비트로 음수를 표현하는 방법으로는 부호 절댓값 방식(**Sign Magnitude**), 1의 보수 방식(**One's complement**), 2의 보수방식(**Two's complement**)으로 이 세 방식이 있다. 현재 우리가 사용하는 대부분의 컴퓨터는 이 세 방식에서 2의 보수 방식을 채택하였다. 비트로 음수를 표현하는 방법들과 2의 보수가 채택된 이유에 대해서 알아보자.

첫 번째 방법으로는 부호 절댓값 방식은 절댓값을 이용한 방법이다. 이 방법을 이용했을 때의 예시를 5로 들자면 양수로는 맨 앞에 존재하는 부호비트를 양수를 나타내는 0으로 설정하고, 5를 의미하는 2진수인 101을 이어 붙여 0101로 표기하게 된다. 음수 5의 경우는 맨 앞 부호비트만을 1로 바꾸어 1101로 -5를 표기한다.

두 번째 방법으로는 1의 보수를 이용하는 방법이다. 5를 예시로 들자면 4비트를 기준으로 할 때 5의 2진수인 0101에 더했을 때 1111이 되도록 하는 수인 1010이 -5를 1의 보수로 표현하는 방법이다. 다르게 설명하자면 5의 비트에 베타적 논리합(exclusive OR)연산인 XOR연산을 적용하게 되면 모든 비트가 반전되어 1의 보수인 1010의 값을 얻을 수 있다.

세 번째 방법으로는 2의 보수를 이용하는 방법이다. 2의 보수를 구하는 방법은 간단하다. 1의 보수를 구한 뒤에 +1을 하게 되면 2의 보수가 된다. 5의 경우는 1의 보수가 1010이므로 2의 보수는 1011이다. 최근에는 2의 보수가 가장 많이 사용되고 있으며, 2의 보수 방식만의 장점 또한 확실하다. 2의 보수 방식을 이용했을 때의 장점을 알기 위해 우선 부호 절댓값 방식의 경우와 1의 보수 방식을 이용한 경우의 단점을 먼저 알아보았다. 단점 및 2의 보수 방식 이용하였을 때의 보완되는 이유는 아래와 같다.

첫 번째 단점으로는 0의 표현 방법이 통일되지 않는다. 2의 보수 방식을 제외한 위에서의 두 방법으로는 0을 표현할 때 0000과 1000으로 두 가지 표현식이 나오게 된다. 이 경우에는 메모리가 낭비될뿐더러 가시성이 떨어지며, 실무단계에서는 협업 과정에서 충돌이 날 수 있다.

두 번째 단점으로는 2의 보수보다 많은 수를 표현할 수 없다. 첫 번째 단점이 원인이 되어 같은 공간으로 비교했을 때 2의 보수와 비교하여 숫자 하나를 표현하지 못하게 된다. 예를 들게 된다면 7을 음수로 표현할 때 부호 절댓값 방식으로는 1111이며, 1의 보수를 이용한 방법의 경우는 1000이다. 같은 4비트의 크기로 표현할 수 있는 숫자의 범위로 비교했을 때 앞의 두 방법은 -7부터 7까지의 수를 표기할 수 있다. -7을 부호 절댓값 방식의 경우에는 -8을 표현하는 것을 시도하면 1111에서 0000이 되며, -7에서 0으로 값이 바뀌게 된다. 1의 보수의 경우에는 1000에서 0111이 되며 -7에서 7로 바뀌게 된다. 이는 0을 두 가지 방식으로 표현하기 때문에 표현할 수 있는 숫자가 한 가지 줄어든 것이다.

그러나 2의 보수의 경우에는 위 두 경우가 보완된다. 0을 표현하는 경우는 1의 보수의 경우 0000과 1111이 있으나 2의 보수의 경우 -0을 표현하는 1의 보수인 1111에서 +1을 하여 0000을 0으로 정하였으며, 1의 보수의 경우에는 0부터 음수를 나열하게 되면 1111, 1110, 1101, … 으로 진행되나, 2의 보수의 경우 0000, 1111, 1110, 1101, … 으로 진행되기에 -7 같은 경우 2의 보수는 1001이 되므로 1000으로 -8을 표현할 수 있게 된다. 1의 보수와 2의 보수 표현 범위를 수식으로 표현하게 되면 다음과 같다. 1의 보수의 경우는 –(2n-1+1) ~ 2n-1-1이며, 2의 보수의 경우는 –2n-1 ~ 2n-1-1이다. 1의 보수의 음수 범위에 +1이 되는 이유는 위에서 설명했듯 -0의 표현인 1111로 인해 생긴 값이다.

이에 따라 생기는 2의 보수의 장점으로는 오버플로우(overflow)와 언더플로우(underflow)의 처리 과정이 편리해진다. -8의 2의 보수인 1000에서 1을 빼게 된다면 0111이 되며 7의 값을 갖는 언더플로우가 발생하게 된다. 오버플로우나 언더플로우의 처리 과정이 편해지는 이유는 다음과 같다. 4비트의 값에 200을 넣으려 할 때 오버플로우가 발생하며 비트로는 1000 즉, -8이 나오게 된다. 200의 오버플로우 후 값은 200을 4비트에서 2의 보수로 표현할 수 있는 16으로 나눴을 때의 나머지인 8이 나오게 된다. 8의 경우 또한 7(0111)에서 오버플로우가 발생하여 -8(1000)의 값을 갖게 됨을 알 수 있다. 이는 즉, 오버플로우나 언더플로우가 발생했을 때, 어느 값으로 인해 해당 경우가 발생하였는지 찾기 쉽기에 에러처리 및 시스템 유지보수에서의 이점을 가질 수 있다. 만약 1의 보수를 이용하게 된다면 비트 계산에서 발생하는 오버플로우나 언더플로우를 처리할 때 더 많은 시간과 노력이 소요될 것이다.

2의 보수의 또 하나의 장점으로는 연산이 더욱 편리해진다는 장점이 있다. 4비트 기준으로 비트의 뺄셈을 진행할 때 7(0111)에 -3(1101)을 하게 되면 7과 -3의 비트값을 더했을 때 10100의 값이 나오며, 이때 4비트를 벗어나 맨 앞에 1이 생기게 되는데 최상위비트를 벗어나 자리가 넘칠 경우 이 값을 캐리라고 한다. 이때 캐리 값을 버리게 되면 7-3을 한 값인 4(0100)가 나오게 된다. 결과가 음수인 뺄셈을 할 경우, 2(0010) – 5(1011)의 계산을 할 때 2와 -5의 비트를 더하게 되면 정상적으로 -3(1101)의 결과가 나오게 된다. 1의 보수 방식을 이용했을 땐 1의 보수에서 2의 보수로 변환하는 과정에서 비트값에 1을 더하듯 1의 보수로 비트연산은 진행하게 되면 연산 후에 비트에 -1을 해야 하므로 추가적인 연산이 필요하다. 이처럼 2의 보수법을 활용했을 때 연산 과정이 복잡하지 않아 빠른 연산이 가능하게 된다.

2의 보수의 장점을 요약하자면 부호 절댓값 방식과 1의 보수 방식보다 한 가지 수를 더 표현할 수 있다는 장점이 되며, 이 경우 생기는 이점으로는 1의 보수와 비교했을 때 같은 저장공간과 대비해서 더 많은 수의 표현이 가능해지기에 메모리의 효율적 사용이 가능해지며, 오버플로우와 언더플로우 같은 예외적 상황에서의 디버깅이 가능하다. 또한 비트 간의 연산이 효율적이기에 빠른 연산속도를 가질 수 있기에 CPU 또한 효율적인 사용이 가능하다.

이와 같은 이유로 현재 많은 컴퓨터에서 음수 표현은 2의 보수 방식으로 사용하고 있다.