate remainders: $2r_{i-1}-D$, $2r_{i-1}$, $2r_{i-1}+D$
- Only most significant bits of 3 remainders generated
- Overall delay: CSA, Qsel, two Mux and final CSA
- May be faster than radix-4 stage - higher complexity of radix-4 PLA

Extending to Radix-8 SRT division

- More complex quotient selection circuit - 3 quotient digits ($q_i, q_{i+1}, q_{i+2}$) generated in parallel
- For $q_{i+1}$: calculate speculative remainders $2r_{i-1}-D, 2r_{i-1}, 2r_{i-1}+D$
- For $q_{i+2}$: calculate $4r_{i-1}-3D, 4r_{i-1}-2D, 4r_{i-1}-D, 4r_{i-1}, 4r_{i-1}+D, 4r_{i-1}+2D, 4r_{i-1}+3D$
  - Only most significant bits of these 7 remainders
- 7 Qsel units required with multiplexors (controlled by $q_i$ and $q_{i+1}$) to select correct value of $q_{i+2}$
- Extend to 4 overlapping radix-2 stages for radix-16 divider: number of Qsel units increases from 11 to 26
- Another alternative for a radix-16 divider: 2 overlapping radix-4 SRT stages
Array Dividers

♦ All division algorithms can be implemented using arrays: n rows with n cells per row for radix-2 division

♦ Restoring: each row forms difference between previous remainder and divisor and generates quotient bit according to sign of difference

♦ No need to restore remainder if quotient bit=0 - either previous remainder or difference transferred to next row

♦ If ripple-carry in every row - n steps to propagate carry in a single row - total execution time of order n²

♦ Nonrestoring array: same speed as restoring; only advantage - handle negative operands in a simple way

♦ Final remainder may be incorrect - sign opposite to dividend

A Non-restoring Array Divider

If T=0 (1) - addition (subtraction) performed:
Subtract - add two's complement of divisor (assumed positive) & forced carry=T
Faster Array Dividers

♦ Previous array dividers - add/subtract with carry-propagation performed in each row

♦ Nonrestoring:
  * Only sign bit of partial remainder needed to select quotient bit
  * Can be generated by using fast carry-look-ahead circuit
  * Other bits of remainder use carry-save adder

♦ Each cell generates $P_i$ and $G_i$ besides sum and carry

♦ $P_i$ and $G_i$ of all cells in a row connected to a carry-look-ahead circuit to generate quotient bit

♦ Execution time - of order $n \log n$ vs. $n^2$

♦ Similarly, high-radix division array with carry-save addition

♦ Small carry-look-ahead adder used to determine most significant bits of remainder to select quotient digit

Fast Square Root Extraction

♦ Similar to division - small extensions to division unit

♦ Nonrestoring algorithm -
  $q_i = 1, \bar{1}$; $Q = 0.q_1, \ldots, q_m$ - calculated square root

♦ Advantages of adding 0 to the digit set of $q_i$:
  * Shift-only operation required when $q_i = 0$
  * Overlap between regions of $r_i$ where $q_i = 1$ or $q_i = 0$ are selected leads to reduced precision inspection of remainder

♦ Nonrestoring - must identify $r_i \geq 0$ to correctly set $q_i$
  * Requires precise determination of sign bit of $r_i$

♦ If $q_i = 0$ allowed - lower precision comparison sufficient, enables use of carry-save adders for $r_i$

♦ Remainder: two sequences - partial sum and carries

♦ Only a few high-order bits of these two sequences must be examined to select $q_i$
Selection of \( q_i = 0 \)

- Square root \( Q \) restricted to normalized fraction - 
  \( 1/2 \leq Q < 1 \), with \( q_1 = 1 \)
- Radicand: \( 1/4 \leq X < 1 \)
- Remainder \( r_{i-1} \) (\( i \geq 2 \)): \(-2(Q_{i-1} - 2^{-i}) \leq r_{i-1} \leq 2(Q_{i-1} + 2^{-i})\) 
  (\( Q_{i-1} \) is partially calculated root at step \( i-1 \) - 
  \( Q_{i-1} = 0.q_1q_2...q_{i-1} \))
- In step \( i \geq 2 \), select \( q_i = 0 \) whenever \( r_{i-1} \) is in range 
  \([- (Q_{i-1} - 2^{-i-1}), (Q_{i-1} + 2^{-i-1})]\)
- If \( q_i = 0 \), \( r_i = 2r_{i-1} \)

Selection Rule for \( q_i \)

\[
q_i = \begin{cases} 
1 & \text{if } r_{i-1} \geq (Q_{i-1} + 2^{-i-1}) \\
0 & \text{if } -(Q_{i-1} - 2^{-i-1}) \leq r_{i-1} \leq (Q_{i-1} + 2^{-i-1}) \\
\bar{1} & \text{if } r_{i-1} \leq -(Q_{i-1} - 2^{-i-1}). 
\end{cases}
\]

- Since \( Q_{i-1} + 2^{-i-1} \) and \( Q_{i-1} - 2^{-i-1} \) are in \([1/2, 1]\), 
  a selection rule which avoids a high-precision comparison is 

\[
q_i = \begin{cases} 
1 & \text{if } 1/2 \leq 2r_{i-1} \leq 2 \\
0 & \text{if } -1/2 \leq 2r_{i-1} < 1/2 \\
\bar{1} & \text{if } -2 \leq 2r_{i-1} < -1/2. 
\end{cases}
\]

- Similar to the SRT rule
Example

\[ X = 0.0111101_2 = 61/128 \]

\[ Q = 0.1011001_2 = 89/128 \]

\[ 2^{-7} r_7 = -113/2^{14} = X - Q^2 = (7808-7921)/2^{14} \]

High-Radix Square Root Extraction

\[ \beta - \text{radix: digit set for } q_i \rightarrow \{\overline{\alpha, \alpha+1, \ldots, 0, 1, \ldots, \alpha}\} \]

\[ \text{Computing new remainder: } r_i = \beta r_{i-1} - q_i (2Q_{i-1} + q_i \beta^{-i}) \]

\[ \text{Example: } \quad \beta = 4, \text{ digit set } \{\overline{2, 1, 0, 1, 2}\} \text{ preferable - eliminates need to generate multiple } 3Q_{i-1} \]

\[ \text{Generation of } q_i (2Q_{i-1} + q_i 4^{-i}) \text{ makes square root extraction somewhat more complex than division} \]

\[ \text{Calculation can be simplified} \]

\[ \text{For } q_i = 1, 2 - \text{subtract } Q001_2 \text{ & } Q010_2, \text{ respectively} \]

\[ \text{For } q_i = \overline{1} - \text{add } Q00\overline{1}_2 \text{ - same as } (Q-1)111_2 \]

\[ \text{For } q_i = \overline{2} - \text{add } Q0\overline{0}0_2 \text{ - same as } (Q-1)110_2 \]

\[ \text{Two registers with } Q \text{ and } Q-1, \text{ updated at every step, simplify execution of square root algorithm} \]
Selecting Quotient Digit

- Only low-precision comparison of remainder needed to select quotient digit -
  - Perform add/subtract in carry-save - small carry-propagate adder to calculate most significant bits of \( r_i \)
  - To provide inputs to a PLA for selecting square root digit \( q_i \)
  - Other inputs to PLA: most significant bits of root multiple

- Several rules for selecting \( q_i \) have been proposed
  - Intervals of remainder determine size of carry-propagate adder (between 7 and 9 bits for base-4 algorithm with digits \( 2,1,0,1,2 \)) and exact PLA entries

- Selected \( q_i \) depends on truncated remainder and truncated root multiple

- In first step no estimated root available

- Separate PLA for predicting first few bits of root

Example - Square Root Using Radix-4 Divider

- \( P-D \) plot for divide also used for square root

- Same PLA (with 19 product terms) used for predicting next quotient digit and root digit

- A separate PLA (with 28 product terms) added
  - Inputs - 6 most significant bits of significand and least significant bit of exponent (indicates whether exponent odd or even)
  - Output - 5 most significant bits of root

\[
\sqrt{1.f \cdot 2^{E-1023}} = \begin{cases} 
0.1f \cdot 2^{(E+1)/2-1023} & \text{if } E \text{ is odd} \\
0.01f \cdot 2^{E/2+1-1023} & \text{if } E \text{ is even}
\end{cases}
\]

- Radicand in \([1/4,1] \Rightarrow \) square root in \([1/2,1]\)