System Program HW 1

학번 / 이름: 2014310407 / 이 준 혁

- 구현

우선 총 길이인 WORD_SIZE, fraction 의 길이인 FRAC_SIZE, exponent 의 길이인 EXP_SIZE, exp bias 인 EXP_BIAS, sfp 상에서의 음의 무한대, 양의 무한대, nan, -0.0 인 NEG_INFINIY, POS_INFINIY, NAN, NEG_ZERO, 그리고 int 상에서의 최댓값, 최솟값인 TMAX 와 TMIN 이 모두를 define 으로 정의했다.

또한 매크로 함수를 통해 MSB, LSB 함수를 정의했는데 이는 각각 sfp 의 most significant bit, least significant bit 을 반환하는 함수로서 정의했다.

1. Int2sfp

가장 먼저 Sign 을 나타내는 sfp 값, exponent 를 나타내는 sfp 값, fraction 을 나타내는 sfp 값 세 변수를 선언 후 0 으로 초기화했다.

만약 입력 정수가 0 이라면, sfp 로 0 을 반환했고, 아닌 경우에는 right shift operation 을 통해 MSB, 즉 sign 값이 무엇인지 파악했다. 이 값이 1 인 경우에는 sign 을 나타내는 sfp 값에 NEG_ZERO(MSB 만 1, 나머지는 0), 즉 - 부호를 나타내는 값을 대입했다.

그 다음에는 exponent 와 fraction 을 알아야 하기 때문에, while 문을 이용해, input 이 0 이 나올 때 까지 2 로 계속 나누며 2 로 나눈 나머지가 1 인 경우에만 fraction 에 1 을 뒤에서부터 추가해줬다. 또 exponent 의 정수 값 역시 구했다. 이 때 exponent 의 정수 값이 16 이 되는 순간이 있으면, 이는 입력 받은 정수가 sfp 가 표현할 수 있는 최대치를 넘었다는 의미이므로 sign 에 따라 NEG_INFINITY 혹은 POS_INFINITY 를 반환했다.

이 과정을 통해 fraction 과 exponent 를 구했다면 fraction 의 위치를 맞춰줘야 하므로 fraction 의 leading 1 이 뒤에서부터 11 번째 위치에 위치하도록 shift 했다. 즉, 1.*** 꼴이 나오도록 한 것이다. 이후 leading 1 은 제거했다.

마지막으로 정수 값으로 구한 exponent 에 exponential bias 를 더해서 2 진수로 바꿔서 sfp data type 에 저장했다. 이렇게 sfp 로 구한 sign, exponent, fraction 을 모두 bit or operation 을 통해 결합시켜 반환했다.

2. Sfp2int

만약 입력이 POS_INFINITY, NEG_INFINITY 인 경우 각각 TMAX, TMIN 을 반환했다.

아닌 경우에는 우선 sfp input 의 MSB 를 구해서 sign 을 알아냈다. 그리고 sign 이 고려된 상황이므로 MSB 를 0 으로 바꿔서 양수인 경우만 생각하게 했다.

그리고 fraction 을 제거해서 exponent 만 남겨둔 후 이 값을 정수로 바꾸고 bias 를 빼 줬다. 만약 이 값이 최댓값, 즉 16 이라면 주어진 input 이 NAN 이라는 뜻이므로(INFINITY 가 될 수 없음) TMIN 을 반환했다.

아닌 경우에는 fraction 만 떼어 온 후에 leading 1 을 복원시켰다. 이후 exponent 에 맞게 shift 를 시켜 정확한 값을 가지게 했다. 이후 type casting 을 통해 값을 저장하고, 값에 sign 을 곱한 값을 반환했다. (sign 이 1 인 경우 -1 을 곱해 반환, 아닌 경우 그대로 반환)

3. float2sfp

Int2sfp 와 같이 Sign 을 나타내는 sfp 값, exponent 를 나타내는 sfp 값, fraction 을 나타내는 sfp 값 세 변수를 선언 후 0 으로 초기화했다.

float 에서도 0 인 경우에는 똑같이 sfp 로 0 을 반환해줬다. 또한 float 에서는 shift operation 을 사용할 수 없으므로 <연산자를 사용해 0 보다 작은 경우에만 sign 을 나타내는 sfp 값을 NEG_ZERO 로 바꿨다. 또 sign 에 대한 정보를 모두 알게 되었으므로 양수로 만들어서 계산했다.

이후에는 정수 부분과 소수 부분을 분리했다. 그리고 정수 부분이 0 이면 exponent 를 처리하는 방식이 달라지므로 이 경우에 대한 flag 를 만들었다.

정수 부분에 대해서는 int 에서 했던 것과 동일하게 진행했다. 소수 부분에 대해서는 2 로 나눈 나머지를 이용할 수 없기 때문에 소수 부분에서 2^(-n)을 계속 빼면서, 0 이 되거나 exponent 가 -15 가 되거나 총 fraction 의 길이가 16 이 될 때까지 반복했다. 이 때 exponent 는 처음 nonzero 인 숫자가 나오는 위치를 의미한다. 만약 정수 부분이 0 이 아니면 이 값은 양수 혹은 0 이 되고, 정수 부분이 0 인 경우에는 음수가 나오게 된다. 이 값이 -15 까지 가게 된다면 denormalized case 가 되기 때문에 종료했다.

즉, exponent 가 0 이상인 경우, -1 ~ -14 인 경우에는 적당한 shift operation 을 통해 fraction 이 제 위치를 찾아가도록 했다. -15 인 경우에는 exponent 를 -14 로 고정하고 계속 2^(-n)을 빼 주며 fraction 의 값을 채워 나갔다.

이후에는 exponent 에 bias 를 더한 후에 exponent 를 나타내는 sfp 에 이진수로 저장했다. 이후 각 sfp 부분을 다 더해서 그 값을 반환했다.

4. Sfp2float

Int 인 경우와 마찬가지로, sign 과 exponent 에 대한 정보를 얻어냈다. 이후 fraction 을 계산해야 하는데, infinity, denormalized case 와 normalized case 가 다르기 때문에 다른 방식으로 계산했다.

Exponent 가 16 인 경우, 즉 infinity case 에서는 float 에서의 infinity 인 2^128 혹은 -2^(128)을 반환했다.

Exponent 가 -15 인 경우, 즉 denormalized case 에는 fraction 을 계산하고 2^(-14)를 곱해서 반환했다. 다만 이 때 leading 1 은 추가하지 않았다.

Exponent 가 -15 가 아닌 경우, 즉 normalized case 에는 fraction 을 계산하고 2 ^ (exp)를 곱해서 반환했는데, 이 때는 fraction 을 바꿀 때 leading 1 을 추가했다.

5. Sfp_add

우선 예외 처리부터 시작했다. 만약 두 수중 하나가 NAN 인 경우에는 NAN 을 반환했다. +∞ 와 -∞를 더하는 경우 역시 NAN 을 반환했다. 또한 +∞와 normal value 를 더하면 +∞, -∞와 normal value 를 더하면 -∞를 반환하도록 했다.

일반적인 경우에는 우선 a 와 b 의 sign 을 MSB 를 통해 계산했고, MSB 를 0 으로 만들었다. 또한 a 와 b 의 exponent 값을 계산했다. 또한 fraction 역시 계산했는데 이 때 normalized case 에서는 leading 1 을 추가했다. 이후 rounding 을 위해 fraction 을 왼쪽으로 2 만큼 shift 시켰다.

이제 exponent 끼리 값을 비교해, exponent 가 작은 쪽의 fraction 을 그 차이만큼 right shift 했다. (동일한 경우에는 shift 를 수행하지 않음) 그 후에는 두 개의 sign 을 비교했다.

1) 두 개의 sign 이 같은 경우

Sign 을 두 개의 sign 으로 설정하고 fraction 은 두 fraction 의 합으로 계산했다. 이 때, overflow 가 발생해서 leading 1 이 다음 위치로 넘어갈 수 있으므로, 이 경우에는 계산된 fraction 을 right shift 해 주고, exponent 를 하나 증가시켜주었다.

2) 두 개의 sign 이 다른 경우

두 개의 frac 이 같은 경우에는, 부호만 다른 같은 수를 더한 것이므로 0 을 반환했다. 아닌 경우에는 sign 을 fraction 이 큰 쪽의 것으로 지정하고 fraction 을 큰 쪽 – 작은 쪽으로 계산했다.

이렇게 계산이 끝났는데, 만약 exponent 과 frac 이 모두 0 이면, 값이 0 이라는 것이므로 0 을 반환했다. (0 + 0 을 계산한 경우가 해당) 또, 만약 leading 1 이 fraction 앞의 위치에 없는 경우에는 있을 때 까지 left shift 를 통해 leading one 을 만들어 주었다.

이렇게 계산한 후에 frac 의 맨 마지막 두 비트, 즉 rounding 할 비트들을 살펴보았다. 만약 11 로 끝나거나 10 으로 끝날 때 10 앞의 비트가 1 이면, 즉 110 이면 round up 을 해 주었다. (Round to even) 아닌 경우에는 아무것도 안 하면서 round down 을 해 주었다.

이제 마지막으로 exponent 가 31, 즉 최댓값인 경우에는 Infinity 인 경우이므로 +∞ 혹은 -∞를 반환했다. 아닌 경우에는 exponent 와 sign 을 계산해서 fraction 과 함께 다 더해서 반환했다.

6. Sfp2bits

마지막 NULL 문자까지 포함해 총 WORD_SIZE + 1 만큼의 크기의 문자 배열을 할당했고, 뒤에서부터 result 의 LSB 에 해당하는 문자로 배열을 채웠다. 매 반복마다 result 를 오른쪽으로 한 번 shift 시켰다. 이 과정을 WORD_SIZE 만큼 반복했다. 이후 배열의 맨 마지막 값을 NULL 문자로 채웠다.