EE 486 lecture 10: Multiply iteration and an introduction to Divide

M. J. Flynn

Patent searching

- In addition to your homework assignment also go to http://www.uspto.gov/patft/index.html and download the first page (image not text) of three arithmetic oriented patents 1) from patent applications 2) from issued patents and 3) one of whose inventors is "Rumynin"

Databases: Patent Grant and Patent Application Full-Text and Full-Page Images
Patent Full-Text and **Full-Page Image Databases**
Issued Patents (full-text since 1976, full-page images since 1790)
Patent Applications (published since 15 March 2001)
Quick Search
Advanced Search
Patent Number Search
Access Full-Page Images Directly
Database Notices and Status
Database Contents
Quick Search
Advanced Search
Publication Number Search
Important Notices!
How to Access Full-Page Images
Problems Accessing the Databases?
Report Data Content Problems
Tools to Help in Searching by Patent Classification
Downloadable Published Sequence Listings
HOME | INDEX | SEARCH | SYSTEM STATUS | BUSINESS CENTER | NEWS&NOTICES |

Multipliers

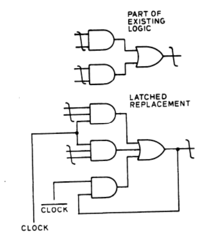
Type	D= Delay (CSAs)	W	D,W for 27b case
Wallace	$2\log_{3/2} n$	depends	7
Binary tree	$2[\log_2 n-1]$	$2[\log_2 n]$	8, 10
ZM (type1)	2 sqrt (n)	5	9, 5
OS (type1)	sqrt(2n-6)	6	8, 6
Higher Order array	2 sqrt (n)	5	9, 5

Trees, arrays and iteration

- Arrays and trees need not be fully built out, but smaller structures can be iterated on to produce the product. While requiring multiple passes through the hardware, the simpler pp generation and reduction can make the processes faster than expected.
- The trick is (as with clocking) low iteration overhead

The Earle latch

- Adds latch to existing logic with little overhead, no delay.
- Pipeline rate is 4 gate delays (2 to transit and 2 to hold).
- Becomes the basis for iterative multipliers.

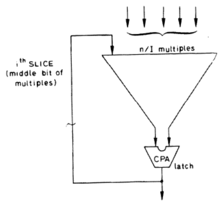


Earle latch

- Cycle time of 4 gate delays (2 for clock and transit and 2 for ~clock).
- No additional transit delay
- Useful in pipelining CSA arrays and trees
- BUT, iterate on what?

Iterate on the CPA

- Brings the reduced product back into the tree.
- The tree now has n/I+1 inputs where I is the number of iterations.
- The CPA output is shifted by k n/I bits, k the multiplier encoder

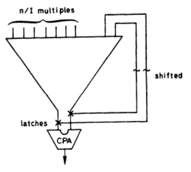


Iterate on the tree

Inputs now (n/I) +2

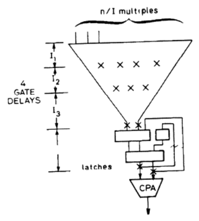
But no longer need the CPA as part of the iteration

Need to shift the CPA output to align the pp's



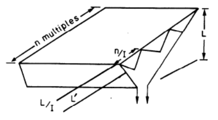
Iterate on a "compressor"

- Add a [4:2] at the bottom of the tree, then iterate on the two outputs.
- Need to balance the iteration delays. Iteration is now 4 gate delays. The tree and the pp generator must support this rate.



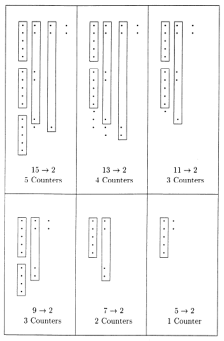
Iteration

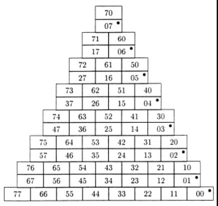
- Iteration reduces the cost of pp generation by I and of pp reduction somewhat less than that.
- If the iteration overhead can be kept low the result may compare favorably to a fully built tree.



Example, 64x64 using 8x8 generators and (5,5,4) counters

- Consider a fully built tree. The height,n, is 15. There are 3 (5,5,4) counter delays in the tree.
- The pp generation uses 64 pp generators and 152 counters and a 128b CPA





Same multiplier with iteration

- Suppose we iterate on a single level of (5,5,4) counters. We reduce 5 pp's in iteration 1 and 3pp's in each iteration thereafter. So 1+4 iterations covers the 15 pp's. Now we need about 30 pp generators and only 32 or so counters. The CPA spans only about 74b.
- The counter delay is 5, ignoring the iteration overhead.

Two 64b multipliers compared

Type	pp's needed to be generated	Depth in (5,5,4)'s	CPA size required
Tree	15	3	128b
Iterative (one level)	3	5 (plus clocking overhead)	74b

Multipliers

- Multipliers (especially n > 50) are big and expensive, with potentially serious wire congestion and length problems.
- Yes, it's possible to make a really awful multiplier that's slower than a better design with half the hardware.
- Indeed, there's room for yet one more multiplier patent! Good luck.

Divide

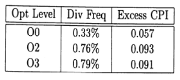
- Divisor/dividend = quotient + rem./dividend
- Basic approaches:
  - Subtractive (restoring, non restoring and SRT)
  - Multiplicative (Newton-Raphson, Binomial expansion in Taylor series)
  - Table look up (bipartite tables)

How important is divide?

- How much hardware should be devoted to divide and square root?
- Can compilers schedule long latency ops?
- What's the role of multiple issue?
- Can the multiplier be shared with MPY and DIV?
- Can the divide be shared with DIV and SQRT?

Compiler effects

- Compiler optimization has 2 effects.
  - It decreases the number of LD/ST instructions, increasing the frequency of DIV
  - Makes DIV relatively more important, but it also can manage dependencies better.



Opt Level	Div Freq	Excess CPI
O0	0.33%	0.057
O2	0.76%	0.093
O3	0.79%	0.091

Table 6.1: Effects of compiler optimization

Multiple issue effects

- Multiple issue machines need access to results earlier, even with optimized code.
- Earlier usage means more CPI delay due to DIV latency
- Note the AT =k behavior for the DIV implementations

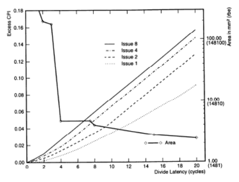


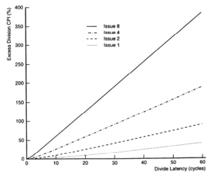
Figure 6.5: CPI and area vs division latency - low latency

Area- time tradeoffs

algorithm	Latency (~)	Excess CPI	Area (rbe)
1b	>40	0.5	2500
2b	20-40	.1-.3	3110
4b, simple NR	10-20	.04-.1	4070
8b	4-10	.01-.07	6665
Very High radix	<4	< .01	>100,000

Multiple Issue effects

- The higher the issue rate the higher the CPI delay due to DIV. The relative effects are even more significant.
- MI processors need good divide support.



Excess CPI as a percentage of base CPI for multiple issue

Sharing a multiplier with a divider

- If a multiplier is shared with the divide unit the conflict cost is low, especially for fast algorithms(.02-.03 excess CPI)
- The problem is more of implementation…a multi-function unit has more overhead.

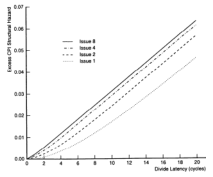


Figure 6.8: Excess CPI due to shared multiplier

Subtractive divide

- Restoring
- Non restoring
- Shift over 0's
- Brute force (multiple subtractors)

Restoring and non restoring

- To find q_n, … q_1, q_0; R(n) = Dividend; trial $q_i = 1$; d is the divisor R(0) = remainder
- $R(i+1) = R(i) - q_i \times 2^i$ d; if result is negative then set $q_i = 0$ and restore … $R(I) + 2^i$ d; if positive then set $q_i = 1$ and proceed.
- Non restore… if negative then set $q_i = 0$ and proceed with $R(i+1) = R(i) + q_i \times 2^i$ d;
- Note that - 2 d + d = -d

Example

$29 - 3 \star 2^4 = -19$ $q_4 = 1$

$-19 + 3 \star 2^4 = +29$ restore $\boxed{q_4 = 0}$

$29 - 3 \star 2^3 = +5$ $\boxed{q_3 = 1}$

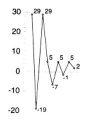
$+5 - 3 \star 2^2 = -7$ $q_2 = 1$

$-7 + 3 \star 2^2 = +5$ restore $\boxed{q_2 = 0}$

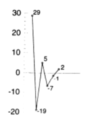
$+5 - 3 \star 2^1 = -1$ $q_1 = 1$

$-1 + 3 \star 2^1 = +5$ restore $\boxed{q_1 = 0}$

$+5 - 3 \star 2^0 = +2$ $\boxed{q_0 = 1}$



Restoring Division

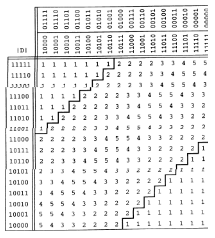


Non-restoring Division

Speeding up division

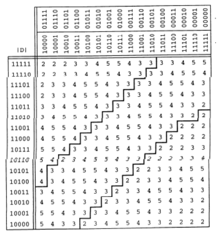
- Skip over 0’s and skip over 1’s; similar to multiply
- Higher radix: radix 4 (2b/iteration) or radix 8 (3b/iteration. Widely used with redundant digit set.
- Simplest approach is to use multiple subtractors… also called brute force. Need 3 for 2b and 7 for 3b.

Quotient bits determined; 1 and 3 pr.



|D|	01111 10000	01110 10001	01101 10010	01100 10011	01011 10100	01010 10101	01001 10110	01000 10111	00111 11000	00110 11001	00101 11010	00100 11011	00011 11100	00010 11101	00001 11110	00000 11111
11111	1	1	1	1	1	1	1	2	2	2	2	3	3	4	5	5
11110	1	1	1	1	1	1	2	2	2	2	3	3	4	5	5	4
11101	1	1	1	1	1	2	2	2	2	3	3	4	5	5	4	3
11100	1	1	1	1	2	2	2	2	3	3	4	5	5	4	3	3
11011	1	1	1	2	2	2	2	3	3	4	5	5	4	3	3	2
11010	1	1	2	2	2	2	3	3	4	5	5	4	3	3	2	2
11001	1	2	2	2	2	3	3	4	5	5	4	3	3	2	2	2
11000	2	2	2	2	3	3	4	5	5	4	3	3	2	2	2	2
10111	2	2	2	3	3	4	5	5	4	3	3	2	2	2	2	1
10110	2	2	3	3	4	5	5	4	3	3	2	2	2	2	1	1
10101	2	3	3	4	5	5	4	3	3	2	2	2	2	1	1	1
10100	3	3	4	5	5	4	3	3	2	2	2	2	1	1	1	1
10011	3	4	5	5	4	3	3	2	2	2	2	1	1	1	1	1
10010	4	5	5	4	3	3	2	2	2	2	1	1	1	1	1	1
10001	5	5	4	3	3	2	2	2	2	1	1	1	1	1	1	1
10000	5	4	3	3	2	2	2	2	1	1	1	1	1	1	1	1

Table 4.13: Shift-length table for divisor-multiple



|D|	01111 10000	01110 10001	01101 10010	01100 10011	01011 10100	01010 10101	01001 10110	01000 10111	00111 11000	00110 11001	00101 11010	00100 11011	00011 11100	00010 11101	00001 11110	00000 11111
11111	2	2	2	3	3	4	5	5	4	3	3	3	3	4	5	5
11110	2	2	3	3	4	5	5	4	3	3	3	3	4	5	5	4
11101	2	3	3	4	5	5	4	3	3	3	3	4	5	5	4	3
11100	2	3	3	4	5	5	4	3	3	3	4	5	5	4	3	3
11011	3	3	4	5	5	4	3	3	3	4	5	5	4	3	3	2
11010	3	4	5	5	4	3	3	3	4	5	5	4	3	3	2	2
11001	4	5	5	4	3	3	3	4	5	5	4	3	3	2	2	2
11000	4	5	5	4	3	3	4	5	5	4	3	3	2	2	2	2
10111	5	5	4	3	3	4	5	5	4	3	3	2	2	2	3	3
10110	5	4	2	3	4	5	5	4	3	3	2	2	2	3	3	4
10101	4	3	3	4	5	5	4	3	3	2	2	3	3	4	5	5
10100	4	3	4	5	5	4	3	3	2	2	3	3	4	5	5	4
10011	3	4	5	5	4	3	3	2	3	3	4	5	5	4	3	3
10010	4	5	5	4	3	3	2	3	3	4	5	5	4	3	3	2
10001	5	5	4	3	3	3	3	4	5	5	4	3	3	2	2	2
10000	5	4	3	3	2	3	4	5	5	4	3	3	2	2	2	2

Table 4.15: Shift-length table for divisor-multiples $3D/2, D,$ a

Redundancy in division

- Multiple subtractors can be replaced with redundant digit sets. This leads to SRT division.(lecture 12)
- Usually limited to 2b / iteration.