



CHUNGNAM NATIONAL UNIVERSITY



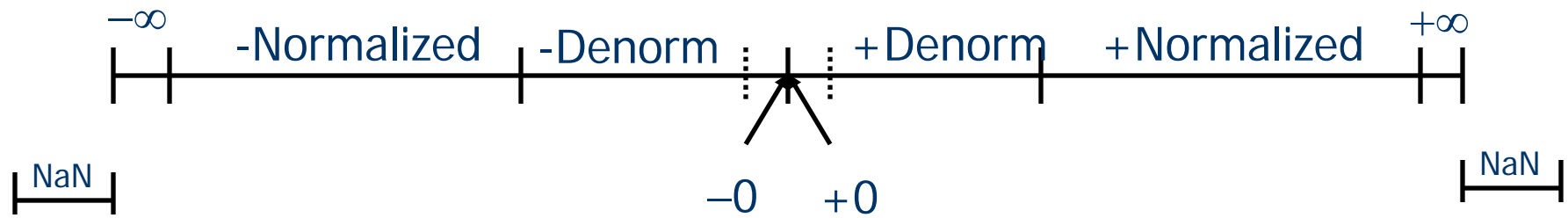
시스템 프로그래밍

강의 3 : 2.4 실수의 표현 및 처리 II

2014년 9월 23일

<http://eslab.cnu.ac.kr>

소수 표현 요약

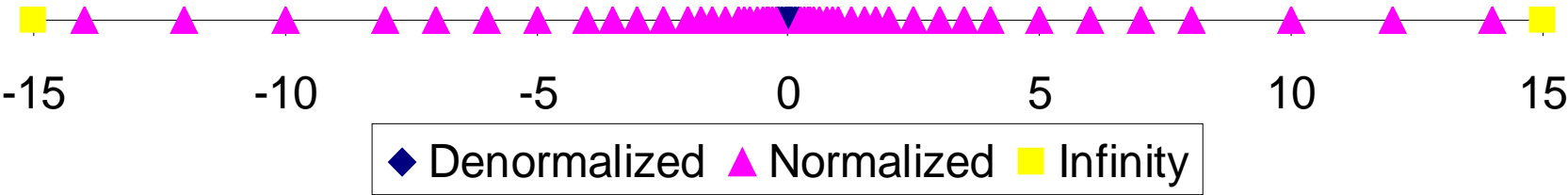


값의 분포 - 6비트 체계

6 비트 IEEE 유사한 형식으로 표시하는 경우

- $e = 3$ 비트 지수
- $f = 2$ 비트 소수
- 바이어스 = 3
- 정규화 최대값은 14

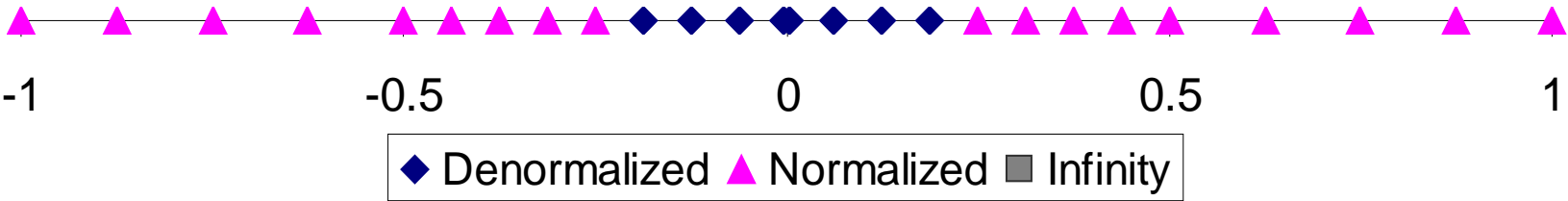
0 부근에서 왜 분포가 조밀해 지는지 파악해 보자



값의 분포(확대)

6 비트 IEEE 유사한 형식으로 표시하는 경우

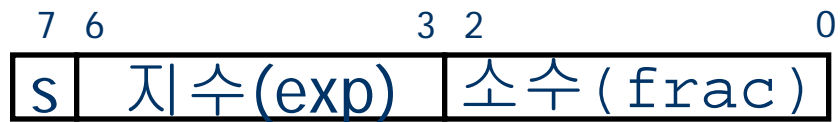
- $e = 3$ 비트 지수
- $f = 2$ 비트 소수
- 바이어스 = 3



간단한 Floating Point 시스템 예제

8-bit Floating Point 표시

- 부호비트는 MSB로 표시.
- 다음 4비트는 바이어스 값 7로 exp로 사용.
- 마지막 3비트는 frac
- IEEE Format 과 동일한 일반적인 형태
 - 정규화, 비정규화 표시 사용
 - 0, NaN, infinity 등 표시 사용



지수 값의 범위

Exp	exp	E	2^E		bias=7
0	0000	-6	1/64	(비정규화)	E=1-bias
1	0001	-6	1/64		
2	0010	-5	1/32		
3	0011	-4	1/16		
4	0100	-3	1/8		
5	0101	-2	1/4		
6	0110	-1	1/2		
7	0111	0	1		E=exp-bias
8	1000	+1	2		
9	1001	+2	4		
10	1010	+3	8		
11	1011	+4	16		
12	1100	+5	32		
13	1101	+6	64		
14	1110	+7	128		
15	1111	n/a		(inf, NaN)	

8비트 시스템 - 표현 가능 수(양수)

	s	exp	frac	E	Value	$(-1)^s M 2^E$
비정규화 수	0	0000	000	-6	0	
	0	0000	001	-6	$1/8 * 1/64 = 1/512$	← 0에 가장 가까운 값
	0	0000	010	-6	$2/8 * 1/64 = 2/512$	
	...					
	0	0000	110	-6	$6/8 * 1/64 = 6/512$	
	0	0000	111	-6	$7/8 * 1/64 = 7/512$	← 최대 비정규화 수
	<hr/>					
정규화 수	0	0001	000	-6	<u>8/8</u> * 1/64 = 8/512	← 최소 정규화 수
	0	0001	001	-6	<u>9/8</u> * 1/64 = 9/512	
	...					
	0	0110	110	-1	$14/8 * 1/2 = 14/16$	
	0	0110	111	-1	$15/8 * 1/2 = 15/16$	← 1에 가장근접한 수
	0	0111	000	0	$8/8 * 1 = 1$	
	0	0111	001	0	$9/8 * 1 = 9/8$	← 1에 가장근접한 수
	0	0111	010	0	$10/8 * 1 = 10/8$	
	...					
	0	1110	110	7	$14/8 * 128 = 224$	
	0	1110	111	7	$15/8 * 128 = 240$	← 최대 정규화 수
	0	1111	000	n/a	inf	

Floating Point 근사(Rounding)

근사 과정

- 먼저 정확한 값을 계산한다
- 그 결과를 원하는 정밀도로 조정한다
 - ◆ 만일 지수부가 너무 크다면 오버플로우일 수 있다
 - ◆ frac 필드의 길이에 맞출 수 있도록 반올림 한다



근사 방법의 선택

	\$1.40	\$1.60	\$1.50	\$2.50	-\$1.50
● 0 방향	\$1	\$1	\$1	\$2	-\$1
● 내림($-\infty$)	\$1	\$1	\$1	\$2	-\$2
● 올림 ($+\infty$)	\$2	\$2	\$2	\$3	-\$1
● <u>인접 짝수 (default)</u>	\$1	\$2	\$2	\$2	-\$2

Q. 원 짝수 ?

Note:

1. 버림: 근사값은 참값보다 클 수 없다.
2. 올림: 근사값은 참값보다 작을 수 없다.

인접 짝수 모드(Round-To-Even)

컴퓨터에서 사용하는 기본(Default) 근사모드

- 어셈블리 언어 수준까지 이해하지 않고는 설명할 수 없다
- 다른 근사 모드들은 모두 통계적으로 편향되어 있다
 - ▶ 여러 개의 근사한 양수들의 합은 과대(또는 과소)평가 하게 된다

십진수에서 근사법의 적용

- 두 개의 가능한 값의 정 중간 값인 경우(그 이외의 경우에는 반올림)
 - ▶ 최소 자리값이 짝수가 되도록 근사
- E.g., 0.01의 자리로 근사하는 경우

1.2349999	1.23	(정 중간 보다 작다)
1.2350001	1.24	(정 중간보다 크다)
1.2350000	1.24	(정 중간 값 - 올림)
1.2450000	1.24	(정 중간 값 - 내림)

이진수에서의 근사법

이진 소수의 표현

- "짝수" 는 최소 유효 값이 0 인 경우를 말함
- 근사의 대상이 되는 중간값의 표시는 근사하는 자리 바로 오른쪽의 비트가 = $xxx100..._2$

Examples

- 1/4 자리로 근사하는 경우(2 진 소숫점이하 2번째 자리로)

값	이진수	근사값	연산	근사값(10진수)
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

종합 적용 : Floating point 수 만들기

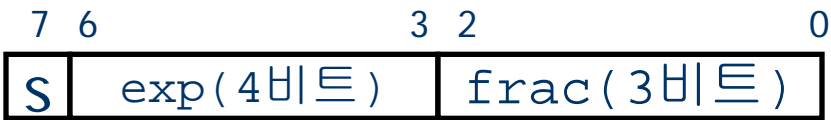
과정

- 1 이 맨 앞에 올 수 있도록 정규화(Normalize)
- 소수점 자리(fraction)에 맞도록 반올림
- 반올림 효과를 반영하기 위한 정규화 후처리(Post-normalize)

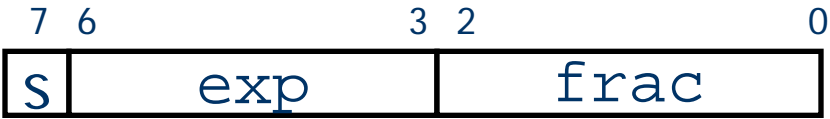
예)

- 8-bit 비부호형 수를 아래와 같은 간단한 floating point format 으로 표시해보자
- Example Numbers(128 => 128.0 으로 표시하기)

128	10000000
15	00001101
33	00010001
35	00010011
138	10001010
63	00111111



1단계 : 정규화하기

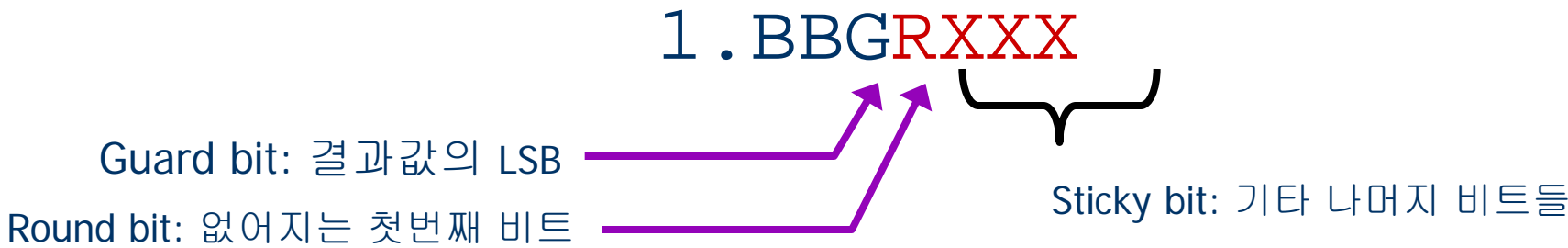


요구사항

- 이진 소수의 위치를 이동해서 1.xxxxxx의 형태가 되도록 한다
- 자리수 만큼 0 을 채운다
 - 지수값을 소수점 위치에 맞도록 조정

Value	Binary	M	E
128	10000000	1.00000000	7
13	00001101	1.1010000	3
17	00010001	1.0001000	4
19	00010011	1.0011000	4
138	10001010	1.0001010	7
63	00111111	1.1111100	5

2단계 : 근사



* frac 필드가 3비트 이므로, 1/8 위치로 근사해야 한다.

반올림 조건

- Round = 1, Sticky = 1 → > 0.5
- Guard = 1, Round = 1, Sticky = 0 → 인접 짝수

Value	소수	GRS	올림?	근사값
128	1.0000000	000	N	1.000
13	1.1010000	100	N	1.101
17	1.0001000	010	N	1.000
19	1.0011000	110	Y	1.010
138	1.0001010	011	Y	1.001
63	1.1111100	111	Y	10.000

3단계 : 정규화 후처리

방법

- 근사화 결과 오버플로우가 발생했을지 모른다
- 우측으로 한 자리 shift 하고, 지수값을 1 증가시킨다(정규화 작업)

Value	근사값	지수	조정후	결과
128	1.000	7		128
13	1.101	3		15
17	1.000	4		16
19	1.010	4		20
138	1.001	7		134
63	10.000	5	1.000 / 6	64



int에서 float으로 타입이 바뀌면 결과값이 달라진다!!!

Practice 5 : IEEE 소수표현 – 인코딩

❖ 다음의 이진수들을 소수점 첫째 자리로 짝수 근사법을 이용하여 근사법을 보이시오

1. 10.010_2

2. 10.011_2

3. 10.110_2

4. 11.001_2

Practice 6 : IEEE 소수표현 – 인코딩

❖ 아래와 같은 시스템이 있다고 할 때 아래소수를 IEEE 소수로 표현하시오. 인접짝수 근사를 이용하시오.

- **8-bit Floating Point**
- 부호비트는 **MSB**로 표시.
- 다음 4비트는 바이어스 값 7로 지수부로 사용.
- 마지막 3비트는 **frac**

1. 10.010_2

2. 10.011_2

3. 10.110_2

4. 11.001_2

FP 곱셈

피연산자

$$(-1)^{s1} M1 2^{E1} \quad * \quad (-1)^{s2} M2 2^{E2}$$

정확한 결과

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

- Sign s : $s1 \wedge s2$
- 소수 M : $M1 * M2$
- 지수 E : $E1 + E2$

정정

- 만일 $M \geq 2$, M 을 우측으로 shift, E 를 1 증가
- 만일 E 값이 범위를 벗어나면, 오버플로우
- M 을 frac 정밀도에 맞추어 근사

구현

- 가장 중요한 부분은 소수들 간의 곱셈

FP 덧셈

피연산자

$(-1)^{s1} M1 2^{E1}$

$(-1)^{s2} M2 2^{E2}$

- $E1 > E2$ 을 가정

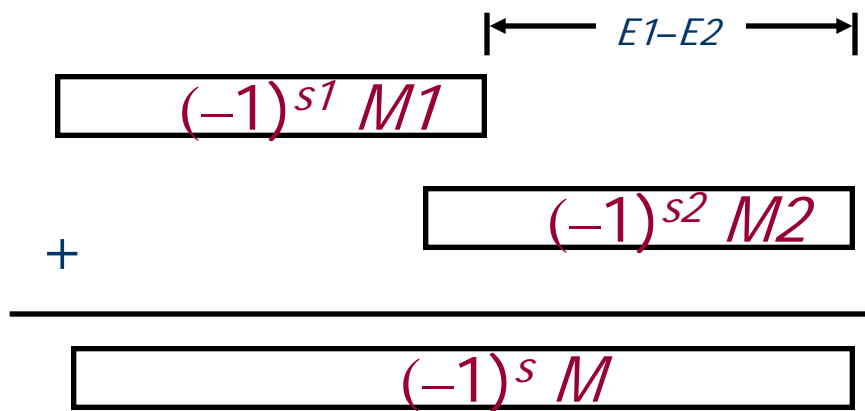
정확한 결과

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

- 부호 s , 소수 M :
 - ▶ 열 맞춤후의 덧셈의 결과
- 지수 E : $E1$

정정

- 만일 $M \geq 2$, M 을 우측으로 shift, E 를 1 증가
- 만일 $M < 1$, M 을 좌측으로 k 자리 shift, E 를 k 만큼 감소
- 만일 E 가 범위를 벗어나면 오버플로우
- M 을 근사시켜서 frac 정밀도에 맞춤



C 언어에서의 Floating Point

C에서는 2개의 정밀도를 제공

float single precision

double double precision

변환

- int, float, double 간의 casting 은 값을 변화시킨다
- Double, float 에서 int 로의 변환
 - ◆ 소수 부분을 제거
 - ◆ 0 방향으로의 근사와 동일
 - ◆ 무한대 또는 NaN 의 경우에는 정의 되어 있지 않음
 - 일반적으로 TMin
- int 에서 double 로
 - ◆ 정확한 변환, 정수가 53비트 보다 짧은 경우에만
- int 에서 float 로
 - ◆ 근사 모드에 따라 근사화

C 에서의 Floating Point 숫자

설명	exp	frac	Numeric Value
0	00...00	00...00	0.0
최소 비정규화 수(+) <ul style="list-style-type: none">Single $\approx 1.4 \times 10^{-45}$Double $\approx 4.9 \times 10^{-324}$	00...00	00...01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
최대 비정규화 수 <ul style="list-style-type: none">Single $\approx 1.18 \times 10^{-38}$Double $\approx 2.2 \times 10^{-308}$	00...00	11...11	$(1.0 - \epsilon) \times 2^{-\{126,1022\}}$
최소 정규화 수(+) <ul style="list-style-type: none">● 최대 비정규화 수보다 아주 조금 크다	00...01	00...00	$1.0 \times 2^{-\{126,1022\}}$
1	01...11	00...00	1.0
최대 정규화 수 <ul style="list-style-type: none">Single $\approx 3.4 \times 10^{38}$Double $\approx 1.8 \times 10^{308}$	11...10	11...11	$(2.0 - \epsilon) \times 2^{\{127,1023\}}$

인코딩의 특징

FP에서의 0 은 정수에서의 0 과 동일하다

- 모든 비트들 = 0

비부호 정수 비교를 사용할 수 있다(거의)

- 먼저 부호비트를 비교해야 한다
- -0 = 0 인 점을 고려해야 한다
- NaN의 경우는 문제가 생긴다
 - ◆ 다른 모든 값들보다는 클 것인가
 - ◆ 비교의 결과는 무엇인가?
- 그렇지 않은 다른 경우는 OK
 - ◆ 비정규화 vs. 정규화
 - ◆ 정규화 vs. 무한대

Ariane 5

- 이륙후 37초 후 폭발
- 위성체 손실규모 5천억원

이유

- 수평 속도를 64비트 floating point 형으로 계산한 후에 16-bit int 타입으로 변환한 후에 오버플로우 발생
- Ariane 4 로켓에서는 이 수평속도 값의 범위를 정의하여, 정상동작
- Ariane 5 로켓에서는 치명적 오류로 판단
 - ◆ 동일한 소프트웨어를 사용해서



Floating Point Puzzles

● 다음의 C 수식에서 :

- ▶ 다음의 수식이 항상 참인지 판별하라
- ▶ 참이 아니라면, 그 이유를 설명하라

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

d, f 는 NaN 가 아님

- `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`

Summary

컴퓨터에서 부동소숫점의 표시 방식을 이해해야 한다.

IEEE Floating Point 표준은 명쾌한 수학적특성을 갖는다

- $M \times 2^E$ 형태의 수 표현
- 실제 구현 방법과 관계없이 연산을 적용할 수 있다
 - ▶ 마치 완벽한 정밀도로 계산한 후에 근사화 한 것 처럼
- 실제 연산과 일치하는 것은 아니다
 - ▶ 교환/배분 법칙에 위배되는 경우가 있다
 - ▶ 컴파일러 개발자나 고급 프로그래머들에게는 큰 위협

부동소숫점의 표현도 다른 데이터 타입간의 혼용 또는 변환을 하는 경우에 주의해야 한다.

**** 예습 숙제 : pp.193-199, 3.2-3.2.2**