2진법/8진법/16진법

Computer vs Human

사람이 일상생활에 사용하여 편리한 10진법은 컴퓨터 내부적으로 직접 처리하지 못합니다. 2진법으로 관리 및 처리되지요. 이는 애초부터 0(off)과 1(on)의 스위칭 체계로 전자회로가 설계되어 왔기 때문입니다.

BIT와 BYTE

전산용어에서 이 On/Off를 관리할 수 있는 하나의 단위를 bit라고 하여 전산처리의 최소 단위가 됩니다.
담배를 한 가치씩 팔지 않듯이 BIT도 하나씩 관리하지 않습니다. 8개의 BIT를 묶어 Byte라는 데이터 단위로 관리하는데요, 이렇게 되면 8개의 on/off 스위치를 가지게 됩니다.
8개의 On/off 스위치를 가지고 생성해 낼 수 있는 조합의 개수는 2^8개 즉, 256가지의 조합을 만들 수 있고, 이런 이유로 Byte 형 데이터는 0~255까지의 정수형(256가지) 데이터가 됩니다.
16개(2Byte)의 스위치가 있다면 2^{16}, 즉 65536가지의 정수형을 관리할 수 있게 됩니다. 이런 이유로 IDL의 Integer형(2Byte : C의 short integer 형)은 0~65535 또는 부호 있는 정수형(디폴트)은 -32768~32767의 정수를 관리할 수 있게 됩니다.
255, 32767, 65535 와 같은, 10진수 체계에서는 별로 중요해 보이지 않는 숫자가 전산에서 계속 등장하는 이유는 바로 컴퓨터가 2진수 체계에 기반하기 때문입니다. 2진수 체계에서 이 숫자들은 꽤 의미있는 모양으로 나타납니다.

10진	2진
255	1111 1111
32767	0111 1111 1111 1111
65535	1111 1111 1111 1111

2진, 8진, 16진법

10진수에서 크지 않은 숫자도 2진수로 풀어 놓으면 어쩔 수 없이 엄청난 자리수가 됩니다. 몇 자리 수인지 세는 게 너무 힘들지요(사람에게 말입니다. 그래서 네 자리씩 띄어쓰기도 합니다). 그래서 2진수를 쉽게 끊어 읽을 수 있는 진법이 전산계에서 함께 발달했습니다. 대표적인 것이 8진수 체계와 16진수 체계입니다. 8진수(8은 2^3)는 BIT를 3자리씩 끊어서 관리할 수 있게 되고, 16진수(16은 2^4)는 4자리씩 끊어 관리할 수 있게 됩니다.
8진수는 0~7의 숫자를 이용하여 표현하며 0~7은 3개의 BIT로 관리할 수 있습니다. 16진수는 0~F의 16가지 숫자(와 일부 알파벳)을 이용하여 표현하며 4개의 BIT를 관리합니다. 특히 16진수는 1 Byte(8 Bit)를 2자리의 문자로 표현하게 되어 전산분야에서 2진수를 대신하여 많이 사용됩니다. 예를 들어 2진수 1111 1111은 16진수 FF로 표현되지요.

10진수를 2진, 8진, 16진으로 변환 표출

FORMAT Code 'B', 'O', "Z'는 각각 2진(Binary), 8진(Octal), 16진(Hex)를 의미합니다.

```
IDL> print, 10, format='(B)'
        1010
IDL> print, 10, format='(O)'
      12
IDL> print, 10, format='(Z)'
       A
```

다른 Format Code와 마찬가지로 자리수를 지정할 수 있으며 Format Code와 자리수 사이에 0(zero)을 쓰면 남은 자리수를 0으로 채워 넣을 수 있습니다(Zero Padding).

```
IDL> print, 10, format='(B08)'
00001010
```

10진수를 2진, 8진, 16진 문자열로 변환

STRING() 함수와 Format Code를 이용하면 10진수를 다른 진법체계로 변환하여 "문자열"로 받아낼 수 있습니다.

```
IDL> Bin=string(10, format='(B08)')
IDL> print, bin
00001010
IDL> Hex=string(255, format='(Z02)')
IDL> print, Hex
FF
```

2진, 8진, 16진 체계의 숫자

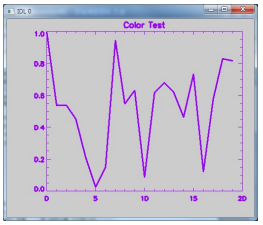
2진수 1111을 바로 변수에 입력하고자 할 때 어떻게 할까요? 2진수 1010과 16진수 FF를 더하고자 할 때 어떻게 해야 할까요? 다음 처럼 2진수, 8진수, 16진수를 직접 쓸 수 있습니다.

```
IDL> a='1111'b
IDL> print, a
       15
IDL> print, '1010'b+'FF'x
      265
```

작은 따옴표 안에 원하는 수를 넣고 바로 붙여서 B(또는 b)를 붙여주면 2진수, O(또는 o)를 붙이면 8진수, X(또는 x)를 붙이면 16진수를 의미합니다.
8진수 755와 10진수 1023을 더하여, 16진수로 표출하면 다음과 같습니다.

```
IDL> print, '755'O+1023, format='(Z)'
     5EC
```

16진 체계를 이용한 색 지정



Direct Graphics에서 RGB 합성 색상체계(Decomposed=1)는 다음과 같이 색을 지정하게 되어 있습니다.

$$R+G*256+B*256^2$$

이 때 R, G, B는 각각 0~255로 각 색의 농도가 됩니다.
이를 이용하여 (R=204, G=204, B=204)인 회색(R, G, B 모두 밝기가 같으면 흰색, 검은색을 포함하는 회색조가 됩니다) 배경에 (R=153, G=0, B=255) 인 보라색 계열의 플롯을 그려봅시다.

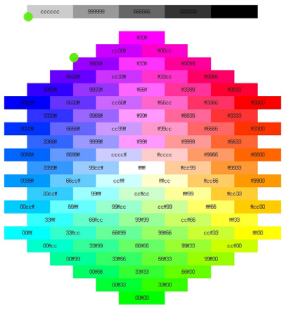
```
IDL> device, decomposed=1
IDL> plot, randomu(seed, 20), $
          Color=153L+0*256L+255*256L^2, $
          background=204L+204*256L+204*256L^2
```

같은 내용을 16진 표기법으로 사용하면 표현이 훨씬 간결해집니다. 204는 16진수로 'CC'x가 되고요, 153은 16진수로 '99'x,

255는 16진수로 'FF'x 가 됩니다. 16진수 표기에 의하면 앞의 귀찮은 R, G, B 공식이 다음과 같이 간결해집니다.

'청색녹색적색'x

```
IDL> plot, randomu(seed, 20), color='FF0099'x, $
        background='CCCCCC'x
```



웹 컬러차트. 초록색점이 예제에 사용한 회색조와 보라색

이러한 색상표는 인터넷에서 쉽게 검색되는, 웹 컬러 차트에도 나와 있습니다. 통상의 웹 컬러 차트는, '적색녹색청색'의 순서로 되어 있지만, IDL에서는 '청색녹색적색'으로 순서가 다르다는 점만 기억해 두십시오.

문자열 상태의 2진, 8진, 16진수를 10진수로

만일, 텍스트 파일 등에서 2진수, 8진수, 16진수로 기록되어 있는 값을 IDL에서 사용하기 위해 10진수로 변환해야 한다면 ReadS 프로시저를 이용해 볼 수 있습니다. reads 는 문자열을 읽는 프로시저인데요, 값을 읽을 때, "이 값은 2진수입니다 Format='(B)'" 또는 "이 값은 16진수입니다 Format='(Z)'" 와 같이 지정해 주는 것입니다.

```
IDL> d_bin='1010'
IDL> reads, d_bin, a, format='(B)'
IDL> print, a
      10
IDL> d_hex='CCFE'
IDL> reads, d_hex, b, format='(Z)'
IDL> print, b
      52478.0
```

비트 연산

비트연산자는 이진수 체계에서 각 자리수에 대해 적용되는 연산자입니다.

NOT

각 자리수의 값을 바꿉니다. 0은 1로, 1은 0으로. 어떤 수의 보수를 구할 때 사용합니다.

예) NOT 0010 = 1101

주의할 점은, 보수의 총 자리수가 데이터 타입에 따라 결정된다는 것입니다.

```
IDL> print, NOT '00001111'B, format='(B)'   ;Integer형(16bit)
1111 1111 1111 0000 ;실제로는 네 자리씩 끊어 보이지는 않아요
IDL> print, NOT Byte('00001111'B), format='(B)'  ;Byte형(8bit)
1111 0000
IDL> print, NOT Long('00001111'B), format='(B)' ;Long(32bit)
1111 1111 1111 1111 1111 1111 1111 0000
```

AND

두 2진수의 해당 자리수가 모두 1일 때만 1이 됩니다.

예) 0110 AND 1111 => 0110

```
IDL> print, 14 and 7
       6
```

14 AND 7 이 6이라니, 이런 뜬금없는 수는 어디서 나오는 것일까요?

```
IDL> print, 14, 7, 14 and 7, format='(B)'
          1110      ;14
           111      ;7
           110      ;6
```

OR

두 2진수의 해당 자리 중 하나라도 1이면 1이 됩니다.

예) 0110 OR 1111 => 1111

```
IDL> print, 14, 7, 14 or 7, format='(B)'
          1110
           111
          1111
```

XOR

두 2진수의 해당 자리 중 한자리만 1이어야 1이 됩니다.

예) 0110 XOR 1111 => 1001

```
IDL> print, 14, 7, 14 xor 7, format='(B)'
          1110
           111
          1001
```

비트 이동(Bit Shift)

IDL의 ISHFT 함수는 2진수 체계에서 지정한 자리수 만큼 비트를 왼쪽/오른쪽으로 이동합니다. 이 때 이동으로 인해 비워지는 자리는 0으로 채워지게 됩니다.

```
IDL> print, ISHFT('0010'b, 1), format='(B04)'
0100
IDL> print, ISHFT('0010'b, -1), format='(B04)'
0001
```

ISHFT는 배열을 받아 처리할 수 있고, 대용량 배열을 받을 경우, 멀티 CPU에 작업을 분배합니다. IDL의 모든 연산자도 이렇게 작동합니다. 그러므로 대규모 데이터에 대한 비트 연산 및 비트 이동을 원할 경우, 꼭 배열 기반의 처리를 하여 속도를 확보하세요.

```
IDL> print, ISHFT([1, 2, 4, 8], 2)  ;네 수를 왼쪽으로 2bit 이동
       4       8      16      32
```

원하는 비트 뽑아내기

비트 이동과 비트 연산자를 이용하여 원하는 비트를 뽑아낼 수 있습니다. 예를 들어, 10110101에서 1101 부분을 (오른쪽 기준으로 3~6 비트(총 4비트)) 뽑아내고자 할 경우, 다음과 같이 합니다.

```
IDL> print, ISHFT('10110101'B, -2) AND '00001111'B, $
              format='(B)'
          1101
```

2진 Bit 중에 필요한 부분을 맨 오른쪽으로 몰아놓고, 필요한 자리수 만큼만 1로 채운 Mask와 AND 연산을 하면 됩니다. 다음과 같이 그 반대의 순서도 결과는 같습니다.

```
IDL> print, ISHFT('10110101'B AND '00111100'B, -2), $
             format='(B)'
          1101
```