#HW 10 2017069598 박상지

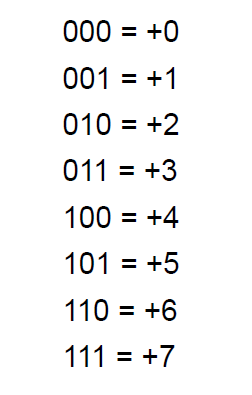
chapter 3 Arithmetic for computers

1. integer addition & subtraction

우리가 사용하는 digital computer에서는 bit 수에 제한이 있다. 이것을 finite precision이라 불리는데 표현 범위가 벗어나기 때문이다. (overflow)

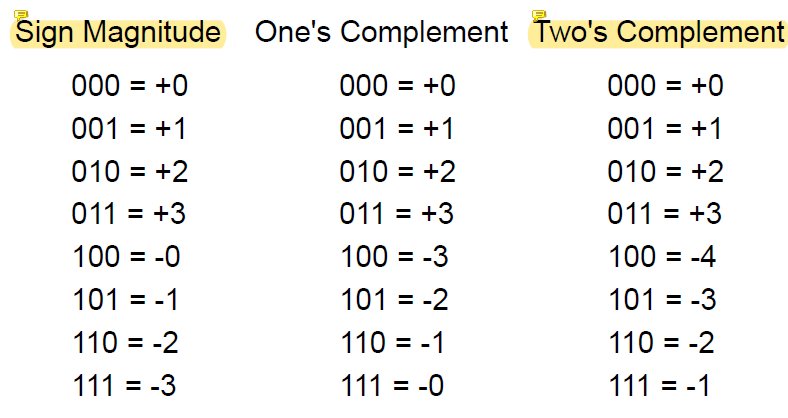
representation of integers

정수 값을 표현할 때 unsigned 와 signed number가 있다. unsigned integers은 말그대로 부호가 없으며 음수 부분을 생각하지 않고 표현을 한다.



하지만 signed number, 음수 부분까지 표현할 때는 여러가지 표현이 있는데 sign magnitude, 1의 보수, 2의 보수 가 있다.

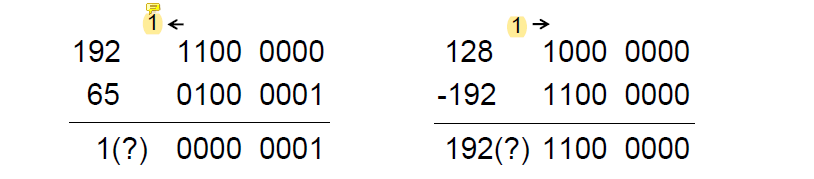
sign magnitude는 맨 위에 첫번째 bit를 부호를 표현하는데 사용된다. 나머지 값은 절댓값으로 사용하는데, 문제는 1와 -1을 더하면 0이 나오질 않는다. 1의 보수도 마찬가지로 1과 -1을 더하면 0의 값이 나오질 않는다. 그렇기 때문에 대게 2의 보수를 사용한다.



또 다른 표현으로는 biased notation이 있는데, 간단히 말하자면 0의 값에 대한 기준점을 잡는다. 예를 들면 Bias 3이라 하면 이진법에서 3의 값이 0이 되는 것이다. 그러면 011 값이 0이 되고 010의 값이 -1이 된다.

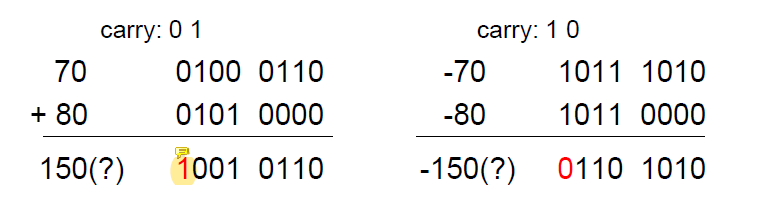
carry/borrow in unsigned arithmetic

carry와 borrow는 표현범위에 벗어 날 때 값이 저장되는데,



192 와 65를 더하면 결과 값이 1이 나온다. 표현 범위에 벗어났기 때문이다.

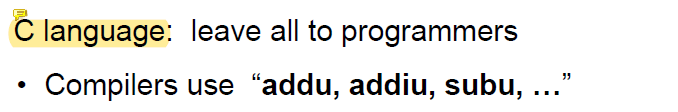
overflow in signed arithmetic



그림과 같이 signed number 연산에서 70과 80을 더하면 150이 나와야 한다. 하지만 signed number에서는 맨 앞의 1 bit은 부호를 판단하기 때문에 표현 범위에 벗어 난다. 즉 150을 표현할 수가 없다. 이런 오류는 carry의 값이 서로 다르면 XOR gate에서 발견해서 overflow 발생 유무를 알린다.

질문) 그림과 같이 150와 같은 overflow는 확실한 오류이다. 그렇다면 이런 문제는 프로그래머에서 해결을 해야 하는가? 아니면 하드웨어에서 해결을 해야 하는가?

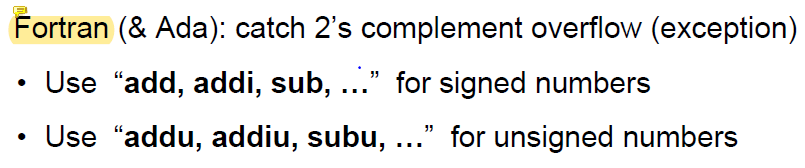
<관점 – 프로그래머 해결>



2의 보수에서 overflow가 나는 것은 전반적으로 프로그래머 잘못이기 때문에 충분한 크기의 데이터 타입을 사용해야 한다. 여기서 하드웨어는 절대로 개입하면 안된다.

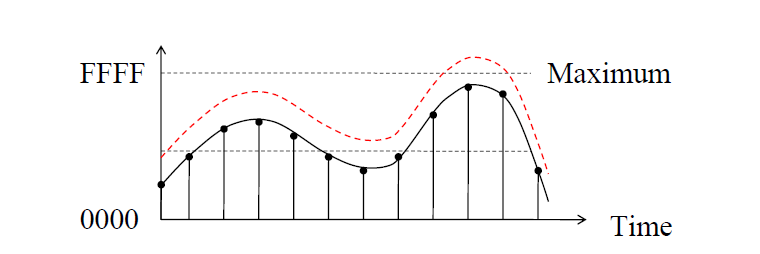
addu, addiu, subu, 와 같은 instruction은 overflow를 무시하는데, 여기서 u는 unsigned를 가리키는 것이 아니라 2의 보수에 대한 오버프로우를 무시하라는 의미이다.

<관전 – 하드웨어 해결>



signed 연산에서 overflow가 나는 것은 백퍼센트 bug이기 때문에 하드웨어는 최소한의 bug를 잡아서 stop을 시켜줘야 한다. add, addi, sub와 같은 instruction을 사용하는데 오버프로우를 예상 또는 발견하는 역할을 한다.

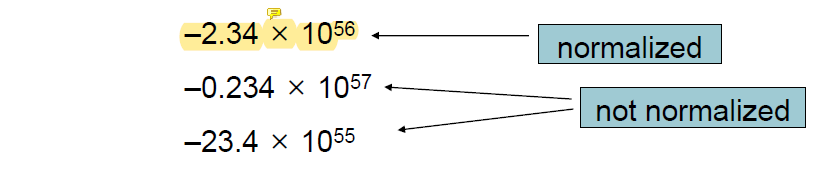
Arithmetic for Multimedia



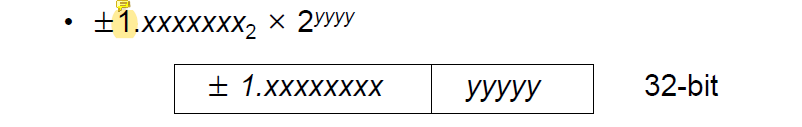
multimedia 연산도 integer 연산인데, 해당 연산에서는 overflow의 개념이 있지 않다. 오디오 데이터를 예를 들면 16-bit integer로 표현을 하는데 폭이 그림과 같이 빨간 선으로 최대값을 넘어서면 표현을 못하는 게 아니라 최대 값 FFFF을 반환해 준다.

1. Floating-Point Real Numbers

FP는 소수점을 표현하기 위해 사용한다.



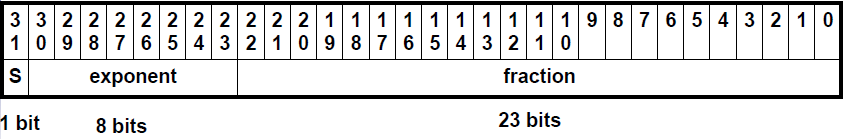
위의 3개의 예를 보면 값은 같지만 표현하는 방식이 다르다. 그렇기 때문에 맨 위에 값을 표준 표현 방식으로 지정하였다.

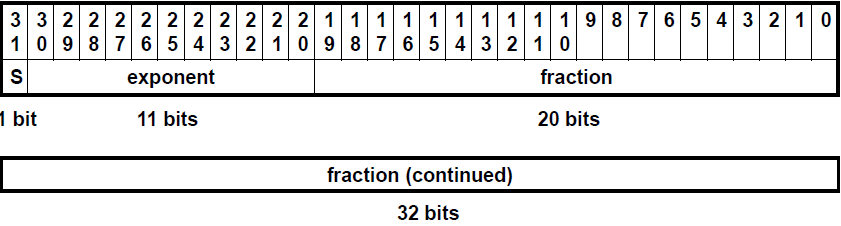


이진법에서는 fraction 부분과 지수부분으로 표현되는데 fraction 부분에서 항상 실제 값이 1로 시작하기 때문에 굳이 표현을 할 필요가 없다. 즉 1 bit을 아낄 수 있다.

Floating Point Standard

1980년도에 제작이 된 표준 표현법은, 컴퓨터가 처음으로 네트워킹이 되던 시절에 floating point를 표현 방식을 컴퓨터 회사간에 독자적으로 표현을 하였기 때문에 호환성이 떨어졌다. 그래서 floating point standard가 생겼다.





single-precision range

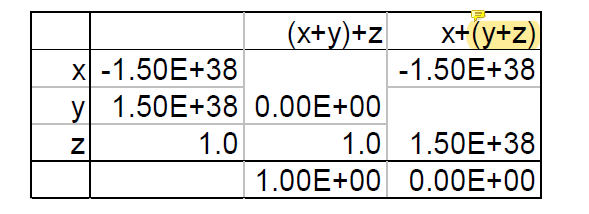
32-bit notation을 사용하면 표현할 수 있는 최대 절대 값은 10^38, 최소 값은 10^(-38)이다.

single-precision과 double-precision 두 가지를 만든 이유는 single-precision으로 표현이 안된 수가 당연히 종종 나왔다. 하드웨어가 미약하였기 때문에 double을 사용하지 않은 이유도 있다. 하지만 하드웨어가 허용되면 일반적으로 double-precision을 사용하는게 더 좋다.

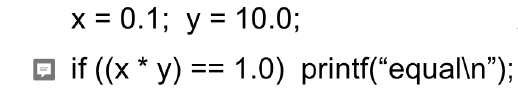
오늘날에 하드웨어는 범용 컴퓨터에서 사용할 때에는 double을 사용한다. float를 쓸 때에는 이유가 있어야 한다. 예를 들면 작고 쉬운 임베디드 연산을 사용할 때 사용하지만 그 외에는 double을 사용해야 한다.

Associativity

x+y+z 연산을 보면 해당 더하기 연산에서는 순서에 상관없이 똑 같은 결과 값이 나와야 한다. 하지만 FP 연산은 근사 계산이기 때문에 순서에 대한 결과 값이 달라진다.

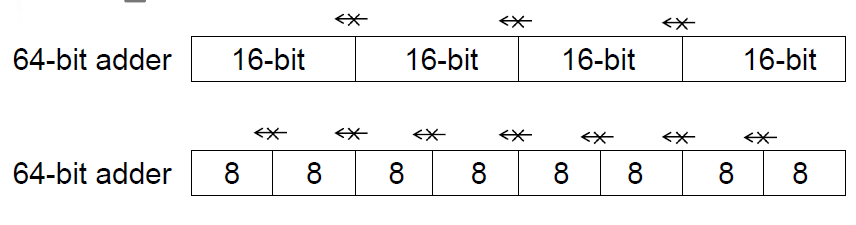


Y와 Z를 더하기 연산을 할 경우 Z에 대한 Y의 지수 값을 맞춰서 더해줘야 한다. 그러면 지수분이 너무나 크기 때문에 잘라 줘야한다. 잘라 줄 경우 결과적으로 반올림을 해야 하기 때문에 Y + Z = Y가 나오는 상황이 나온다. 그 상태에서 X랑 더하면 0이 나오는데 해당 작업은 순서에 따라 결과 값이 달라 진다. 하지만 과학 계산에서 오차가 1만큼 나오는 것은 문제가 되지 않는다.



해당 작업에서 equal 결과 값이 나오지 않는다. fraction bit에 영향이 가는데, FP으로 연산을 하면 메모리 저장될 때 반올림에서 올라가기 때문에 1보다 약간 큰 값으로 근사 계산으로 되어진다. 그렇기 때문에 1과 같지 않아진다.

1. Arithmetic for multimedia



프로세서들은 multimedia을 어떻게 효과적으로 처리를 할까?

640bit adder가 있다 가정하에 16-bit carry라인을 끝어 버리면 4개의 독립적인 16-bit adder을 사용한다. 16-bit multimedia 데이터를 4개를 64-bit adder로 저장하고 이것을 4개를 병렬로 연산할 수 있다. 4개의 multimedia를 데이터를 동시에 연산할 수 있기 때문에 4배속으로 할 수 있다

.