Chapter 3

Arithmetic for Computers
(Only the Part on Floating Point)

Contents - Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding

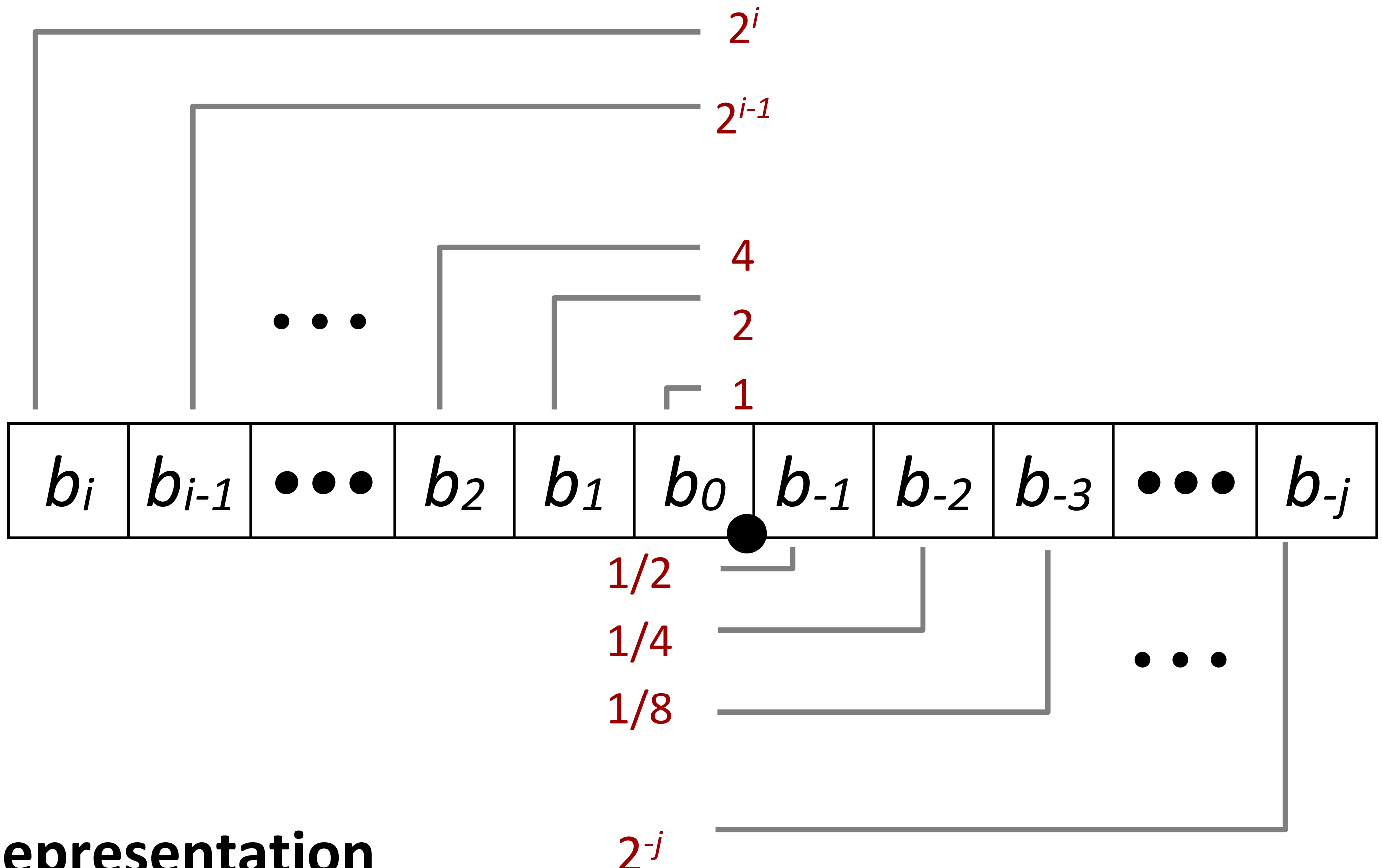
Contents - Floating Point

- **Background: Fractional binary numbers**
- IEEE floating point standard: Definition
- Example and properties
- Rounding

Fractional binary numbers

- **What is 1011.101_2?**

Fractional Binary Numbers



- **Representation**
  - Bits to right of “binary point” represent fractional powers of 2
  - Represents rational number:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Fractional Binary Numbers: Examples

- **Value** | **Representation**

Value	Representation
5 3/4	101.11_2
2 7/8	10.111_2
63/64	1.0111_2

- **Observations**
  - Divide by 2 by shifting right
  - Multiply by 2 by shifting left
  - Numbers of form $0.111111\ldots_2$ are just below 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Use notation $1.0 - \varepsilon$

Representable Numbers

- **Limitation**
  - Can only exactly represent numbers of the form $x/2^k$
  - Other rational numbers have repeating bit representations

- **Value** **Representation**
  - 1/3 $0.0101010101[01]\ldots_2$
  - 1/5 $0.001100110011[0011]\ldots_2$
  - 1/10 $0.0001100110011[0011]\ldots_2$

Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding

IEEE Floating Point

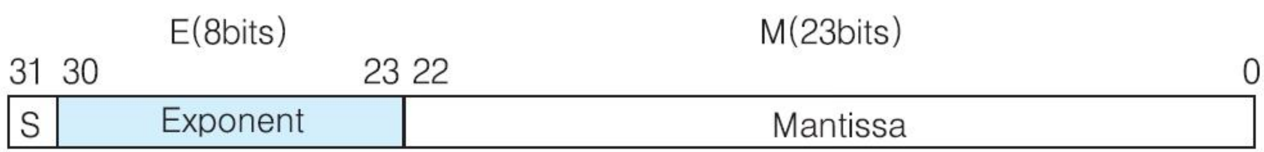
- **IEEE Standard 754**
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs

- **Driven by numerical concerns**
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard

(참고슬라이드) 부동소수점

❖ 실수

- 소수점의 위치를 고정하지 않는 부동 소수점(floating point) 표현법 사용
- 국제 표준 IEEE 754
- 헤더 파일 float.h에 표현 범위 등과 관련한 정보가 정의됨
- 단정도 실수 표현
  - 첫 번째 비트를 부호(S) 표현에 사용
  - 나머지 8개 비트를 지수(E) 표현에 사용
    - 지수 표현 범위: -127 ~ 128
  - 지수도 부호가 있을 수 있으므로 E=127을 0으로 취급(biased-127)
  - 1.M 형태로 정규화해서 표현하며, 1.은 실제로는 표현하지 않고M만 23비트로 표현



[그림 2-9] 부동 소수점에 의한 실수(float) 표현

(참고슬라이드) 부동소수점

[예제 2-5] 실수 -0.75의 단정도 부동 소수점 2진수 표현

1. -0.75를 2진수로 표현하고 정규화

   $- 0.11 = -1.1 \times 2^{-1}$

   $X = (-1)^{1} \times 1.1 \times 2^{-1+127}$

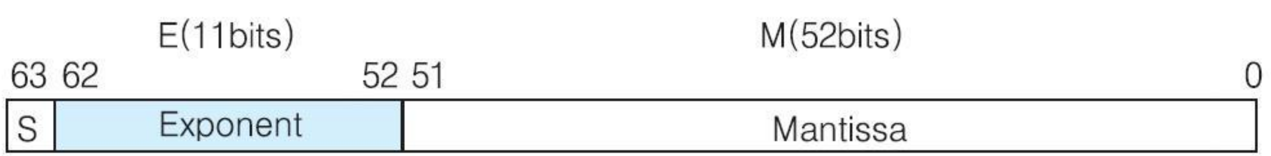
2. 식과 비교해 보면 S = 1, E = 0111 1110, M = 100 0000 0000 0000 0000 0000이므로, -0.75의 단정도 2진수 표현은 1011 1111 0100 0000 0000 0000 0000 0000이며, 16진수 0xBF400000

- 배정도 실수 표현
  - 첫 번째 비트를 부호(S) 표현에 사용
  - 나머지 11개 비트를 지수(E) 표현에 사용
    - 지수 표현 범위: -1023 ~ 1024
  - 유효 자리는 1.M 형태로 정규화해서 표현하며, 1.은 표현하지 않고 M을 52비트로 표현

S : sign bit(0:양수, 1: 음수)

E : 11bits Exponent(지수 부분), Biased−1023

M : 52bits Mantissa(Normalized fraction with hidden 1)



(참고슬라이드) 부동소수점

[예제 2-6] 실수 -0.75의 배정도 부동 소수점 2진수 표현

1. -0.75를 2진수로 표현하고 정규화
   $X = (-1)^{1} \times 1.1 \times 2^{-1+1023}$

2. 식과 비교해 보면 S = 1, E = 011 1111 1110, M = 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000이므로, -0.75의 배정도 부동 소수점을 16진수로 표현하면 0xBFE8000000000000

특수 값	조건	비고
Zero	E: 모든 비트가 0 M: 모든 비트가 0	+0과 −0이 존재
Denormalized numbers	E: 모든 비트가 0 M: 모든 비트가 0은 아님	정규화되지 않은 수의 해석: $-X = (-1)^{s} \times 2^{-126} \times 0.M$(float) $-X = (-1)^{s} \times 2^{-1022} \times 0.M$(double)
Infinity	E: 모든 비트가 1 M: 모든 비트가 0	+Infinity, −Infinity
NAN(Not A Number)	E: 모든 비트가 1 M: 모든 비트가 0은 아님	

(참고슬라이드) 부동소수점

- **실수 표현의 유효 자리**
  - **실수를 제한된 비트**(32, 64)**로 표현하면 모든 실수를 정확하게 표현할 수 없게 되어**, **실제** 10**진수와 부동 소수점에 의한** 2**진 비트로 표현된 수 사이에 오차가 발생할 수 있음**
  - 10**진수 유효 자리**(significant decimals): **부동 소수점 수가 해당 유효 자리까지는 실제** 10**진수와 일치함을 나타냄**
    - 단정도 실수(float): 10진수 유효 자리 6
      - 10진수로 변환했을 때 6자리까지는 신뢰할 수 있음
    - 배정도 실수(double): 10진수 유효 자리 15
      - 10진수로 변환했을 때 15자리까지는 신뢰할 수 있음
  - **실수는 제한된 비트의 부동 소수점 표현으로 정확히 표현되지 않기 때문에 두 실수가 같은지 비교할 때 주의 필요**
    - 두 실수가 응용 프로그램에서 요구하는 오차 범위 안에 있으면 같다고 판단하거나, 유효 자리까지 같으면 같다고 판단해야 함
      - 10진수 0.1을 float로 표현하면 0x3DCCCCCD이고, 이를 다시 10진수로 소수점 이하 20자리까지 출력하면 0.10000000149011612000이 출력되어 오차 발생
      - 유효 자리 6까지만 보면 0.100000으로 0.1과 같음. 즉, 10진수 0.1과 0.10000000149011612000을 단정도 실수 32비트로 표현하면 둘 다 16진수로 0x3DCCCCCD

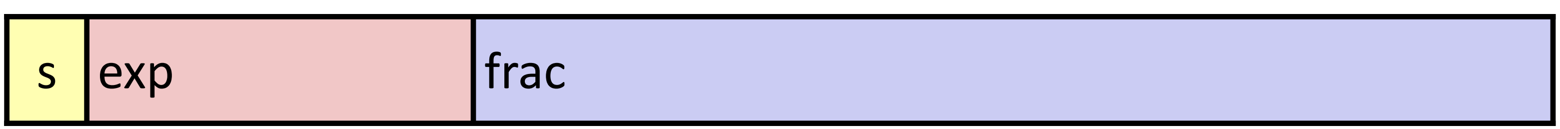
Floating Point Representation

- **Numerical Form:**

$$(-1)^s \, M \; 2^E$$

  - **Sign bit *s*** determines whether number is negative or positive
  - **Significand *M*** normally a fractional value in range [1.0,2.0).
  - **Exponent *E*** weights value by power of two

- **Encoding**
  - MSB s is sign bit ***s***
  - exp field encodes ***E*** (but is not equal to E)
  - frac field encodes ***M*** (but is not equal to M)

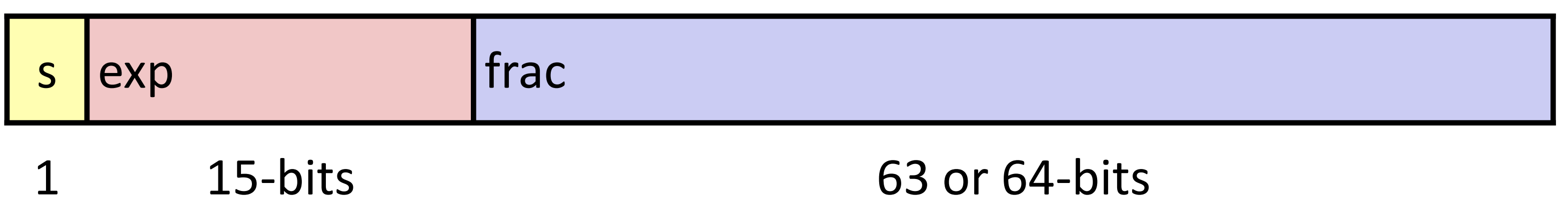


Precisions

- **Single precision: 32 bits**

- **Double precision: 64 bits**

- **Extended precision: 80 bits (Intel only)**



Normalized Values

- **Condition: exp ≠ 000…0 and exp ≠ 111…1**

- **Exponent coded as *biased* value: *E* = *Exp* – *Bias***
  - *Exp*: unsigned value exp
  - *Bias* = 2^{k-1} - 1, where *k* is number of exponent bits
    - Single precision: 127 (Exp: 1…254, E: -126…127)
    - Double precision: 1023 (Exp: 1…2046, E: -1022…1023)

- **Significand coded with implied leading 1: *M* =** $1.\text{xxx}\ldots\text{x}_2$
  - xxx…x: bits of frac
  - Minimum when 000…0 (*M* = 1.0)
  - Maximum when 111…1 (*M* = 2.0 – ε)
  - Get extra leading bit for "free"

Normalized Encoding Example

- **Value:** `Float F = 15213.0;`
  - $15213_{10} = 11101101101101_2$
    $= 1.1101101101101_2 \times 2^{13}$

- **Significand**

  M = $1.\underline{1101101101101}_2$

  `frac` = $\underline{1101101101101}0000000000_2$

- **Exponent**

  E = 13

  $Bias$ = 127

  Exp = 140 = 10001100_2

- **Result:**

s	exp	frac
`0`	`10001100`	`11011011011010000000000`

Denormalized Values

- **Condition:** exp = 000…0

- **Exponent value: $E = -Bias + 1$ (instead of $E = 0 - Bias$)**
- **Significand coded with implied leading 0: $M = 0.xxx...x_2$**
  - **`xxx…x`**: bits of **`frac`**
- **Cases**
  - **`exp`** = 000…0, **`frac`** = 000…0
    - Represents zero value
    - Note distinct values: +0 and –0 (why?)
  - **`exp`** = 000…0, **`frac`** ≠ 000…0
    - Numbers very close to 0.0
    - Lose precision as get smaller
    - Equispaced

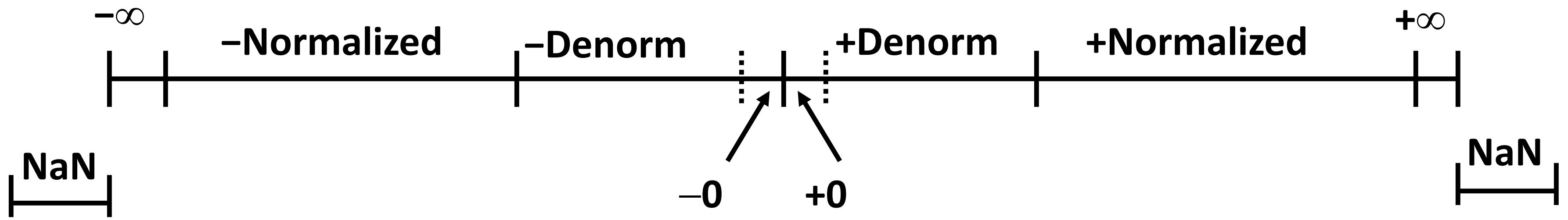
Special Values

- **Condition: `exp` =** 111…1

- **Case: `exp` =** 111…1, **`frac` =** 000…0
  - Represents value ∞ (infinity)
  - Operation that overflows
  - Both positive and negative
  - E.g., 1.0/0.0 = −1.0/−0.0 = +∞, 1.0/−0.0 = −∞

- **Case: `exp` =** 111…1, **`frac` ≠** 000…0
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., sqrt(–1), ∞ − ∞, ∞ × 0

Visualization: Floating Point Encodings



Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- **Example and properties**
- Rounding

Tiny Floating Point Example

s	exp	frac
1	4-bits	3-bits

- **8-bit Floating Point Representation**
  - the sign bit is in the most significant bit
  - the next four bits are the exponent, with a bias of 7
  - the last three bits are the **`frac`**

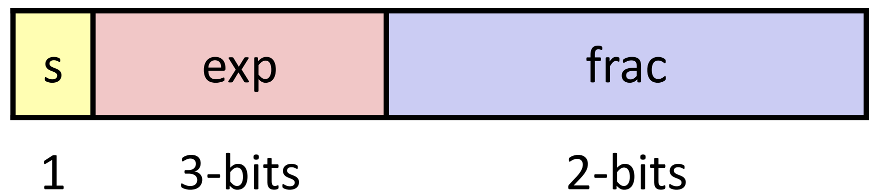
- **Same general form as IEEE Format**
  - normalized, denormalized
  - representation of 0, NaN, infinity

Dynamic Range (Positive Only)

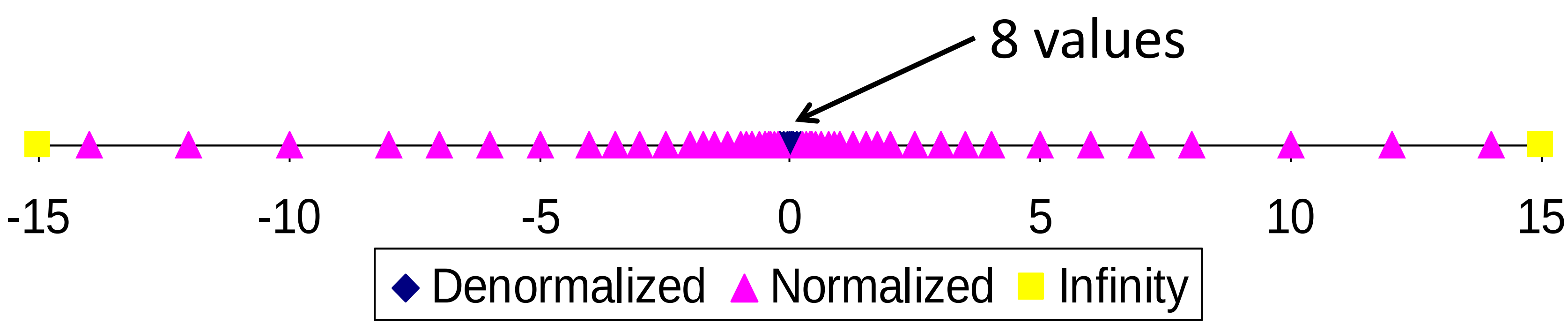
	s	exp	frac	E	Value	
	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	closest to zero
Denormalized numbers	0	0000	010	-6	2/8*1/64 = 2/512	
	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	largest denorm
	0	0001	000	-6	8/8*1/64 = 8/512	smallest norm
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	closest to 1 below
Normalized numbers	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	closest to 1 above
	0	0111	010	0	10/8*1 = 10/8	
	…					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	largest norm
	0	1111	000	n/a	inf	

Distribution of Values

- **6-bit IEEE-like format**
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 23-1-1 = 3

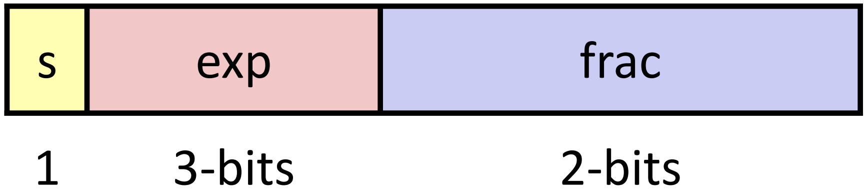


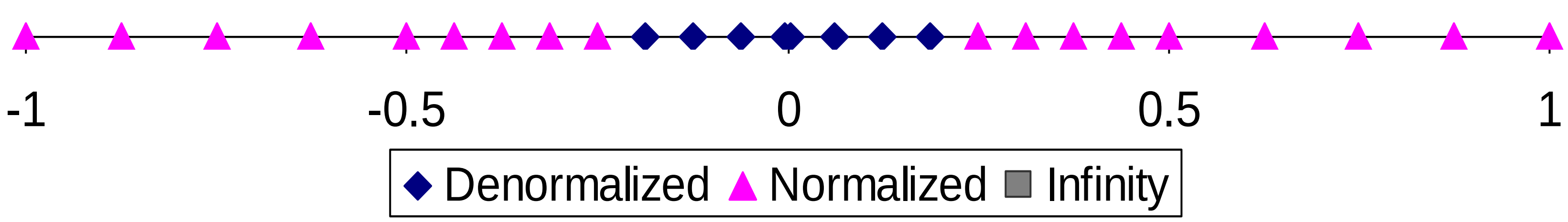
- **Notice how the distribution gets denser toward zero.**



Distribution of Values (close-up view)

- **6-bit IEEE-like format**
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 3





Interesting Numbers `{single,double}`

Description	*exp*	*frac*	*Numeric Value*
Zero	**00…00**	**00…00**	**0.0**
Smallest Pos. Denorm.	**00…00**	**00…01**	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
Single ≈ 1.4×10^{-45}			
Double ≈ 4.9×10^{-324}			
Largest Denormalized	**00…00**	**11…11**	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
Single ≈ 1.18×10^{-38}			
Double ≈ 2.2×10^{-308}			
Smallest Pos. Normalized	**00…01**	**00…00**	$1.0 \times 2^{-\{126,1022\}}$
Just larger than largest denormalized			
One	**01…11**	**00…00**	**1.0**
Largest Normalized	**11…10**	**11…11**	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$
Single ≈ 3.4×10^{38}			
Double ≈ 1.8×10^{308}			

Special Properties of Encoding

- **FP Zero Same as Integer Zero**
  - All bits = 0

- **Can (Almost) Use Unsigned Integer Comparison**
  - Must first compare sign bits
  - Must consider −0 = 0
  - NaNs problematic
    - Will be greater than any other values
    - What should comparison yield?
  - Otherwise OK
    - Denorm vs. normalized
    - Normalized vs. infinity

Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- **Rounding**

Floating Point Operations: Basic Idea

- $x +_f y = \mathrm{Round}(x + y)$

- $x \times_f y = \mathrm{Round}(x \times y)$

- **Basic idea**
  - First compute exact result
  - Make it fit into desired precision
    - Possibly overflow if exponent too large
    - Possibly round to fit into `frac`

Rounding

- **Rounding Modes (illustrate with $ rounding)**

	$1.40	**$1.60**	**$1.50**	**$2.50**	**–$1.50**
Towards zero	$1	$1	$1	$2	–$1
Round down (−∞)	$1	$1	$1	$2	–$2
Round up (+∞)	$2	$2	$2	$3	–$1
Nearest Even (default)	$1	$2	$2	$2	–$2

- **What are the advantages of the modes?**

Closer Look at Round-To-Even

- **Default Rounding Mode**
  - Hard to get any other kind without dropping into assembly
  - All others are statistically biased
    - Sum of set of positive numbers will consistently be over- or under-estimated

- **Applying to Other Decimal Places / Bit Positions**
  - When exactly halfway between two possible values
    - Round so that least significant digit is even
  - E.g., round to nearest hundredth

1.2349999	1.23	(Less than half way)
1.2350001	1.24	(Greater than half way)
1.2350000	1.24	(Half way—round up)
1.2450000	1.24	(Half way—round down)

Rounding Binary Numbers

- **Binary Fractional Numbers**
  - "Even" when least significant bit is 0
  - "Half way" when bits to right of rounding position = $100\ldots_2$

- **Examples**
  - Round to nearest 1/4 (2 bits right of binary point)

Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2

7. 부동소수점 산출연산의 부정확성은 심각한 결과를 초래할 수 있다. 1991년 2월 25일 1차 걸프 전쟁 기간 중에 사우디아라비아 Dharan에 위치한 미국 패트리엇 미사일 부대는 날아오는 이라크의 스커드 미사일을 격추하는 데 실패했다. 스커드 미사일은 미 육군 막사에 떨어져 28명의 대원이 사망했다. 미국 일반 조사위는 이 실패에 관해 상세한 조사를 수행하였으며 수치 계산의 부정확성이 주요 원인이라는 결론을 내렸다. 이 예제에서는 조사위 부석의 일부분을 검증하게 된다.

패트리엇 시스템은 내부 클럭을 가지고 있으며, 이것은 매 0.1초마다 증가하는 카운터로 구현되어 있다. 시간을 초로 계산하기 위해 프로그램은 이 카운터 값을 24비트 값으로 곱해주는데, 이것은 $\frac{1}{10}$로의 비율 이진수 근사한 것이다. 특히 $\frac{1}{10}$의 이진 표시는 비결정성 수열 $0.000110011[0011]\cdots_2$ 이며, 여기서 []안은 무한 반복된다. 이 프로그램은 χ의 값으로 0.1을 근사하였는데, 이진 소수점 우측의 수열에서 앞부분의 23비트만을 이용하였다: χ = 0.00011001100110011001100.

A. 0.1 − χ의 이진수 표시는 어떻게 되는가?

B. 0.1 − χ의 근사한 십진수 값은 얼마인가?

C. 클럭은 시스템에 최초로 전원이 공급되면 0에서 시작해서 계속 증가한다. 이 경우, 시스템은 약 100시간 동안 동작하였다. 이때 실제 시간과 소프트웨어가 계산한 시간과의 차이는 얼마인가?

D. 스커드 미사일이 약 초속 2,000미터로 날아갔다면, 이 예측은 얼마나 틀리게 되는가?

Problem 7 Solution:

In most cases, the limited precision of floating-point numbers is not a major problem, because the *relative* error of the computation is still fairly low. In this example, however, the system was sensitive to the *absolute* error.

A. We can see that 0.1 − χ has binary representation

$$0.0000000000000000000000001100[1100]\cdots_2$$

B. Comparing this to the binary representation of $\frac{1}{10}$, we can see that it is simply

$2^{-20} \times \frac{1}{10}$, which is around 9.54×10^{-8}.

C. $9.54 \times 10^{-8} \times 100 \times 60 \times 60 \times 10 \approx 0.343$ seconds.

D. $0.343 \times 2000 \approx 687$ meters.

8. 문제 7에서 패트리엇 미사일 소프트웨어가 0.1을 $\chi = 0.00011001100110011001100_2$ 로 근사한다는 것을 알았다. 대신에 이들이 IEEE 짝수 근사 모드로 이진 소수점의 우측 23비트를 이용해서 χ'을 0.1로 근사하였다고 가정하자.

A. χ'는 이진수로 어떻게 표시하는가?
B. $\chi' - 0.1$을 근사한 십진수 값은 얼마인가?
C. 계산한 클럭은 100시간 후에 얼마나 큰 오차를 갖게 되었는가?
D. 프로그램의 스커드 미사일 위치 오차는 얼마나 되겠는가?

Problem 8 Solution:

A. Looking at the nonterminating sequence for 1/10, we can see that the 2 bits to the right of the rounding position are 1, and so a better approximation to 1/10 would be obtained by incrementing x to get $x' = 0.00011001100110011001101_2$, which is larger than 0.1.

B. We can see that $x' - 0.1$ has binary representation:

$$0.0000000000000000000000000[1100].$$

Comparing this to the binary representation of $\frac{1}{10}$, we can see that it is $2^{-22} \times \frac{1}{10}$, which is around 2.38×10^{-8}.

C. $2.38 \times 10^{-8} \times 100 \times 60 \times 60 \times 10 \approx 0.086$ seconds, a factor of 4 less than the error in the Patriot system.

D. $0.343 \times 2000 \approx 171$ meters.