NET'teki Decimal Türünün Bitsel Organizasyonu

Kaan Aslan
27 Aralık 2009

128 bit (16 byte) uzunluğundaki *decimal* türünün en belirgin özelliği yuvarlama hatası (*rounding error*) olmadan 29 basamak mantise sahip noktalı sayıları tam olarak tutabilmesidir. Bu türden bir değişkene yerleştirilen noktalı sayı daha sonra tam olarak geri alınabilir. Aritmetik işlemler sonucunda elde edilen değerlerde de bir bozulma söz konusu olmaz. *Decimal* türü *float* ve *double* türlerinden daha yüksek nokta duyarlılığına (*mantissa*) fakat daha düşük basamak duyarlılığına (*magnitude*) sahiptir.

Decimal türü özellikle yuvarlama hatalarının istenmediği finansal hesaplamalarda tercih edilmektedir. Doğal bir tür olmadığı için *decimal* türüyle işlemler *float* ve *double* türlerine göre oldukça yavaştır.[1] Bu nedenle *decimal* türünün amaç dışı bir biçimde gerçek sayı türlerinin yerine kullanılması kötü bir tekniktir.

C#'ta tüm tamsayı türlerinden *decimal* türüne otomatik dönüştürme (*implicit conversion*) tanımlıdır. Yani biz herhangi bir tamsayı türünü doğrudan *decimal* türüne atayabiliriz. Fakat *float* ve *double* türlerinden *decimal* türüne, *decimal* türünden de *float* ve *double* türlerine otomatik dönüştürme tanımlı değildir. Bu demektir ki, biz *float* ya da *double* türünü *decimal* türüne *decimal* türünü de *float* ve *double* türüne doğrudan atayamayız. Örneğin:

```
decimal dc;

dc = 10;            // geçerli
dc = 10F;           // error!
double db = 10M;    // error!
```

Ayrıca, *decimal* türünün (*System.Decimal* yapısının) *CLS* (*Common Language Specification*) uyumlu bir tür olmadığını da belirtelim. Yani bu tür *.NET* uyumlu her dilde bulunmak zorunda değildir.

Decimal türünü oluşturan 128 bitin 96 biti (yani 12 byte'ı) mantis tutmak için, 5 biti noktanın yerini tutmak için ve 1 biti de işaret tutmak için kullanılmaktadır. Bu türde kullanılmayan 26 bitlik bir alan vardır ve bu alan içerisinde sıfır değerleri bulunur. *Decimal* türünün 96 bitlik mantis kısmı [0, 2^{96} – 1] (yani [0, 79228162514264337593543950335]) arasındaki sayılardan oluşturulabilir. *Decimal* türünde üstel kısım için bir normalizasyon uygulanmaz. Başka bir deyişle üstel kısım her zaman sıfır ya da pozitiftir ve mantisin en sağından soluna doğru kaç basamak gidilerek noktanın yerleştirileceğini belirtir. Tabi bütün bu bölümlerin sayı içerisinde 2'lik sistemde tutulduğunu biliyorsunuz. Aşağıda bazı sayılar için mantis, üstel ve işaret bitlerinin değerleri verilmektedir (parantez içerisindeki değerler mantisin 16'lık sistemdeki karşılıklarını gösteriyor) :

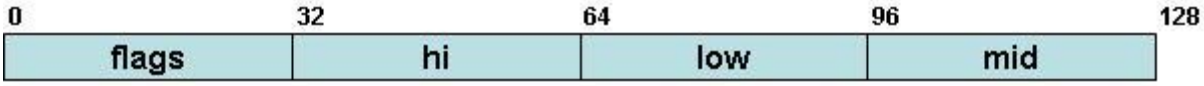
Sayı	Mantis Kısmı (96 bit)	Üstel Kısım	İşaret Biti
12345	12345 (00000000000000000003039)	0	0
12.345	12345 (00000000000000000003039)	3	0
-12.345	12345 (00000000000000000003039)	3	1
123.45	12345 (00000000000000000003039)	2	0
-0.012345	12345 (00000000000000000003039)	6	1
0.0000000012345	12345 (00000000000000000003039)	13	0
7.92281625142643375935439503 35	79228162514264337593543950335 (FFFFFFFFFFFFFFFFFFFFFFFF)	28	0
0	0 (00000000000000000000000)	0	0
0.0000000000000000000000000001	1 (00000000000000000000001)	28	0

Matematiksel olarak *decimal* türünün tuttuğu değerler $(-1)^s \times c \times 10^{-e}$ biçiminde ifade edilebilir. Burada *s* işareti belirtir ve 1 bittir (0 ise pozitif 1 ise negatif). *c* ise 96 bittir ve $0 <= c <= 2^{96}-1$ arasında değer alır; *e* ise $0 <= e <= 28$ arasında değer alabilen 5 bitten oluşur. Ayrıca *decimal* türü için +∞, -∞ ve *NAN* biçiminde özel değerlerin tanımlı olmadığını belirtelim.

Şimdi gelelim *decimal* türünün bitsel organizasyonuna. Mantis kısmı, üstel kısım ve işaret biti bu 128 bitin nerelerinde bulunuyor? Sizi hayal kırıklığına uğratmak istemeyiz fakat bu formatın bitsel düzeyde kesin bir tanımlaması yok. *Decimal* türü *.NET* sistemlerinde *System.Decimal* isimli yapıyla temsil ediliyor ve bu türden bir değişkenin değeri yapının içerisindeki 4 adet *int* türden veri elemanında tutuluyor. Bu 4 *int* değer birbirinden bağımsız olarak ele alındığı için kendi aralarındaki dizilimin nasıl olduğunun da bir önemi yok. Aşağıda *.NET* sınıf kütüphanesinin son sürümündeki *Decimal* yapısının bir kısmını görüyorsunuz:[2]

```
[Serializable, StructLayout(LayoutKind.Sequential),
ComVisible(true)]
public struct Decimal : IFormattable, IComparable,
IConvertible,
                        IComparable<decimal>,
IEquatable<decimal>
{
    private int flags;
    private int hi;
    private int lo;
    private int mid;
    //...
}
```

Gördüğünüz gibi 128 bit 4 ayrı *int* türden veri elemanıyla temsil edilmiş. Yapı içerisindeki *low*, *mid* ve *height* elemanları mantis bilgisinin 4'er byte'lık kısımlarını, *flags* elemanı ise sayının işaretini ve üstel kısmını tutuyor. *flags* içerisinde kullanılmayan bitler de var. Yapının *structLayout(LayoutKind.Sequential)* özniteliğine (*attribute*) sahip olduğuna dikkat ediniz. Bu durumda yapı elemanları ilk bildirilen düşük adreste olacak biçimde ardışıl dizilecektir. O halde biz *decimal* türünden bir değişkenin aşağıdaki gibi bir eleman dizilimine sahip olacağını söyleyebiliriz:



Fakat yukarıda da belirttiğimiz gibi *Decimal* yapısı *flags*, *hi*, *low* ve *mid* elemanlarını bağımsız elemanlar olarak kullanmaktadır. Yani bu elemanların yapı içerisindeki yerleri değişse bile *Decimal* yapısının çalışmasında bir değişiklik oluşmaz.

Decimal yapısının *GetBits* isimli *public static* metodu bize *low*, *mid*, *high* ve *flags* değerlerini belirttiğimiz sırada *int* bir dizi olarak verir:

```
public static int[] GetBits(decimal d)
```

Bu metot aşağıdaki gibi yazılmıştır:

```
public static int[] GetBits(decimal d)
{
    return new int[] { d.lo, d.mid, d.hi, d.flags };
}
```

Aşağıdaki gibi bir programlamla klavyeden girilen decimal değerleri *flags*, *high*, *mid*, *low* sırasıyla ekrana yazdırabilirsiniz:

```
class App
{
    public static void Main(string[] args)
    {
        decimal val;
        int[] bits;
        string format = "{0,20:X8}{1,10:X8}{2,10:X8}{3,10:X8}";

        do
        {
            Console.Write("Sayı giriniz: ");
            val = decimal.Parse(System.Console.ReadLine());
            bits = decimal.GetBits(val);
            Console.WriteLine(format, bits[3], bits[2],
                              bits[1], bits[0]);
        } while (val != 0);
    }
}
```

32 bitlik *flags* elemanın bitsel yerleşimi şöyledir:

Kullanılmayan alanlar içerisinde 0 değerleri vardır. Yapı içerisindeki yerleşimin *GetBits* metodu ile verilenden farklı olduğunu bir kez daha yineleyelim. *Decimal* türü ile toplama, çıkarma, çarpma, bölme gibi işlemler *Decimal* yapısı içerisindeki operatör metotlarla yapılmaktadır. Örneğin:

```
decimal d1, d2, d3;

d1 = 10;
d2 = 20;
d3 = d1 + d2;
```

gibi bir işlem için derleyici aşağıdakine benzer kod üretir:

```
Decimal d1, d2, d3;

d1 = new Decimal(10);
d2 = new Decimal(20);
d3 = d1 + d2;
// d3 = public static Decimal operator +(Decimal a, Decimal b)
```

Decimal türünden bir değer herhangi bir ortama bellekte bulunduğu sıraya göre değil *low*, *mid*, *high* ve *flags* sırasına göre aktarılmaktadır. *Decimal* yapısının *GetBytes* isimli *internal* metodu bu değerleri byte dizisi olarak verir:

```
internal static void GetBytes(decimal d, byte[] buffer)
{
    buffer[0] = (byte) d.lo;
    buffer[1] = (byte) (d.lo >> 8);
    buffer[2] = (byte) (d.lo >> 0x10);
    buffer[3] = (byte) (d.lo >> 0x18);
    buffer[4] = (byte) d.mid;
    buffer[5] = (byte) (d.mid >> 8);
    buffer[6] = (byte) (d.mid >> 0x10);
    buffer[7] = (byte) (d.mid >> 0x18);
    buffer[8] = (byte) d.hi;
    buffer[9] = (byte) (d.hi >> 8);
    buffer[10] = (byte) (d.hi >> 0x10);
    buffer[11] = (byte) (d.hi >> 0x18);
    buffer[12] = (byte) d.flags;
    buffer[13] = (byte) (d.flags >> 8);
    buffer[14] = (byte) (d.flags >> 0x10);
    buffer[15] = (byte) (d.flags >> 0x18);
}
```

Örneğin *BinaryWriter* sınıfının decimal parametreli *Write* metodu şöyle aktarım yapmaktadır:

```
public virtual void Write(decimal value)
{
    decimal.GetBytes(value, this._buffer);
    this.OutStream.Write(this._buffer, 0, 0x10);
}
```

---

[1] *float* ve *double* türleri doğal türlerdir. Bu türler üzerinde işlemler matematik işlemciye sahip pek çok işlemci tarafından tek bir makina komutuyla elektriksel düzeyde yapılabilmektedir. Halbuki *decimal* türü ile işlemler algoritmik kodların çalıştırılmasıyla yapılırlar.

[2] *.NET* kütüphanesine ilişkin *C#* kodları *Reflector* isimli *decompiler* kullanılarak elde edilmiştir. (http://www.red-gate.com/products/reflector/)

Kaynaklar

1. MSDN Library Reference (http://msdn.microsoft.com/en-us/library/default.aspx).
2. Petzold, C. (2004). *Programming In the Key of C#* (sayfa 92-98). Redmond, Washington: Microsoft Press.
3. *Standard ECMA-334 C# Language Specification, 4th Edition.* (2009). http://www.ecma-international.org/ adresinden elde edilmiştir.
4. *Standard ECMA-335 Common Language Infrastructure(CLI), Partions I to IV. 4th Edition.* (2009). http://www.ecma-international.org/ adresinden elde edilmiştir.