System Programming
숙제 3

인천대학교

시스템 프로그래밍 | 지도교수 박문주

송병준 | 201701562

0.목차

1. 과제 1
2. 환경 1
3. 문제 풀이 1
- 2.87 1
- 2.88 3
- 2.90 4

System Programming

숙제 3

송병준
2019년 9월 29일

1. 과제

Computer System, A Programmer's Perspective 2장 연습 문제 풀이
- 2.87
- 2.88
- 2.90

2. 환경

운영체제: macOS 10.14.6 (Darwin Kernel Version 18.7.0 x86_64)
프로세서: Intel Core i5
편집기: vim 8.0.1365
빌드: Apple LLVM version 10.0.1 (clang-1001.0.46.4)

3. 문제 풀이

- **2.87**

IEEE 754-2008로 명명된 IEEE 부동소수점 2008년 표준은 16비트 길이의 "절반 정밀도"를 갖는 부동소수점 표시를 포함한다. 이것은 본래 컴퓨터그래픽 회사들이 16비트 정수로 얻을 수 있는 것보다 더 높은 동적범위로 데이터를 저장하기 위해 고안한 것이다. 이 포맷은 1개의 부호 비트, 지수 5비트($k = 5$), 유효숫자 10비트($n = 10$)를 갖는다. 지수 조정값 bias는 $2^{5-1} - 1 = 15$다.

다음 표에서 주어진 숫자들에 대해 다음 지시사항을 고려하여 각 열의 값들을 채우시오.

Hex: 인코딩 형식을 설명하는 16진수 4자리 수
M: 유효숫자 값. 이 값은 x나 x/y 같은 값들의 연속.
x는 정수, y는 2의 제곱임. 예. 0, $\frac{67}{64}$, $\frac{1}{256}$
E: 지수의 정수 값
V: 표시된 형태의 숫자 값. x 또는 $x \times 2^z$의 형태를 사용할 것. x, z는 정수임.
D: `printf`문에서 `%f`를 이용하여 출력되는 숫자 값(근사값일 수도 있음)

예를 들어, 숫자 $\frac{7}{8}$을 나타내면, $s = 0$, $M = \frac{7}{4}$, $E = -1$이다. 따라서 이 숫자의 지수 필드는 01110_2(십진수로 $15 - 1 = 14$), 유효숫자 필드 1100000000_2가 되어 16진수로 `3B00`이다. 숫자로는 0.875다.

"—"로 표시된 칸은 채우지 않아도 된다.

Description	Hex	*M*	*E*	*V*	*D*
−0	_____	_____	_____	−0	−0.0
Smallest value > 2	_____	_____	_____	_____	_____
512		_____	_____	_____512	512.0
Largest denormalized	_____	_____	_____	_____	_____
−∞	_____	—	—	−∞	−∞
Number with hex representation 3BB0	3BB0	_____	_____	_____	_____

16비트 부동소수점 표기법은 1자리의 부호 비트, 5자리의 지수 비트, 10자리의 유효숫자 비트로 실수를 나타낸다. 이 표기법에는 몇 가지 특별한 경우와 값이 존재하는데, 다음과 같다.

Type	Exp	Fraction	Sign
Positive Zero	0	0	0
Negative Zero	0	0	1
Denormalized numbers	0	non zero	any
Nomalized numbers	1 ~ 30	any	any
Infinities	31 (all 1s)	0	any
NAN	31 (all 1s)	non zero	any

이 표기법에는 양의 0과 음의 0이 각각 존재한다. 또한 무한대와 NAN(Not A Number) 또한 존재한다.

Exponent가 0인데 유효숫자가 0이 아닌 것은 정규화되지 않은 숫자이다. 부동소수점으로 나타내어지는 모든 수는 정규화되어야 한다. 즉, 숫자가 $1.F \cdot 2^E$의 형태로 맞추어져야 한다는 것이다. 반면 비정규화된 부동소수점의 경우 숫자가 $0.F \cdot 2^{-14}$ 형태가 된다.

정규화된 부동소수점의 값은 $V = (-1)^s \cdot (1.F) \cdot 2^{E-15}$이며, 비정규화된 부동소수점의 값은 이와 조금 다른 $V = (-1)^s \cdot (0.F) \cdot 2^{-14}$이다. 여기서 주목해야 할 것은 비정규화 표현에서는 2의 −15제곱이 아닌 −14제곱을 곱한다는 것이다. 비정규화 표현은 사실 정규화 표현에서 E가 가장 작은 경우보다 하나 더 작은 경우이기 때문이다.

비정규화된 표현을 사용하면 작은 숫자를 더 세밀하게 표현할 수 있다. 예를 들어 $1.0_{(2)} \cdot 2^{-16}$은 정규화 표현으로 나타낼 수 없다. 따라서 비정규화를 거쳐야 한다. 지수가 정규화 지수 범위인 [−14, 15] 안에 포함될 때까지 유효숫자를 shift하면 $0.010_{(2)} \cdot 2^{-14}$이 된다. 유효숫자의 1의 자리를 1로 상정하는 정규화 표현에서는 나타낼 수 없다.

아주 작은 숫자도 표현할 수 있다. 조금 극단적인 예시를 들면, $1_{(2)} \cdot 2^{-25}$와 같이 매우 작은 숫자는 정규화된 지수 표현의 범위를 벗어나기 때문에 표현이 힘들다. 하지만 비정규화 표현에서는 E가 0, M이 0000000001로 표현이 가능하다. 값은 $0.0000000001_{(2)} \cdot 2^{-14}$을 가지게 된다.

주어진 표를 채우면 다음과 같다.

Description	Hex	Mantissa	Exponent	Value	Dump
-0	0000	0	0	-0	0.0
Smallest value > 2	4401	1025/ 1024	1	1.0000000001 * 2^1	1.00097656
512	6000	1	9	1.0 * 2^9	512
Largest denormalized	03FF	1023/1024	-14	0.1111111111 * 2^(-14)	0.00006096
-∞	FC00	0	16	-∞	-∞
Number with hex representation 3BB0	3BB0	123/64	-1	1.1110110000 * 2^(-1)	0.9609375

- **2.88**

다음과 같은 IEEE 부동소수점 형식을 따르는 두 가지의 9비트 부동소수점 표시가 있다고 하자.

1. 형식 A
   - 한 개의 부호 비트
   - $k = 5$개의 지수 비트, 지수 바이어스는 15
   - $n = 3$개의 비율 비트
2. 형식 B
   - 한 개의 부호 비트
   - $k = 4$개의 지수 비트, 지수 바이어스 7
   - $n = 4$개의 비율 비트

아래 표에는 형식 A에 의한 비트 패턴들이 주어져 있으며, 해야 할 일은 이들을 형식 B의 가장 가까운 값들로 변환하는 것이다. 근사가 필요하다면 $+\infty$ *방향으로 근사*해야 한다. 추가로, 형식 A, B 비트 패턴으로 주어진 숫자들의 값을 제시해야 한다. 이 숫자들을 정수(예. 17)나 비율(예. 17/64, $17/2^6$)로 나타내어라.

Format A		Format B	
Bits	Value	Bits	Value
1 01111 001	$\frac{-9}{8}$	1 0111 0010	$\frac{-9}{8}$
0 10110 011	_____	_____	_____
1 00111 010	_____	_____	_____
0 00000 111	_____	_____	_____
1 11100 000	_____	_____	_____
0 10111 100	_____	_____	_____

A 형식으로 주어진 부동소수점을 B 형식으로 바꾸어야 한다. 이때 표현 가능한 지수의 범위가 줄어들기 때문에 필연적으로 근사가 일어나게 된다.
먼저 지수를 5자리에서 3자리에 맞도록 옮긴 다음 유효숫자를 이에 맞추어 옮긴다.

변환된 값들은 다음과 같다.

Bits	Value	Bits	Value
1 01111 001	-9/8	1 0111 0010	- 9/8
0 10100 011	52	0 1100 0110	52
1 00111 010	-5/1024	1 0000 0101	- 5/1024
0 00000 111	7/131072	0 0000 0001	1/1024
1 11100 000	-8192	1 1110 1110	-112
0 10111 100	384	0 1110 1111	120

- **2.90**

2^x의 부동소수점 표시를 계산하는 C 함수를 작성하려 한다. 구현 방법으로 결과 값을 IEEE 단일정밀도로 직접 구성하는 것을 선택하였다. x가 매우 작을 때는 0.0을 리턴하고, x가 너무 크면 $+\infty$를 리턴한다. 아래 코드에 빈칸을 채워서 정확한 계산을 할 수 있도록 하라. 함수 u2f는 넘겨받은 비부호형 인자와 동일한 비트 표현을 갖는 부동소수점 값을 리턴한다고 가정하라.

```
float fpwr2(int x)
{
    /* Result exponent and fraction */
    unsigned exp, frac;
    unsigned u;

    if (x < _________) {
        /* Too small.  Return 0.0 */
        exp = _________;
        frac = _________;
    } else if (x < _________) {
        /* Denormalized result */
        exp = _________;
        frac = _________;

    } else if (x < _________) {
        /* Normalized result. */
        exp = _________;
        frac = _________;
    } else {
        /* Too big.  Return +oo */
        exp = _________;
        frac = _________;
    }

    /* Pack exp and frac into 32 bits */
    u = exp << 23 | frac;
    /* Return as float */
    return u2f(u);
}
```

네 가지 경우가 존재한다.

- $x < -149$

첫번째는 비정규화 표현으로 표시하기 어려운 아주 작은 수이다. 비정규화 부동소수점으로 나타낼 수 있는 가장 작은 수는 $0.00000000000000000000001 \cdot 2^{-126}$이다. 이는 $1 \cdot 2^{-(126+23)}$와 같다. 따라서 지수 x가 −149보다 작은 경우에 해당한다.

- $-149 \leq x < -126$

두번째는 비정규화 표현으로만 나타낼 수 있는 경우이다.

- $-126 \leq x < 128$

세번째는 정규화 표현의 범위에 해당하는 경우이다.

- $128 \leq x$

네번째는 x가 커서 단일 정밀도 표현법으로 나타낼 수 없는 경우이다.

[소스 코드]

```
float fpwr2(int x) {
        /* Result exponent and fraction */
        unsigned exp;
        unsigned frac;
        unsigned u;

        if (x < -126 /* Minimun denormailzed exponent value */ +
                -23 /* Length of fragment to represent the smallest number */) {
                /* too small. return 0.0 */
                exp = 0;
        frac = 0;
        } else if (x < -126 /* Minumum denormalized exponent value. */) {
                /* Denormalized result */
                exp = 0;
                frac = 1 << (unsigned)(23 /* Length of fragment */ +
                                       x +
                                       126 /* Min exponent * -1 */);
        } else if (x < 128 /* Maximum normalized exponent value + 1. */) {
                /* Normalized result */
                exp = 127 /* Bias */ + x;
                frac = 0;
        } else {
                /* Too big, return +oo */
                exp = 0xFF;
                frac = 0;
        }

        /* Pack exp and frac into 32 bits */
        u = exp << 23 | frac;
        /* Result as float */
        return u2f(u);
}
```

[실행 결과]

```
==== Test 6 Started ====
Assertion succeeded: fpwr2(4) == 16.0f
Assertion succeeded: fpwr2(5) == 32.0f
Assertion succeeded: fpwr2(7) == 128.0f
Assertion succeeded: fpwr2(-126 - 23) != 0.0f
Assertion succeeded: fpwr2(-126 - 23 - 1) == 0.0f
Assertion succeeded: fpwr2(127) != INFINITY
Assertion succeeded: fpwr2(128) == INFINITY
fpwr2(0) is 1.000000.
fpwr2(1) is 2.000000.
fpwr2(79) is 604462909807314587353088.000000.
==== Test 6 Succeeded ====
```