CHUNGNAM NATIONAL UNIVERSITY

시스템 프로그래밍

강의 3 : 2.4 실수의 표현 및 처리 I
2014년 9월 16일
http://eslab.cnu.ac.kr

부동소수점 퍼즐

- 아래의 C 프로그램을 보고 :
  - 모든 값에 대해 항상 성립하는 관계인지 생각해 보라
  - 그렇지 않은 이유는 무엇인지 설명해 보라

```
int x = …;
float f = …;
double d = …;
```

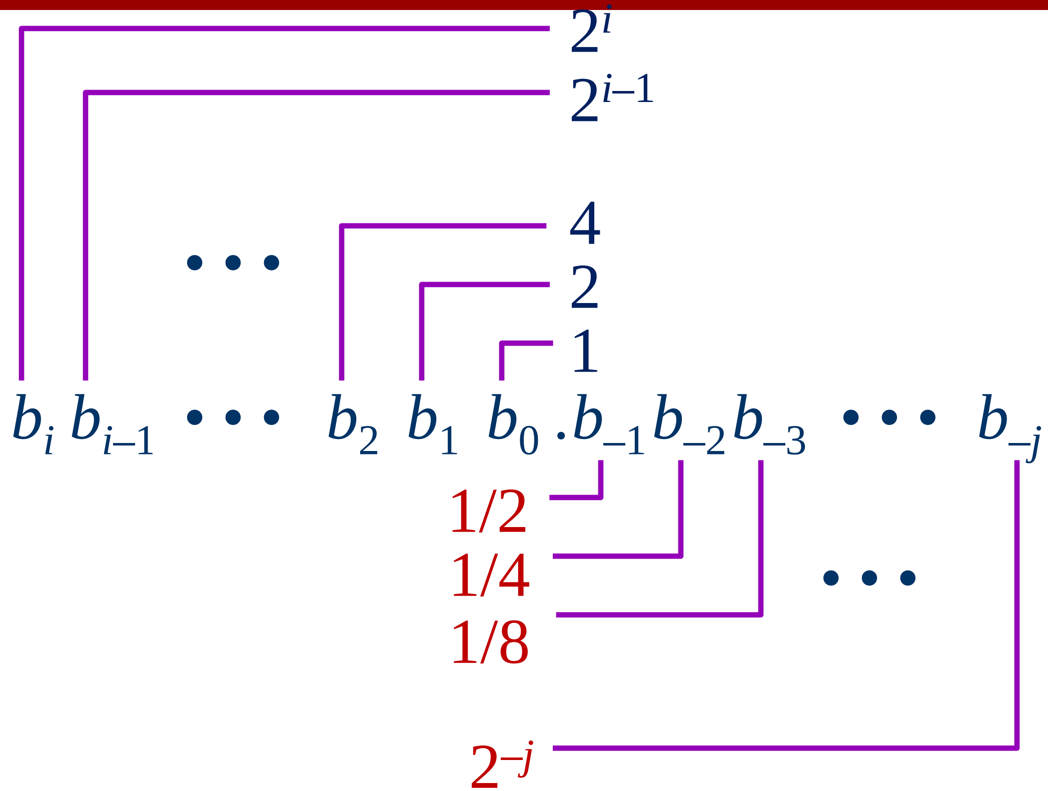
d 나 f는 NaN 은 아니다

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0` ⇒ `((d*2) < 0.0)`
- `d > f` ⇒ `-f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

IEEE Floating Point

- IEEE 표준 754
  - 실수(floating point) 연산을 위한 단일 표준으로 1985년에 제정
    - 이전까지는 다양한 형태의 실수 표시법이 존재하였음
    - 인텔사의 지원으로 8087 프로세서 개발목적으로 추진
  - 현재 모든 주요 CPU에서 IEEE Floating Point라는 이름으로 지원됨
- 수치해석적인 측면을 고려하여 정의됨
  - rounding, overflow, underflow 등에 유용함
  - 빠른 연산을 수행하기가 어려운 단점이 있다
    - 표준 정의 시 수치해석자들의 수가 하드웨어 연구자들 보다 많았다
- Floating point 와 프로그래머
  - 무관심 , 흥미상실, 이해하기 어려운 내용
  - 그렇지만 실제로는 우아하고 이해할만한 내용

2진 소수



표현방법

- 이진 소수점 우측의 비트들은 2의 분수제곱을 의미
- 소수는 다음과 같이 표시:

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

2진 소수 예제

값	2진 소수 표시
5 와 3/4	101.11_2
2 와 7/8	10.111_2
63/64	0.111111_2

관찰

- 우측으로 쉬프트 하면 2로 나눈 효과를 얻음
- 좌측으로 쉬프트하면 2를 곱하는 효과를 얻음
- 1.0 에 매우 근접하는 $0.111111\ldots_2$ 과 같은 수들은 다음과 같이 표시한다
  - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
  - $1.0 - \varepsilon$

표시 가능한 수

한계

- **$x/2^k$ 형태 만 정확히 표시가능**
- 소숫점 이하의 비트들이 무한 반복되는 경우는 정확히 표시할 수 없다
- 5 x 2^{100} 은 어떻게 표시되겠는가? => 101 0000 0000 0000 ……
  - 큰 수의 표시는 자리값을 이용해 표시하는 방법을 사용할 수 없다

값	표시
1/3	$0.0101010101[01]\ldots_2$
1/5	$0.001100110011[0011]\ldots_2$
1/10	$0.0001100110011[0011]\ldots_2$

- 다른 방법
  - $x \times 2^y$ 의 형태로 수를 표시하고, x, y를 이용해 표시하는 방법을 이용하는 방법도 있다

Practice 1 : 2진 소수의 표시

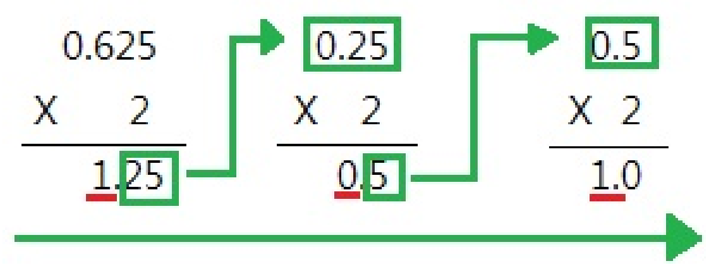
다음의 빈칸에 알맞은 숫자들로 변환하시오.

Representation	Value	Decimal
0.0_2	$\frac{0}{2}$	0.0_{10}
0.01_2	$\frac{1}{4}$	0.25_{10}
0.010_2	$\frac{2}{8}$	______
0.0011_2	______	0.1875_{10}
0.00110_2	$\frac{6}{32}$	0.1875_{10}
0.001101_2	$\frac{13}{64}$	______
0.0011010_2	______	0.203125_{10}
0.00110011_2	$\frac{51}{256}$	0.19921875_{10}

Practice 2 : 2진 소수

다음의 빈칸에 알맞은 숫자들로 변환하시오.

분수 값	2진 소수	10진수
1/8	0.001_2	0.125
		0.75
25/16		1.5625
5/32	0.00101_2	



0.101_2

Practice 3 : 패트리엇 미사일 예제

내부 클럭이 0.1초마다 증가하며, 이 카운터와 1/10의 24비트 이진수 표시값과 곱하여 초를 계산한다. 1/10은 이진수로 $0.000110011[0011]...._2$ 이다. 이 프로그램에서는 소숫점 우측의 23비트를 이용하여 0.1의 근사값 x를 이용하고 있다.

A. 0.1-x 는 이진수로 어떻게 표시되는가?

B. 0.1-x의 근사한 십진수 값은 얼마인가? 9.54×10^{-8}

C. 시스템을 100시간동안 동작시켰을 때, 실제시간과 이 소프트웨어가 계산한 시간과의 차이는 얼마인가?

D. 스커드미사일이 약 초속 2000미터의 속도로 날아간다면, 100시간후(약4일 후)오차는 얼마가 되겠는가?

IEEE Floating Point 표준

소수의 표현방법

- $(-1)^s\ M\ 2^E$
  - 부호비트 s 는 양수/음수 여부를 표시
  - 유효숫자 M 은 [1.0,2.0) 또는 [0.0, 1.0) 사이의 실수 값을 표시
  - 지수 E 은 2의 지수제곱을 표시

인코딩

- MSB 는 부호 비트
- exp 필드는 E 를 인코딩
- frac 필드는 M 를 인코딩

인코딩 방법

필드의 정의

s	exp	frac

- MSB 는 부호 비트
- exp 필드는 *E* 부분을 인코딩
- frac 필드는 *M* 부분을 인코딩

필드의 길이

- Single 정밀도(float): 8 exp 비트, 23 frac 비트
  - 총 32 비트
- 더블(double): 11 exp 비트, 52 frac 비트
  - 총 64 비트
- 확장(Extended precision): 15 exp 비트, 63 frac 비트
  - Intel 호환 컴퓨터에서만 사용
  - 80 비트표시
    - 1 비트는 사용하지 않음

인코딩은 exp 값에 따라 세 가지의 경우로 달라진다
(정규화, 비정규화, 특수값)
(Normalized, Denormalized, Special values)

Case 1 : 정규화된 값(Normalized values)

s	exp	frac

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

조건

- exp ≠ 000…0 그리고 exp ≠ 111…1 인 경우에 사용

지수(E)는 조정된 값(*biased* value) 의 형식으로 표시된다

$E = exp - bias,\ exp = E + bias$

- *exp* : unsigned value
- *bias* : 조정값 Bias value
  - *bias* = 2^{e-1} - 1, 여기서 e 는 지수비트의 갯수임
  - Single precision: 127 (*exp*: 1…254, *E*: -126…127)
  - Double precision: 1023 (*exp*: 1…2046, *E*: -1022…1023)
  - 조정값으로 인해 +, -범위의 E 값을 비부호형으로 전환

유효숫자(M)는 묵시적으로 1로 시작하는 것으로 간주한다

$M = 1.xxx\ldots x_2$

- xxx…x: frac 부분
- 000…0 일 때 최소(M = 1.0)
- 111…1 일 때 최대(M = 2.0 – ε)
- 이와 같이 함으로써 1비트를 **무료로** 표시할 수 있게 된다

정규화 값의 인코딩 예제

Value

```
float F = 15213.0;
```

- 15213_{10} = 11101101101101_2 = 1.1101101101101_2 X 2^{13}

유효숫자(Significand), 23비트

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

frac = $\underline{1101101101101}0000000000_2$ <= 23 비트

지수(Exponent), 8비트

E = 13

$Bias$ = 127

exp = 140 = 10001100_2 <= 8 비트

Floating Point Representation:

```
Hex:      4    6    6    D    B    4    0    0
Binary: 0100 0110 0110 1101 1011 0100 0000 0000

140:     100 0110 0

15213:             1110 1101 1011 01
```

Case 2 : 비정규화 인코딩(Denormalized value)

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

적용 조건

- exp = 000…0

인코딩

- 지수(Exponent value) $E = 1 - Bias$
- 유효숫자(Significand value) $M =$ 0.xxx…x_2
  - xxx…x: frac 비트

비정규화 인코딩의 사용

- case I : exp = 000…0, frac = 000…0
  - 0을 표시 (정규화 방식에서는 항상 M>= 0)
  - +0 과 –0 의 경우 표시가 다르다는 점에 주의
- case II : exp = 000…0, frac ≠ 000…0
  - 0.0에 매우 근접한 소수값을 표시(underflow 수의 표시)
  - "점차적인 언더플로우"특성 : 0.0 부근의 숫자들이 동일 간격으로 분포한다

Case 3 : 특수값(Special Values)

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

적용조건

- `exp` = `111…1`

두 가지 경우에 사용

- `exp` = `111…1`, `frac` = `000…0`
  - 무한대 ∞ (infinity)를 표시(+/-)
  - 오버플로우를 표시할 수 있다.
  - E.g., 1.0/0.0 = –1.0/–0.0 = +∞, 1.0/–0.0 = –∞
- `exp` = `111…1`, `frac` ≠ `000…0`
  - Not-a-Number (NaN)
  - 숫자로 표시할 수 없는 결과를 나타낼 때 사용
  - E.g., sqrt(–1), $\infty - \infty$, $\infty * 0$

Practice 4 : IEEE 실수표현 – 인코딩

❖ 다음의 숫자들을 32비트 single 정밀도로 IEEE 표준 754로 인코딩 하고 2진수와 16진수로 표현하시오

1. -15213.0 : $15213_{10} = 11101101101101_2$

2. 9.6875

소수 표현 요약