Addition and Division method

เสนอ

ผศ.ดร.สุรินทร์ กิตติธรกุล

จัดทำโดย

61010914 นายลัทธพล แพ่งสภา

61011405 นายพรรษา บุญทวีกุลสวัสดิ์

61011433 นายเสฏฐวุฒิ ทิพย์กรรภิรมย์

เอกสารการประกอบการนำเสนอนี้ เป็นส่วนหนึ่งของ

วิชา 01076009 ชื่อวิชา Computer Organization and Assembly Language

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ADDITION AND DIVISION METHOD

Content

- ADDER
  - Ripple Carry Adder (RCA)
  - Carry Lookahead Adder (CLA)
- DIVISION
  - Slow Division Method
    - Restoring Division
    - Non-Restoring Division
  - Fast Division Method
    - SRT
    - Newton-Raphson Division
      - Variant Newton-Raphson Division
    - Goldschmidt division
      - Binomial theorem

ADDER

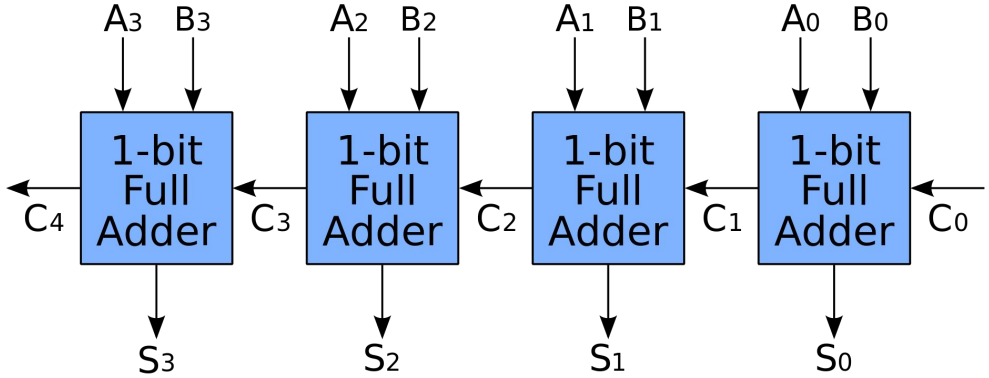
วงจรบวก

Ripple Carry Adder (RCA)

การทดแบบริปเปิล (ระลอก)

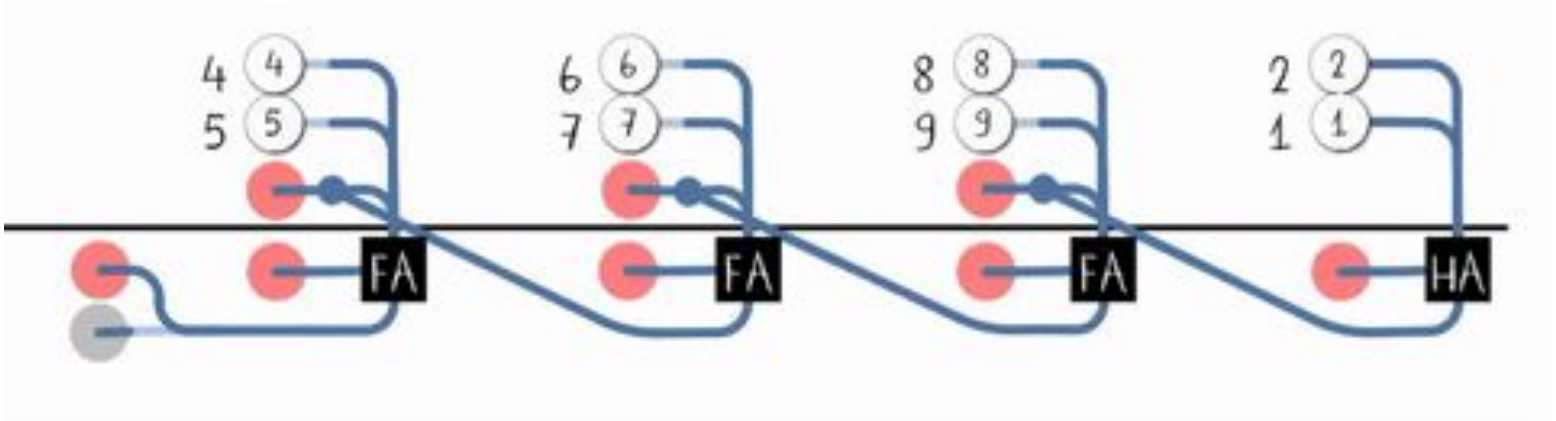
Ripple Carry Adder (RCA)

Ripple carry adder หรืออีกชื่อหนึ่งคือ วงจรบวกแบบขนาน (Parallel Adder) เป็นวิธีบวกเลขที่คล้ายกับวิธีที่คนใช้ในการบวกเลข โดยวงจร Ripple carry adder เป็นการนำวงจรบวกแบบคิดค่าตัวทด (Full Adder) มาต่อขนานกันเพื่อทำให้บวกได้จำนวนบิตมากขึ้น



รูปที่ 1. ตัวอย่างวงจร Ripple Carry Adder 4 บิต

Ripple Carry Adder (RCA)



รูปที่ 2. ตัวอย่าง Animation Ripple Carry Adder 4 บิต (ตัวอย่างเป็นการบวกในรูปเลขฐานสิบ)

Carry Lookahead Adder (CLA)

การทดแบบดูตัวทดล่วงหน้า

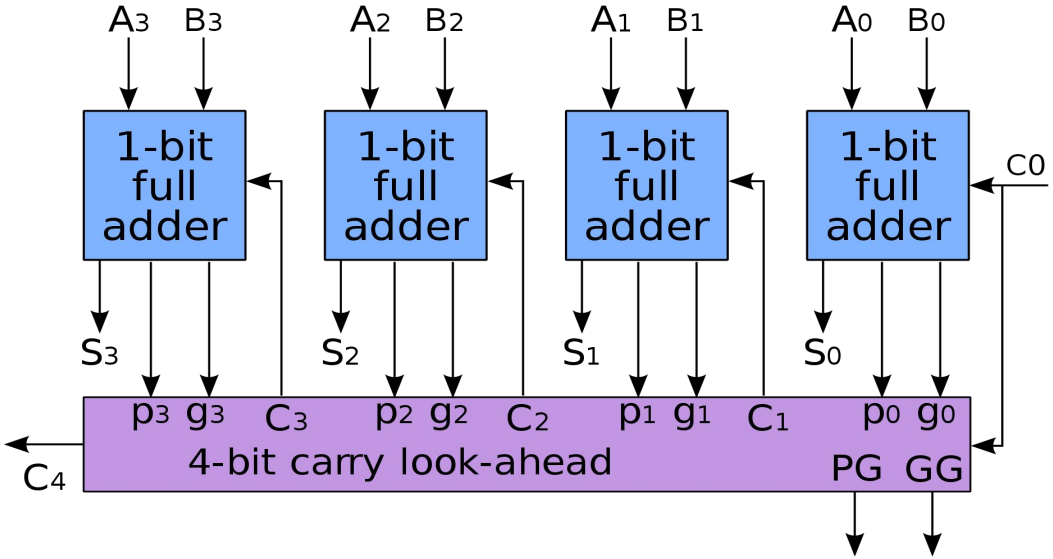
Carry Lookahead Adder (CLA)

เนื่องจากการใช้เวลาที่น้อยแต่ได้ผลลัพธ์ที่ถูกต้องเป็นสิ่งสำคัญในการเพิ่มประสิทธิภาพในการทำงานของคอมพิวเตอร์ ดังนั้นเราจึงทำวงจร Carry Lookahead Adder เพื่อลดเวลาในขณะที่กำลังคำนวณการบวก ซึ่งวิธีนี้เป็นการลดเวลาที่ใช้ในการทดเลข โดยใช้การคำนวณล่วงหน้าว่าตัวทดควรจะเป็นอะไร

Carry Lookahead Adder (CLA)

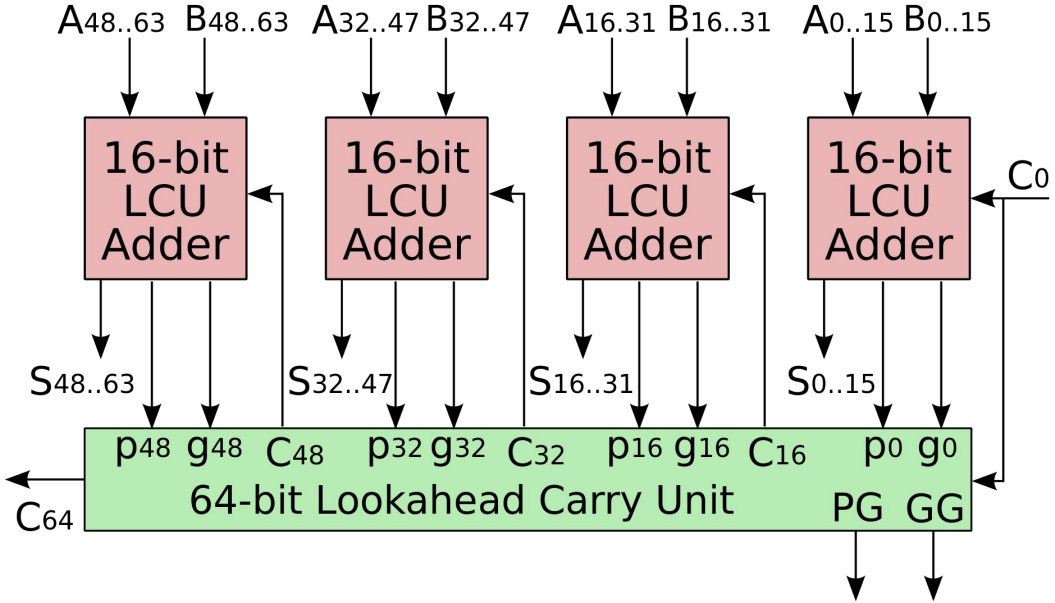
การบวกแบบ CLA นั้นจะเร็วกว่าแบบ RCA อย่างเห็นได้ชัดก็ต่อเมื่อมีการบวกจำนวนเป็นจำนวนบิตมากๆ โดยในการใช้จริง จะมีการใช้ CLA 4 bit เป็นชุดๆ เพื่อหา C4, C8, C12 และ C16 และนำวงจร CLA 16 bit นี้มาต่อเป็นชุดๆอีกเช่นกัน เพื่อหา C16, C32, C48 และ C64 (ดังรูปที่ 4.)

Carry Lookahead Adder (CLA)



รูปที่ 3. ตัวอย่าง Carry Lookahead Adder 4 บิต

Carry Lookahead Adder (CLA)



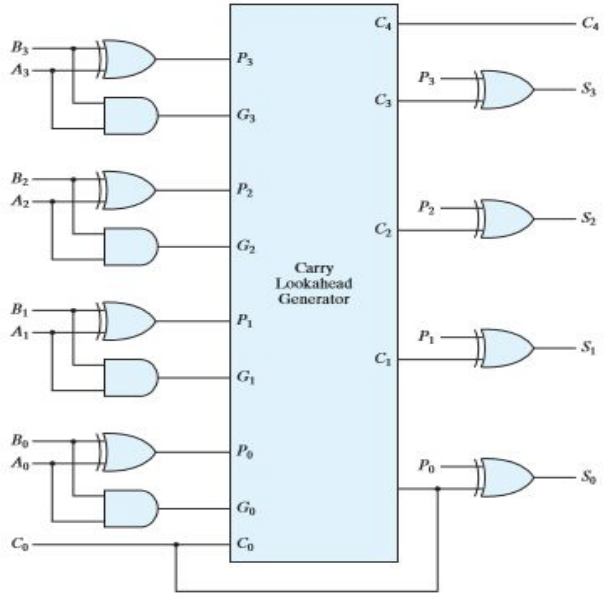
รูปที่ 4. ตัวอย่าง Carry Lookahead Adder 64 บิต

Carry Lookahead Adder (CLA)

A	B	Carry In	Carry Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

ตารางที่ 1. ตารางค่าความจริงของวงจร Full Adder

Carry Lookahead Adder (CLA)

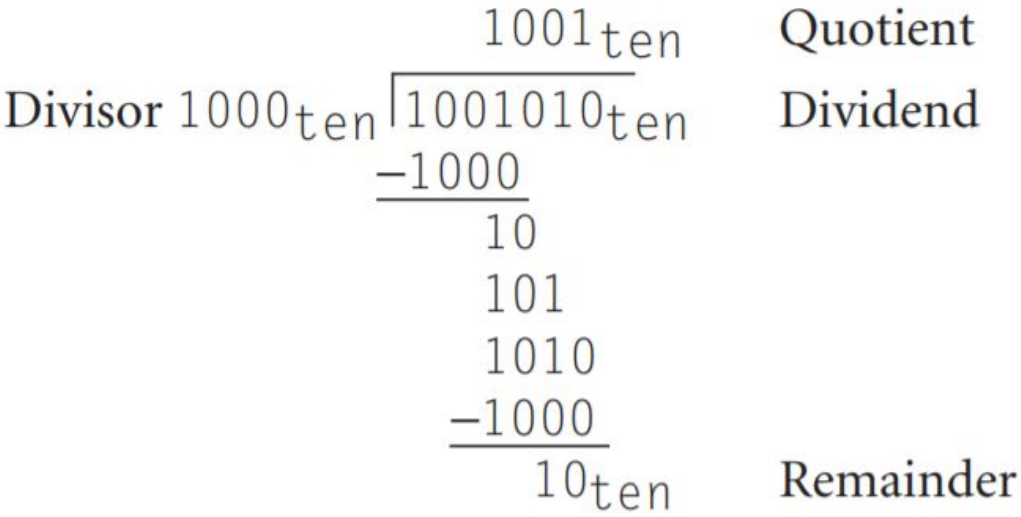


รูปที่ 5. CLA Logic

DIVISION

วงจรหาร

Division



รูปที่ 6. ตัวอย่างการหารแบบทั่วไป

การหารแบบปกติที่มนุษย์ใช้ ประกอบด้วย

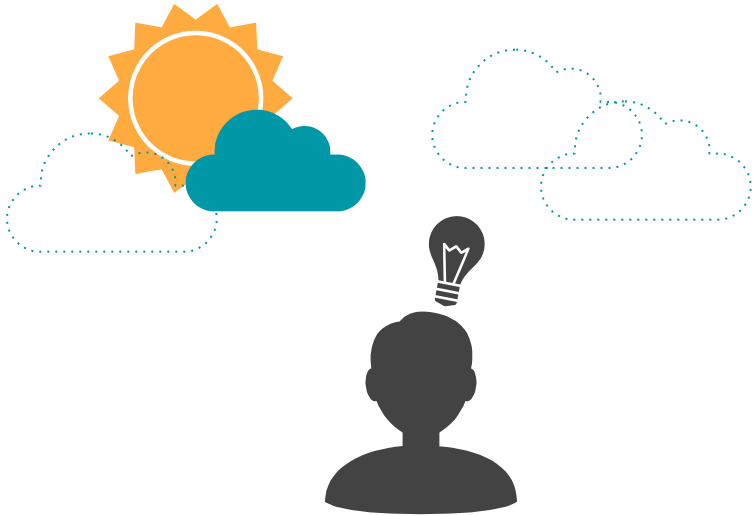
- **Divisor :** ตัวหาร
- **Dividend :** ตัวตั้ง
- **Quotient :** ผลการหาร
- **Remainder :** เศษการหาร

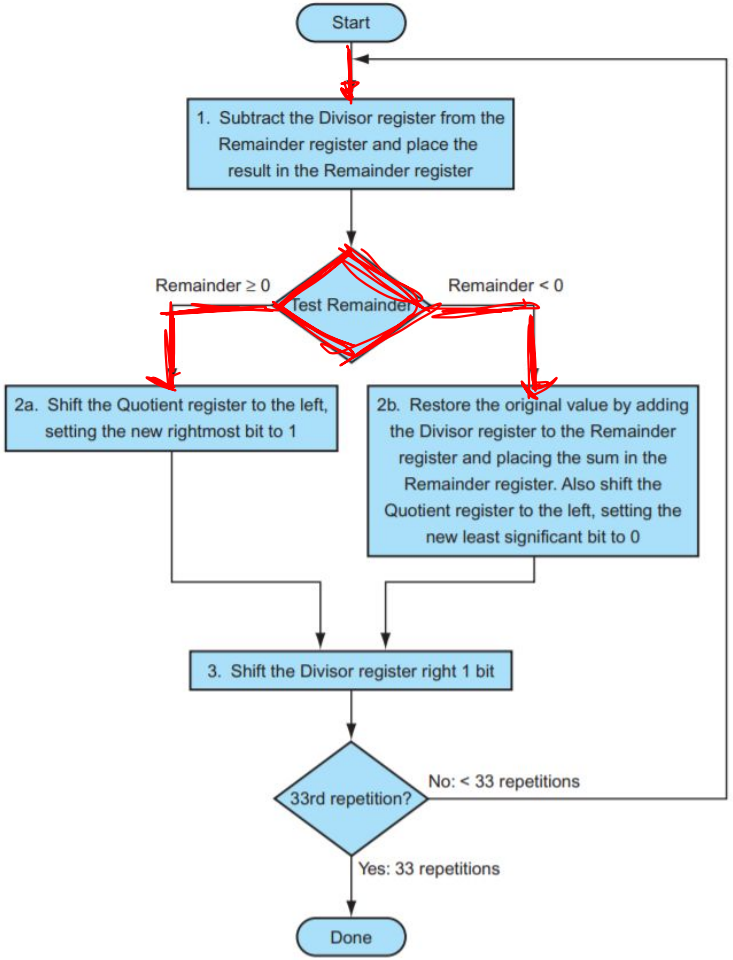
สรุปเป็นสมการได้ว่า

Dividend = Divisor x Quotient + Remainder

Slow Division Method

วิธีหารแบบช้า



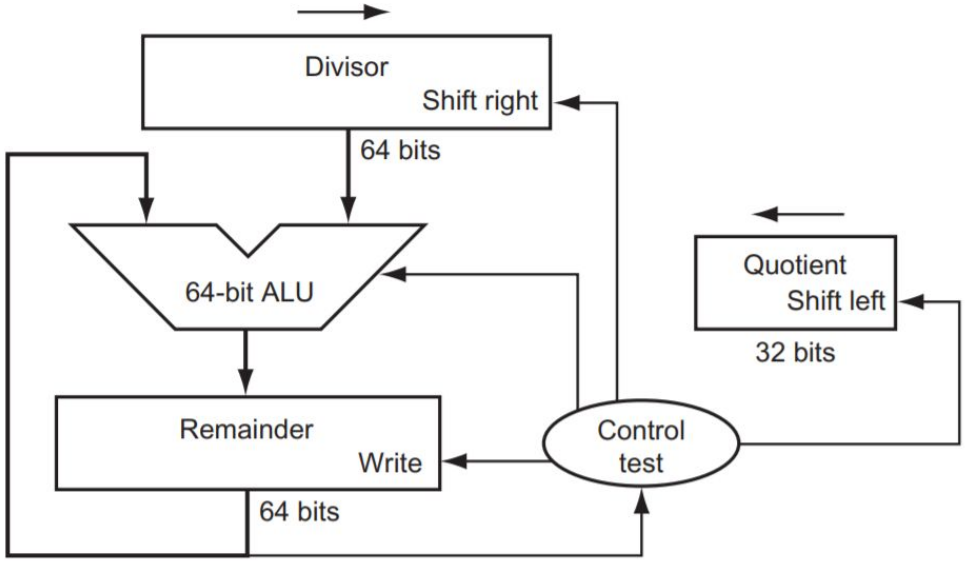


รูปที่ 7. Flowchart ของ Restoring Division Method

Restoring Division

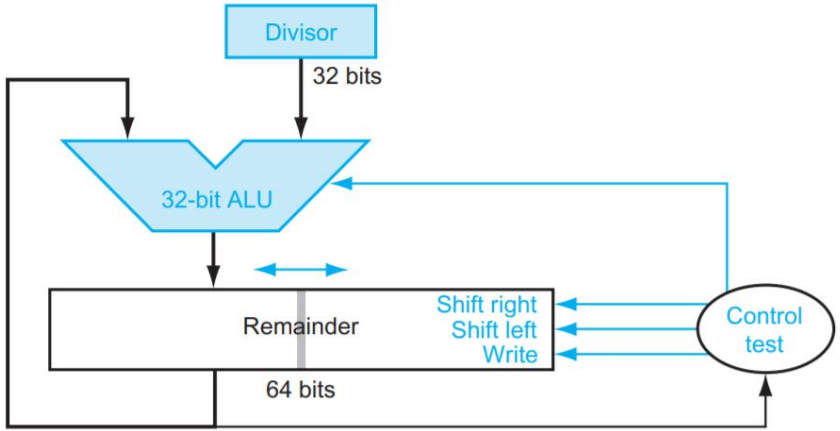
Algorithm

Restoring Division



รูปที่ 8. ตัวอย่างวงจรการหารเวอร์ชันแรก

Restoring Division



รูปที่ 9. ตัวอย่างวงจรการหาร MIPS

เป็นวงจรหารเวอร์ชั่นที่พัฒนาขึ้นมาถัดจากเวอร์ชันแรก เรียกว่า MIPS

ส่วนที่เป็นสีฟ้าคือที่ส่วนที่ถูกพัฒนาขึ้นมาจากเวอร์ชั่นแรก

การพัฒนาที่สำคัญคือ การทำวงจรโดยไม่สนใจ Overflow แต่ให้ Software เช็คการ Overflow แทนที่จะใช้ Hardware

Restoring Division

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

ตารางที่ 2. ตัวอย่างการหารด้วยวงจร Restoring Division

Restoring Division

Signed Number :

จากสมการ Quotient = (Dividend - Remainder)/Divisor จะเห็นว่าเครื่องหมายของ Quotient จะขึ้นอยู่กับ Dividend และ Divisor (ไม่ขึ้นอยู่กับ Remainder เพราะ Remainder มีค่าน้อยกว่าหรือเท่ากับ Dividend จึงไม่ทำให้เครื่องหมายของส่วนด้านบนเปลี่ยน)

- เครื่องหมายของ Dividend กับ Divisor เหมือนกัน Quotient จะเป็น +
- เครื่องหมายของ Dividend กับ Divisor ต่างกัน Quotient จะเป็น -

Restoring Division

Signed Number (Example) :

1. +7/+2

เมื่อคิดออกมาแล้วจะได้ Quotient = +3,
Remainder = +1

ตรวจคำตอบ:

+7 = (+3 x 2) + (+1)

+7 = +7 ✓

2. −7/+2

เมื่อคิดออกมาแล้วจะได้ Quotient = −3,
Remainder = −1

ตรวจคำตอบ:

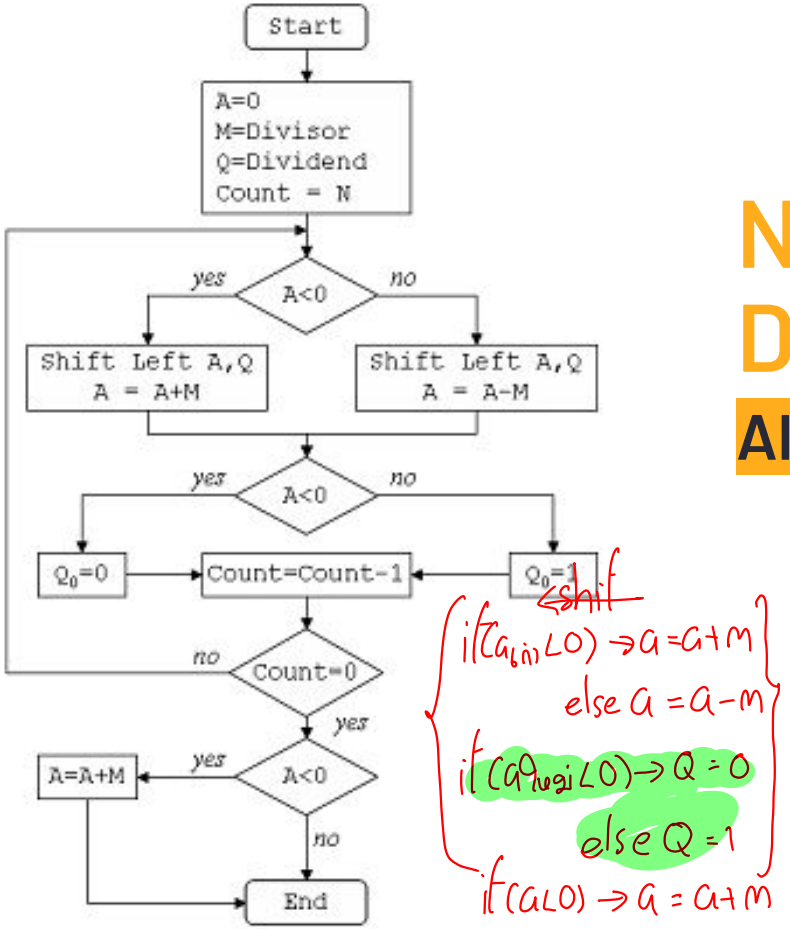
−7 = (−3 x 2) + (−1)

−7 = −7 ✓

** แล้วถ้า Quotient = −4, Remainder = +1 ล่ะ?

Non - Restoring Division

Algorithm



รูปที่ 10. Flowchart ของ Non-Restoring Division Method

Non - Restoring Division

Non - Restoring Division เป็นวิธีหารที่มีความเร็วมากกว่าแบบ Restoring เล็กน้อย เพราะ ไม่มีการเก็บค่าของ Remainder กลับมาใหม่ แต่เปลี่ยนเป็นการเพิ่มค่า Dividend เข้าไปยัง Remainder แทน

Non - Restoring Division

Example : Dividend = 101110,
Divisor = 010111

Set A = Dividend = 000000
Set Q = Dividend = 101110
จะได้ว่า AQ = 000000 101110

Set M = Divisor = 010111

M'= 101001

Action	A	Q	Count
Initial	000 000	101 110	6
A > 0 => SHL (AQ)	000 001	011 10□	
A = A-M	101 010	011 10□	
A < 0 => Q0 = 0	101 010	011 100	5
A < 0 => SHL (AQ)	010 100	111 00□	
A = A+M	101 011	111 00□	
A < 0 => Q0 = 0	101 011	111 000	4
A < 0 => SHL (AQ)	010 111	110 00□	
A = A+M	101 110	110 00□	
A < 0 => Q0 = 0	101 110	110 000	3
A < 0 => SHL (AQ)	011 101	100 00□	
A = A+M	110 100	100 00□	
A < 0 => Q0 = 0	110 100	100 000	2
A < 0 => SHL (AQ)	101 001	000 00□	
A = A+M	000 000	000 00□	
A < 0 => Q0 = 1	000 000	000 001	1
A > 0 => SHL (AQ)	000 000	000 01□	
A = A+M	101 001	000 01□	
A < 0 => Q0 = 1	101 001	000 010	0
Count has reached Zero, So final steps			
A < 0 => A = A+M	000 000	000 010	
	Reminder	Quotient	

ตารางที่ 3. ตัวอย่างการหารด้วยวงจร Non - Restoring Division

SRT Division

Non – Restoring Division เป็นวิธีหารที่มีความเร็วมากกว่าแบบ Restoring เล็กน้อย เพราะ ไม่มีการเก็บค่าของ Remainder กลับมาใหม่ แต่เปลี่ยนเป็นการเพิ่มค่า Dividend เข้าไปยัง Remainder แทน

C

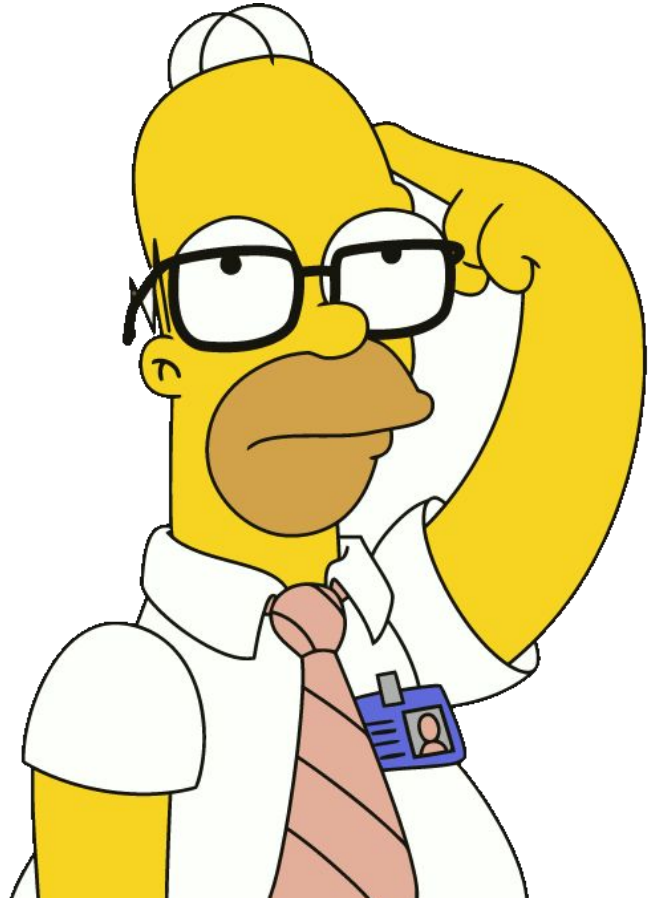
Non - Restoring Division เป็นวิธีหารที่มีความเร็วมากกว่าแบบ Restoring เล็กน้อย เพราะ ไม่มีการเก็บค่าของ Remainder กลับมาใหม่ แต่เปลี่ยนเป็นการเพิ่มค่า Dividend เข้าไปยัง Remainder แทน

Fast Division Method

วิธีหารแบบเร็ว

What is Fast Division Method?

Moore’s Law ได้ถูกนำมาใช้กับการหาร เช่นเดียวกันกับการคูณ แต่ทว่าการหารนั้นต่างจากการคูณ เราไม่สามารถที่จะเพิ่มฮาร์ดแวร์เพื่อช่วยให้การหารนั้นไวขึ้นได้ นั่นเป็นเหตุผลที่เราต้องใช้อัลกอริทึมบางอย่างเพื่อช่วยให้การหารเลขจำนวนมาก ๆนั้นมีความเร็วเพิ่มขึ้น



SRT Division

Algorithm

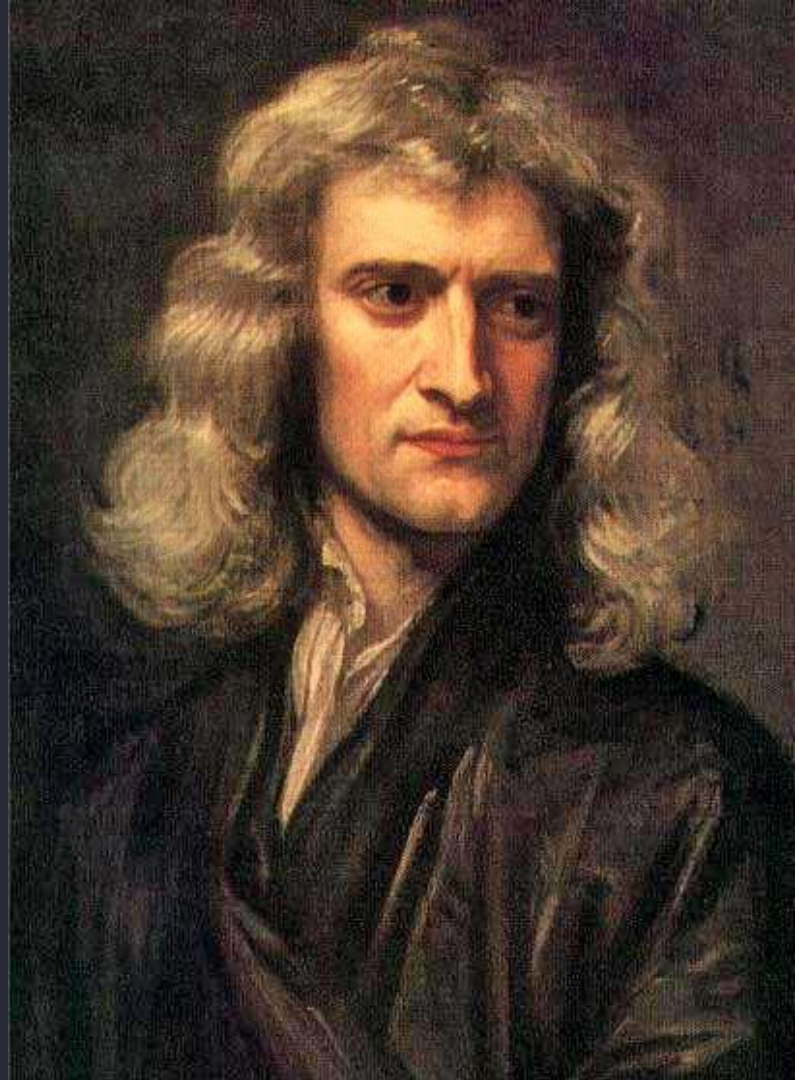
SRT Division

โดยปกติ การหารแบบ Slow Division นั้นจะทำให้ได้ Quotient 1 บิต ต่อขั้นตอน จึงได้มีการคิดค้นวิธี SRT Division ขึ้นเพื่อให้ได้จำนวน Quotient มากขึ้นใน 1 ขั้นตอน

ซึ่งวิธีนี้ใช้การทำนาย Quotient หลายๆบิตใน 1 ขั้นตอนโดยดูจาก Lookup Table และในขั้นตอนถัดไปจะมีการแก้ Quotient ที่ทำนายผิดพลาด

SRT Division

โดยวิธีนี้จะใช้เลข 6 บิต จาก Reminder และ 4 บิต จาก Divisor เพื่อใช้ในการทำนายเทียบกับ Table ซึ่งความแม่นยำของวิธีนี้ก็จะขึ้นอยู่กับความเหมาะสมของตัว Table



Newton-Raphson Division

Algorithm

Newton-Raphson Division

Newton-Raphson เป็นการนำ Newton's method มาใช้ ซึ่ง Newton's method เป็นวิธีที่ใช้ในการหาราก โดย Newton-Raphson Division มีขั้นตอนดังนี้

1. ประมาณค่า X_0 เป็น 1/D (ส่วนกลับของ Divisor D)
2. คำนวณซ้ำ ๆ โดยประมาณค่า $X_1, X_2, X_3, \ldots, X_S$ เพื่อให้เกิดความแม่นยำมากขึ้น
3. คำนวณ Quotient จากการนำ Dividend N มาคูณกับ X_S : $Q = NX_S$

Newton-Raphson Division

สมการที่ใช้ในการคำนวณหา $X_0, X_1, X_2, \ldots, X_S$ โดยที่ i เป็นจำนวนเต็ม

$$X_{i+1} = X_i - \frac{f(X_i)}{f'(X_i)} = X_i - \frac{1/X_i - D}{-1/X_i^2} = X_i + X_i(1 - DX_i) = X_i(2 - DX_i)$$

เนื่องจากวิธีนี้เป็นการลู่เข้า ทำให้สมการเป็นสมการกำลังสองแน่นอน จึงมีการคำนวณค่า S ที่เหมาะสมได้เป็น $S = \left\lceil \log_2 \frac{P+1}{\log_2 17} \right\rceil$ และ P คือจำนวนบิตสูงสุดที่คำนวณได้โดยที่ยังแม่นยำอยู่

Newton-Raphson Division

สรุปสูตรของ Newton–Raphson iteration ได้ว่า $X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)}$

ตัวอย่างจงหา $f(x) = \frac{1}{x} - b$ ที่ X_3และให้ b มีค่าเท่ากับ 7

หา $f'(x)$ เพื่อใช้ในการแทนลงในสูตร

$$f'(x) = -\frac{1}{x^2}$$

ดังนั้นเมื่อใช้สูตรจะแทนค่าได้ว่า $X_{n+1} = X_n - \frac{\frac{1}{X_n} - b}{\frac{1}{X_n^2}} * \frac{-X_n^2}{-X_n^2}$

$$= X_n - (-X_n + bX_n^2)$$

$$= 2X_n - bX_n^2$$

$$= X_n(2 - bX_n)$$

Newton-Raphson Division

หา $X_0 = \frac{1}{10}$

หา $X_1 = \frac{1}{10} * \left(2 - 7 * \frac{1}{10}\right) = \frac{1}{10}\left(\frac{20}{10} - \frac{7}{10}\right) = \frac{1}{10} * \frac{13}{10} = \frac{13}{100}$

หา $X_2 = \frac{13}{100} * \left(2 - 7 * \frac{13}{100}\right) = \frac{13}{100}\left(\frac{200}{100} - \frac{91}{100}\right) = \frac{13}{100} * \frac{109}{100} = \frac{1417}{10000}$

หา $X_3 = \frac{1417}{10000} * \left(2 - 7 * \frac{1417}{10000}\right) = \frac{1417}{10000}\left(\frac{20000}{10000} - \frac{9919}{10000}\right) = \frac{1417}{10000} * \frac{10081}{10000} = \frac{14284777}{100000000}$

ซึ่งถ้าเราสังเกต $\frac{1}{b} = \frac{1}{7} = 0.1428571428571429$

เมื่อเรานำมาเทียบกับ $X_3 = 0.14284777$ จะเห็นว่าหลังจุดทศนิยม 4 ตำแหน่งแรกมีค่าเท่ากัน

สรุปได้ว่าถ้าหากจำนวณ X_nเมื่อ n มากขึ้น ค่าความแม่นยำจะเพิ่มขึ้นด้วย

Variant Newton-Raphson Division

Variant Newton-Raphson เป็นการพัฒนาวิธี Newton-Raphson ให้เร็วขึ้นเล็กน้อยโดยการ shift N และ D เพื่อให้ D อยู่ในช่วง [0.5, 1.0] โดยจะเริ่มโดยให้

$$X := \frac{140}{33} + D \cdot \left(\frac{-64}{11} + D \cdot \frac{256}{99} \right)$$

นี่เป็นสมการกำลังสองที่เหมาะสมที่สุดกับ 1/D และมีค่า error น้อยกว่าหรือเท่ากับ 1/99

และการทำงานซ้ำในแต่ละรอบจะทำให้เกิด error ($Y{\cdot}E$ เป็น term ใหม่)

$$\begin{aligned} E &:= 1 - D \cdot X \\ Y &:= X \cdot E \\ X &:= X + Y + Y \cdot E. \end{aligned}$$

Variant Newton-Raphson Division

โดยจำนวนครั้งในการทำงานจะไม่เกิน

$$\left\lceil \log_3 \left(\frac{P+1}{\log_2 99} \right) \right\rceil$$

และ Quotient ของเลข P บิต จะมีค่าเป็น

$$Q := N \cdot X$$

Goldschmidt Division

Algorithm

Goldschmidt Division

Goldschmidt Division เป็นการหา Quotient ด้วยการคูณทั้ง Dividend(N) และ Divisor(D) ด้วย Factor F_i จนกว่า Divisor จะเข้าใกล้เลข 1 จึงจะหยุดคูณ โดยมีรูปสมการดังนี้

$$Q = \frac{N}{D}\frac{F_1}{F_1}\frac{F_2}{F_2}\frac{F_{\dots}}{F_{\dots}}.$$

Goldschmidt Division

ขั้นตอนในการหารด้วยวิธี Goldschmidt :

1. สร้างค่าประมาณของ Factor F_i
2. คูณทั้ง Dividend และ Divisor ด้วย F_i
3. ถ้า Divisor เข้าใกล้ 1 ให้หยุดทำและให้ Quotient มีค่าเท่ากับ Dividend แต่ถ้า Divisor ยังไม่เข้าใกล้ 1 ให้วนกลับไปทำข้อ 1 ใหม่

Goldschmidt Division

สมมติว่า N/D ถูกปรับอัตราส่วนเพื่อให้ 0<D<1 โดย F_i จะอิงตาม D ดังสมการ

$$F_{i+1} = 2 - D_i$$

และเมื่อคูณ N/D ด้วย F จะได้เป็น

$$\frac{N_{i+1}}{D_{i+1}} = \frac{N_i}{D_i}\frac{F_{i+1}}{F_{i+1}}.$$

Goldschmidt Division (Binomial Theorem)

เป็นการนำ Binomial theorem มาปรับใช้กับ Goldschmidt Division โดยสมมติว่า N/D ถูกปรับอัตราส่วนให้อยู่ในรูปของเลขยกกำลังของสอง (2^n)

โดยที่ $D \in (\frac{1}{2}, 1]$.

กำหนดให้ D = 1 – x และ $F_i = 1 + x^{2i}$

จะทำให้ได้สมการดังนี้

$$\frac{N}{1-x} = \frac{N \cdot (1+x)}{1-x^2} = \frac{N \cdot (1+x) \cdot (1+x^2)}{1-x^4} = \cdots = Q' = \frac{N' = N \cdot (1+x) \cdot (1+x^2) \cdots (1+x^{2^{(n-1)}})}{D' = 1-x^{2^n} \approx 1}.$$

Goldschmidt Division (Binomial Theorem)

จากสมการก่อนหน้าหลังจากทำเป็นจำนวน n ครั้ง $(x \in [0, \frac{1}{2}))$ จะเห็นว่าตัวหารด้านล่าง 1 - x^{2n} สามารถปัดเป็น 1 ได้ โดยจะมี Relative error เป็น

$$\varepsilon_n = \frac{Q' - N'}{Q'} = x^{2^n}$$

ซึ่งจะมีค่าสูงสุดเป็น 2^{-2n} เมื่อ x = ½ ดังนั้นมันจึงมีความแม่นยำสำหรับตัวเลขอย่างน้อย 2^n บิต