



01076009

องค์ประกอบคอมพิวเตอร์และภาษาแอสเซมบลี

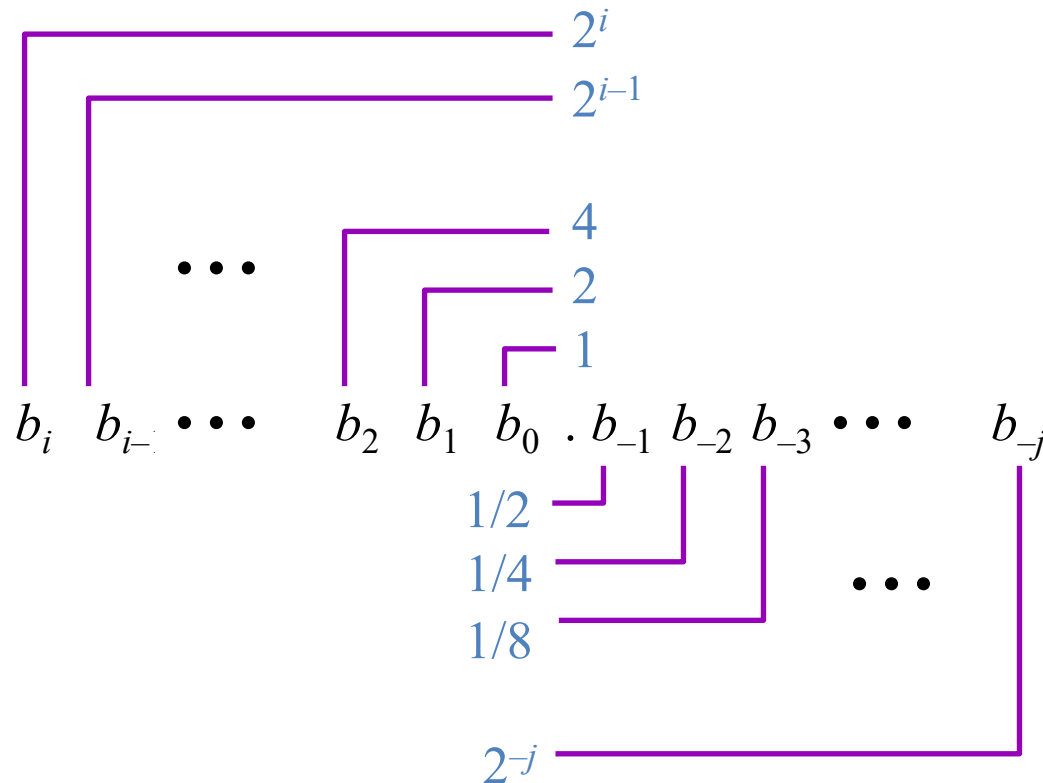
Computer Organization and Assembly Language

Floating Point



Floating Point

- นอกจากการประมวลผลเลขจำนวนเต็มแล้ว การประมวลผลเลขทศนิยมหรือจำนวนจริงก็มีความจำเป็นอย่างมาก ทุกภาษาก็มีการกำหนดตัวแปรเป็น float
- บิตด้านขวามีค่าเป็นกำลังสองของบิตด้านซ้ายเสมอ



Floating Point



$1/2 = 2^{-1} = 0.5$	$1/512 = 2^{-9} = 0.001953125$
$1/4 = 2^{-2} = 0.25$	$1/1024 = 2^{-10} = 0.0009765625$
$1/8 = 2^{-3} = 0.125$	$1/2048 = 2^{-11} = 0.00048828125$
$1/16 = 2^{-4} = 0.0625$	$1/4096 = 2^{-12} = 0.000244140625$
$1/32 = 2^{-5} = 0.03125$	$1/8192 = 2^{-13} = 0.0001220703125$
$1/64 = 2^{-6} = 0.015625$	$1/16384 = 2^{-14} = 0.00006103515625$
$1/128 = 2^{-7} = 0.0078125$	$1/32768 = 2^{-15} = 0.000030517578125$
$1/256 = 2^{-8} = 0.00390625$	$1/65536 = 2^{-16} = 0.0000152587890625$



Floating Point Example

- Value Representation
 - 5-3/4 101.11_2
 - 2-7/8 10.111_2
 - 63/64 0.111111_2
- ข้อสังเกต
 - เมื่อ shifting right จะเท่ากับการหารด้วย 2
 - เมื่อ shifting left จะเท่ากับการคูณด้วย 2
 - ตัวเลขที่อยู่ในรูป $0.111111..._2$ เมื่อเพิ่มจำนวนบิตไปเรื่อยๆ จะมีค่าเข้าใกล้ 1.0
 - $1/2 + 1/4 + 1/8 + ... + 1/2^i + ... \rightarrow 1.0$



Exercise

- จงแปลง 1.0001_2 ให้เป็นเลขฐาน 10
 $= 1, .001 = 1 \times 2^{-3} = .125$
 $= 1.125$
- จงแปลง 3.90625 ให้เป็นเลขฐาน 2
 $= 11, .90625 = 0.5 + 0.25 + 0.125 + 0.03125 = 0.11101$
 $= 11.11101$
- จงแปลง 1.11111_2 ให้เป็นเลขฐาน 10
 $= 1, .11111 = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125$
 $= 1.96875$



Floating Point

- ข้อจำกัด
 - สามารถแทนค่าตัวเลขได้เฉพาะที่เขียนในรูป $x/2^k$ ได้เท่านั้น
 - ตัวเลขบางค่าที่เป็นทศนิยมไม่รู้จบ ไม่สามารถแทนได้

Value

Representation

1/3

0.0101010101[01]...₂

1/5

0.001100110011[0011]...₂

1/10

0.0001100110011[0011]...₂



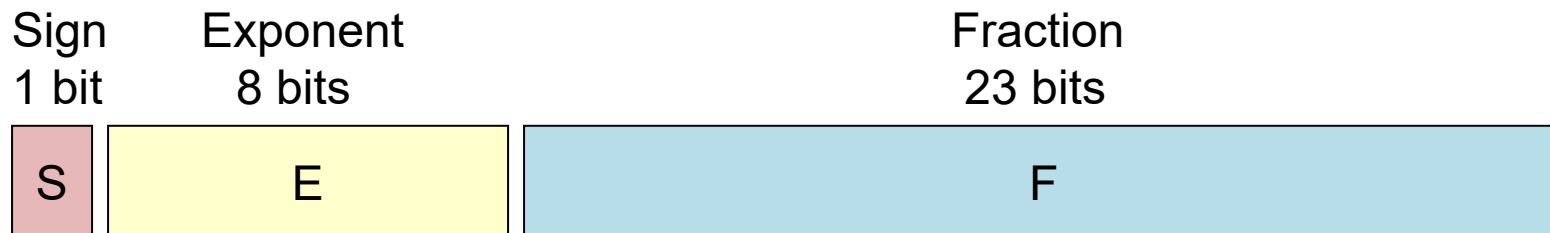
Floating Point

- การแทนทศนิยมมีหลากหลายแบบ แต่เพื่อให้เป็นแบบเดียวกัน คอมพิวเตอร์จะใช้การแทนที่เรียกว่า Normalized scientific notation ประกอบด้วย ตัวเลข 1 หลักหน้าจุด ที่เหลืออยู่หลังจุด และตามด้วยเลขยกกำลัง เช่น 3.5×10^9

$$1.010001 \times 2^5_{\text{two}} = (1 + 0 \times 2^{-1} + 1 \times 2^{-2} + \dots + 1 \times 2^{-6}) \times 2^5_{\text{ten}}$$

- การกำหนดรูปแบบที่เป็นมาตรฐาน จะทำให้การแลกเปลี่ยนข้อมูลระหว่างเครื่องทำได้ง่าย เครื่องคอมพิวเตอร์จึงมักใช้มาตรฐาน IEEE 754 ในการกำหนดว่าจะเก็บเลขทศนิยมอย่างไร

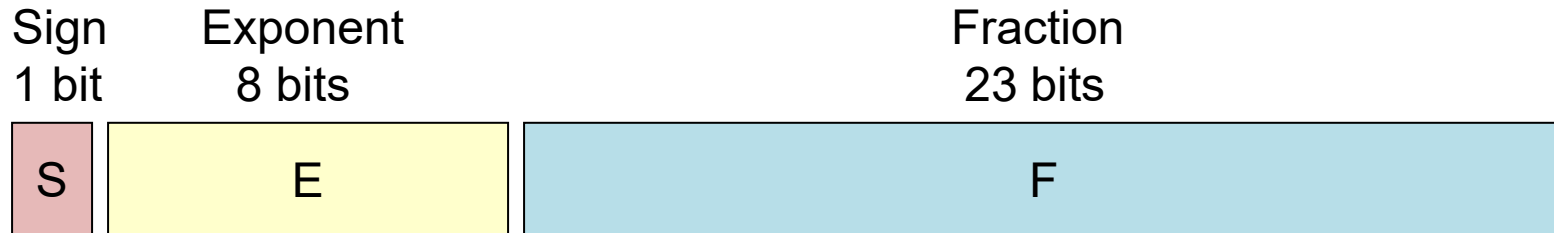
IEEE 754



- ค่าที่เก็บ = $(-1)^S \times F \times 2^E$
- หากเพิ่มจำนวนบิตใน Exponent จะทำให้ขนาดของข้อมูลที่เก็บได้เพิ่มขึ้น (กรณี 8 บิต จะได้ $2^{-127} - 2^{128}$ (ประมาณ $10^{-38} - 10^{38}$)
- หากเพิ่มจำนวนบิตใน Fraction จะทำให้สามารถเก็บค่าได้ละเอียดมากขึ้น
- เนื่องจากเป็น Normalize Number ดังนั้นตัวเลขจะอยู่ในรูปแบบ 1.xxxx (ฐาน 2) อย่างแน่นอน ดังนั้นใน IEEE 754 จึงละเลข 1 ออก ดังนั้นค่าที่เก็บจะได้จาก $(-1)^S \times (1+F) \times 2^E$

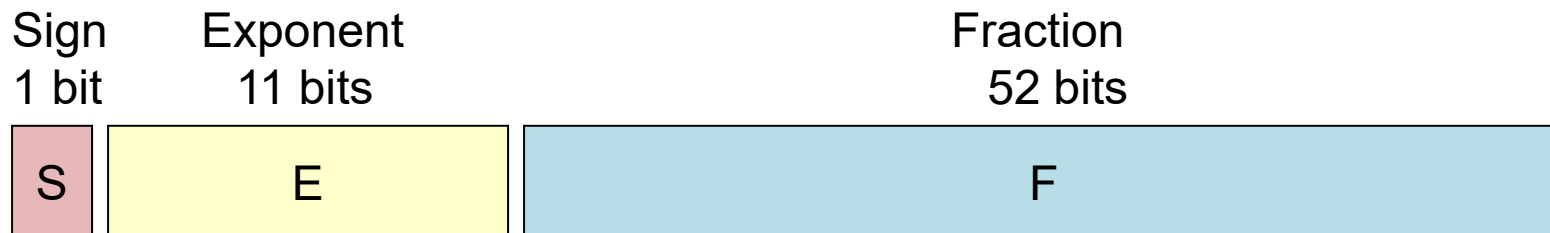


IEEE 754



- ค่าที่มากที่สุดที่สามารถแทนได้ คือ 2.0×2^{128}
- ค่าที่น้อยที่สุดที่สามารถแทนได้ คือ 1.0×2^{-127}
- Overflow คือ ตัวเลขที่มากกว่าค่ามากที่สุดที่เก็บได้
- Underflow คือ ตัวเลขที่น้อยกว่าค่าน้อยที่สุดที่เก็บได้
- มากกว่าและน้อยกว่า จะวัดจาก Exponent อย่างเดียว
- แต่จำนวนทศนิยมที่เก็บได้ จะวัดจาก Fraction

IEEE 754



- Double Precision คือ การเก็บเลขทศนิยมโดยใช้พื้นที่ขนาด 64 บิต หรือ รีจิสเตอร์ 2 ตัว
- ขอบเขตขนาดของข้อมูล คือ 1.0×2^{-308} ถึง 2.0×2^{308}
- และเนื่องจากขนาดของ Exponent มีค่าเพิ่มขึ้น จึงเก็บเลขทศนิยมได้เพิ่มขึ้นไปด้วย



Exponent Representation

- บิตแรก จะเป็นบิตเครื่องหมาย
- แต่ในส่วนของ exponent เนื่องจากต้องเก็บทั้งจำนวนบวกและจำนวนลบ แต่จะใช้ 2's Complement ไม่ได้ เนื่องจากไม่สามารถเปรียบเทียบมากกว่า/น้อยกว่าดังนั้นจึงใช้วิธี bias notation เพื่อให้ค่าที่น้อยที่สุดมีค่าเป็น 00...0 และค่าที่มากที่สุดเป็น 11...1
- Bias notation คือ การนำค่าที่เรียกว่า bias ไปลบออกจากค่าใน exponent เพื่อให้ได้ค่า exponent ที่แท้จริง (ช่วงข้อมูลจะเปลี่ยนเป็น $2.0 \times 2^{127} - 1.0 \times 2^{-126}$)
- IEEE 754 single-precision จะใช้ 127 เป็นค่า bias สำหรับ double precision จะใช้ค่า bias เป็น 1023

Final representation: $(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$



Example

Final representation: $(-1)^s \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- ให้แทน -0.75 ในรูปแบบ in single precision
 - Single (1 + 8 + 23)
 - Double (1+ 11 + 52)
- เปลี่ยน 0.75 เป็นฐาน 2 = $2^{-1} + 2^{-2} = 0.11_2$
- Normalize = 1.1×2^{-1} ดังนั้นส่วน fraction = 1000...000 และส่วน exponent = $-1+127 = 126 = 0111\ 1110$
- ดังนั้นจะได้ Single Precision ดังนี้ **1 0111 1110 1000...000**



Example

Final representation: $(-1)^s \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- ให้แทน 32 ในรูปแบบ in single precision
- เปลี่ยน 32 ให้เป็นฐาน 2 = 100000_2
- เขียนในรูปแบบ Normalize = 1×2^5
- ดังนั้นส่วน Fraction จะเป็น 0000...000
- ส่วน Exponent = $5 + 127 = 132 = 10000100$
- ดังนั้นจะได้ Single Precision ดังนี้ **0 1000 0100 0000...000**



Exercise

Final representation: $(-1)^s \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- ให้แทน 0.0625_{10} ในรูปแบบ in single precision
 - $2^{-4} = 0.0001 = 1.0 \times 2^{-4}$ ดังนั้นส่วน Fraction = 0 ส่วน Exponent = $-4+127$
= 0 01111011 0000...000
- ให้แทน -26.625_{10} ในรูปแบบ in single precision
 - $26 = 11010$, $0.625 = 0.5+0.125 = 0.101 = 11010.101$
 - 1.1010101×2^4 ส่วน Fraction = 1010101 ส่วน Exponent = $4+127 = 131$
 - 1 10000011 1010101...000



Example

Final representation: $(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- ให้อหว่าเลขฐาน 2 ในแบบ Single Precision แทนค่าเลขใด

1 1000 0001 01000...0000

- Singed Bit เป็นลบ
- ส่วนของ Fraction คือ 1.01
- ส่วนของ Exponent = $129 - 127 = 2$
- รวมกัน = $-1.01_2 \times 2^2 = -101_2 = -5$



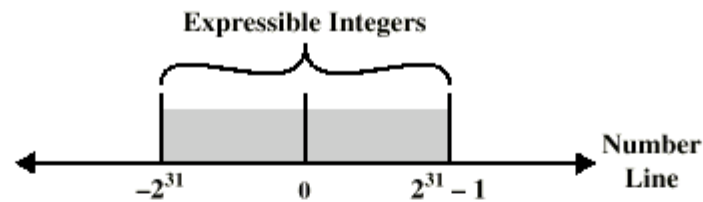
- จงแปลง IEEE 754 ต่อไปนี้ ให้เป็นเลขฐาน 10

-3.75

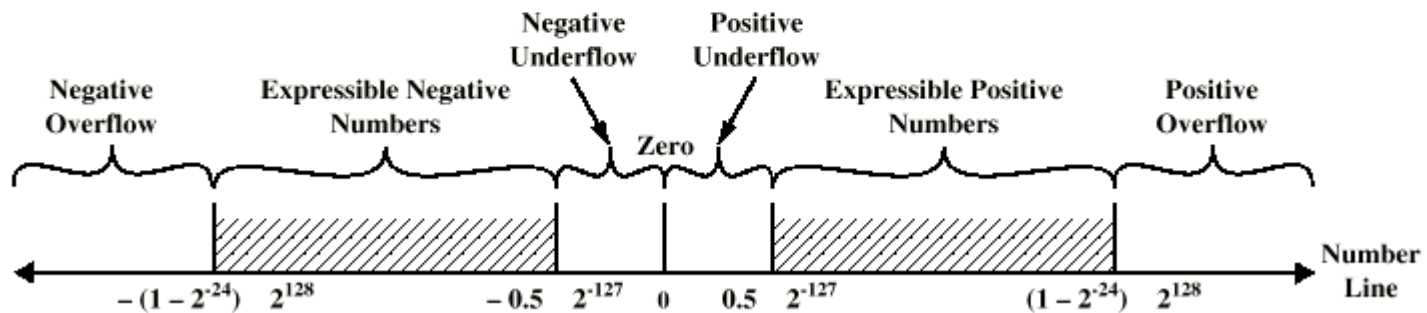
$$\text{Exponent} = 162 - 127 = 35$$

$= 55834574848$

Expressible Numbers



(a) Two's Complement Integers



(b) Floating-Point Numbers



FP Addition

- ลองพิจารณาการบวกเลขฐาน 10 ในแบบ Normalize (กำหนดให้เก็บทศนิยมเพียง 4 ตำแหน่ง เพื่อให้ง่าย)
 - $9.999 \times 10^1 + 1.610 \times 10^{-1}$
 - เริ่มต้นต้องปรับส่วนของเลขยกกำลัง (Exponent) ให้มี magnitude เดียวกัน โดยยึดค่าของเลขยกกำลังที่มากกว่า
 - $9.999 \times 10^1 + 0.016 \times 10^1$
 - จากนั้นก็บวกเลข 2 จำนวนเข้าด้วยกัน = 10.015×10^1
 - และ Normalize = 1.0015×10^2
 - เนื่องจากกำหนดให้มีทศนิยมเพียง 4 ตำแหน่ง จึงปัดเศษ = 1.002×10^2



FP Addition

- กลับมาพิจารณาการบวกเลข FP ที่เป็นฐาน 2 วิธีการก็จะคล้ายกับฐาน 10
 - $1.010 \times 2^1 + 1.100 \times 2^3$
 - ปรับส่วนของเลขยกกำลังให้มี magnitude เดียวกัน
 - $0.0101 \times 2^3 + 1.1000 \times 2^3$
 - ทำการบวก ได้ 1.1101×2^3
 - เนื่องจากอยู่ในรูปของ Normalize Form แล้ว จึงไม่ต้อง Normalize อีก
 - ตรวจสอบ Overflow / Underflow ซึ่งกรณีนี้ไม่เกิด
 - ทำการปัดเศษ (Round) ซึ่งกรณีนี้ไม่ต้องทำ
 - จากนั้นแปลงเป็น IEEE 754

0 10000010 110100000000000000000000

FP Addition



- สรุปขั้นตอนการบวกเลขทศนิยม
 1. เปรียบเทียบ Magnitude ของส่วนยกกำลัง ถ้าไม่เท่ากันก็ปรับให้เท่ากัน โดยยึดค่าของเลขยกกำลังที่มากกว่า
 2. บวกตัวเลขส่วนที่ไม่ยกกำลังเข้าด้วยกัน
 3. ทำการ Normalize
 4. ตรวจสอบ Overflow, Underflow -> Exception
 5. ปิดเศษ
 6. ตรวจสอบว่ายังต้อง Normalize หรือไม่ ถ้าต้องกลับไปข้อ 3



Example

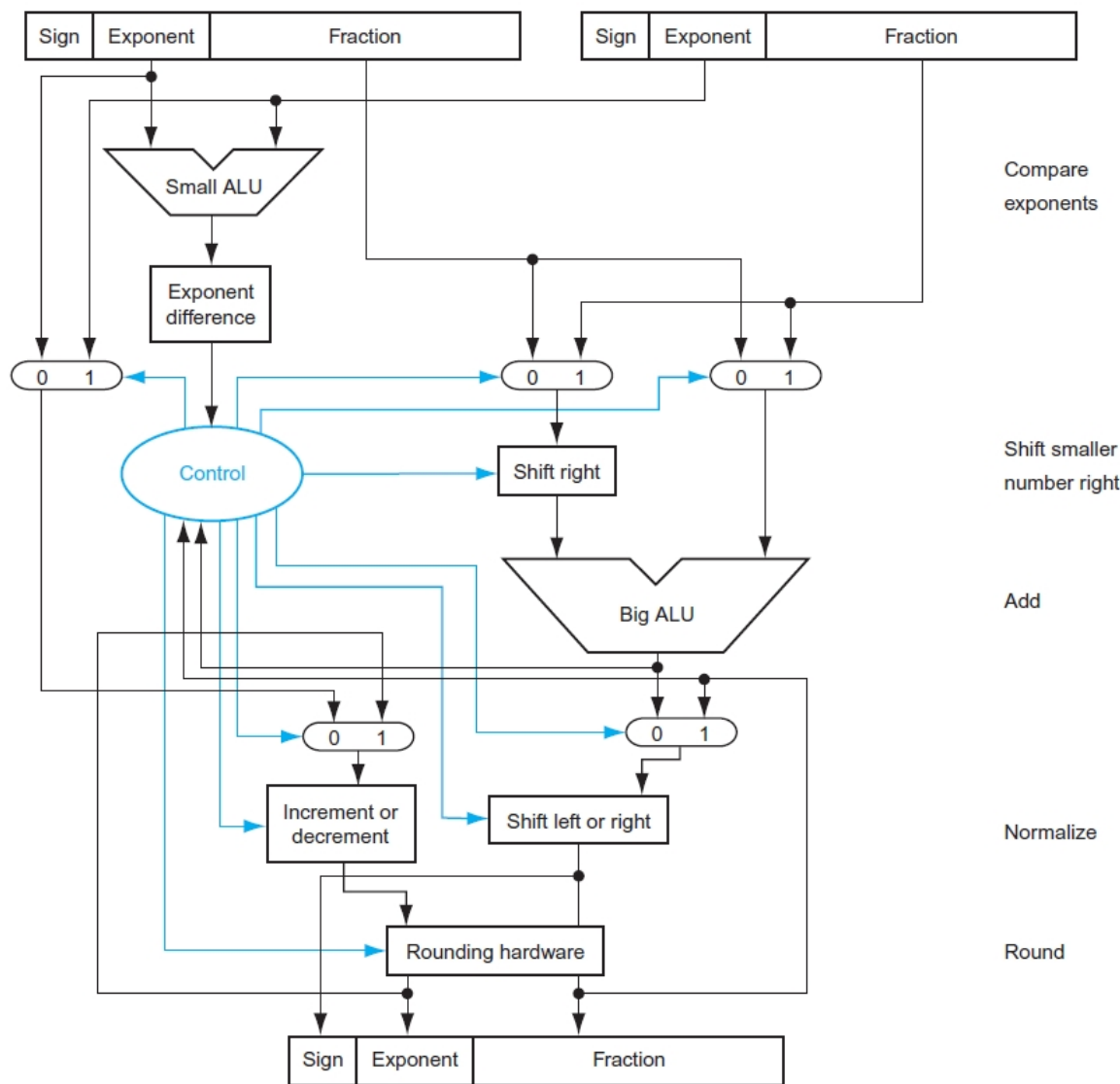
- ใ้ห้บวก 0.5_{10} กับ -0.4375_{10}
- $0.5_{10} = 0.1_2 = 0.1_2 \times 2^0 = 1.000_2 \times 2^{-1}$
- $-0.4375_{10} = -0.0111_2 = -1.110 \times 2^{-2}$
- ขั้นที่ 1 เปลี่ยนให้ magnitude เท่ากัน
— $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
- ขั้นที่ 2 บวกเข้าด้วยกัน $= 0.001 \times 2^{-1}$
- ขั้นที่ 3 Normalize และตรวจสอบ Overflow, Underflow $= 1.000 \times 10^{-4}$
- ขั้นที่ 4 ปิดเศษ (ได้เท่าเต็ม)
- $1.000 \times 10^{-4} = 0.0001_2 = 1/2^4 = 1/16 = 0.0625 = 0.5 - 0.4375$



Exercise

- ใ้ห้บวก 5.66015625 และ 8.59375
 - $5.66015625 = 101.10101001 = 1.0110101001 \times 2^2$
 - $8.59375 = 1000.10011 = 1.00010011 \times 2^3$
 - เริ่มจากปรับส่วน magnitude โดยปรับมาที่ 2^3
 - $= 1.00010011 + 0.10110101 \times 2^3$ (ปัดเศษ)
 - $= 1.11001 \times 2^3$
 - เป็นตัวเลขที่ Normalize แล้ว และไม่ Overflow, Underflow
 - 14.25

FP Addition





FP Multiplication

- สำหรับการคูณแล้ว ขั้นตอนจะเป็นดังนี้
 - คำนวณส่วนของเลขยกกำลัง (Exponent)
 - คูณส่วนที่ไม่ยกกำลัง (Significand)
 - Normalize
 - ปิดเศษ
 - กำหนดเครื่องหมาย



Example

- หาผลคูณของ $1.110_{10} \times 10^{10} \times 9.200_{10} \times 10^{-5}$
- เริ่มจากส่วน exponent : $10 + (-5) = 5$
- แต่เนื่องจากส่วน exponent เก็บแบบ biased notation ซึ่ง $10 + 127 = 137$ และ $-5 + 127 = 122$ ดังนั้น $137 + 122$ จะได้ 259 ซึ่งไม่สามารถเก็บใน 8 บิตได้ ดังนั้นให้ลบออกด้วย 127 = $259 - 127 = 132$ ($5 + 127$)
- ต่อไปเป็นการคูณส่วน significand โดย $1.110 \times 9.200 = 10.212000$
- ดังนั้นผลลัพธ์ = 10.212×10^5 เมื่อ Normalize จะได้ 1.0212×10^6
- เมื่อปัดเศษให้เหลือ 3 หลัก จะได้ 1.021×10^6
- สุดท้ายใส่เครื่องหมายจะได้ $+1.021 \times 10^6$



Example

- หาผลคูณของ $0.5_{10} \times -0.4375_{10}$
- $0.5 = 1.000_2 \times 2^{-1}$ และ $-0.4375_2 = -1.110 \times 2^{-2}$
- เริ่มจากบวกส่วน exponent = $-1 + (-2) = -3$
- เมื่อเขียนในแบบ biased notation = $(-1+127) + (-2+127) - 127 = (-1-2) + (127 + 127 - 127) = -3 + 127 = 124$
- คูณส่วน significand 1.000×1.110 คูณแบบจำนวนเต็ม = 1110000 จากนั้น
ค่อนนำทศนิยมมาใส่ = $1.110000_2 \times 2^{-3}$
- ปิดเศษให้เหลือ 3 บิต = $1.110_2 \times 2^{-3}$
- ตัวเลขข้างต้น Normalize แล้ว และไม่ Overflow หรือ Underflow
และไม่จำเป็นต้องปิดเศษอีก จึงได้คำตอบเท่ากับ $1.110_2 \times 2^{-3}$ (-0.21875)



Exercise

- ให้คูณ 5.66015625 และ 8.59375
 - $5.66015625 = 101.10101001 = 1.0110101001 \times 2^2$
 - $8.59375 = 1000.10011 = 1.00010011 \times 2^3$
 - เริ่มจากบวกส่วน exponent $= 2 + 3 = 5$
 - เขียนในแบบ biased notation $5+127 = 132$
 - คูณส่วน significand $1.0110101 \times 1.00010011 = 1.100001001101111$
 - ปิดเศษให้เหลือ 3 บิต $1.100001001101111 \times 2^3$
 - ตัวเลขข้างต้น Normalize แล้ว และไม่ Overflow หรือ Underflow
 - จึงได้คำตอบเท่ากับ $1.100001001101111_2 \times 2^3 = 48.6084$



FP Instruction in ARM

- ใน Raspberry Pi จะมี Coprocessor ที่ทำหน้าที่คำนวณ Floating Point มีชื่อว่า VFPv3
- คำสั่ง FP จะไม่ได้ใช้ register ชุดเดียวกับการคำนวณแบบจำนวนเต็ม โดยจะมีการสร้าง register ขึ้นมาอีกชุด คือ s0-s31 สำหรับใช้กับ single precision และ d0-d15 สำหรับ double precision
- ดังนั้นก่อนที่จะคำนวณจะต้องโหลดข้อมูลมาใส่ Sx เสียก่อน

LDR r6, =fvalue

VLDR s0, [r6] ; ตัวตั้ง

VLDR s1, [r6, #4] ; ตัวคูณ

VMUL.f32 s0, s0, s1

VMOV r0, s0 ; เก็บผลลัพธ์



FP Instruction in ARM

- Addressing Mode จะใช้บาง Addressing Mode ไม่ได้

VLDR	s16, [r6, #4]	; ได้
VLDR	s16, [r6, #4]!	; ไม่ได้
VLDR	s16, [r6] #4	; ไม่ได้

- กรณีต้องการคำนวณแบบ Double Precision

LDR	r6, =fvalue	
VLDM	r6!, {D0, D1}	; ตัวตั้ง และ ตัวคูณ
VMUL.f64	d0, d1	
VCVT.f32.f64	s0, d0	; แปลง 64 -> 32
VMOV	r0, s0	



FP Instruction in ARM

- คำสั่งที่ใช้ได้

VADD.f32	s0, s1, s2	; s0 = s1+s2
VSUB.f32	s0, s2, s4	; s0 = s2 - s4
VDIV.f64	d4, d5, d1	; d4 = d5/d1
VMUL.f32	s2, s4, s1	; s2 = s4 * s1
VNMUL.f64	d4, d3, d2	; d4 = -(d3*d2)
VMAL.f64	d4, d3, d2	; d4 = d4+(d3*d2)
VABS.f32	s0, s1	; s0 = abs(s1)
VNEG.f32	s2, s3	; s2 = - s3
VSQRT.f64	d0, d1	; d0 = sqrt(d1)
VCMP.f32	s0, s1	
VMRS	APSR_nzcv, FPSCR ; copy status	



Fixed Point

- การคำนวณแบบเลขทศนิยมนั้น จะช้ากว่าการคำนวณแบบจำนวนเต็มมาก
- ดังนั้นในกรณีที่ทำงานกับเลขทศนิยมน้อยๆ อาจใช้วิธีที่เรียกว่า Fixed Point
- เช่น 0.5×0.9 ก็ทำการคูณ 0.5 ด้วย 10 และคูณ 0.9 ด้วย 10 จะได้เป็น 5 กับ 9 จากนั้นก็คำนวณแบบปกติ เมื่อได้ผลลัพธ์ออกมาแล้ว จึงค่อยหารกลับไปเป็นทศนิยมอีกครั้ง
- การทำงานแบบนี้ จะทำให้การคำนวณทำได้เร็วขึ้นมาก



Homework #5

- กำหนดตัวเลข 2 จำนวน 6.18×10^2 และ 5.796875×10^1
- ให้แสดงการบวกเลข 2 จำนวนตามแบบ IEEE 754 ตามขั้นตอน
- ให้แสดงการคูณเลข 2 จำนวนตามขั้นตอน
- ให้บอกด้วยว่าผลการทำงาน เกิด Overflow หรือ Underflow หรือไม่



For your attention