Convert Little Endian to Big Endian

21

11

I just want to ask if my method is correct to convert from little endian to big endian, just to make sure if I understand the difference.

I have a number which is stored in little-endian, here are the binary and hex representations of the number:

```
0001 0010 0011 0100 0101 0110 0111 1000

12345678
```

In big-endian format I believe the bytes should be swapped, like this:

```
1000 0111 0110 0101 0100 0011 0010 0001

87654321
```

Is this correct?

Also, the code below attempts to do this but fails. Is there anything obviously wrong or can I optimize something? If the code is bad for this conversion can you please explain why and show a better method of performing the same conversion?

```
uint32_t num = 0x12345678;
uint32_t b0,b1,b2,b3,b4,b5,b6,b7;
uint32_t res = 0;

b0 = (num & 0xf) << 28;
b1 = (num & 0xf0) << 24;
b2 = (num & 0xf00) << 20;
b3 = (num & 0xf000) << 16;
b4 = (num & 0xf0000) << 12;
b5 = (num & 0xf00000) << 8;
b6 = (num & 0xf000000) << 4;
b7 = (num & 0xf0000000) << 4;

res = b0 + b1 + b2 + b3 + b4 + b5 + b6 + b7;

printf("%d\n", res);
```

c endianness

share improve this question

edited Jul 29 '17 at 7:11

Shaun Wilson
7,479 ●1 ●39 ●45

asked Oct 9 '13 at 15:13

JeckyPorter
118 ●1 ●1 ●6

2

You could use a better example bit pattern like "0001 0010 0011 0100 0101 0110 0111 1000" – chux Oct 9 '13 at 15:17

8

Your code is nibble-based (4-bit) instead of byte-based (8-bit). It's taking a 32-bit value and reversing the order of the nibbles. I think you wanted 64-bit values done byte-based. Also, the shifts won't work since they shift, not rotate. So you'll lose bits "off the end". And to tidy it up a bit, consider using an array instead of discrete `b1`, `b2`, etc. – lurker Oct 9 '13 at 15:21

1

I am doing this based on the following task: "A 32bit numerical value represented by the hexadecimal representation (st uv wx yz) shall be recorded in a four-byte field as (st uv wx yz)." So If I would do the same but instead taking 8bits (1 byte) it will work ? – JeckyPorter Oct 9 '13 at 15:28

Your examples are 64-bit. So you really meant 32-bit? – lurker Oct 9 '13 at 15:32

2

Don't do the conversion yourself, most platforms provide functions to do this: `htobe32`, `htonl`, etc. if you want portability, use a header like this. – legends2k Dec 25 '14 at 3:35

show **2** more comments

11 Answers

active oldest votes

33

OP's sample code is incorrect.

Endian conversion works at the bit and 8-bit byte level. Most endian issues deal with the byte level. OP code is doing a endian change at the 4-bit nibble level. Recommend instead:

```
// Swap endian (big to little) or (little to big)
uint32_t num = 9;
uint32_t b0,b1,b2,b3;
uint32_t res;

b0 = (num & 0x000000ff) << 24u;
b1 = (num & 0x0000ff00) << 8u;
b2 = (num & 0x00ff0000) >> 8u;
b3 = (num & 0xff000000) >> 24u;

res = b0 | b1 | b2 | b3;

printf("%" PRIX32 "\n", res);
```

If performance is truly important, the particular processor would need to be known. Otherwise, leave it to the compiler.

[Edit] OP added a comment that changes things.
"32bit numerical value represented by the hexadecimal representation (st uv wx yz) shall be recorded in a four-byte field as (st uv wx yz)."

It appears in this case, the endian of the 32-bit number is *unknown* and the result needs to be store in memory in *little* endian order.

```
uint32_t num = 9;
uint8_t b[4];
b[0] = (uint8_t) (num >>  0u);
b[1] = (uint8_t) (num >>  8u);
b[2] = (uint8_t) (num >> 16u);
b[3] = (uint8_t) (num >> 24u);
```

[2016 Edit] Simplification

> ... The type of the result is that of the promoted left operand.... Bitwise shift operators C11 §6.5.7 3

Using a `u` after the *shift* constants (right operands) results in the same as without it.

```
b3 = (num & 0xff000000) >> 24u;
b[3] = (uint8_t) (num >> 24u);
// same as
b3 = (num & 0xff000000) >> 24;
b[3] = (uint8_t) (num >> 24);
```

share improve this answer

edited Apr 9 '16 at 14:58

answered Oct 9 '13 at 15:25

chux

86.6k ● 8 ● 75 ● 160

+1 Now it's correct :) – LihO Oct 9 '13 at 15:34

@LihO yeah, brain hiccup. – chux Oct 9 '13 at 15:38

Ahh. Now I understand. Great Thank you very much! I assume I then only cast the byte array as an uint32 if needed right ? – JeckyPorter Oct 9 '13 at 16:03

@JeckyPorter difficult to "cast the byte array". Instead, unionize. `union JPEndian { uint32_t u32; uint8_t u8[4]; };` `JPEndian.u8[0] = (uint8_t) (num >> 0u); ... printf("%" PRIX32 "\n", JPEndian.u32);` – chux Oct 9 '13 at 16:19

add a comment

23

I think you can use function `htonl()`. Network byte order is big endian.

share improve this answer

answered Oct 9 '13 at 15:24

aoak

633 ● 7 ● 18

Probably the best answer here which avoids reinventing the wheel when a standard solution already exists. – legends2k Dec 25 '14 at 3:32

1

This won't work on big endian machines: stackoverflow.com/questions/21311435/... – CurtisJC Aug 17 '15 at 8:27

Note that `htonl()` is not in the standard C library unfortunately necessitating re-implementation. Some implementations return `uint32_t`, others `unsigned long`. hton32() fixes the size. – chux Jul 7 '16 at 18:41

add a comment

11

Sorry, my answer is a bit too late, but it seems nobody mentioned built-in functions to reverse byte order, which in**very important in terms of performance**.

Most of the modern processors are little-endian, while all network protocols are big-endian. That is history and more on that you canfind on Wikipedia. But that means our processors convert between little- and big-endian millions of times while we browse the Internet.

That is why most architectures have a dedicated processor instructions to facilitate this task. For x86 architectures there is `BSWAP` instruction, and for ARMs there is `REV`. This is **the most efficient way to reverse byte order**.

To avoid assembly in our C code, we can use built-ins instead. For GCC there is `__builtin_bswap32()` function and for Visual C++ there is `_byteswap_ulong()`. Those function will generate **just one processor instruction** on most architectures.

Here is an example:

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

int main()
{
    uint32_t le = 0x12345678;
    uint32_t be = __builtin_bswap32(le);

    printf("Little-endian: 0x%" PRIx32 "\n", le);
    printf("Big-endian:    0x%" PRIx32 "\n", be);

    return 0;
}
```

Here is the output it produces:

```
Little-endian: 0x12345678
Big-endian:    0x78563412
```

And here is the disassembly (without optimization, i.e. `-O0`):

```
        uint32_t be = __builtin_bswap32(le);
   0x0000000000400535 <+15>:    mov    -0x8(%rbp),%eax
   0x0000000000400538 <+18>:    bswap  %eax
   0x000000000040053a <+20>:    mov    %eax,-0x4(%rbp)
```

There is just one `BSWAP` instruction indeed.

So, if we do care about the **performance**, we should **use those built-in functions instead** of any other method of byte reversing. Just my 2 cents.

share improve this answer

answered Feb 14 '18 at 18:10

Andriy Berestovskyy
4,941 ● 2 ● 8 ● 25

add a comment

8

"I swap each bytes right?" -> yes, to convert between little and big endian, you just give the bytes the opposite order. But at first realize few things:

- size of `uint32_t` is 32bits, which is 4 bytes, which is 8 HEX digits
- mask `0xf` retrieves the 4 least significant bits, to retrieve 8 bits, you need `0xff`

so in case you want to swap the order of 4 bytes with that kind of masks, you could:

```
uint32_t res = 0;
b0 = (num & 0xff) << 24;        ; least significant to most significant
b1 = (num & 0xff00) << 8;       ; 2nd least sig. to 2nd most sig.
b2 = (num & 0xff0000) >> 8;     ; 2nd most sig. to 2nd least sig.
b3 = (num & 0xff000000) >> 24;  ; most sig. to least sig.
res = b0 | b1 | b2 | b3 ;
```

share improve this answer

answered Oct 9 '13 at 15:28

LihO
33.1k ● 7 ● 73 ● 139

add a comment

4

You could do this:

```
int x = 0x12345678;

x = ( x >> 24 ) | (( x << 8) & 0x00ff0000 )| ((x >> 8) & 0x0000ff00) | ( x << 24)  ;

printf("value = %x", x);  // x will be printed as 0x78563412
```

share improve this answer

edited Mar 14 '15 at 21:14

Spotlight
263 ● 1 ● 10 ● 20

answered Oct 17 '14 at 3:18

ayyappa ch
67 ● 6

add a comment

2

I am assuming you are on linux

Include `"byteswap.h"` & Use `int32_t bswap_32(int32_t argument);`

It is logical view, In actual see, `/usr/include/byteswap.h`

share improve this answer

edited Jun 17 '16 at 4:18

ketan
15.5k ● 12 ● 38 ● 66

answered Jun 17 '16 at 3:57

0x47-sci-tech
30 ● 5

add a comment

1

One slightly different way of tackling this that can sometimes be useful is to have a union of the sixteen or thirty-two bit value and an array of chars. I've just been doing this when getting serial messages that come in with big endian order, yet am working on a little endian micro.

union MessageLengthUnion {

```
uint16_t asInt;
uint8_t asChars[2];
```

};

Then when I get the messages in I put the first received uint8 in .asChars[1], the second in .asChars[0] then I access it as the .asInt part of the union in the rest of my program. If you have a thirty-two bit value to store you can have the array four long.

share improve this answer

answered Oct 9 '13 at 15:37

DiBosco
509 ● 4 ● 17

add a comment

1

one more suggestion :

```
unsigned int a = 0xABCDEF23;
a = ((a&(0x0000FFFF)) << 16) | ((a&(0xFFFF0000)) >> 16);
a = ((a&(0x00FF00FF)) << 8) | ((a&(0xFF00FF00)) >>8);
printf("%0x\n",a);
```

share improve this answer

answered Dec 30 '15 at 14:56

Saurabh Sengar
489 ● 1 ● 7 ● 15

How about explaining how it works and what benefits this approach has? – SamB Dec 30 '15 at 19:53

in first instruction it swap the 16 bits sets (ie word length) and in second instruction it swaps the 8 bits sets (ie character length) resulting in big endian to little endian and vice versa conversion. Output will be : 23EFCDAB. and yeah the benefit is no extra variable and less steps – Saurabh Sengar Dec 31 '15 at 5:38

add a comment

1

OP's code is incorrect for the following reasons:

- The swaps are being performed on a nibble (4-bit) boundary, instead of a byte (8-bit) boundary.
- The shift-left `<<` operations of the final four swaps are incorrect, they should be shift-right `>>` operations and their shift values would also need to be corrected.
- The use of intermediary storage is unnecessary, and the code can therefore be rewritten to be more concise/recognizable. In doing so, some compilers will be able to better-optimize the code by recognizing the oft-used pattern.

Consider the following code, which efficiently converts an unsigned value:

```
// Swap endian (big to little) or (little to big)
uint32_t num = 0x12345678;
uint32_t res =
    ((num & 0x000000FF) << 16) |
    ((num & 0x0000FF00) << 8) |
    ((num & 0x00FF0000) >> 8) |
    ((num & 0xFF000000) >> 16);

printf("%0x\n", res);
```

The result is represented here in both binary and hex, notice how the bytes have swapped:

```
0111 1000 0101 0110 0011 0100 0001 0010

78563412
```

Optimizing

In terms of performance, leave it to the compiler to optimize your code when possible. You should avoid unnecessary data structures like arrays for simple algorithms like this, doing so will usually cause different instruction behavior such as accessing RAM instead of using CPU registers.

share improve this answer

answered Jul 29 '17 at 7:57

Shaun Wilson
7,479 ● 1 ● 39 ● 45

add a comment

1

A Simple C program to convert from little to big

```
#include <stdio.h>

int main() {
unsigned int little=0x1234ABCD,big=0;
unsigned char tmp=0,l;

printf(" Little endian little=%x\n",little);

for(l=0;l < 4;l++)
{
    tmp=0;
    tmp = little | tmp;
    big = tmp | (big << 8);
    little = little >> 8;
}
printf(" Big endian big=%x\n",big);

return 0;
}
```

share improve this answer

edited Nov 6 '17 at 17:41

answered Nov 6 '17 at 17:35

Naresh pothula
21 ● 2

add a comment

0

You can use the lib functions. They boil down to assembly, but if you are open to alternate implementations in C, here they are (assuming int is 32-bits) :

```
void byte_swap16(unsigned short int *pVal16) {

//#define method_one 1
// #define method_two 1
#define method_three 1
#ifdef method_one
    unsigned char *pByte;

    pByte = (unsigned char *) pVal16;
    *pVal16 = (pByte[0] << 8) | pByte[1];
#endif

#ifdef method_two
    unsigned char *pByte0;
    unsigned char *pByte1;

    pByte0 = (unsigned char *) pVal16;
    pByte1 = pByte0 + 1;
    *pByte0 = *pByte0 ^ *pByte1;
    *pByte1 = *pByte0 ^ *pByte1;
    *pByte0 = *pByte0 ^ *pByte1;
#endif

#ifdef method_three
    unsigned char *pByte;

    pByte = (unsigned char *) pVal16;
    pByte[0] = pByte[0] ^ pByte[1];
    pByte[1] = pByte[0] ^ pByte[1];
    pByte[0] = pByte[0] ^ pByte[1];
#endif


}
```

And the usage is performed like so:

```
unsigned short int u16Val = 0x1234;
byte_swap16(&u16Val);
unsigned int u32Val = 0x12345678;
byte_swap32(&u32Val);
```

share improve this answer

answered Aug 10 '17 at 13:19

netskink
1,187 ● 14 ● 22

add a comment

Your Answer

B

Sign up or log in

Sign up using Google | Sign up using Facebook | Sign up using Email and Password

Post as a guest

Name

Email
Required, but never shown

Post Your Answer

By clicking "Post Your Answer", you agree to our terms of service, privacy policy and cookie policy

Not the answer you're looking for? Browse other questions tagged c endianness or ask your own question.

asked	5 years, 8 months ago
viewed	129,197 times
active	1 year, 4 months ago

Blog

- Stack Overflow and Pursuit: Nurturing A New

Generation of Developers

Featured on Meta

- Unicorn Meta Zoo #4: What makes for a healthy community?

Linked

86
convert big endian to little endian in C [without using provided func]

22
read bitmap file into structure

7
DOES htonl() change byte order on BIG ENDIAN machine?

4
How can I reorder the bytes of an integer?

-1
Confusion regarding pointer size

0
C++ - Reading number of bits per pixel from BMP file

0
iOS app Modbus read/write floating numbers

0
Reason for & and | in endianess conversion

-1
What would be the Get Big Endian function looks like in Objective C?

Related

179
How do I convert between big-endian and little-endian values in C++?

59
C# little endian or big endian?

90
Does Java read integers in little endian or big endian?

100
C Macro definition to determine big endian or little endian machine?

86
convert big endian to little endian in C [without using provided func]

3
Bit manipulation of integers depending on endianess

59
C program to check little vs. big endian

19
Little Endian vs Big Endian?

-2
binary to int with strtol in C

2
Converting a structure from Little endian to Big endian

Hot Network Questions

How to write a nice frame challenge?

Can the pre-order traversal of two different trees be the same even though they are different?

Scaling an object to change its key

Is the author of the Shu"t HaRidvaz the same one as the one known to be the rebbe of the Ariza"l?

- How to make all magic-casting innate, but still rare?
- Leaving job close to major deadlines
- How to best clean this sealed rotary encoder / volume knob?
- What is that ceiling compartment of a Boeing 737?
- Synaptic Static - when to roll the d6?
- Can a character learn spells from someone else's spellbook and then sell it?
- Justifying Affordable Bespoke Spaceships
- Is declining an undergraduate award which causes me discomfort appropriate?
- The Amazing Sliding Crossword
- Setting up the trap
- Is there a term for the belief that "if it's legal, it's moral"?
- Why is it easier to balance a non-moving bike standing up than sitting down?
- What is the highest power supply a Raspberry pi 3 B can handle without getting damaged?
- I found a password with hashcat but it doesn't work
- How can a warlock learn from a spellbook?
- Why do you need to heat the pan before heating the olive oil?
- Slow Performance When Changing Object Data [2.8]
- Densest sphere packing
- Teferi's Time Twist and Gideon's Sacrifice
- How can I restore a master database from its bak file?

Question feed

STACK OVERFLOW

Questions
Jobs
Developer Jobs Directory
Salary Calculator
Help
Mobile
Disable Responsiveness

PRODUCTS

Teams
Talent
Advertising
Enterprise

COMPANY

About
Press
Work Here
Legal
Privacy Policy
Contact Us

STACK EXCHANGE NETWORK

Technology
Life / Arts
Culture / Recreation
Science
Other

Blog
Facebook
Twitter
LinkedIn