

2진수의 기계적인 활용

2021 동탄고등학교 30925 전한결

April 16, 2021

Contents

1 10진수와 2진수	3
1.1 10진수	3
1.2 기계 장치에서 정보의 전달	3
1.3 2진수	4
1.4 2진수 읽기	4
1.5 기계 장치에서 2진수의 활용	5
2 2진수 처리	5
2.1 자료 저장에 대한 컴퓨터 구조	5
2.2 정수	6
2.2.1 정수의 표현법	6
2.2.2 2진수의 덧셈	7
2.2.3 정수의 연산	8
2.3 유리수	9
2.3.1 고정 소수점 방식 실수 표현	9
2.3.2 부동 소수점 방식 실수 표현	10
2.4 문자	10
2.5 이미지	12
2.5.1 이미지의 물리적 특성과 전산화 방법	12
2.5.2 이미지 압축	13
2.6 오디오	13
2.6.1 소리의 물리적 특성과 전산화	13
2.6.2 오디오 압축	13
2.7 영상	14
3 논리 연산	14
3.1 논리합과 논리 부정	14
3.2 논리합과 논리 부정을 통한 기타 논리 연산의 구현	15
4 나의 생각	15
4.1 세상을 인식하는 방법	15
4.2 기술 발전에 따른 기술 갈등 발생	16
5 참고문헌	17

Abstract

우리가 10진수 체계를 사용하는 것처럼, 컴퓨터 속에서는 2진수 체계를 사용해 자료를 처리한다. 진수 체계는 수를 그 진수에 대한 방정식으로 나타냈을 때 그 계수들을 나열한 것이고, 2진수는 2를 방정식의 문자로 한다. 2진수로는 일정한 방식에 따라 여러 자료를 나타낼 수 있으며, 그 안에는 자료의 정확도를 벼려 자료의 처리 효율성을 위하는 처리 과정도 있다. 2진수를 사용한 모든 연산은 OR 연산과 NOT 연산으로 풀어서 나타낼 수 있으며, 해당 연산이 합쳐져 복잡한 연산을 가능하게 한다. 컴퓨터는 내가 세상을 인식하는 방법을 바꿨으며, 컴퓨터를 통해 사물들의 인식에 대해 다시 생각해 볼 기회를 가질 수 있었다. 기술 발전은 우리가 예상하지 못한 방향으로 진행되므로, 기술 발전에 대비하기 위한 전지구적 합의가 필요하다.

1 10진수와 2진수

1.1 10진수

우리는 살면서 10진수를 통해 수를 표현한다. 인류가 굳이 10진수를 사용하는 이유는 정확히 밝혀지지 않았지만, 대부분의 인류학자들은 그것이 사람의 손가락으로 수를 세는 것에서 수가 기원했기 때문이라고 말한다.

10진수는 0부터 9까지의 10가지 모양으로 없음에 해당하는 수부터 아홉에 해당하는 수까지를 표현한다. 이 숫자를 일의 자리 숫자라고 한다. 만약 아홉 이상의 수를 표현해야 할 때에는 숫자 옆에 하나에 해당하는 수를 쓰고 숫자가 할 수 있는 경우의 수가 한 번 다 한 후에 몇이 더 큰지를 일의 자리 숫자에 나타낸다. 이때 일의 자리 숫자 옆에 있는 숫자를 십의 자리 숫자라고 한다. 십의 자리 숫자도 일의 자리 숫자와 같이 10가지의 경우의 수를 가질 수 있으며, 이 경우의 수를 모두 소진했을 때에는 또 옆에 숫자를 쓰고, 이때 십의 자리 숫자 옆에 쓴 숫자를 백의 자리 숫자라고 한다. 우리가 사용하는 10진수에서는 이러한 진행을 이어가며 그 값을 나타낸다.

24

1.2 기계 장치에서 정보의 전달

컴퓨터는 기계적인 신호로 이루어져 있고, 기계적인 신호는 전압이 강한지 약한지를 통해 그 정보를 전달한다. 대부분의 기계 장치에서는 전력이 흐르지 않는, 즉 전압이 0인 지점과 전압이 최대¹인 지점의 두 가지를 일정한 기준으로 나누어 구분한다. 이때 일정한 지점이 어디냐에 따라서 CMOS와 TTL 방식으로 구분한다. 즉, 기계장치에서는 전압의 세기로 총 두 가지 경우를 나타낼 수 있는 것이다.

하지만 두 가지 경우로 의미있는 정보를 전달하기는 쉽지 않다. 수학자들은 당연히 이러한 문제를 알고 있었으며, 이 문제를 해결하기 위해 10진수에서 값을 표현하는 방식을 빌려왔다. 이것을 2진수고 부른다.

¹보통은 이 최대점을 전원전압이라고 하고, 전원전압을 어떻게 설정할지는 기계를 설계하는 사람이 정한다.

1.3 2진수

10진수의 경우, 글자 하나를 써서 표현할 수 있는 상황의 개수가 10개이기 때문에 10가지를 넘는 정보를 표현하기 위해서 숫자 옆에 다른 숫자를 써서 10가지를 초과하는 정보를 표현한다. 2진수도 이 방식을 적용해서 사용할 수 있다. 단지, 수를 구성하는 숫자가 2개밖에 없을 뿐이다.

2진수는 0부터 1에 해당하는 2가지 모양으로 없음에 해당하는 수부터 하나에 해당하는 수까지를 표현한다. 이 숫자를 10진수에서와 같이 일의 자리 숫자라고 한다. 만약 하나 이상의 수를 표현해야 할 때에는 숫자 옆에 하나에 해당하는 수를 써서 숫자가 할 수 있는 경우의 수가 한 번 다 한 후에 몇이 더 큰지를 이의 자리 숫자에 나타낸다. 이 숫자를 이의 자리 숫자고 한다.² 이와 같은 방식으로 사의 자리 숫자, 팔의 자리 숫자, 십육의 자리 숫자 등을 정의하여 수를 표현한다.³ 이 과정은 10진수에서 숫자를 표현하는 방법과 그 방법이 동일하다.

11000₂

1.4 2진수 읽기

위에 11000₂라고 2진수로 표현한 수는 우리가 24라고 하면 아는 수와 같은 값을 나타낸다. 2진수를 보고 그 크기는 어떻게 가늠할 수 있을까?

우리가 10진수를 읽을 수 있는 것은 10진수에 대해 배웠기 때문이다. 2진수도 배운다면 누구나 읽어낼 수 있다. 일단 10진수에 대해서 다시 한 번 알아보자.

예를 들어, 우리 눈 앞에 세 자리 숫자가 있다고 하자. 이 숫자는 10진수로 표현되었다.

794

이 숫자는 백의 자리 숫자가 7, 십의 자리 숫자가 9, 일의 자리 숫자가 4이다. 이 값은 다음과 같이 표현될 수도 있다.

$$7 \times 100 + 9 \times 10 + 4 \times 1$$

또한 수학적으로는 다음과 같은 방식으로 표현할 수도 있다. 다음은 10에 대한 2차식이다.

$$7 \times 10^2 + 9 \times 10^1 + 4 \times 10^0$$

10진수는 가장 오른쪽에 있는 수(일의 자리 숫자)에다가 10^0 을 곱하고, 그 값에 그 다음 자리에 있는 수(십의 자리 숫자)에다가 10^1 을 곱한 값을 더하고, 이러한 과정을 숫자가 끝날 때까지 반복한 것이다. 이를 일반화하면 다음과 같다. n 진수는 가장 오른쪽에 있는 수(n^0 의 자리 숫자)에다가 n^0 을 곱하고, 그 값에 그 다음 자리에 있는 수(n^1 의 자리 숫자)에다가 n^1 을 곱한 값을 더하고, 그 다음 자리에

²왜냐하면 2진수 10에 해당하는 수를 우리가 이라고 부르기 때문이다. 10진수의 경우도 10에 해당하는 수를 우리가 십이라고 부르기 때문에 십의 자리 숫자라고 하는 것이다.

³2진수는 10진수 숫자와 헷갈릴 수 있기 때문에 그 숫자가 어떤 표기법을 통해 표현되었는지를 숫자 옆에 아랫첨자로 나타낸다.

있는 수(n^2 의 자리 숫자)에다가 n^2 을 곱한 값을 더하고... 결국 n 에 대한 다항식으로 표현할 수 있는 수를 말한다. 그렇다면 2진수 11000은 다음과 같은 (2에 대한) 다항식으로 표현할 수 있다.

$$\begin{aligned} & 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\ & = 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1 \\ & = 1 \times 16 + 1 \times 8 \\ & = 16 + 8 \\ & = 24 \end{aligned}$$

1.5 기계 장치에서 2진수의 활용

다시 수학자들의 이야기로 돌아와보자. 수학자들은 n 진수로 표현한 숫자가 결국 n 에 대한 다항식으로 표현 가능한 수라는 것을 발견해내었다. 2진수 숫자를 하나의 전기 신호로 표현하고, 그 전기 신호를 여러개 작성하면 더 다양한 값을 나타낼 수 있다는 사실을 알아낸 것이다. 전선을 여러개 사용하는 방법과 시간차를 두고 여러 값을 보내는 방식 중에서 수학자들은 시간차를 두고 여러 값을 보내는 방식을 선택했다. 컴퓨터와 같은 기계장치들은 시간차를 두고 전송되는 2진수 전기 신호를 일련의 방식에 따라 이해하고 처리한다.

2 2진수 처리

컴퓨터가 2진수를 사용하여 숫자를 전송하고 처리한다는 것을 알게 되었다. 하지만 컴퓨터는 우리가 생각하는 것만큼 그렇게 단순하게 작동하지 않는다.

우리가 수라고 할 때에는 보통 실수를 나타낸다. 실수에는 유리수와 무리수가 있고, 각각에는 0보다 큰 양수와 0보다 작은 음수, 그리고 0이 있다. 앞으로는 이러한 수들을 종류에 따라 컴퓨터에서 어떻게 표현하는지 알아보고, 컴퓨터에서 그 표현방식에 따라 어떤 연산을 취하는지를 알아보고자 한다.

2.1 자료 저장에 대한 컴퓨터 구조

컴퓨터 구조에 대해서 이야기하자면 양이 너무나 방대해 한 권의 책을 써야 할 정도이겠지만, 이번 장에서는 간단히 설명에 필요한 만큼만 서술하려고 한다.

컴퓨터의 구조는 크게 입력부분, 연산부분, 저장부분, 출력부분⁴으로 나눌 수 있다. 각각의 부분들은 그 이름에 맞게 작업을 수행하는데, 이번에 알아볼 부분은 저장부분이다. 컴퓨터는 2진수로 정보를 저장하고 전달하고 활용한다고 했다. 따라서 0과 1을 많이 저장할 수 있게 고안된 장치를 사용한다. 우리가 사용하는 컴퓨터에서 그 기능은 주 기억 장치와 보조 기억 장치가 수행한다.

주 기억 장치는 자주 책상에 비교되곤 한다. 우리가 업무를 볼 때에 당장 작업해야 하는 문서나 참고자료, 필통이나 도장 등은 책상에 놓고 사용하기 때문에 주 기억 장치가 넓을수록 한 번에 많은 작업을 수행할 수 있다.

⁴이 이름은 설명을 위해 임의로 지은 것이다.

반면에 보조 기억 장치는 책장에 비교되곤 한다. 당장 쓰지는 않지만 언젠가 쓸 일이 있을 때에 그 때 그 때 꺼내서 책상에 올려놓기 위해 사용되는 자료들을 보관해놓는 곳이다. 따라서 보조 기억 장치가 넓을수록 많은 정보를 오래 기록해놓을 수 있다.

이 주 기억 장치와 보조 기억 장치의 크기는 **비트(bit)** 또는 **바이트(Byte)**라는 단위로 나타낸다. 기억 장치에 수용할 수 있는 2진수의 자릿수를 비트(bit)로 나타낸다. 예를 들어 지금 16개의 똑딱이 스위치가 있다고 치자. 이 스위치는 순서를 가지고 있고 각각 켜지거나 꺼진 상태를 나타낼 수 있다. 이 16개의 스위치들은 함께 해서 16자리 2진수 숫자를 나타낼 수 있으므로 16비트 기억장치라고 볼 수 있다. 바이트는 나중에 다시 설명하겠지만, 8개의 비트를 한 단위로 묶은 것이다. 왜 굳이 8개를 하나의 바이트로 묶었는지는 나중에 설명하게 될 것이다. 어쨌든 16비트 기억장치인 16개의 똑딱이 스위치는 2비트짜리 기억장치가 되는 것이다.

$$16 \text{ bit} = 2 \text{ Byte}$$

1바이트는 8비트이기 때문에 총 $2^8 = 256$ 개의 경우를 표현해낼 수 있다.

참고로 16진수는 0부터 9까지의 숫자에 A부터 F까지의 알파벳을 함께 하여 만든 수 체계로, 4비트를 한 글자로 표현할 수 있다. 2진수를 10진수로 변환하면 숫자와 변환된 숫자의 연관성이 나타나지 않아서 불편했기 때문에 16진수 체계가 고안되었다. 16진수 표기법 따르면 다음 등식이 성립한다.

$$11010010_2 = D2_{16}$$

16진수는 알파벳을 사용하는 이유로 익숙하지 않아서 16진수 대신 8진수 체계를 사용하기도 했는데, 16진수가 표현할 수 있는 비트 수는 2의 거듭제곱 꼴로 나타낼 수 있지만 8진수는 3개의 비트를 나타내기 때문에 16진수를 사용하는 경우가 더 많다. 한 바이트는 8비트고 이는 2자리 16진수로 완벽히 표현가능한데, 이 역시 8진수보다 16진수를 선호하는 이유 중 하나이다.

2.2 정수

2.2.1 정수의 표현법

정수는 크게 음의 정수, 양의 정수, 0으로 나눌 수 있다. 음의 정수와 양의 정수는 각각 일대일 대응이 가능하고 순서가 있으므로 어떤 정수를 표현할 때에 몇 번째 정수인지를 표현하면 쉽게 이진수로 정수를 표현할 수 있다. 또한 음의 정수인지를 양의 정수인지를 첫 번째 비트로 표현하면 모든 정수를 표현할 수 있게 된다. 이렇게 된다면 n 비트 정수가 -2^{n-1} 부터 2^{n-1} 까지를 표현할 수 있게 된다. 이 방식을 **부호화 절댓값 표기법**이라고 한다.

음의 정수의 부호가 1이고 양의 정수의 부호가 0이라고 한다면 다음 등식이 성립한다.

$$10011000_2 = -24$$

하지만 이 방식을 사용하면 0이 2개가 생기는 문제가 발생한다. 이 방식에 따르면, 다음 두 4비트 정수는 같은 값을 가리킨다.

$$1000_2 = -0 = 0$$

$$0000_2 = +0 = 0$$

따라서 0이 겹쳐지는 문제를 해결하기 위해 컴퓨터에서는 의 보수 표기법을 사용한다.

2의 보수 표기법은 사전적으로 2의 보수 관계에 있는 두 2진수로 절댓값이 같고 부호가 다른 두 10진수를 표현하는 것으로 정의된다. 2의 보수란 어떤 수를 커다란 2의 제곱수에서 빼서 얻은 이진수 쌍 또는 그 중 하나를 의미하는데, 이 두 이진수 중 어떤 특징에 따라 하나는 음수, 하나는 양수로 표현하겠다는 의미이다. 이 표기법은 다음과 같은 특징을 가진다.

- 양의 정수를 표현할 때에는 부호 비트(가장 처음 비트)를 제외한 비트를 사용하여 표현한다.
- 음의 정수를 표현할 때에는 양의 정수를 비트 반전한 뒤 반전한 수에 1을 더한다. 이때 올림 비트는 무시한다.
- 음의 정수를 통해 양의 정수를 표현할 때에는 음의 정수를 비트 반전한 뒤 반전한 수에 1을 더한다. 이때 올림 비트는 무시한다.

예를 들어, 24를 8비트 이진수 2의 보수 표기법으로 표현한 것에 대해 다음 등식이 성립한다.

$$00011000_2 = 24$$

이때, -24 를 표현하기 위해서는 양의 정수를 비트 반전한 뒤 반전한 수에 1을 더한다.

00011000_2	24
11100111_2	비트 반전
11101000_2	1을 더한다. -24 완성

이렇게 한다면 0의 표현법이 2개가 되는 문제가 해결된다.

00000000_2	0
11111111_2	비트 반전
00000000_2	1을 더한다.

이때 계산하는 수는 8비트 이진수이기 때문에 9번째 비트를 제외하고 생각한다. 이 방식을 사용하면 정수의 연산을 더욱 쉽게 할 수 있다.

2.2.2 2진수의 덧셈

2진수의 덧셈은 AND 연산과 XOR 연산으로 진행된다. 각각 연산에 대해서는 다음 표로 설명할 수 있다.

A	B	A AND B	A XOR B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

XOR 연산은 연산 후 수의 일의 자리 숫자를, AND 연산은 연산 후 수의 이의 자리 숫자를 나타낼 수 있다. 만약 이의 자리 숫자가 연산하는 숫자의 일의 자리 숫자에 해당한다면 그 덧셈의 이의 자리 숫자를 사의 자리 숫자로 적용할 수 있고, 이를 통해서 두 자리 이상의 연산도 가능하다.

$$\begin{array}{r}
 & \begin{array}{cc} 1 & 0 \\ + & 1 \\ \hline 0 & 1 \end{array} \\
 & \text{XOR} \\
 & \begin{array}{cc} 1 & 0 \\ + & 1 \\ \hline 1 & 0 \end{array} \\
 & \text{AND} \\
 & \begin{array}{cc} 1 & 0 \\ + & 1 \\ \hline \end{array} \\
 & +
 \end{array}$$

2.2.3 정수의 연산

2의 보수 표기법이 효율적인 표기법인 이유는 연산을 직관적으로 수행할 수 있기 때문이다. 2의 보수 표기법을 사용하면 별도의 추가적인 연산 없이 그냥 두 비트에서 더하기 연산을 수행해도 원하는 결과를 얻을 수 있다는 말이다. 예를 들어 94 – 35를 이진수로 수행한다고 해보자. 이때 사용하는 비트수는 8이다.

이때 94는 이진수로 01011110₂이고, 35는 이진수로 00100011₂이다. 하지만 35에는 빼기 연산을 적용하므로 -35를 더하는 연산을 수행해야 하기 때문에 -35를 2의 보수 표기법에 의거하여 이진수로 표현해보면 11011101₂이다. 이제 두 수를 더하기 연산하면 원하는 결과를 얻을 수 있다.

$$\begin{array}{r}
 & \begin{array}{ccccccccc} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ + & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array} \\
 & \text{XOR} \\
 & \begin{array}{ccccccccc} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ + & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array} \\
 & \text{AND} \\
 & \begin{array}{ccccc} 1 & 0 & 0 & 1 & 1 \\ + & & & & \\ \hline \end{array} \\
 & +
 \end{array}$$

하지만 우리는 8비트 연산을 수행하기 때문에 9번째 비트를 무시하여야 한다. 그러면 우리가 얻은 결과는 다음과 같다.

$$00111011_2 = 59$$

컴퓨터는 위와 같이 덧셈과 뺄셈 연산을 수행한다.

A 와 B 의 곱하기 연산은 A 에다가 A 를 더하면서 B 를 1씩 빼는데, 이때 B 가 0이 되면 연산을 종료하는 것으로 구현이 가능하다.⁵

A 와 B 의 나누기 연산은 A 에서 B 를 빼면서 A 가 B 보다 작아지는 지점이 몇 번을 뺀 후인지 알아보는 것으로 구현이 가능하다.

⁵ $B < 0$ 인 경우 수가 0에 도달하지 않는데, 이때는 부호비트가 0으로 바뀔 때에 수 비트에 해당하는 모든 비트가 0으로 초기화되게 된다. 이것을 기준으로 연산을 종료한다. 이때 부호 비트 역시 덧셈의 결과에 적용되어야 한다.

2.3 유리수

실수의 표기법에 대해 설명하기 전에, 컴퓨터와 이론의 차이에 대해서 간단히 짚고 넘어가겠다. 우리가 대표적인 무리수라고 하는 π 의 값을 대부분 알고 있을 것이다.

$$\pi \simeq 3.14159265358979\dots$$

하지만 π 의 값을 숫자로 정확히 표현한 것을 눈으로 확인한 사람은 없을 것이다. 이는 모든 실수를 표현하는 데에 한계가 있는 우리 수 표기법 또는 실수의 특징의 탓이다. 즉, 숫자를 통해서는 모든 실수를 표현할 수 없다. 이는 컴퓨터에서도 마찬 가지이다. 컴퓨터에서는 무리수를 표현할 수 없다. 무리수에 대한 이론적인 성질을 통해 무리수를 활용할 수는 있겠지만, 유리수를 대하는 것처럼 무리수를 대할 수가 없다는 것이다.⁶

컴퓨터에서 실수(유리수)를 표현할 때에는 부동 소수점 방식이나 고정 소수점 방식을 사용한다.

2.3.1 고정 소수점 방식 실수 표현

고정 소수점 방식은 부호화 절댓값 표기법과 비슷하다. 가장 처음 비트를 부호 비트로 사용하고 그 후 일정한 부분을 정수부, 그 후 일정한 부분을 소수부로 활용한다. 예를 들어, 고정 소수점 방식의 실수를 16비트 2진수로 표현한다고 하자. 이때 처음 비트는 부호 비트로 작용하고 그 후 7개의 비트는 정수부를 나타내는 비트, 그 후 8개는 소수부를 나타내는 비트가 된다. 예의 예를 들어, 다음 2진수가 나타내는 유리수는 다음 방정식을 만족한다.

$$\begin{array}{r} 1000011110100000_2 \\ 1 | 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 | 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \end{array}$$

$$\begin{aligned} & 1000011110100000_2 \\ &= - (1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3}) \\ &= - \left(4 + 2 + 1 + \frac{1}{2} + \frac{1}{8} \right) \\ &= - \frac{61}{8} \\ &= - 7.625 \end{aligned}$$

이러한 고정소수점 방식은 구현하기 편리하지만 표현 가능한 수의 범위가 적고 정밀도가 떨어지기 때문에 정확도보다 구현의 용이성이 필요한 시스템에서 자주 사용한다. 이러한 문제점을 해결한 것이 부동 소수점 방식이다.

⁶무리수 처리를 위해서는 추가적인 연산이 필요하기 때문에 자료 저장 및 처리 방식에 차이가 생긴다.

2.3.2 부동 소수점 방식 실수 표현

부동 소수점 방식은 더 많은 범위를 실수로 표현하기 위해서 고안된 실수 표기법이다. 가장 처음 비트를 부호 비트로 사용하고 그 후 일정한 부분을 지수 부분 (*exponential bias*), 그 후 일정한 부분을 기수(*fraction* 또는 *mantissa*)로 사용한다. 예를 들어, 부동 소수점 방식의 실수를 16비트 2진수로 표현한다고 하자. 이때 처음 비트는 부호 비트로 작용할 것이고 그 후 7개의 비트는 지수부를 나타내는 비트, 그 후 8개는 가수부를 나타내는 비트가 될 것이다. 이 표기법으로 실수 -7.625 를 표현하면 다음과 같다.

$$100000101110100_2$$

이 표기법은 직관적이지 않은데, 이 수를 해석하는 방식은 다음과 같다.

$$\begin{array}{r} 1 \mid 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \mid 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \end{array}$$

1. 가수부의 숫자를 이진 소수점으로 두는 이진수 유리수를 하나 만든다. 이때 정수부는 언제나 1이다.
 1.11101_2
2. 지수부를 2의 보수 표기법으로 해석한 수를 e 라고 할 때, 1.의 숫자에 10_2^e 를 곱한다.
 $1.11101_2 \times 10_2^{10_2}$
3. 이 수에 부호를 붙여 부호화 절댓값 표기법으로 해석한다.
 $-1 \times 1.11101_2 \times 10_2^{10_2}$

이 수를 해석하면 다음과 같다.

$$\begin{aligned} & -1 \times 1.11101_2 \times 10_2^{10_2} \\ &= -1 \times 1.90625 \times 2^2 \\ &= -1.90625 \times 4 \\ &= -7.625 \end{aligned}$$

부동 소수점 표기법은 추가적인 연산이 많이 필요하고 고정 소수점 표기법에 비해 구현도 어렵지만 정확도가 뛰어나고 표현할 수 있는 수의 범위가 크기 때문에 대부분의 시스템에서 사용하는 방식이다.

2.4 문자

2진수로 표현하는 방식을 알아보기 전에 어떻게 표현하는지 가장 궁금했던 것이 바로 이 문자의 표기이다. 우리가 수학 시간에 배우는 숫자들로는 절대 메시지를 표현할 수 없을 것이라고 생각했기 때문이다. 하지만 수학자들은 2진수만을 이용하여 문자, 심지어는 문자열까지 표현하는 방법을 고안해냈다.

위에서 비트(bit)와 바이트(Byte)의 차이점에 대해서 말했다. 바이트는 비트보다 2^8 배 더 많은 경우의 정보를 저장할 수 있다. 여기서 2^8 이라는 숫자는 문자의 표기를 설명하면서 빼놓을 수 없는 요소이다. 컴퓨터에서는 어떻게 문자를 표현하는 것일까?

살면서 아스키(ASCII) 코드라는 말은 들어본 적이 많을 것이다. ASCII는 American Standard Code for Information Interchange의 약자로, 정보 교환을 위한 미국 표준 부호라는 뜻이다. 소프트웨어는 이 아스키라는 표준에 맞춰 글자를 표현한다.

아스키는 7개의 비트를 이용하여 컴퓨터에 명령을 내리거나 문자를 쓰기 위해 고안된 표준이다. 총 33개의 제어 문자와 95개의 출력 가능한 문자가 있는데, 이중 33개의 제어 문자는 대부분 현재 사용되지 않는다. 이 아스키 표의 일부를 보면 다음과 같다.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\	^	-	-
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:

이 표는 영어 문자와 숫자를 대응시킨 것이다. 즉, 40₁₆ 행의 8이 H라는 말은 40₁₆ + 8₁₆ = 48₁₆ = 72라는 숫자가 H에 대응된다는 뜻이다. 아스키 코드의 모든 문자는 7비트로 표현이 가능하므로 이는 다음과 같이 나타낼 수 있다.

1001000₂

만약 HELLO라는 문자열이 나타내고 싶다면 35개의 비트가 필요하다.

10010001000101100110010011001001111₂

그런데 8비트가 문자와 관련이 있다는 이유는 무엇일까? 아스키 코드에서는 7비트로 문자를 표현하지만 Latin-1이라는 인코딩 방식에서는 8비트로 문자를 표현하기 때문이다. 이 인코딩 방식에는 영어, 스페인어, 독일어 등까지 표현이 가능한 문자 목록이 포함되어 있다.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
B0	°	±	²	³	'	¶	¶	.	¹	º	»	¼	½	¾	¿	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:

이 방식으로 Schrödinger를 표현하면 다음과 같다.

53636872F6C4696E676572₁₆

하지만 시간이 지남에 따라 유럽 이외의 지역으로도 컴퓨터가 보급되면서 각국의 언어에 맞춰 인코딩 방식을 개발하는 시도가 늘어났고, 이때 한글 조합형, 완성형 토론 등이 이뤄지기도 했다. 하지만 이 역시도 그 나라에서 사용하지 않는 소프트웨어를 설치하면 글씨가 깨져보이는 등의 문제가 발생해 사람들을 불편하게 했고, 결국 전세계적인 문자 표를 만들기 위한 유니코드가 출범한다. 이 유니코드 표에

따르면 완성형 한글은 AC00번부터 D7A3번까지 총 11172글자가 배당되어있다. 전 세계의 모든 글자를 담기에 256개는 너무 적은 수였기 때문에 그것을 넘어가는 글자들은 대부분 한 글자에 2바이트를 차지한다. 따라서 16진수로 나타낼 때에는 4글자로 나타내야 한다.⁷ 한글을 16진수로 나타내면 다음과 같다.

D55CAE00₁₆

이렇게 한글은 4글자, 유럽의 글자는 2글자의 16진수를 한 글자로 계산하는데, 이렇게 다양한 해석 방식에 표준을 정하고자 했으니, 그것이 바로 인코딩 방식이다. 이 인코딩 방식이 달라짐에 따라서 멀쩡한 글씨가 보이지 않거나 하는 일이 일어나기도 한다.

2.5 이미지

우리가 컴퓨터를 사용하면서 대부분의 메모리 공간을 차지하는 것이 바로 이미지 정보이다. 우리가 자주 사용하는 홈페이지의 로고, 검색 버튼, 윈도우의 닫기 버튼과 구성 그 자체도 사실 이미지가 없으면 멋밋하기 그지없는 디자인일 것이다.

컴퓨터에서 이미지를 저장하는 방법에 따라서 이미지 파일의 확장자가 달라지게 된다. 우리가 잘 알고 있는 jpg나 jpeg, 무손실 압축 방식이라는 png 등이 이미지 확장자에 속한다.

컴퓨터 정보로서의 사진을 구현하기 위해서는 여러가지 요소가 필요하다. 이미지의 크기(해상도), 구성하는 색상의 배치, 색상의 정확도 등이 그에 속한다. 이 정보들 중에서 가장 중요한 것은 구성하는 색상의 배치로, 대부분의 이미지 파일 중 가장 많은 영역을 차지하는 정보일 것이다.

2.5.1 이미지의 물리적 특성과 전산화 방법

빛의 3원색은 빨간색, 초록색, 파란색을 말한다. 이 세가지 색을 조절함으로 우리가 볼 수 있는 가시광선의 모든 색들을 구현할 수 있다는 것이다. 흰색은 빨간색 초록색 파란색 빛이 모두 함께 있을 때에 보이는 색이며, 검은색은 빛이 하나도 없는 색이다.

컴퓨터에서 사진을 아주 많이 확대해 보면 한 부분을 이루는 단색 정사각형을 볼 수 있을 것이다.⁸ 이것을 통해 우리는 사진이 작은 정사각형으로 이루어져 있음을 알 수 있을 것이다. 실제로 컴퓨터는 이 원리를 이용하여 사진 정보를 저장한다.

가장 왼쪽 위에 있는 픽셀의 색상 정보부터 오른쪽으로, 제일 위 한 줄이 모두 기록되면 다음 줄에 있는 정보를 위와 같은 방법으로, 모든 직사각형을 채울 때까지 색에 대한 정보를 반복하고 나면 사진이 완성된다.

최근에는 한 가지 색을 표현할 때에 빨간 정도와 초록색인 정도, 파란 정도, 색의 투명한 정도를 256($= 2^8$)등분하여 한 색당 4바이트⁹의 용량을 차지한다.

예를 들어, 다음 16진수는 빨간 정도, 초록색인 정도, 파란 정도, 투명도를 각각 8비트씩 할당하여 한 가지 색상을 표현하고 있다.

FDDE59FF₁₆

⁷ 2바이트가 표현할 수 있는 경우의 수는 $(2^8)^2 = 65,536$ 개이다.

⁸ 이 작은 정사각형을 화소(素) 또는 픽셀(pixel)이라고 한다.

⁹ 색을 256등분하여 나타내면 그 색의 정도는 한 바이트로 완벽히 나타낼 수 있다.

2.5.2 이미지 압축

한 색상이 4비트로 나타내진다면, FHD 화질의 바탕화면을 표현하는 데에 이론적으로 8.1MB의 공간이 사용된다. 만약 4K 화질의 바탕화면을 표현하는 데에는 32.4MB의 공간이 사용된다. 이는 실로 엄청난 수치이다. 만약 하루 종일 활동한 내용을 사진으로 찍을 때¹⁰ 100장의 사진을 찍는다면 하루에 3.24GB씩의 용량을 사용하게 될 것이고, 용량이 32GB인 휴대전화로는 열흘을 채 넘기지 못하고 더 이상의 사진을 찍을 수 없게 될 것이다. 즉, 아무것도 없이 사진만 저장한다고 해도 1000장을 넘기지 못하게 될 것이라는 말이다.

수학자들은 이 문제를 인식하고 있었고, 더 효율적으로 사진을 저장하기 위해 여러 방법을 연구했다. 그 중에는 사진을 손실시키면서도 획기적으로 파일의 용량을 줄이는 .jpg와 용량은 크지만 사진 원본 정보가 손실되지 않는 .png를 연구해내게 되었다.

2.6 오디오

우리는 시각적인 정보를 넘어 청각적인 정보까지 자료화하여 사용하기에 이르렀다. 소리 역시 시스템을 풍성하게 만드는 자료이며, 보다 효과적으로 상황의 분위기를 전달하기 위해 많은 소프트웨어에서 오디오를 활용한다.

2.6.1 소리의 물리적 특성과 전산화

소리는 공기로 전달되는 파동이다. 따라서 디지털 신호로 그 파동을 표현할 수만 있다면 소리 역시 디지털로 표현해낼 수 있다. 연속적인 색상을 무시 가능할 정도로 작은 단위로 구분함으로서 연속성을 구현해냈듯이, 소리도 시간상 거리가 가까운 연속적인 값들로 나타낼 수 있다.

예를 들어, 파동은 진폭과 진동수, 파형으로 특정할 수 있다. 따라서 파동을 전달하는 매질의 움직임을 수치로 나타내면 소리를 나타낼 수 있다. 1초에 몇 개의 위상을 전송하는지를 나타내는 프레임레이트, 한 위상을 몇 개의 위치로 나누어 판단할지를 나타내는 비트레이트, 실제로 전송되는 위상의 정보를 모두 표현한다면 일정한 길이의 소리를 전산화할 수 있는 것이다.

우리가 자주 사용하는 파일 형식 중 이 세가지를 가장 근본적인 방법으로 나타내는 것이 *Windows Audio Wave Format(.wav)*이다. .wav 파일을 유심히 관찰할 일은 많지 않았겠지만, .wav 파일은 같은 프레임레이트, 같은 비트레이트, 같은 길이를 가지고 있는 음악이 어떤 내용을 담고 있는지에 상관 없이 같은 용량을 차지하게 된다. 가령 아무런 소리도 담겨있지 않아도, 엄청나게 복잡한 수열로 표현되어 있어도 같은 용량을 차지한다는 것이다.¹¹

2.6.2 오디오 압축

하지만 오디오 역시 적은 용량으로 많은 파일을 표현하기 위한 노력이 가해졌다. 대표적인 오디오 압축 포맷으로는 .mp3가 있다.

.mp3는 이산 코사인 변환(*Discrete Cosine Transform*)을 통해 압축된 데이터 형태로, 오디오 프레임을 하나씩 저장하기보다는 일정 부분을 일정한 주파수의 합으로 표현하여 주변의 비슷한 데이터들까지도 쉽게 표현할 수 있도록 만든 것이다.

¹⁰최근의 휴대전화는 기본적으로 4K화질의 사진을 찍도록 설정되어있다.

¹¹이는 .bmp 형식의 사진과 그 상황이 비슷하다.

즉, 이전의 데이터 기록 방식이 점을 찍어 하나하나의 점을 잇는 방식이었다면 mp3는 그 점들의 규칙을 수식으로 나타내 수식을 통해 점의 자취를 구현하는 방식이라고 할 수 있다. 이 과정에서 데이터의 손실이 일어나지만, 비트레이트와 진동수를 높게 구현하여 원본과 비슷한 오디오를 효과적으로 구현하는 것이 가능하다.

원래 이러한 방식은 손실 이미지 압축 방식인 .jpg에 적용되는 방식인데, 이것을 오디오로 확장시킨 것이다.

2.7 영상

사실 영상은 우리가 접하는 컴퓨터 자료의 끝이라고 해도 과언이 아니다. 영상은 우리가 현실을 인식하는 방법을 컴퓨터로 구현한 것으로, 우리가 가장 많이 소비하는 전산화된 자료이기도 하다.

이미지와 오디오를 표현하는 방법을 알고 있는 한 영상을 표현하는 것은 그렇게 어렵지 않을 것이다. 1초에 몇 개의 화면을 보일지를 나타내는 프레임레이트, 각각의 프레임이 어떻게 구성되어 있는지를 나타내는 사진 정보, 그 프레임에서 재생될 오디오를 나타낸 음성 정보를 조합하면 영상 자료를 구성할 수 있다.

하지만 사진과 오디오만으로도 엄청난 양의 용량을 차지하는 만큼, 영상 정보 역시 압축이 필요하다. 수학자들은 이미지 포맷인 .jpg에서 사용되는 압축의 방식을 오디오인 .mp3를 넘어 영상인 .mp4까지로 적용시켰다.

3 논리 연산

2.2.3 장에서 정수의 덧셈에 대해서 알아보면서 AND연산과 XOR연산에 대해 보았다. 이러한 연산을 논리 연산이라고 한다. 하지만 논리 연산에 AND와 XOR만 있는 것은 아니다. 논리 연산은 NOT과 OR에서 시작한다.

3.1 논리합과 논리 부정

OR 연산은 입력 값 2개 중 하나라도 1이 있다면 1을 연산의 결과로 가지는 연산이다. OR의 논리표는 다음과 같다.

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

OR은 기호로 $A|B$, $A + B$ 또는 $A \vee B$ 로 표현한다. 다른 말로는 논리합이라고 한다.

NOT 연산은 입력 값이 1개인 연산이다. 말 그대로 신호의 부정을 연산의 결과로 가진다. NOT의 논리표는 다음과 같다.

A	not A
0	1
1	0

NOT은 기호로 $\neg A$, \overline{A} 또는 $\neg A$ 로 표현한다. 다른 말로는 논리 부정이라고 한다.

3.2 논리합과 논리 부정을 통한 기타 논리 연산의 구현

OR과 NOT, 이 두 연산이 구현되어있는 기계에서는 어떤 연산도 구현할 수 있다. 가령 AND의 경우,

$$A \times B = A \cdot B = \overline{\overline{A} + \overline{B}}$$

로 구현이 가능하다.

덧셈 연산에서 꼭 필요한 XOR은 다음과 같이 구현할 수 있다.

$$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

이외에도 AND를 뒤집은 NAND, OR를 뒤집은 NOR, XOR을 뒤집은 NXOR 등도 있다.

$$A \text{ nand } B = \overline{A \times B}$$

$$A \text{ nor } B = \overline{A + B}$$

$$A \text{ nxor } B = \overline{A \oplus B}$$

A	B	A nand B	A nor B	A nxor B
0	0	1	1	1
0	1	1	0	0
1	0	1	0	0
1	1	0	0	1

결국 둘 중 하나가 켜져있는지와 거꾸로 하는 연산만으로 모든 기계와 알고리즘의 구현이 가능하다는 것이다.

4 나의 생각

4.1 세상을 인식하는 방법

내가 세상을 보는 방법은 컴퓨터를 알기 전과 컴퓨터를 알게 된 후로 나뉜다고 해도 과언이 아닐 것이다.

사실 2진수를 통해 정보를 표현하는 방법은 공통적인 특징을 가지고 있다. 바로 연속적인 자료를 이산적으로 표현했다는 것이다. 컴퓨터를 설계한 수학자들은 디지털로 연속적인 자료를 표현하는 것이 불가능하다는 것을 이르게 알아챘고, 이산적인 자료로 연속적인 것들을 최대한 표현해내기 위해서 많은 노력을 기울였다. 사실 생각해보면 우리가 컴퓨터 밖에서 보는 것들은 모두 연속적인 자료로 이루어져 있다. 이 세상에 존재하는 어떤 직선도 완벽하게 곧지 않고, 이 세상에 존재하는 어떤 구도 완벽한 구형이 아니라는 것이다. 이러한 상황을 기반으로, 컴퓨터가 인간을 이해지 못하는 분야는 사고의 연속성이라고 생각한다.

인간이 사용하는 언어와 컴퓨터가 사용하는 언어는 차이가 많다. 컴퓨터에게 사용하는 언어는 단어에 깊은 뜻을 담을 수 없으며, 깊은 뜻이라고 해도 결국 추상화¹²에 지나지 않는다. 인식 주체에 따라서 다르게 해석될 여지가 있는 자연어와는

¹²복잡한 것을 간단하게 나타내는 것. 예를 들어, $A \text{ xor } B = A \cdot \overline{B} + \overline{A} \cdot B$ 이지만 간단히 $A \oplus B$ 라고 나타낸다.

두드러지는 차이점이다. 우리가 자연어로 어떤 단어를 언급할 때에, 우리는 우리가 생각하는 그 단어의 중심 의미 주변의 일정한 범위를 그 단어와 같은 의미로 생각한다. 예를 들어 가진다라는 의미에는 실제로 물건을 손에 쥐는 것 뿐만이 아니라 어떤 생각을 하게 되는 것(나쁜 생각을 가지게 되다.), 어떤 시간을 보내는 것(끔찍한 시간을 가지다.)이 포함된다. 하지만 대부분의 프로그래밍 언어에서 가진다는 것은 어떤 자료가 여러 자료를 포함할 수 있는 자료에 포함되는 단 한 상황만을 의미한다. 컴퓨터적인 관점에서 보면 중의적인 언어 사용은 컴퓨터와의 의사소통에서 배제되어야하는 것이겠지만, 이 역시 인간 간의 의사소통과 컴퓨터에게 하는 작업 수행의 차이점이라고 할 수 있다.

컴퓨터를 분석하게 되면서 컴퓨터에 대해 알게 되었고, 컴퓨터 속의 것들과 컴퓨터 밖의 것들을 비교하며 우리가 생활 속에서 인식하지 않고 지나가는 당연한 것들을 다시 한 번 생각해보는 기회를 가질 수 있었다. 컴퓨터가 정보를 처리하는 방법에 대해서 생각해보다 보니 사람이 정보를 처리하는 방법과 비교하며 사람에 대해서 생각해보게 되었다. 우리가 자주 마주하는 사회문제는 대부분 사람들이 가치를 두는 것의 순서가 다르기 때문에 다른 사람과 자신의 차이를 알지 못하는 것에서 발생한다. 우리는 기계적인 사고에 익숙하나, 아무래도 인간이듯이 모든 사람이 서로 다른 생각을 하고 있음을 알아야 할 것이다. 나는 컴퓨터를 통해 사회문제를 분석하는 방법을 배웠고, 앞으로 분석한 사회문제를 어떻게 해결해나갈 수 있을 것인지를 생각해보고자 한다.

4.2 기술 발전에 따른 기술 갈등 발생

나는 최근에 뇌파 분석 기술에 대해서도 관심을 가지기 시작했다. 뇌파를 분석하여 어떤 생각을 하고 있는지 컴퓨터를 통해 처리할 수 있게 되면 의사소통에 어려움을 겪고 있는 장애인에게, 가상현실에서 하는 공동작업을 필요로 하는 사람들에게 많은 도움을 줄 수 있을 것이다.

하지만 뇌파 분석 기술이 생각의 주체를 기계로 바꿔버린다는 윤리적 문제도 제기되고 있다. 이러한 기술을 독재의 수단으로 사용하여 사람들이 하는 생각을 통제하는 용례도 발생할 수 있다는 것이다. 뇌파 분석 기술 뿐만이 아니더라도, 컴퓨터와 정보 기술이 발전됨에 따라서 컴퓨터는 점점 더 많은 문제를 발생시킬 것이다. 요즘 사회적으로 뜨거운 화제가 되고 있는 자율주행자동차의 윤리적 딜레마, 산업의 기계화에 따른 실업률 증가 문제 등이 현대 사회의 대표적인 기술 갈등이다.

물론 너무 이상적으로 생각하는 것일 수 있겠지만, 내가 생각하기에, 대부분의 윤리적 문제는 발생하지 않을 수 있으나, 그 기술을 사용하는 행동 주체의 철학의 부재에 의해 발생한다고 본다. 어쩔 수 없는 피해 중 어느것에 더 가치를 둘 것이나 하는 문제는 어쩔 수 없는 윤리적 갈등이 발생하기 마련이지만, 정보 오용에 의한 피해는 인간의 인식 변화를 통해 충분히 막을 수 있는 일이라는 말이다. 이런 이야기에 대해서 하자면 노벨의 다이너마이트의 이야기를 빼놓을 수 없다. 다이너마이트는 만들어진 목적이 무엇이든 전쟁에서 효과적인 승리를 위해 사용되었다. 이는 사상(死傷)적 피해를 발생시킨 기술의 오용으로 보인다. 기술이 발전함에 따라서 특별한 전문가가 아니더라도 기술을 쉽게 이용할 수 있게 되면서, 발전된 기술을 통한 고도의 범죄 역시 상당수 발생하고 있다. 국제 사회는 이러한 기술의 오용을 방지하기 위해서 전세계적인 기술 사용의 표준을 정하고, 그것을 관리할 필요가 있다.

하지만 기술이 결국 사람을 위한 것인 만큼, 사람들은 기술을 사용하기 위해 빠르게 준비되어야 할 것이다. 따라서 기술의 발전 속도에 따라 인간들의 인생관도

진보적인 변화를 보여야 한다는 것이 내 의견이다.

5 참고문헌

- 1.2. Logic Levels - learn.sparklearn.com
- 2.2.1. 2의 보수 (2021. 4. 16 17:28판) - ko.wikipedia.org
- 2.3. 컴퓨터에서의 실수 표현: 고정소수점 vs 부동소수점
 - gsmesie692.tistory.com
- 2.4. ASCII/ISO 8859-1 (Latin-1) Table with HTML Entity Names
 - cs.stanford.edu
- 2.6.2. [H.264] Discrete Cosine Transform - crynut84.tistory.com
- 2.6.2. MP3 - ko.wikipedia.org