# Endianness

Allen Baker

August 19, 1996

Computers, like people, differ in the way they write. Some write data "left-to-right" and others "right-to-left". A computer can read its own data just fine - problems happen when one type of computer stores data and a different type tries to read it. With computers, the left-to-right vs right-to-left dichotomy is called “endianness”.

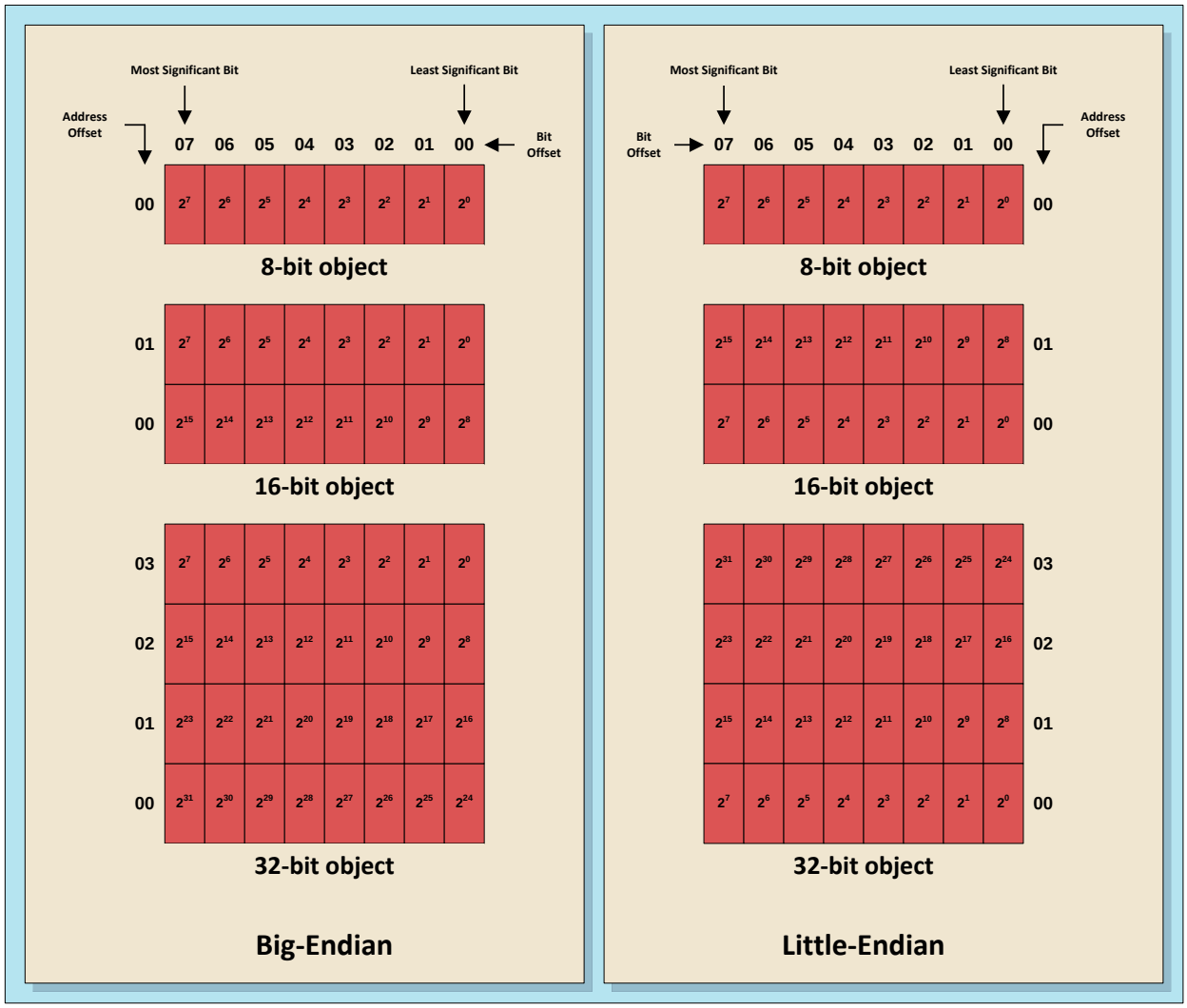
“Big-endian” and “little-endian” are terms that describe the order in which a sequence of bytes are stored in computer memory. Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address). Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first. For example, in a big-endian computer, the two bytes required for the hexadecimal number 4F52 would be stored as 4F52 in storage (if 4F is stored at storage address 1000, for example, 52 will be at address 1001). In a little-endian system, it would be stored as 524F (52 at address 1000, 4F at 1001).

Endianness becomes an issue any time data is transferred from one computer to another. The receiving computer must somehow be told (because it cannot detect) the endianness of the data. Without knowing, the receiving computer, if it guesses wrong will present the data as a completely indecipherable mess. Additionally, when a file format is developed on one type of computer, the file format usually takes on the endianness of that type of computer. This makes the file format unnatural for computers of the opposing endianness. For example, here are some common file formats and their endianness:

- Adobe Photoshop -- Big Endian
- BMP (Windows and OS/2 Bitmaps) -- Little Endian
- DXF (AutoCad) -- Variable
- GIF -- Little Endian
- IMG (GEM Raster) -- Big Endian
- JPEG -- Big Endian
- FLI (Autodesk Animator) -- Little Endian
- MacPaint -- Big Endian
- PCX (PC Paintbrush) -- Little Endian
- QTM (Quicktime Movies) -- Little Endian (on a Mac!)
- Microsoft RIFF (.WAV & .AVI) -- Both
- Microsoft RTF (Rich Text Format) -- Little Endian
- SGI (Silicon Graphics) -- Big Endian
- Sun Raster -- Big Endian
- TGA (Targa) -- Little Endian
- TIFF -- Both, Endian identifier encoded into file
- WPG (WordPerfect Graphics Metafile) -- Big Endian (on a PC!)

For some reason, for many people, it is very difficult to assimilate the concept of endianness to the level of clarity necessary to actually implement software that gracefully handles it. All the verbal or written explanation in the world does not seem to “sink in” very easily. Take for example, the second paragraph above. After reading it, are you ready to start coding? Probably not.

This is a topic where a picture can be worth a thousand words. Accordingly, here is a diagram which I think clearly conveys the concept of endianness without resorting to a long winded and baffling explanation.




On a Little Endian architecture (e.g. Intel) the least significant byte (and bit) is located at the lowest address. The most significant at the highest address. (The address of the byte is at the Little End of the byte). A 32-bit word is laid out like this on a little endian machine:

```
byte address |0       |1       |2       |3       |
bit          |00000000|00111111|11112222|22222233|
             |01234567|89012345|67890123|45678901|
content      |01234567|89abcdef|ghijklmn|opqrstuv|
```

On a Big Endian architecture (e.g. Motorola) the most significant byte (and bit) is located at the lowest address. The least significant at the highest address. (The address of the byte is at the Big End of the byte). A 32-bit word is laid out like this on a big endian machine:

```
byte address |0       |1       |2       |3       |
bit          |00000000|00111111|11112222|22222233|
             |01234567|89012345|67890123|45678901|
content      |vutsrqpo|nmlkjihg|fedcba98|76543210|
```

Data travels across the wire on a network in big endian byte order. To get from little endian on a sending machine to big endian on a receiving machine the following steps occur:

Start with little endian order:

```
byte address |0       |1       |2       |3       |
bit          |00000000|00111111|11112222|22222233|
             |01234567|89012345|67890123|45678901|
content      |01234567|89abcdef|ghijklmn|opqrstuv|
```

Swap the bytes from little endian order to big endian order – application layer:

```
byte address |0       |1       |2       |3       |
bit          |00000000|00111111|11112222|22222233|
             |01234567|89012345|67890123|45678901|
content      |opqrstuv|ghijklmn|89abcdef|01234567|
```

Swap the bits from little endian order to big endian order – probably data link or hardware layer:

```
byte address |0       |1       |2       |3       |
bit          |00000000|00111111|11112222|22222233|
             |01234567|89012345|67890123|45678901|
content      |vutsrqpo|nmlkjihg|fedcba98|76543210|
```

Write sequence of bits to wire:

```
←  vutsrqponmlkjihgfedcba9876543210 ← <SENDER>
```

Read sequence of bits from wire:

```
<RECEIVER> ←  vutsrqponmlkjihgfedcba9876543210 ←
```

Place bits into big endian bit positions beginning at low address and proceeding to high address – probably data link or hardware layer:

```
byte address |0       |1       |2       |3       |
bit          |00000000|00111111|11112222|22222233|
             |01234567|89012345|67890123|45678901|
content      |vutsrqpo|nmlkjihg|fedcba98|76543210|
```