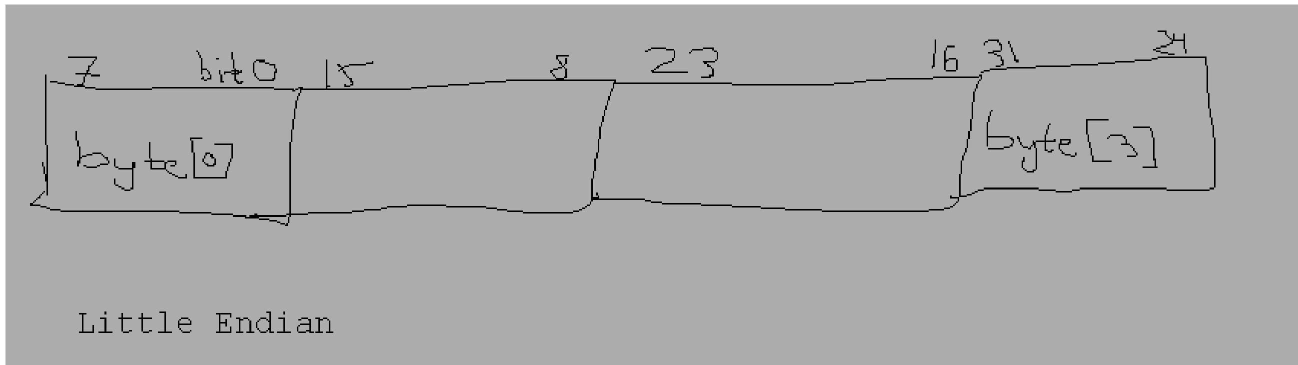
Littleendian And Bigendian Byte Order Illustrated

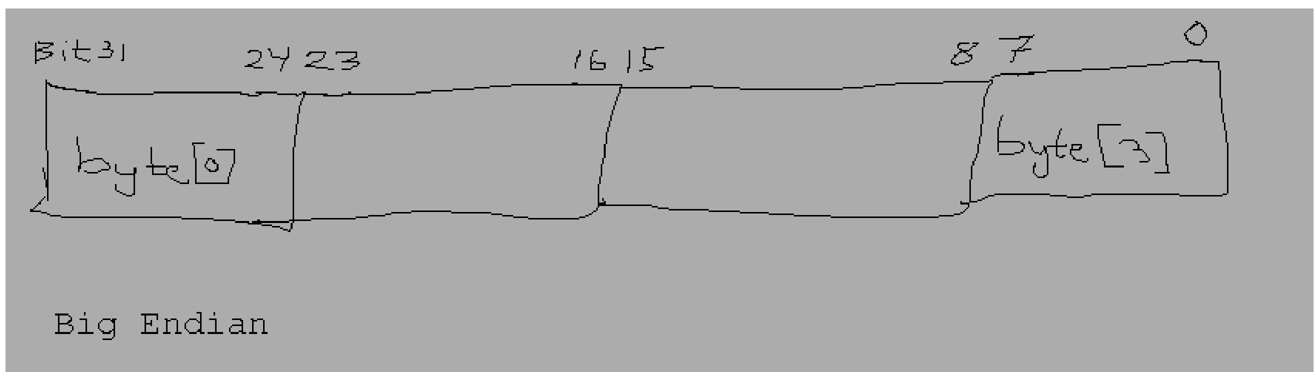
Due to extremely high loads on the server because of the popularity of these two pictures, they have been moved to a page of their own.

A pedagogically correct picture describing little endian byte ordering drawn by me:



0 is the least significant bit, as any reasonable person would think.

The followup to the successful Little Endian drawing: Big Endian



When do I have to care about Endianess?

If you program C/C++ and you don't do any special tricks, then you almost never have to care about the endianess of the target processor. All the arithmetic funtions work the same, e.g 1 << 16 will yield the same results on a 32-bit number on a PowerPC and an Intel processor, C would be useless otherwise.

But you will have to think about the endianess when you deal directly with the memory where 32-bit and 16-bit numbers are stored. E.g. this program:

```
#include<inttypes.h>
#include<stdio.h>


int main(int argc, char** argv )
{
   uint32_t nbr32 = 0x12345678;
   uint16_t* p16 = (uint16_t*)&nbr32;
```

```
   printf("%x\n", *p16 );
}
```

will print 1234 on Sparc/PowerPC and 5678 on an x86. The question is why anyone would write such a program for real.

A more realistic example would be writing binary data to a file. If you have an uint32_t you would like to write to a file, it could be tempting to write

```
size_t write_uint32_to_file( FILE* file, uint32_t nbr )
{
   return fwrite( file, &nbr, 4, 1 );
}

size_t read_uint32_from_file( FILE* file, uint32_t* nbr )
{
   return fread( file, nbr, 4, 1 );
}
```

The downside is that now you cannot read data written by a little endian host on a big endian host. But if you write it like this:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <inttypes.h>

size_t write_uint32_to_file( FILE* file, uint32_t nbr )
{
   nbr = htonl( nbr ); // Host to network long
   return fwrite( file, &nbr, 4, 1 );
}

size_t read_uint32_from_file( FILE* file, uint32_t* nbr )
{
   size_t res = fread( file, nbr, 4, 1 );
   *nbr = ntohl( *nbr ); // Network to host long
   return res;
}
```

then you will be able to read it on any host, since you now know that it will be in network order (which is big endian). The advantage of using these functions is that they exist in one form or another on all operating systems that have internet functionality (Linux, *BSD, Mac OS X, Solaris, Windows etc.) and that they are often optimized. There are four different functions: htonl and ntohl for 32 bit numbers and htons and ntohs for 16 bit numbers.