$$\therefore \quad k, \ ((\ln r - 1)/(\ln r)^2) = 0$$

$$\Rightarrow \quad (\ln r - 1) /(\ln r)^2 = 0$$

$$\therefore \quad \ln r = 1$$

$$r = e$$

โดยที่ e คือ ค่า 2.71828

ดังนั้น r ที่เป็นไปได้คือ 2 หรือ 3 ถ้า r เป็น 2 ก็คือเลือกเลขฐาน 2 และถ้าค่า r เป็น 3 ก็คือเลือกเลขฐาน 3 จะเห็นได้ว่า ทางที่เป็นไปได้ก็คือสร้างหน่วยความจำแต่ละเซลให้เป็นตามฐาน คือ ถ้าเป็น r = 2 ก็สร้างให้แต่ละเซลมีความสามารถแสดงได้ 2 หนทาง 0, 1 และเป็น r = 3 ก็จะต้องสร้างหน่วยความจำ คือแต่ละเซลให้มีความสามารถแสดงได้ถึง 3 หนทางคือ 0, 1, 2

ในความเป็นจริงจะมี magnetic device บางอย่างสามารถแสดงผลได้เป็นได้ถึง 3 ทาง ตัวอย่างเช่น square loop coil จะมีทิศทางปรากฏถึง 3 ทาง ถึงแม้ว่าจะมีหนทางในการสร้างอุปกรณ์ประเภทนี้ก็จริง แต่วิศวกรเลือกที่จะใช้ two-state system มากกว่า เพราะง่ายกว่าในการใช้งานและยังตรงกับเงื่อนไขอย่างอื่นในเรื่องเกี่ยวกับอุปกรณ์ไฟฟ้า ที่มีเพียง ON หรือ OFF ในขณะที่การตัดสินใจ ก็จะมีเพียง TRUE หรือ FALSE เท่านั้น

การคำนวณในระบบเลขฐาน 2

การบวก

การบวกในเลขฐานใดๆ ก็มีแนวคิดเดียวกับเลขฐาน 10 เพียงแต่ว่าเลขในแต่ละฐานนั้นจะมีจำนวนเลขแตกต่างกันไป

ตัวอย่างที่ 1

```
  11010010110  +
      1111011
  11100010001  <------ sum
      1111111  <------ carries
```

Addition of two single binary digits

1110

11010

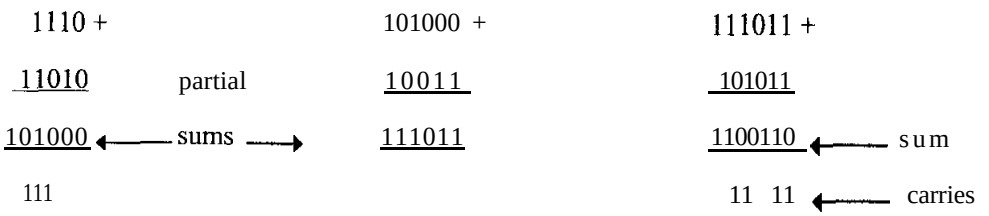
10011 +

101011

1100110 ← sum

22121 ← carries

รูปแสดงวิธีการบวก



การลบในระบบเลขฐานสอง

การลบเลขในระบบเลขฐานสองนั้น จะแตกต่างกับการบวก ข้อเท็จจริงอย่างหนึ่งที่เราจะต้องทราบก็คือ คอมพิวเตอร์นั้นจะทำการปฏิบัติการทางคณิตศาสตร์ทางตัวเลขได้เพียงอย่างเดียว คือการบวก ดังนั้น ปฏิบัติการทางคณิตศาสตร์อย่างอื่นๆ จำเป็นจะต้องเปลี่ยนให้เข้ามาสู่ระบบของการบวกเสมอ จึงจะคำนวณได้ ดังเช่น ปฏิบัติการลบ จะต้องใช้วิธีการของการสร้างตัวเลขส่วนเติมเต็ม (Complement) เข้ามาช่วย ความหมายของตัวเลขส่วนเติมเต็ม ก็คือ ถ้าเรามีเลขจำนวนหนึ่ง (100111) ฐาน 2 ดังนั้น one's Complement ของ100111 คือ 011000 (สังเกตได้ว่าหลักใดเป็น 1 Complement จะเป็น 0 และถ้าหลักใดเป็น 0 Complement จะเป็น 1)

ให้พิจารณาตัวอย่างการลบกันของเลขจำนวน ซึ่งดำเนินการได้ดังนี้คือ

110010 - 10001 = 110010 - 010001

= 110010 - (11111 - 101110)

= 110010 + 101110 - 11111

= 1100000 - (1000000 - 1)

= 1100000 - 1000000 + 1

= 100001

กระบวนการลบที่ปรากฏนี้ จะสรุปได้เป็นขั้นตอนดังนี้

110010 -	110010 +	
010001	101110	← complement
	11000 → 100000+	
	1	← end-around-carry
	100001	← result

ตัวอย่างต่อไปนี้จะแสดงวิธีการเก็บเลขจำนวนหนึ่งๆ ในระบบคอมพิวเตอร์

(a) + 192 = + 11000000 = + 0.11 x 2^8	(0) 11000000000	(0) 01000
(b) + 1920 = + 11110000000 = + 0.1111 x 2^{11}	(0) 11110000000	(0) 01011
(c) + 19.25 = + 10011.01 = 0.1001101 x 2^5	(0) 10011010000	(0) 00101
(d) - 192 = -11000000 = -0.11 x 2^8 = 1.01000000 x 2^8	(1) 01000000000	(0) 01000
(using twos complement)		
(e) - 19.5 = -10011.1 = -0.100111 x 2^5	(1) 01100100000	(0) 00101
= 1.011001 x 2^5		
(using twos complement)		
(f) -0.125 =-0.001 =-0.1 x 2^{-2} = 1.1 x 2^{-2}	(1) 10000000000	(1) 11110
(using twos complement for both mantissa and index)		
(g) + 0.046875 = + 0.000011 = + 0.11 x 2^{-4}	(0) 11000000000	(1) 11100

ข้อเปรียบเทียบระหว่างการใช้ one's Complement และ two's Complement

Ones complement	Twos complement
(1) Rules for complementation	
(a) Change 1's to 0 and O's to 1.	(a) Change l's to 0 and O's to 1.
	(b) Add 1 to least significant digit.
(2) Rules for addition with complementary numbers	
(a) Add in binary.	(a) Add in binary and discard the carry from the most significant digit.
(b) If there is a '1' carry from the most significant digit, add it to the least significant digit, and action any further carries if necessary.	
(3) Advantages and disadvantages	
(a) Simplest process for complementation.	(a) More complicated complementation because of need to add 1 to least significant digit.
(b) More complicated addition because of 'end-around-carry'.	(b) Addition simple because most significant carry discarded.
(c) Inconvenient for serial arithmetic because of 'end-around-carry'.	(c) Well adapted for serial arithmetic.
(d) Two values for zero, one negative and one positive.	(d) One value only for zero.
(e) Unsatisfactory for detection of overflow.	(e) Easy detection of overflow.
(f) Floating point arithmetic difficult because 'end-around-carry'. must be from most significant part to least significant part of the number.	(f) Floating point arithmetic simple to implement.

การคูณเลข หลักการคูณของเลขฐานใดๆ จะมีวิธีการเช่นเดียวกับเลขฐาน 10 ดูจากตัวอย่างต่อไปนี้

ตัวอย่างการคูณเลข

```
   1001                    11101
     11  x                 1101  x
   1001                    11101      (a)
  1001                     11101      (b)
 11011                    11101       (c)
                         101111001
```

มีข้อน่าสังเกตสำหรับการคูณเลขฐานจำนวน a ด้วยเลขจำนวน b โดยที่จำนวน b นั้นจะเป็นเลขที่อยู่ในรูปของเลข 2 ยกกำลัง x ก็หมายถึงการเคลื่อนย้ายบิทของเลข a ไปทางซ้ายมือ เท่ากับ x บิทแล้วเติม 0 เข้าไปเท่ากับจำนวนบิทที่ย้าย ดูจากตัวอย่างดังนี้

$1001 \times 10 = 10010$

$1001 \times 100 = 100100$

$1001 \times 1000 = 1001000$

จะเห็นได้ว่า 10 ก็คือ 2^1

100 ก็คือ 2^2

1000 ก็คือ 2^3

ดังนั้น การคูณด้วยเลข 2^1 ก็คือเพิ่มบิท 0 ทางขวามือไป 1 บิท และถ้าคูณด้วย 2^x ก็คือการเพิ่มบิท 0 ทางขวามือเป็นจำนวน x บิท เป็นต้น

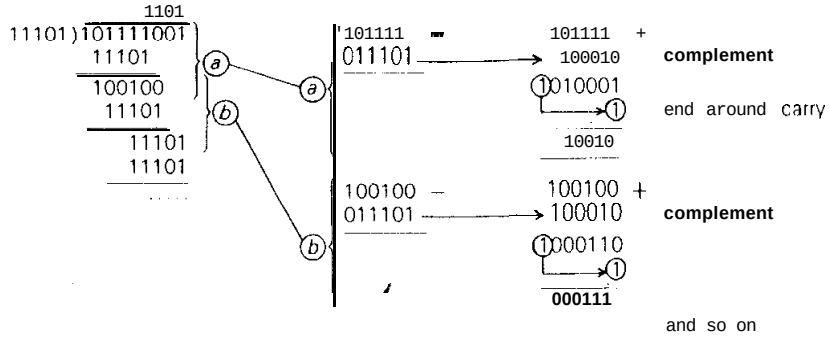
การหารเลข การหารเลขฐานสองนั้น ค่อนข้างจะยุ่งยากที่สุดในกระบวนการคำนวณทั้งหมด สิ่งที่เราจะต้องระลึกไว้อยู่เสมอ ก็คือ

$(0.1)_2 = (1/2)_{10}$

$(0.10)_2 = (1/4)_{10}$

$(0.100)_2 = (1/8)_{10}$

ตัวอย่างที่ 1 $(101111001)_2 \div (11101)_2$



$101111001_2 \div 11101_2 = 1101_2$

101111001, ÷ 11101, = 1101,

ตัวอย่างที่ 2

1. 0.1010, = $(1/2 + 1/8)_{10} = (5/8)_{10}$

ตัวอย่างที่ 3

2. $(1\ 1/32)_{10} = (8/32 + 2132 + 1/32)_{10} = (1/4 + 1116 + 1/32)_{10}$

$= 0.(1/2) + 1.(1/4) + 0.(1/8) + 1.(1/16) + 1.(1/32)$

$= 0.01011,$

ตัวอย่างที่ 4

3. $(1\ 1/16)_{10} = (101\ 1/10000)_2$

$= 0.1011,$

(An alternative method of calculating such fractions is binary long division as described in the previous section.)

การกำหนดตัวเลขในระบบคอมพิวเตอร์

(Number Representation)

การเก็บจำนวนเลขจำนวนใดจำนวนหนึ่งภายในพื้นที่ส่วนหนึ่งของสมองคอมพิวเตอร์นั้น จะมีแนวทางในการให้ความหมายของจำนวนเลขนั้นในรูปแบบของ sign และ magnitude โดยที่ sign หมายถึงเครื่องหมายทางคณิตศาสตร์ คือ + หรือ - ซึ่งในระบบเลขฐานสองนั้นจะกำหนดให้ sign + ก็คือเลข 0 ส่วน sign - ก็คือ 1 นั่นเอง

ส่วนวิธีการแทนความหมายของเลขลบอีกแบบที่เราดำเนินการผ่านมาแล้วในเรื่องของการลบก็คือ การแทนที่เลขลบนั้น ค่าของส่วนเติมเต็ม (Complement) ของมันนั่นเอง

ตัวอย่างต่อไปนี้จะแสดงวิธีการแทนค่าของเลขจำนวนบวกและลบ โดยมีข้อตกลงว่า ใช้ระบบ 6 บิท/word

Assuming a 6 bit word, write down as binary decimals the fractions $\frac{9}{16}, \frac{3}{4}, \frac{7}{8}$ and then write down $-\frac{9}{16}$, $-\frac{3}{4}$, $-\frac{7}{8}$ in twos complement form.

		2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
	$+\frac{9}{16}$ =	0	.	1	0	0	1	0
	$\frac{9}{16}$ (ones complement) =	1	.	0	1	1	0	1
$-\frac{9}{16}$ =	$\frac{9}{16}$ (twos complement) =	1	.	0	1	1	1	0
	$+\frac{3}{4}$ =	0	.	1	1	0	0	0
	$\frac{3}{4}$ (ones complement) =	1	.	0	0	1	1	1
$-\frac{3}{4}$ =	$\frac{3}{4}$ (twos complement) =	1	.	0	1	0	0	0
	$+\frac{7}{8}$ =	0	.	1	1	1	0	0
	$\frac{7}{8}$ (ones complement) =	1	.	0	0	0	1	1
$-\frac{7}{8}$ =	$\frac{7}{8}$ (twos complement) =	1	.	0	0	1	0	0

การเก็บจำนวนเลขในรูปของ Floating Point

การกำหนดพื้นที่เพื่อใช้เก็บข้อมูลที่เป็นตัวเลขนั้น เราสามารถกระทำได้ 2 แบบ โดยที่แบบแรกก็คือ การเก็บเลขจำนวนนั้นในรูปของเลขจำนวนเต็ม ซึ่งการจัดเก็บนั้น จะอยู่ในรูปของ Fixed Point คือรูปแบบที่ปรากฏในการจัดเก็บดังที่กล่าวมาแล้วในส่วนแรก ส่วนกรณีของการจับ

จองพื้นที่เป็นประเภทที่สองคือ Floating Point นั้น เราต้องการที่จะให้พื้นที่ๆ กำหนดนั้นสามารถจะเก็บได้ทั้งเครื่องหมาย เลขส่วนที่เป็นจำนวนเต็ม และทศนิยมพร้อมๆ กัน การเก็บเลขในรูปของ Fixed Point ข้อตกลงสำหรับพื้นที่ 7 บิท นั้นเป็นที่ทราบอยู่ว่าเราจะเลขบวกและลบได้ในช่วง -64 ถึง +63 ดังนั้นถ้าหากเลขเกินไปกว่านี้ เช่น 98 หรือ ต่ำกว่านี้ เช่น -100 จะจัดเก็บไม่ได้ ในทางปฏิบัติงานของคอมพิวเตอร์ จะเรียกสภาพนี้ว่า ความผิดพลาดประเภท Overflow หรือ Underflow นั่นเอง

การจัดเก็บข้อมูลประเภท Floating Point นั้น มีหลักการ ดังนี้เลขจำนวนใดจำนวนหนึ่งจะจัดเก็บเป็น Floating Point จะถูกแบ่งให้เป็น 2 ส่วน ดังนี้ $a \times x^b$ โดยที่ a ก็คือส่วนของ mantissa ที่มีค่าอยู่ระหว่าง $-1 \leq x < 1$ และ b เป็นเลขจำนวนเต็มใดๆ ซึ่งอาจจะเป็นลบหรือบวกก็ได้

ดังนั้น เลขจำนวน 96 ที่จะจัดเก็บรูปของ Floating Point จะถูกจัดการกำหนดส่วน ให้เป็นรูปแบบปรากฏดังนี้คือ

$$(96)_{10} = (1100000)_2$$
$$= .11 \times 2^7$$

	7
.110	0111

ตัวอย่าง $(128)_{10} = (100000000)_2$

$$= .10 \times 2^8$$

	8
.100	1000

โครงรูปโดยทั่วไปของ Floating Point จะเป็นรูปแบบดังนี้คือ

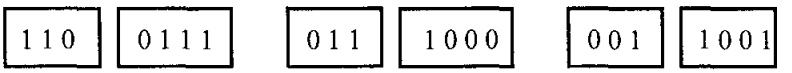
+ a	±b
mantissa	index

ตัวอย่างเช่น ถ้าเรากำหนดให้ 1 word ของเครื่องคอมพิวเตอร์ระบบหนึ่ง ประกอบด้วย 32 บิท ดังนั้น จำนวน 32 บิท จะถูกแบ่งออกมาเป็นสองส่วนคือส่วนแรก 23 บิท จะถูกใช้เป็นพื้นที่ส่วน a และส่วนที่สองคืออีก 9 บิทที่เหลือจะถูกกำหนดให้เป็นพื้นที่ส่วน b ดังนั้นขอบเขตของเลขที่

ปรากฏในพื้นที่ส่วน b ก็จะมีค่าอยู่ระหว่าง -2^8 ถึง $(2^9 - 1)$ (หรือจำนวน -256 ถึง +255 ในระบบเลขฐานสิบ) ดังนั้น ค่าของส่วนของ index จะมีค่าอยู่ระหว่าง 2^{-256} ถึง 2^{255}

ระดับความถูกต้องของการคำนวณตัวเลขโดยระบบคอมพิวเตอร์นั้นมีผลจากการกำหนดขนาดของบิทในการเก็บข้อมูล เพราะถ้าหากมีจำนวนบิทมาก พื้นที่ส่วนของ a และ b ก็จะมาก การเก็บส่วนที่เป็นเลขนัยสำคัญ (Significant number) ก็จะเก็บได้มาก ซึ่งผลจากการคำนวณก็จะได้ถูกต้องมากยิ่งขึ้น (ทั้งนี้เนื่องจากการคำนวณนั้นมักจะมีการปัดเศษทิ้งอันเนื่องจากจำนวนตัวเลขที่เกิดจากการคำนวณ จะมีขนาดยาวกว่าพื้นที่ๆ จะจัดเก็บนั่นเอง)

ในส่วนของพื้นที่ a คือ mantissa นั้น จะเริ่มจาก (0) 100 . . . 0 ถึง (0) 111 . . . 111 สำหรับเลขบวก และ (1) 000 . . . 000 จนถึง (1) 011 . . . 111 สำหรับเลขลบ โดยที่เลขที่อยู่ในวงเล็บหมายถึง เครื่องหมายบวกหรือลบดังที่อธิบายมาแล้ว ดูตัวอย่างวิธีการเก็บต่อไปนี้



(a) 110×2^7 (b) 011×2^8 (c) 001×2^9

โดยทั่วไป เราจะคำนวณค่าที่เป็นไปได้ของ mantissa จากสูตรต่อไปนี้

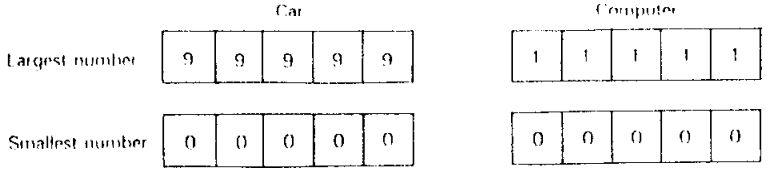
$(1/R) \leq a < 1$ โดยที่ R จะหมายถึง ฐานของข้อมูลนั้นๆ ตัวอย่างเช่น ถ้าเป็นระบบเลขฐาน 10 ค่าของ mantissa จะอยู่ในช่วง $.1 \leq |a| < 10$

ตัวอย่างการเก็บข้อมูล เช่นเลขจำนวนหนึ่ง ฐาน 10 มีค่าเท่ากับ -0.00405 จะถูกจัดเก็บในรูป -0.405×10^{-2} ส่วนเลข 23456.78 จะถูกจัดเก็บในรูปแบบของ 0.2345678×10^5 เป็นต้น

ตัวอย่างต่อไปนี้จะแสดงวิธีการเก็บ Floating Point Number คือ $a \times 2^b$ โดยมีข้อตกลงว่าจะใช้รีจิสเตอร์ที่มีส่วนของ mantissa เท่ากับ 12 บิทและส่วนของ index 6 บิท

Register capacity

The idea of a register is commonplace, and a typical electro-mechanical register is the mileometer in a car which operates in base 10. In a computer the register is electronic and operates in base 2. Figure 2.8 illustrates the limits of the register contents, and it is desirable



ความผิดพลาดอันเกิดจากการคำนวณ

ให้ย้อนกลับไปดูวิธีการแปลงเลขฐานใดๆ ในส่วนของเลขเศษส่วน จะเห็นได้ว่าบางครั้งเราไม่สามารถที่จะแปลงเลขจากฐานหนึ่งไปสู่อีกฐานหนึ่งได้อย่างถูกต้องครบถ้วน ขอให้พิจารณาดูจากตัวอย่างต่อไปนี้ ให้แปลง $(.6)_{10}$ ไปเป็นเลขฐานสอง

$0.6 = 3/5 = 11/101$

```
                0.10011001
          101 ) 11.00000
                101
                 1000
                  101
                 1000
                  101
                   11     the pattern repeats
                          itself
```

ถ้าเรานำเลข $(0.6)_{10}$ มาเก็บเป็นเลขฐาน 2 บนรีจิสเตอร์ซึ่งมี mantissa 8 บิท และ index 4 บิท เราจะได้ค่าดังกล่าวปรากฏดังนี้

$$0.1001 = 0.1001 \times 2^{0} \qquad \boxed{(0)\ 1001100} \quad \boxed{(0)\ 000}$$

ซึ่งในส่วนค่าของ mantissa นั้น ก็คือค่าของ 19/32 ซึ่งจะมีค่าแตกต่างจากค่าจริงไป เท่ากับ 3/5 - 19/32 = 1/160 ซึ่งคิดเป็นความผิดพลาดสูงถึง (1/160 - 3./5) x 100 ≈ 1% ซึ่งความผิดพลาดนี้จะส่งผลต่อไปถึงการคำนวณซึ่งยังไม่นับรวมถึงความผิดพลาดจากส่วนอื่นที่เกิดสะสมเพิ่มขึ้นอีกด้วย

เราจะวิเคราะห์โครงสร้างของความผิดพลาดอันเกิดจากการจัดเก็บ การคำนวณ การปัดเศษทิ้ง ได้ดังนี้

เลขจำนวนหนึ่งแบ่งเป็น 2 ส่วน โดยมีรูปแบบดังนี้คือ $x_3 = x_1 + x_2$ ----- (1)

กำหนดให้ประมาณค่าตัวเลขดังกล่าว และจัดเก็บไว้ในรูปแบบ ดังนี้คือ

$$y_3 = y_1 + y_2 \qquad \text{----- (2)}$$

(โดยที่ y_1, y_2 และ y_3 เป็นค่าประมาณที่จัดเก็บ)

นำ (2) - (1) จะได้ผลลัพธ์คือ

$$(y_3 - x_3) \approx (y_1 - x_1) + (y_2 - x_2)$$

ดังนั้น $A_3 \approx A_1 + A_2$

โดยที่ A_i หมายถึง ความผิดพลาดสัมพัทธ์ (Absolute Error) ในเทอมที่ i

ดังนั้น เราอาจจะเขียนรูปแบบความผิดพลาดได้ดังนี้คือ

$$|A_3| \approx |A_1| + |A_2|$$

ตัวอย่างต่อไปนี้จะแสดงผลจากความผิดพลาดในการตัดเศษ, ในการคำนวณของเลข 2 จำนวน คือ 8.42 + 1.357 = 9.777

กำหนดให้ A_1 : maximum error in 8.42 = 0.005

และ A_2 : maximum error in 1.357 = 0.0005

ดังนั้น maximum error ของค่า 9.777 ก็คือ

$$A_3 \leq A_1 + A_2$$

$$A_3 \leq .005 + .0005$$

$$\leq .0055$$

ดังนั้นค่าที่เกิดจากการคำนวณของ 2.42 + 1.357 จะมีค่าเป็นไปได้คือ 9.777 ± 0.0055 ดังนั้นค่าขั้นต่ำที่เป็นไปได้คือ 9.7715 และค่าขั้นสูงที่เป็นไปได้ก็คือ 9.7825 ขากขอบเขตของค่าขั้นต่ำและสูงที่เป็นไปได้ จะมีเลขนัยสำคัญ 2 หลักคือ 9.8

ปฏิบัติการทางคณิตศาสตร์ที่ทำให้เกิด relative error เพิ่มมากขึ้น ก็คือ การปฏิบัติการคูณนั่นเอง ซึ่งจะแสดง relative error ที่เกิดขึ้นได้ดังนี้

กำหนดให้ x_3 เป็นค่าจริงของเลขจำนวนหนึ่งที่แบ่งข้อมูลเป็น 2 ส่วน คือ x_1, x_2 และในทำนองเดียวกัน กำหนดให้ y_3 เป็นค่าที่ประมาณขึ้น โดยเกิดจากตัวเลขประมาณ 2 ส่วน คือ y_1 และ y_2 ดังนั้นจะได้ว่า

$$x_3 = x_1.x_2$$

และ $$y_3 = y_1.y_2$$

ดังนั้น $$y_3 \cdot x_3 = y_1.y_2 \cdot x_1.x_2$$

$$\Rightarrow \quad \frac{y_3 - x_3}{x_3} = \frac{y_1.y_2 - x_1.x_2}{x_3}$$

ดังนั้น $$\frac{y_3 - x_3}{x_3} \quad \frac{y_1.y_2 \cdot x_1.x_2}{x_1x_2}$$

กำหนดให้ R_3 คือ relative error product

$$R_3 = \frac{Y3 - x_3}{x_3}$$

$$\Rightarrow \quad R_3 = \frac{y_3 - x_3}{x_1x_2}$$

ในทำนองเดียวกัน

$$R_1 = \frac{y_1 - x_1}{x_1} \qquad \text{และ} \qquad R_2 = \frac{y_2 - x_2}{x_2}$$

ดังนั้น $y_1 = (R_1 + 1)x_1$ และ $y_2 = (R_2 + 1)x_2$ แทนค่า y_1 และ y_2 ในสมการของ R_3 จะได้ว่า

$$R_3 = \frac{(R_1 + 1)(R_2 + 1)x_1x_2 - x_1x_2}{x_1x_2}$$

$$\Rightarrow \quad R_3 = (R_1 + 1)(R_2 + 1) - 1$$

$$= R_1R_2 + R_1 + R_2$$

สืบเนื่องมาจากค่าของ R_1 และ R_2 คือ percentage error ซึ่งมีค่าน้อยมาก ดังนั้น $R_1R_2 \longrightarrow 0$ เราจึงตัด R_1R_2 ทิ้ง

$$\therefore \quad R_3 \leq R_1 + R_2$$

แต่ด้วยเหตุที่ว่า ค่าของ R_3 อาจจะเป็นบวกหรือลบก็ได้ ดังนั้นจะได้ว่า

$$|R_3| \leq |R_1| + |R_2|$$

ตัวอย่าง กำหนดให้นำเลข 2 จำนวนคือ 0.26 และ 1.35 นำมาคูณกัน จงวิเคราะห์หา relative error

" " $0.26 \times 1.35 = 0.3510$

กำหนดให้ relative error ของ 0.26 คือ $0.005/0.26 \approx 0.0192 = R_1$

และกำหนดให้ relative error ของ 1.35 คือ $0.0005/1.35 \approx 0.0037 = R_2$

ดังนั้น relative error ที่สูงที่สุด ที่เกิดจากการคูณของ 0.26 x 1.35 คือ

$$R_3 \leq R_1 + R_2$$

$$\leq 0.0192 + 0.0037$$

$$R_3 \leq 0.0229$$

แต่เนื่องจาก

$$R_3 = \frac{A_3}{\text{product}}$$

$$A_3 = R_3 \times (\text{product})$$

ดังนั้น

A_3 = 0.0229 x 0.3510

= 0.0080

โดยที่ product จะมีค่าที่เป็นไปได้อยู่ระหว่างค่าที่สูงสุดคือ 0.3430 และค่าที่สูงสุดคือ 0.3590 ดังนั้น ค่าของ product จะมีเลขนัยสำคัญตั้งแต่ 0 ถึง 1 หลัก ทั้งนี้เนื่องจากค่าจากการคำนวณจะมีขอบเขต ดังนี้ 0.351 ± 0.008

<u>ปฏิบัติการการเลื่อนบิท</u> (Shift bit operation) การคำนวณในเลขฐานสองนั้น จำเป็นจะต้องอาศัยปฏิบัติการที่เรียกว่า การเลื่อนบิทเข้ามาช่วย เพื่อประสิทธิภาพของการทำงานดังตัวอย่างที่เคยนำมาให้ดูในเรื่องของการคูณเลข

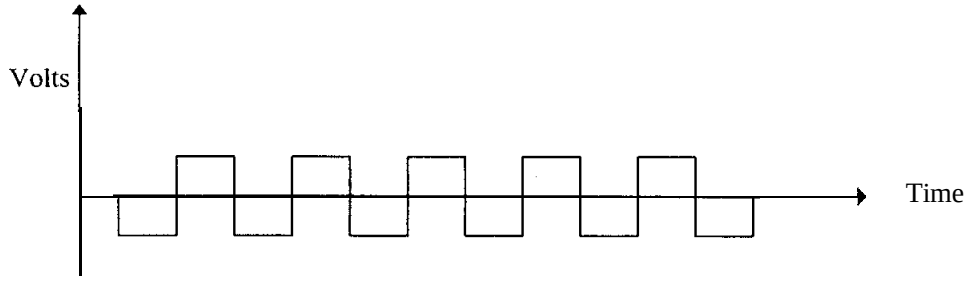
ดังนั้น สำหรับการคูณเลขฐานสองจึงมีวิธีดำเนินการดังนี้คือ

	Overflow	Double-length register	
Multiplicand (M)		11101	
Multiplier (L)		01101	
Add multiplicand M		11101	00000
* Shift partial product, a, right		01110	10000
** Shift right		00111	01000
Add M	1	00100	01000
* Shift right		10010	00100
Add M	1	01111	00100
* Shift right		10111	10010
** Shift right		01011	11001
Answer		1011	11011

จากรูป กำหนดให้ M และ L เป็นเลข 2 จำนวนที่จะทำการคูณกัน โดยขั้นตอนที่ปรากฏนั้นเป็นการปฏิบัติการในเครื่อง

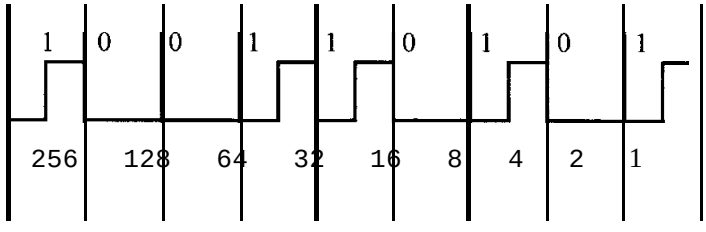
<u>วิธีการแทนข้อมูลในระบบคอมพิวเตอร์</u>

สัญญาณที่เปิดจากอุปกรณ์ทางอิเล็คโทรนิคส์นั้น จะนำใช้แสดงถึงข้อมูลที่ใช้ในระบบคอมพิวเตอร์ สัญญาณที่ปรากฏนั้นจะอยู่ในรูปแบบคล้ายรูปสี่เหลี่ยม โดยมี amplitude อยู่ระหว่าง 6 ถึง 10 โวลต์ ดังรูป



รูป 17.1 Pulse train

ภาพที่แสดงนั้น เรียกว่า pulse train จะเห็นได้ว่า สัญญาณนี้จะส่งผ่านสายไปในอัตราสม่ำเสมอ ดังนั้น เมื่อเมื่อเลขจำนวนหนึ่งถูกส่งผ่านสายไปนั้น จะหมายความว่า สัญญาณแรกสุด จะหมายถึงเลข 1 หลักในระบบเลขฐานสอง และลำดับถัดไป ก็จะหมายถึง เลขฐานสองลำดับถัดไปเรื่อยๆ เช่น เลข $(309)_{10} = (100110101)_2$ จะมีการส่งสัญญาณตามรูปที่ปรากฏต่อไปนี้



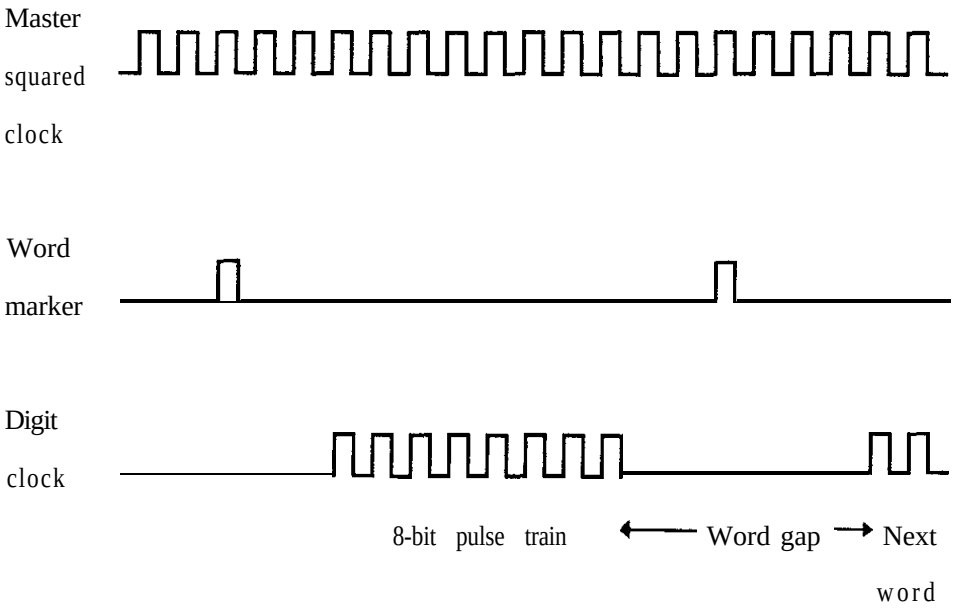
รูป 17.2 Pulse train representing 100110101

ความถี่ของ pulse train ที่ปรากฏบนระบบคอมพิวเตอร์นั้น อาจจะเป็นเพียงเลข 1 ตัว (one digit) หรือมากกว่าก็ได้ ในช่วงระยะเวลาต่อ 1 ไมโครวินาที beat ของ pulse time ซึ่งก็คือ beat ของระบบเครื่องคอมพิวเตอร์จะถูกนำไปเปรียบเทียบกับ regular heart beat ซึ่งในระบบเครื่อง

คอมพิวเตอร์ heart beat ก็คือ **clock rate** นั่นเอง โดยที่ **clock rate** นั้นจะถูกควบคุมโดย

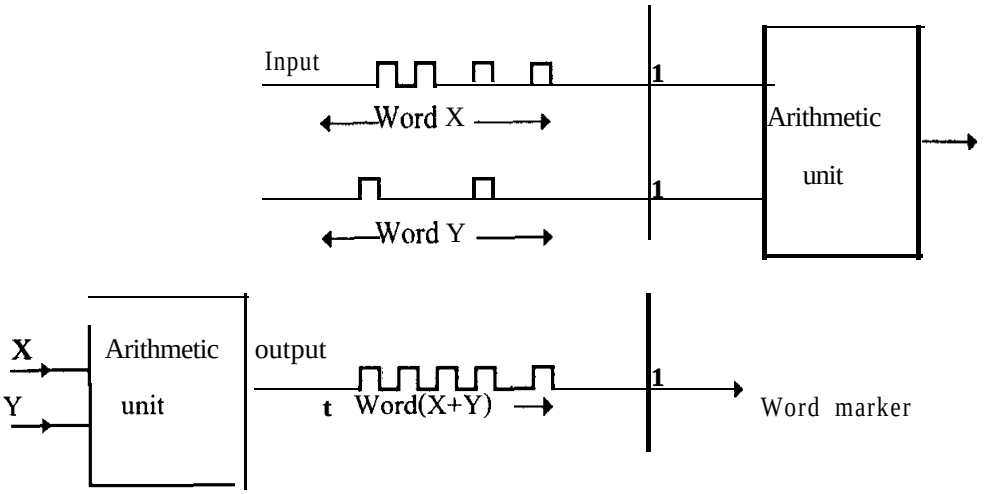
clock pulse generator ด้วยกลไกของการส่งสัญญาณและการรับสัญญาณไปปฏิบัติงานของระบบเครื่องคอมพิวเตอร์ จึงทำให้เราสามารถรับและอ่านสัญญาณ เพื่อแปลงเป็นข้อมูล นำไปประมวลผลต่อไปได้ ระบบคอมพิวเตอร์ซึ่งมีคุณลักษณะเป็น two-beat machine นั้น ในรุ่นแรกๆ จะมี clock rate เท่ากับ 500 kHz. ซึ่งหมายความว่า เราจะใช้เวลาในการส่งสัญญาณเท่ากับอัตรา**ส่วนกลับ** ของ 2 μs ตัวอย่างเช่น เวิร์ดหนึ่งที่มีความยาว 40 บิท จะใช้เวลาในการส่งสัญญาณ เท่ากับ 40 x 2μs = 80 μs คอมพิวเตอร์รุ่นหลังๆ มี clock rate สูงขึ้นถึง 100-200 Mhz. ซึ่งหมายถึงจะใช้เวลาในการส่งสัญญาณ 1 หลักเท่ากับ 100 ns หรือ 200 ns เป็นต้น

Word gap ระหว่าง pulse train นั้น ถูกกำหนดขึ้นเพื่อแบ่งช่วงระยะของการอ่านและการเขียน ทั้งนี้เพื่อแบ่งกลุ่มของแต่ละ pulse train การแบ่งกลุ่มเช่นนี้จะกระทำโดยการใช้ word marker ทั้งนี้เพื่อจะได้จำแนก 2 word ออกจากกัน ในขณะที่ส่งข้อมูลในรูปแบบของ pulse train ตัวอย่างต่อไปนี้จะแสดงปฏิบัติการคำนวณ เลข 2 จำนวน โดยที่แต่ละจำนวนจะอยู่ในรูปของ 8 bit/word



รูป 17.3 Master clock, word marker and digit clock

รูปต่อไปนี้จะแสดงการบวกของเลข 2 จำนวน คือ X และ Y



รูป 17.4 Addition of two numbers by arithmetic unit

แบบฝึกหัด

1. $(11101000111)_2 = (\ ?\ )_{10}$
2. $(15747)_8 = (\ ?\ )_{10}$
3. $(1BE7)_{16} = (\ ?\ )_{10}$
4. $(64E5)_{16} = (\ ?\ )_{10}$
5. $(64E5)_{16} = (\ ?\ )_{10}$ ทำโดยไม่ใช้ตารางแล้วเปรียบเทียบผลที่ได้กับวิธีใช้ตาราง
6. จงเปลี่ยน $(327)_{10}$ ให้เป็นเลขฐาน 2 เลขฐาน 8 และเลขฐาน 16
7. จงเปลี่ยน $(1BE7)_{16}$ ให้เป็นเลขฐาน 2 และเลขฐาน 8 (ใช้วิธีการเปลี่ยนเลขฐาน 16 ให้เป็นเลขฐาน 2 เสียก่อน แล้วจึงเปลี่ยนจากเลขฐานที่หาได้ไปเป็นเลขฐาน 8)
8. จงเปลี่ยน $(15747)_8$ ให้เป็นเลขฐาน 2 และเลขฐาน 16
9. $(64E5)_{16} = (\ ?\ )_2$
10. $(6C4E5)_{16} = (\ ?\ )_2$
11. $(64E5)_{16} = (\ ?\ )_8$
12. $(6C4E5)_{16} = (\ ?\ )_8$
13. $(100010101001)_2 = (\ ?\ )_{10}$
14. $(37456)_8 = (\ ?\ )_2$
15. $(112234)_8 = (\ ?\ )_2$
16. ในแต่ละข้อย่อย จงบวกเลขฐาน 2

    (a) 101110 (b) 111011 (c) 101010 (d) 100011 (e) 110101

    10110 11001 11011 10011 11111

17. Find the ones complement of the following binary numbers:

    (a) 10110 (b) 11001 (c) 11011 (d) 10011 (e) 11111

18. For each pair of binary numbers given in question 16, subtract the second number from the first.
19. Add the following sets of binary numbers:

    (a) 111011 (b) 1101 (c) 11111

    11001 11001 1001

    100011 11111 1110

    10011 1110 10001

20. Multiply the following pairs of binary numbers:

(a) 1001 (b) 1100 (c) 1100 (d) 101 10 (e) 10101

11 10 11 1 01 1 111

21. For each pairof binary numbers given in question 20, divide the second number into the first.

22. Convert the following binary decimals into denary fractions:

(a) 0.01 1 (bj 0.111 (c) 0.1100 (d) 0.0101 (e) 0.1 1111

23. Convert the following dcnary fractions to binary decimals:

(a) 9/16 (b) 13/32 (c) 7/8 (d) 25/32 (e) 15/16

24. Convert the following decimals to binary decimals:

(a) 0.53 175 (b) 0.28175 (c) 0.40675 (d) 0.9375 (e) 0.21925

25. Convert the following binary decimals to decimals:

(a) 0.1111 (b) 0.0101 (c) 0.10001 (d) 0.10101 (e) 0.11011

26. Convert the following octal numbers to binary:

(a) 5236 (b) 7025 (c) 3144 (d) 2617 (e) 4502

27. Convert the following binary numbers to octal:

(a) 1011011 (b) 10011 (c) 110010 (d) 1011011 (e) 1110101110

28. Convert the following denary numbers to octal and then binary:

(a) 192 (b) 265 (c) 312 (d) 495 (e) 518

29. Convert the following binary numbers to octal and then denary:

(a) 11101011 (b) 101 11001 (c) 11001001 (d) 1011101101 (c) 100001110

30. Perform the following binary additions for numbers in the range $-1 \leq x < 1$, written in twos complement form, and check their consistency by changing each line to the decimal cquivalcnt. Indicate, where appropriate, whether an overflow or underflow takes place.

(a) 0.1001 (b) 1.1001 (c) 0.1101 (d) 0.1111 (e) 1.0100

0.0101 1.1100 0.0011 1.0101 1.0011

3 1. Express the following 6 bit words, writtcti in twos complement, as (I) signed decimals in therange $-1 \leq x < 1$ and (ii) signed integers using the scaling factor 2^5.

(a) 011100 (b) 000111 (c) 111100 (d) 101010 (e) 110101 (f) 010011 (g) 011111

(h) 111000 (I) 101011 (j) 001111

32. In a 12 bit register, the first 8 bits are used to represent the mantissa and the last 4 the index, both in twos complement form. Write the following register contents in the form $ax\ 2^b$, and hence find the denary value.

(a)011000000010 (b) 010100000101 (c) 111000000110 (d) 101100100010

(e) 011100001100 (f) 101010001111

33. Write the following numbers in the floating point form $a \times 2^b$, where a is optimized. Insert the information in a 12 bit register such as that described in exercise 3.

(a) +96 (b) +69 (c) +30.5 (d) -1.25 (e) -0.046875 (f) -0.375 (g) +28.75

34. Using a register such as the one described in question 32, show its contents when recording the following numbers, and calculate the percentage error in each case.

(a) 0.4 (b) 1.4 (c) 4.7

35. Calculate the size of the absolute errors of the following expressions, in which the numbers have been rounded off, and hence round off the results to a meaningful number of figures.

(a) 4.53 + 7.234 (b) 0.32 x 1.48 (c) 6.543 + 3.2786 (d) 3.56 x 2.13

36. Using the layout of Fig. 2.11, perform the following multiplications:

(a) 11111 x 10110 (b) 10001 x 01101