1. (2.59)

Least significant byte는 오른쪽의 8bit이다. 따라서 x의 왼쪽 1byte(8bit)와 y의 오른쪽 3byte(24bit)를 합치자. 함수명을 combination이라고 하였다.

1. (2.65)

최대 사용 가능 연산자가 12개이므로, 단축된 연산자를 사용해야 한다. 1이 홀수개면 1, 짝수개면 0을 반환하자. 기본적으로 XOR 연산을 이용해 반씩 shift 연산으로 옮기며 1과 1, 0과 0(짝수개의 1)이 만나면 0이 되도록 저장하자.

1. (2.69)

X를 n만큼 left shift한 값과 x를 32-n만큼 right shift하고 왼쪽 32-n bit만큼 0을 채운 값을 or 연산을 하면 된다.

1. (2.74)

기본적으로 x-y는 x+(-y)이다. -y는 2’s complement를 이용하면 쉽게 구할 수 있다. 덧셈에서 overflow가 일어나려면 기본적으로 두 부호가 같아야 한다. 부호가 다른 경우는 어떤 경우에도 overflow가 없다. Overflow가 일어난 경우는 더하기 전후 부호가 다른 경우이다.

1. (2.80)

Overflow가 있으면 안 되므로 2, 4로 나눈 값을 더하자. 이 때, 양수의 경우는 정수가 4로 나눈 나머지가 3인 경우 1을 더해주어야 하고 음수는 나머지가 0이 아닌 경우 1을 더해주되 나머지가 1인 경우는 추가로 1을 더해주어야 한다.

Check1 변수는 음수이면서 4로 나눈 나머지가 0이 아닌 경우 1이다.

Check2 변수는 음수이면서 -x를 4로 나눈 나머지가 1인 경우 1이다.

Check3 변수는 양수이면서 4로 나눈 나머지가 3인 경우 1이다.

1. (2.88)

우리는 실수형을 기본적으로 (-1)S×M×2E 라고 생각할 것이다.

먼저 0 10110 011의 경우 S=0, M=1+3/8, E=22-15=7이다. 즉, (11/8)×27 = 176.0이다. 이를 binary float형으로 표현하면 1.011×27이므로 exp는 7+7=14=1110(2)이다. 따라서 이를 format B로 표현하면 0 1110 0110(=176.0)이다.

1 00111 010의 경우 S=1, M=1+2/8, E=7-15=-8이다. 즉, (-10/8)×2-8 = -5/210이다. 이를 binary float형으로 표현하면 -1.010×2-8이므로 exp는 7-8=-1이다. Exp가 음수일 수는 없으므로 이 경우는 denormalized form을 사용해야 한다. 이 때 E=1-7=-6이고 -5/210 = (-5/24)×2-6 이므로 따라서 이를 format B로 표현하면 1 0000 0101(=-5/210)이다.

0 00000 111의 경우 denormalized 상태이다. S=0, M=7/8, E=-14이다. 즉, (7/8)×2-14 = 7/217이다. 이 값은 format B의 가장 작은 양수인 0 0000 0001 => 2-10보다 작다. 우리는 rounding을 +∞로 진행하므로 format B의 가장 작은 양수인 0 0000 0001(=2-10)로 rounding 가능하다.

1 11100 000의 경우 S=1, M=1, E=28-15=13이다. 즉, -1.0×213 = -8192.0이다. 이를 binary float형으로 표현하면 -1.0×213이므로 exp는 13+7=20이다. 그런데 20은 4bit로 표현이 불가하다. 따라서 +∞로 rounding을 진행하면 -∞의 바로 위에 있는 값을 반환해야 한다. 즉, 1 1110 1111로 표현 가능하며 이 값은 S=1, M=31/16, E=14-7=7이어서 (-31/16)×23=-248.0이다.

0 10111 100의 경우 S=0, M=1+1/2, E=23-15=8이다. 즉, (3/2)×28 = 384.0이다. 이를 binary float형으로 표현하면 1.1×28이므로 exp는 8+7=15이다. 그런데 이를 4bit로 표현하면 1111이어서 NaN 혹은 +∞이 되어버린다. 여기서 우리는 +∞로 normalize 하므로 +∞로 표현해야 하며 따라서 format B로 표현하면 0 1111 0000이다.

1. (2.95)

f가 NaN이면 f를, 아닌 경우는 0.5\*f를 반환하자. 절댓값이 감소하므로 overflow는 고려하지 않아도 된다.

1. (2.97)

정수형의 범위는 -231 ~ 231-1 이고, float 형으로 표현 가능한 +∞이 아닌 가장 큰 값은 01111111011111111111111111111111 = 1.11…1×2127 이므로 범위를 넘어가는 부분에 대한 rounding은 고려할 필요 없다. 다만 frac이 잘려서 발생하는 rounding은 고려해야 한다. 음수의 경우는 부호비트를 1로 정하고 2’s complement를 사용해 양수로 바꿔서 변환하도록 한다.

어떠한 특정 수를 표현하지 않는 경우는 NaN과 ±∞이다. 따라서 우리는 2에 NaN의 수를 합한 만큼 전체 32bit로 표현 가능한 수에서 빼면 된다. 32bit로 표현 가능한 값들의 개수는 232개이다. NaN은 exp가 0xFF이면서 frac이 0이 아닌 경우이고 ±∞는 frac이 0인 경우이므로 exp가 0xFF인 경우를 제외하면 된다. Exp는 8bit이므로 224개의 NaN과 ±∞이 있다. 따라서, 전체 표현 가능한 수의 개수는 232-224개이다.

곱셈의 경우를 먼저 생각해보자. Sign은 XOR 연산으로 처리한다. Exp는 서로의 지수를 더하되 overflow 여부를 잘 판단한다. Frac은 적당히 14bit만 남기고 곱해 2보다 커지는 경우 지수에 1을 더한다. 14bit만 남기는 이유는 overflow 없이 곱셈을 하면서 23bit보다 많은 bit가 생성되게 하기 위함이다.

덧셈의 경우는 고려해야 할 상황이 많아 간단히 알고리즘만 제시하려 한다. 먼저 x와 y의 부호가 같을 경우, exp를 비교해 둘의 차이만큼 x와 y 중 exp가 더 작은 변수의 frac을 right shift한 뒤 더한다. 이 때 overflow 여부를 판단해 overflow가 일어나면 적절히 ±∞를 반환한다. 만일 지수가 증가하는 경우라면 exp에 1을 더한다. 부호비트가 다른 경우, 둘의 지수를 비교하되 지수가 같으면 frac을 비교한다. 절댓값이 더 큰 변수의 부호가 합한 후의 부호가 되며 frac을 앞선 과정과 동일하게 shift하고 연산을 진행한다. 이후 필요에 따라 exp를 조절한다.