CS740 Computer Systems
Computer Arithmetic

Yirng-An Chen
Oct.12, 1995

Integer representations

Let bits $x_{n-1},....,x_2,x_1,x_0$ represent integers:

Unsigned

- $X = 2^{n-1}x_{n-1}+....+2^2x_2+2x_1+x_0$

Sign magnitude

- $X = (-1)^{Xn-1}(2^{n-2}x_{n-2}+....+2^2x_2+2x_1+x_0)$
- Dual representations for 0.

1's complement

- $X = (-2^{n-1}+1)x_{n-1}+....+2^2x_2+2x_1+x_0$
- Dual representations for 0.

2's complement

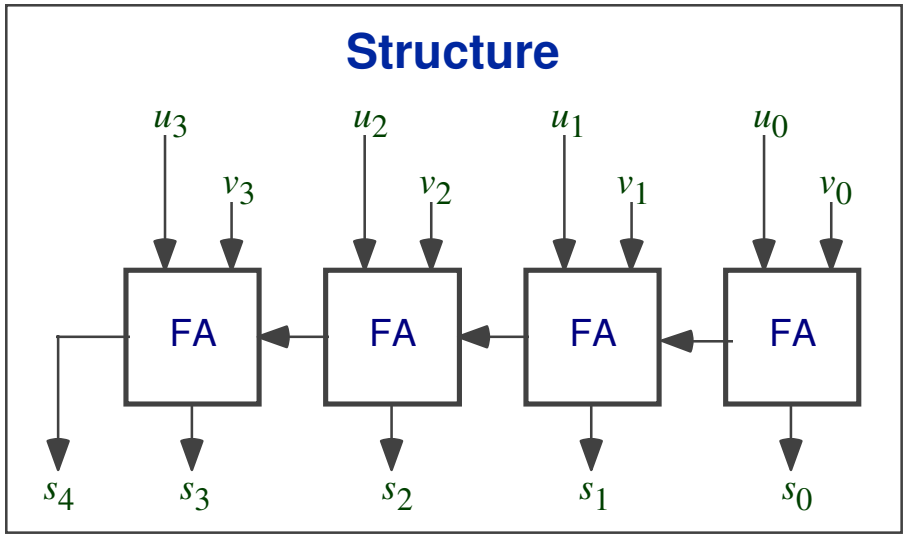
- $X = -2^{n-1}x_{n-1}+....+2^2x_2+2x_1+x_0$

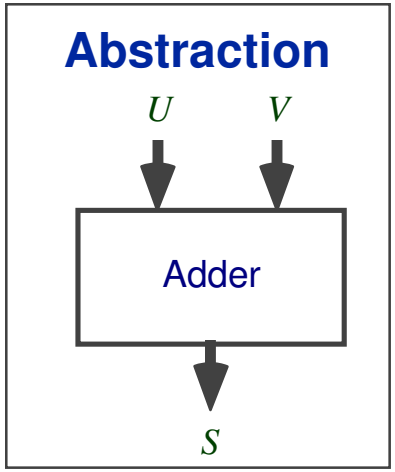
Biased by K

- $X = 2^{n-1}x_{n-1}+....+2^2x_2+2x_1+x_0 -K$

Bits	Unsign	Sign	1's	2's	Biase
000	0	0	0	0	-4
001	1	1	1	1	-3
010	2	2	2	2	-2
011	3	3	3	3	-1
100	4	-0	-3	-4	0
101	5	-1	-2	-3	1
110	6	-2	-1	-2	2
111	7	-3	-0	-1	3

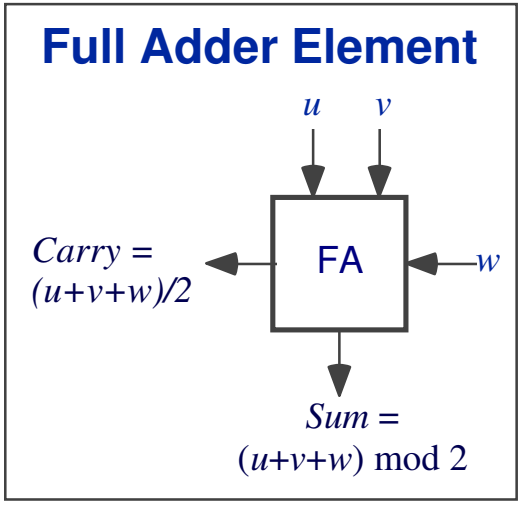
Ripple Carry Adder Design





Shortcoming

- Must propagate carries across all cells
- Can improve with more costly hardware (Carry lookahead adder, Carry skip adder, Carry select adder)



Simple Divider

Restoring

- Shift <P,A> left and Subtract B
- If result is negative, set quotient to 0 and restore old value
- otherwise, set quotient to 1

Nonrestoring

- Allow P be negative.
- If P is negative, shift <P,A> and add B, else shift <P,A> and subtract B
- If P is negative, quotient=0 else 1.

Sequential Multiplier

Basic idea

- Addition and Shift
- O(n) additions and shifts

See Figure A.2(a)

Booth Recoding(Radix-2)

Let X be multiplicand and Y ($y_{n-1}...y_1y_0$) be the multiplier

Basic idea

- At each iteration add $K=(y_{i-1} - y_i) *X*2^i$
- Shift one bit per iteration

Correctness

$$\sum_{i=0}^{n-1} K = X(-2^{n-1}y_{n-1}+2^{n-2}y_{n-2}+...+2y_1+y_0) +Xy_{-1}$$

Example −6*−5

```
P      A
0000 1010 A=-6(1010),B=-5
0000 0101 add 0 and shift
0101      add -B and shift
0010 1010
1011      add B and shift
1110 1101 (sign extended)
0101      add -B and shift
0001 1110 result=30.
```

Booth Recoding(Radix-4)

Ideas

- At each iteration, it consumes 2 bits of multiplier.
- Allow adding 2,1,0,-1, or -2 times multiplicand.

Improvements

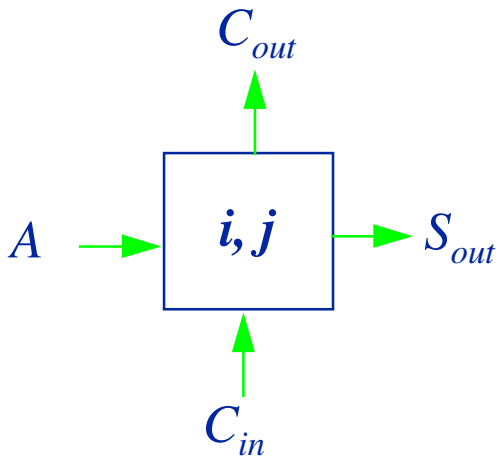
- Reduce the number of partial product by 2

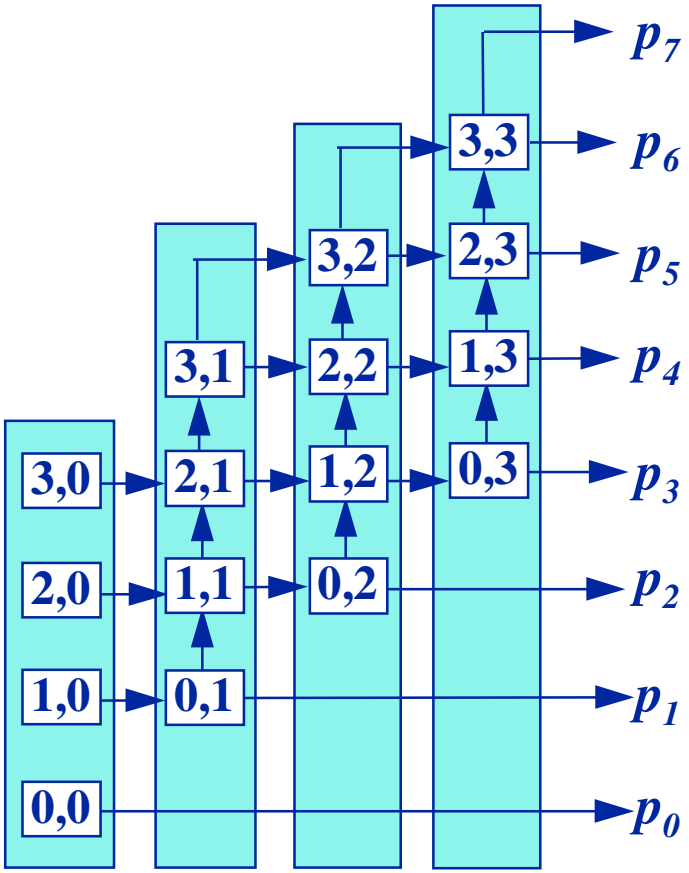
Bit 2*i+1	Bit 2*i	Bit 2*i-1	Multiple
0	0	0	0
0	0	1	+X
0	1	0	+X
0	1	1	+2X
1	0	0	-2X
1	0	1	-X
1	1	0	-X
1	1	1	0

Array Multiplier

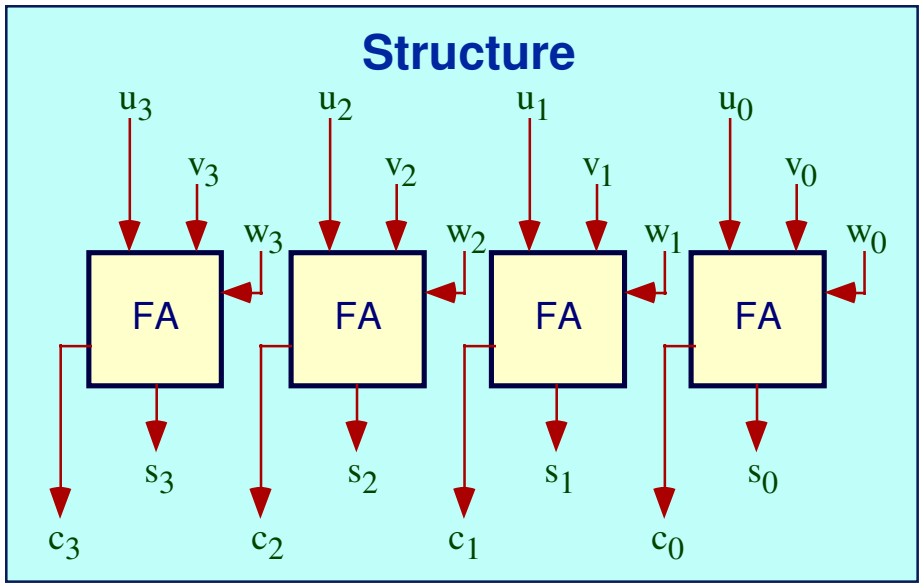
Circuit Structure

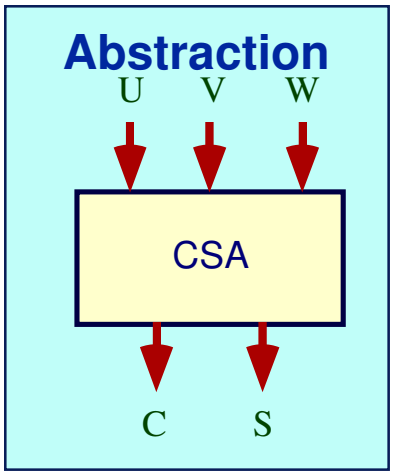
- **4-bit word size**
- **array of cells**
  - Generate partial product $x_i * y_i$
  - Sum with other products of weight *i+j*





Carry Save Adders





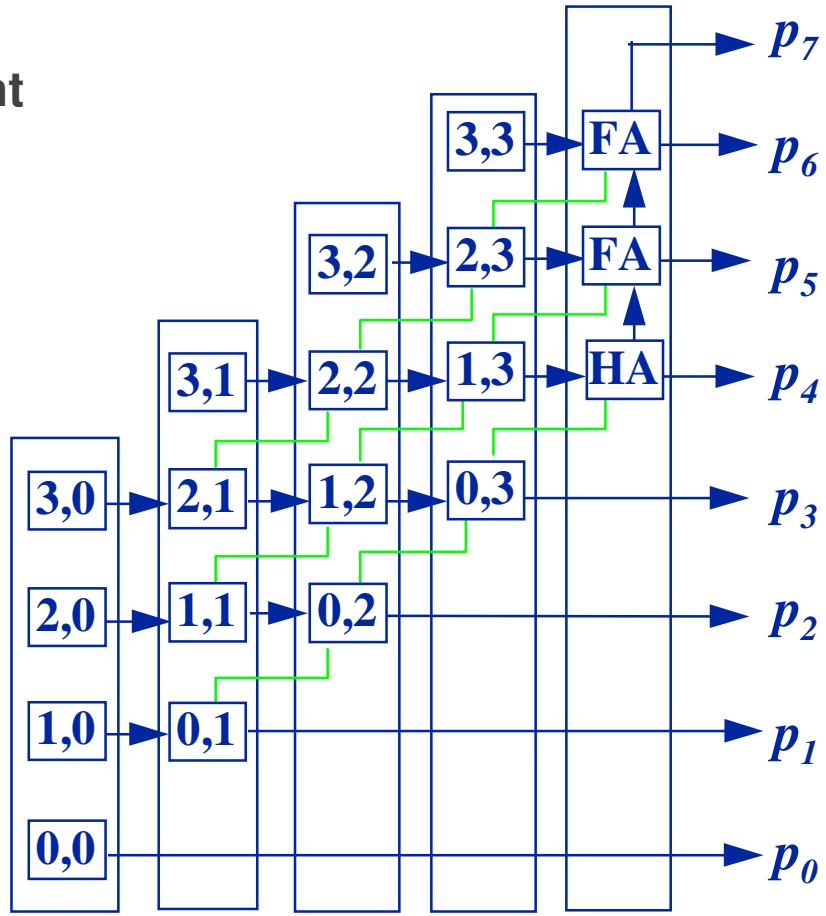
$$S + 2C = U + V + W$$

- Rearrange elements to eliminate carry chain
- Three input words
- Two output words
- 3 to 2 reducer

CSA Multiplier

Circuit Structure

- Carry outputs shifted to right
- Shortens longest path by factor of ~2

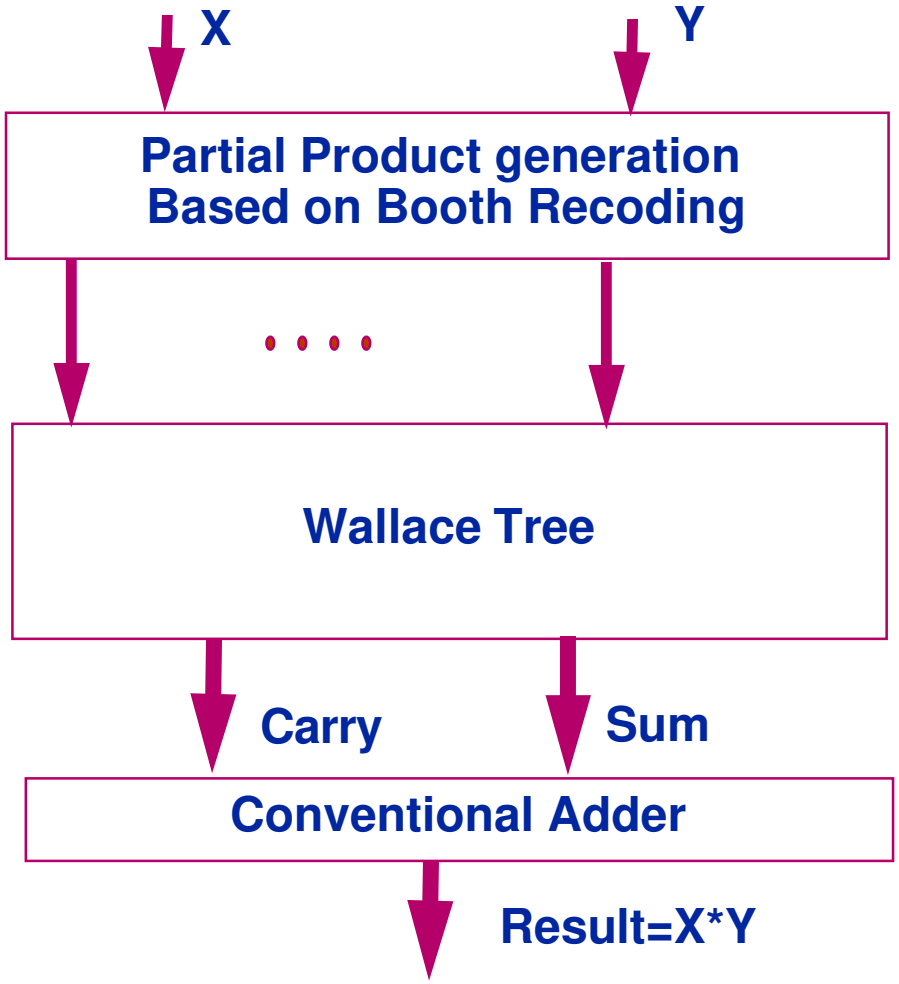


Wallace Tree

Reduce Time to log N.

Common Multiplier Design

Combined Booth recoding and Wallace tree



Floating point

IEEE standard

- **$(-1)^s(1.\text{mantissa})2^{(\text{Exponent-Bias})}$**
- **Different precision**
  - Single: x=8, y=23
  - Double: x=11, y=52
- **Bias: $2^{x-1}-1$.**
- **Max and Min Values:**
  - Exponent: Max=$2^{x-1}-1$, Min= $-2^{x-1}+2$
  - Mantissa: Min=1.0 Max=$2.0-2^{y}(1.11....111)$.
  - Special representations

Floating Point Addition

Let $A=(-1)^{Sa}(1.Ma)2^{Ea}$ and $B= (-1)^{Sb}(1.Mb)2^{Eb}$

Addition:

- A + B

Subtraction:

- A - B

Floating Point Rounding

Let the mantissa be $m_0m_1.m_2.....m_nGRS$

Case 1: $m_0=0$

- Use G, R, and S to generate rounding bit

Case 2: $m_0=1$

- Right shift : $m_0.m_1m_2.....m_nGRS$
- Increase exponent by 1.
- New S = R | S, New R = G New G = m_n

Four Rounding Modes:

- Positive Infinite
- Negative Infinite
- Zero
- Nearest

Renormalization

Floating Point Multiplication

Let $A=(-1)^{Sa}(1.Ma)2^{Ea}$ and $B= (-1)^{Sb}(1.Mb)2^{Eb}$

Multiplication: A * B

- Exponent : Ea+Eb
- Mantissa : 1.Ma * 1.Mb
- Sign : Sa ^ Sb
- Normalization
- Rounding.

Floating Point division

Given

- **Dividend mantissa** p
- **Divisor mantissa** d
- **Normalized forms:** $\texttt{1.xxxx}_2$
  - Range Constraint: $1 <= p, d < 2$

Compute

- **Quotient mantissa** q
  - To sufficient precision
- **Exponent**
  - Based on exponents of dividend & divisor

Radix 4 Division

Conventional "Restoring" Algorithm

- Select quotient digit q_i
  - 0, 1, 2, or 3
- Subtract $q_i d$ from p
- Shift p left

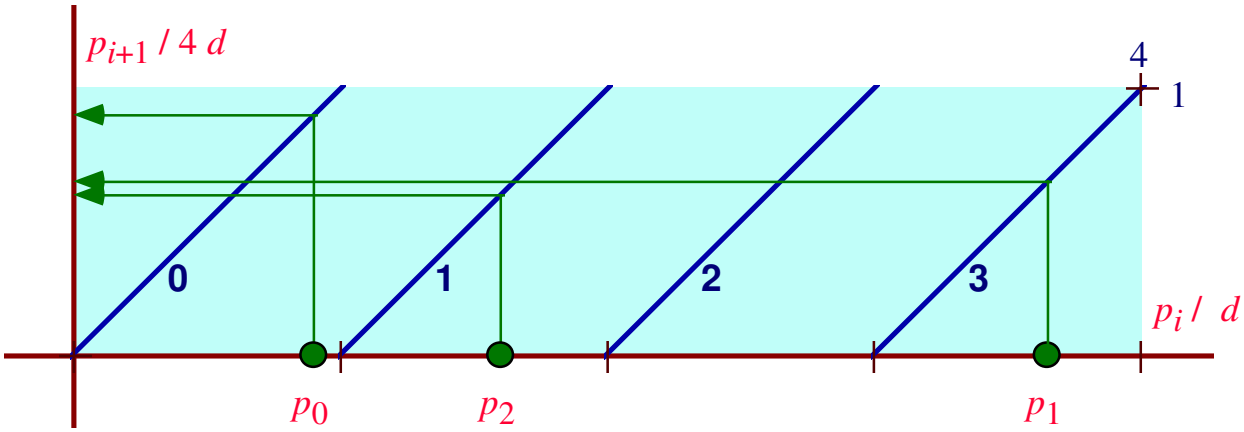
Divisor	d
X 1	1.301
X 2	3.202
X 3	11.103

- Must consider all digits to select digit q_i
- Must form "awkward multiple" $3d$

Example

p_0	1.2130	q_1	0
$-q_1 d$	- 0.		
	1.2130		
p_1	12.1300	q_2	3
$-q_2 d$	-11.1030		
	1.0210		
p_2	10.2100	q_3	2
$-q_3 d$	- 3.2020		
	1.0020		
	10.0200		

Visualization of Digit Selection



Interpretation

- **Horizontal axis indicates scaled partial remainder**
  - In range [0, 4)
- **Vertical axis indicates result of subtracting quotient digit**
  - Unique choice of digit
  - In range [0, 1)

SRT Division

History

- **Radix 4 version due to Atkins (1968)**

Goals

- **Predict quotient digit based on incomplete information about partial remainder and divisor**
- **No awkward multiples**

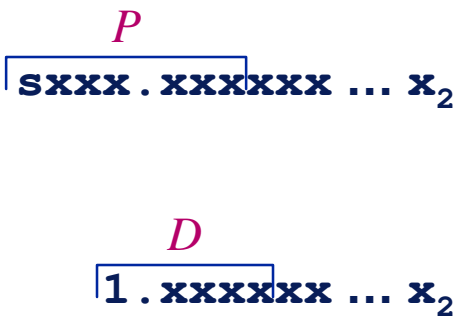
Key Ideas of Method

- **Quotient Digits –2, –1, 0, +1, +2**
  - Redundancy allows imperfect digit prediction
  - Power of two multiples
    - » Implement with shift and/or complement

SRT Iteration Step

Given

- **Partial Remainder** p_i
  - Two's complement representation
  - Can bound range $-16/3 < p_i < 16/3$
- **Divisor** d
  - Range Constraint: $-8/3\ d <= p_i <= 8/3\ d$



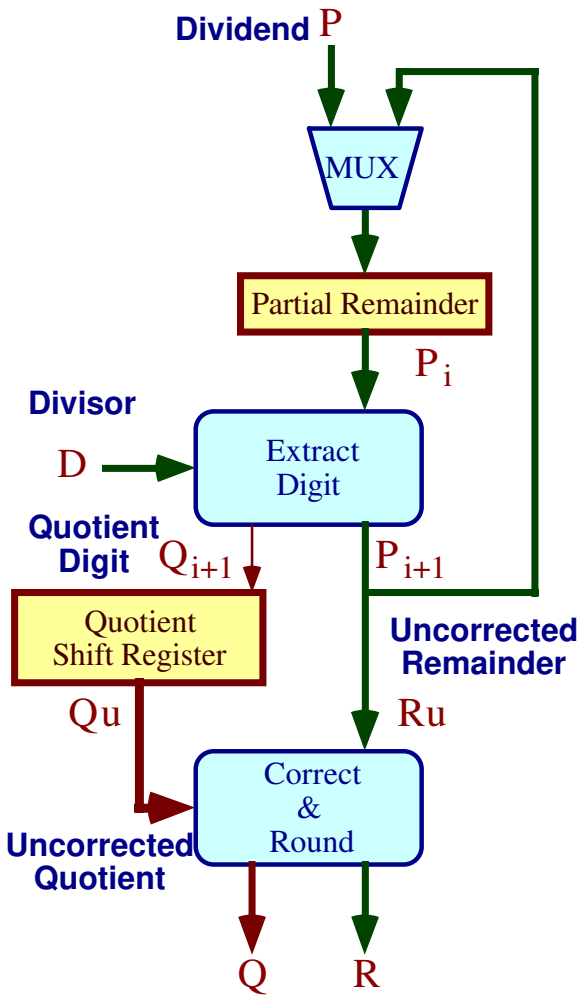
Select

- **Quotient Digit** q_{i+1}
  - By table lookup
  - Using truncated values P and D
- **Compute** $p_{i+1} = 4\,[p_i - q_{i+1}\,d\,]$

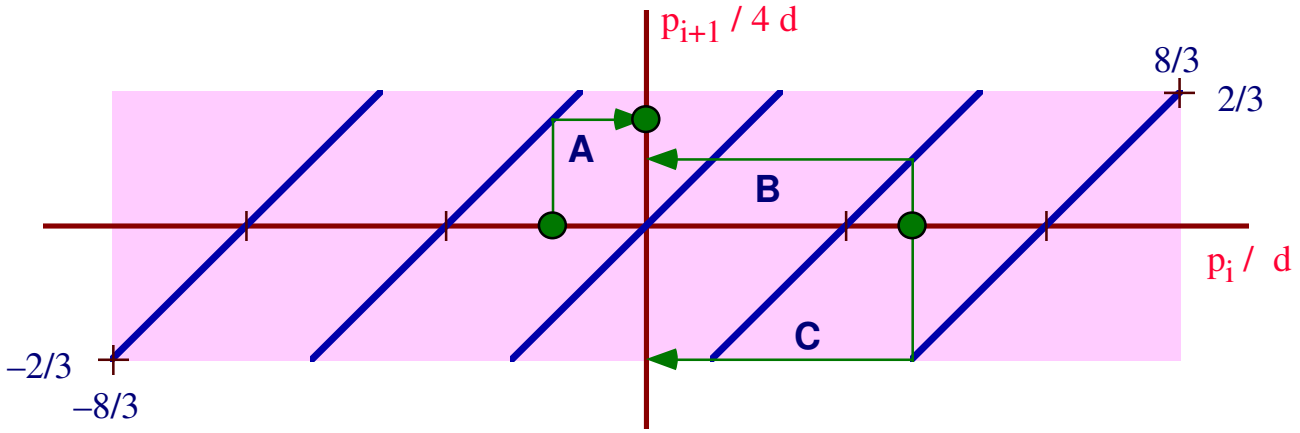
SRT Divider Circuit

Registers

- **Hold partial remainder**
  - Initially loaded with dividend
- **Divisor (not shown)**
  - Unchanged for entire computation
- **Quotient shift register**
  - Accumulates digits from iterations



Quotient Digit Selection



Examples

	p_i	q_{i+1}	p_{i+1}	
A	–7/15 d	–1	4 [8/15] d	= 32/15 d
B	4/3 d	+1	4 [1/3] d	= 4/3 d
C	4/3 d	+2	4 [–2/3] d	= –8/3 d

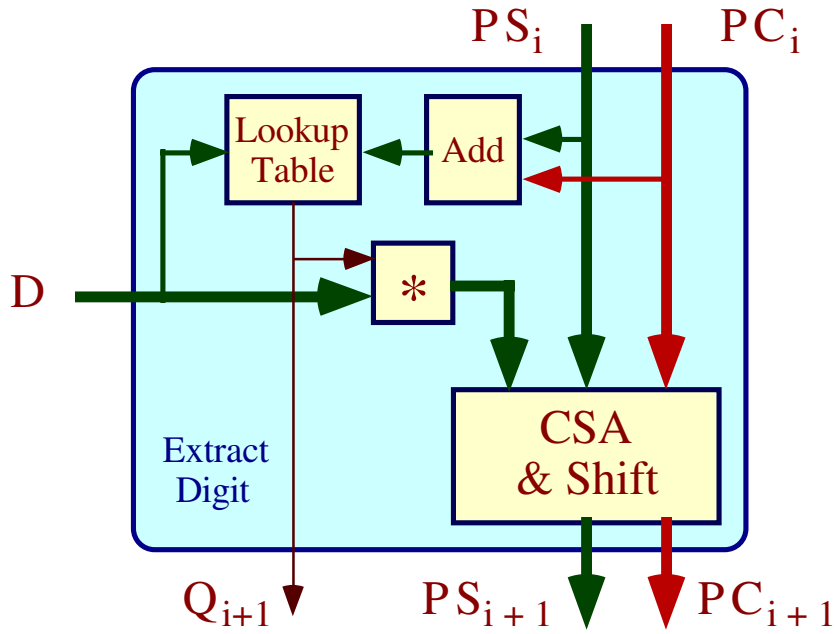
Invariant Preserved

$$-8/3\ d \ <= \ p_{i+1} \ <= \ 8/3\ d$$

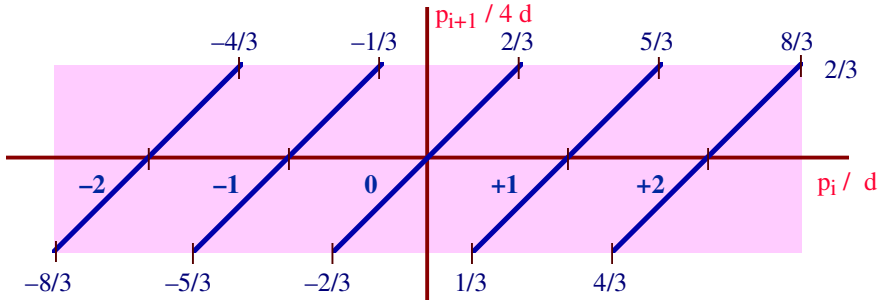
SRT Division Step

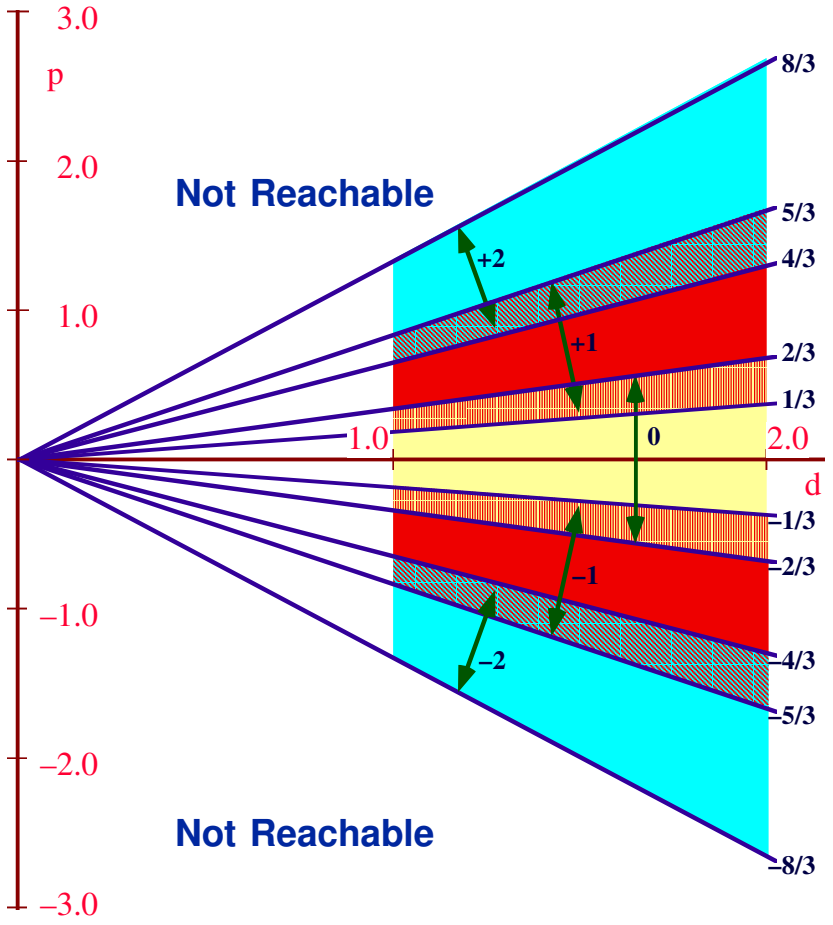
Elements

- **Carry Save Adder to subtract weighted divisor**
- **7-bit adder to get estimate of partial remainder**
  - High order bits
- **Lookup table to determine quotient digit**
  - 4 “interesting” bits of divisor
    - » Leading bit always 1
  - Implemented as PLA
  - Where Intel made mistake
- **Shift/complementer to weight divisor**
  - Multiply by quotient digit



Lookup Table Design





Digit Selection

- Depends on ratio of *d* and *p*
- Redundancy leads to overlapping regions
  - May choose either digit

Discretization of P & D

Values

- **Least Resolvable Unit (LRU)**
  - Weight of least significant bit
- **Partial Remainder** p_i
  - Approximate by P
  - Potentially underestimates by two LRUs
  - One each from pc and ps
- **Divisor** d
  - Approximate by D
  - Potentially underestimates by one LRU

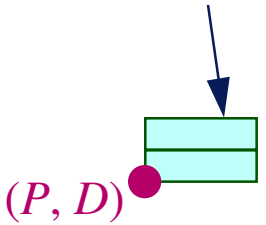
P

`sxxx.xxxxxx ...`

D

`1.xxxxxx ...`

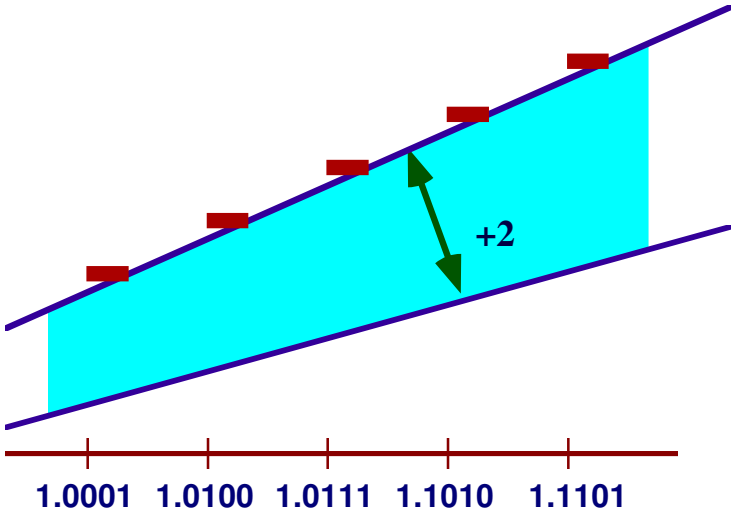
Range of Values for (p_i, d)



Compensating for Inaccuracy

- **Redundancy enables valid digit selection for all cases**
  - As long as have enough bits of p_i and d

Intel's Table Errors

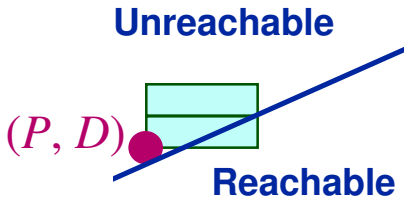


Actual Bug

- Erroneous entries for 5 cases
- Table generates 0 instead of +2

Occurs Only under Marginal Conditions

- (P, D) appear to be out of range
- D must under-estimate d
- P must closely estimate p_i
  - Unlikely with carry-save format



Example of Error

Correct Behavior

p_0	0100.111000	q_1	+2
$+-q_1 d$	1100.010010		
	0001.001010		
p_1	0100.101000		

Incorrect Behavior

p_0	0100.111000	q_1	0
$+-q_1 d$	0000.000000		
	0100.111000		
p_1	0011.100000		

Leading digits drop off end

Divisor d

X	1	0001.110111
X	2	0011.101110
X	–1	1110.001001
X	–2	1100.010010

- Once hit bad table entry, cannot recover

When Can Error Occur?

Requires at Least 7 Iterations

- **Hard to get to bad positions**
- **Worst case error 6×10^{-5}**

Only for "At Risk" Divisors

- **Stay in single column for entire division**
- **Remaining part of divisor should have lots of 1's**
  - To ensure that D underestimates d
- **Coe's Example**
  - $d = 3145727_{10}$ `10111111111111111111`
  - Correct result: 1.33382044913624100
  - Pentium's result: 1.3337390689020375 9
  - **At Risk!**
- **Nicely's Example**
  - $d = 824633702441_{10}$ `10111111111111111111011100000101001`

How Serious is the Error?

Intel's Claims

- **Most applications don't do much division**
  - 1000 per day
- **Many applications can tolerate occasional errors**
  - E.g., image generation
- **Maximum error fairly small**
  - Once every 1.5 X 10^9 divides
- **Statistical likelihood remote**
  - Once every 27,000 years

IBM's Counterclaims

- **Even spreadsheet users do lots of divides**
  - Up to 4.2 Million per day
- **Error more likely for numbers just below integer**
  - Up to once every 10^8 divides

Other Concerns

Error is not Random

- **Get same result every time on every Pentium**

Numbers Used in Practice NOT Uniformly Distributed

- **E.g., `4.999999 / 14.999999`**
  - gives `0.33332922` instead of `0.33333329`
  - Vaughan Pratt's "bruised integers"

Error is Hard to Work Around

- **Can shut off floating point**
  - Machine then runs slower than 486
- **Can recompile with workaround divide subroutine**
  - But most users only have binaries

Bad Numerical Properties

- **E.g, `4.999999 / 15.0 > 4.999999 / 14.999999`**
  - Problem for iterative methods

Who is Affected?

Average PC User

- **Not likely**

Heavy Users of Numerical Software

- **Significant subset of Pentium customers**
- **Concern for numerical accuracy very high**
  - Single inaccurate result can be magnified by later steps
- **Ethical/legal responsibilities**
  - Negligent to knowingly use flawed system
  - Would you buy a bridge from a civil engineer who had used a Pentium?

Why Didn't Intel Discover it?

Standard Steps in Verifying Design

- **Simulate many cases on high-level software model**
  - Probably had correct P-D table entries
- **Simulate/emulate final logic design**
  - Hardware emulators costing $Millions
  - Run complete chip model at ~ 100Hz
    - » $< 10^{-6}$ X real time
  - Not clear which version of table was used
- **Run tests on initial production chips**
  - Feasible to run billions of tests
  - Should have caught error here

Observations

- **Hard to test all aspects of such a complex system**
  - But this is a weak excuse

What about Formal Verification?

Currently Used Tools

- **Most based on Ordered Binary Decision Diagrams (OBDDs)**
- **Check transistor circuit design against low-level software model**
- **Compare two logic networks for equivalence**
  - E.g., pre- and post- optimization

Other Alternatives

- **Sequential verification by symbolic model checking**
  - Used on protocols and small sequential circuits
- **Tools based on automatic theorem provers**
  - Unpopular with industry

What About Dividers?

- **Some have been done with theorem prover, but messy**
- **BDD-based tools cannot handle complete algorithm**
- **Our own research shows promise**

Verifying Single Step

Specification

- **Define**

  $P_i = PS_i + PC_i$

  $P_{i+1} = PS_{i+1} + PC_{i+1}$

- **Input Constraint**

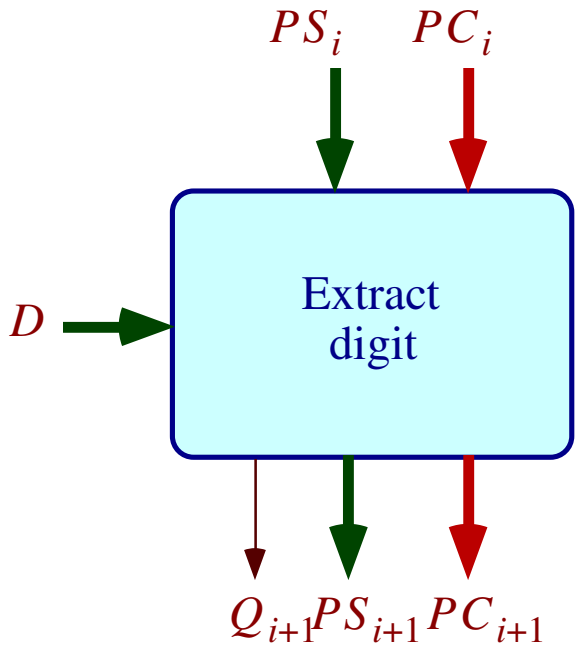
  $|P_i| <= 8/3\ D$

- **I/O Relation**

  $P_{i+1} = 4\ [P_i - Q_{i+1} D]$

- **Output Constraint**

  $|P_{i+1}| <= 8/3\ D$



Verification

- **Can do with OBDDs**
- **Create “checker circuits”**
- **Would uncover Intel’s problem**

System Considerations

Microprocessor Complexity Increasing at Rapid Pace

- **Pentium has features historically found in high-end mainframes**
  - Hardware support for multiplication, division, square root, and transcendental functions
  - Sophisticated instruction pipeline
- **Exercising all possible system behaviors especially difficult**
- **Future generations even more complex**
  - Speculative execution

Must Get it Right First Time

- **Cannot send field service to replace boards**
- **Typically have more people working on verification than on design**

Ubiquitous Systems of Unprecedented Complexity

- **Industry's headaches caused by own successes**

Impact on Intel

Short Term

- **Embarrassing to make mistake**
- **$475 Million is a lot of money**
  - Still had very profitable year, including $372 Million for 4Q94

Long Term

- **Intel still way ahead of competition**
  - Intel has announced next generation processor
  - 75% of $11 Billion microprocessor market
- **Everyone will be more careful in the future**
  - Good opportunities for research on formal verification
- **Intel benefits even from this publicity**
  - as playing key role in computer industry