```
   mantissa = FetchLong((unsigned long *)(buffer+2));
   exp = 30 - *(buffer+1);
   while (exp--)
   {
     last = mantissa;
     mantissa >>= 1;
   }
   if (last & 0x00000001) mantissa++;
   return(mantissa);
}
```

Of course, you may need a complementary routine to take a sample rate as an unsigned long, and put it into a buffer formatted as that 80-bit floating point value.

```
/* ************************ StoreLong() ******************************
 * Fools the compiler into storing a long into a char array.
 ******************************************************************** */

void StoreLong(unsigned long val, unsigned long * ptr)
{
   *ptr = val;
}

/* ************************* StoreFloat() *****************************
 * Converts an unsigned long to 80 bit IEEE Standard 754 floating point
 * number.
 ********************************************************************** */

void StoreFloat(unsigned char * buffer, unsigned long value)
{
   unsigned long exp;
   unsigned char i;

   memset(buffer, 0, 10);

   exp = value;
   exp >>= 1;
   for (i=0; i<32; i++) { exp>>= 1;
      if (!exp) break;
   }
   *(buffer+1) = i;

   for (i=32; i; i--)
   {
      if (value & 0x80000000) break;
      value <<= 1; } StoreLong(value, buffer+2); #ifdef INTEL_CPU FlipLong((unsigned long *)(buffer+2)); #endif }
```

Multi-sampling

Many MIDI samplers allow splitting up the MIDI note range into smaller ranges (for example by octaves) and assigning a different waveform to play over each range. If you wanted to store all of those waveforms into a single data file, what you would do is create an IFF LIST (or perhaps CAT if you wanted to store FORMs other than AIFF in it) and then include a embedded FORM AIFF for each one of the waveforms. (ie, Each waveform would be in a separate FORM AIFF, and all of these FORM AIFFs would be in a single LIST file). See About Interchange File Format for details about LISTs.

```
Content-Type: text/html; charset=iso-8859-1; name="New WAVE RIFF Chunks.htm"
Content-Disposition: inline; filename="New WAVE RIFF Chunks.htm"
Content-Base: "file:///D|/Source/Sound/Format/New%20W
 AVE%20RIFF%20Chunks.htm"

X-MIME-Autoconverted: from 8bit to quoted-printable by skynet.usb.ve id TAA05017
```

New WAVE RIFF Chunks

Added: 05/01/92
Author: Microsoft, IBM

Most of the information in this section comes directly from the IBM/Microsoft RIFF standard document.

The WAVE form is defined as follows. Programs must expect (and ignore) any unknown chunks encountered, as with all RIFF forms. However, **<'fmt'-ck>** must always occur before , and both of these chunks are mandatory in a WAVE file.

```
→
RIFF( 'WAVE'
<'fmt'-ck> // Format
[] // Fact chunk
[] // Cue points
[] // Playlist
[] // Associated data list
) // Wave data
```

The WAVE chunks are described in the following sections.

Fact Chunk

The stores file dependent information about the contents of the WAVE file. This chunk is defined as follows:

```
→ fact( )
```

<dwSampleLength> represents the length of the data in samples. The <nSamplesPerSec> field from the wave format header is used in conjunction with the <dwSampleLength> field to determine the length of the data in seconds.

The fact chunk is required for all new WAVE formats. The chunk is not required for the standard WAVE_FORMAT_PCM files.

The fact chunk will be expanded to include any other information required by future WAVE formats. Added fields will appear following the field. Applications can use the chunk size field to determine which fields are present.

Cue Points Chunk

The <cue-ck> cue-points chunk identifies a series of positions in the waveform data stream. The is defined as follows:

```
→ cue( // Count of cue points
... ) // Cue-point table

→ struct {
DWORD dwName;
DWORD dwPosition;
FOURCC fccChunk;
DWORD dwChunkStart;
DWORD dwBlockStart;
DWORD dwSampleOffset;
}
```

The <cue-point> fields are as follows:

Field	**Description**
dwName	Specifies the cue point name. Each record must have a unique dwName field.
dwPosition	Specifies the sample position of the cue point. This is the sequential sample number within the play order. See "Playlist Chunk," later in this document, for a discussion of the play order.
fccChunk	Specifies the name or chunk ID of the chunk containing the cue point.
dwChunkStart	Specifies the position of the start of the data chunk containing the cue point. This should be zero if there is only one chunk containing data (as is currently always the case).
dwBlockStart	Specifies the position of the start of the block containing the position. This is the byte offset from the start of the data section of the chunk, not the chunk's FOURCC.
dwSampleOffset	Specifies the sample offset of the cue point relative to the start of the block.

Examples of File Position Values

The following table describes the field values for a WAVE file containing a single 'data' chunk:

Cue Point Location	Field	Value
Within PCM data	**fccChunk**	FOURCC value 'data'.
	dwChunkStart	Zero value.
	dwBlockStart	File position of the sample (nBlockAlign aligned bytes) relative to the start of the data section of the 'data' chunk (not the FOURCC).
	dwSampleOffset	Sample position of the cue point relative to the start of the 'data' chunk.
In all other 'data' chunks	**fccChunk**	FOURCC value 'data'.
	dwChunkStart	Zero value.
	dwBlockStart	File position of the enclosing block relative to the start of the data section of the 'data' chunk (not the FOURCC). The software can begin the decompression at this point.
	dwSampleOffset	Sample position of the cue point relative to the start of the block.

Playlist Chunk

The playlist chunk specifies a play order for a series of cue points. The is defined as follows:

```
® plst(
// Count of play segments
... ) // Play-segment table

® struct {
DWORD dwName;
DWORD dwLength;
DWORD dwLoops;
}
```

The <play-segment> fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the cue-point table.
dwLength	Specifies the length of the section in samples.
dwLoops	Specifies the number of times to play the section.

Associated Data Chunk

The associated data list provides the ability to attach information like labels to sections of the waveform data stream. The is defined as follows:

```
→ LIST( 'adtl'

// Label

// Note

} // Text with data length
```