BitScan (/BitScan)

Bearbeiten

7 (/BitScan#discussion)

7 (/BitScan#discussion)

7 (/BitScan#discussion) 231 (/page/history/BitScan) ... (/page/menu/BitScan) ome \* Board Representation \* Bitboards \* BitScan

M C Esch Eve. 1946 [1

a function that determines the bit-index of the least significant 1 bit (LS1B) or the most significant 1 bit (MS1B) in an integer such as bitboards. If exactly one bit is set in an unsigned integer, representing a numerical value of a power of two , this is equivalent to a base-2 logarithm . Many implementations have been devised since the advent of bitboards, as described on this page, and some implementation samples of concrete open source engines listed for didactic purpose.

Table of Contents

Hardware vs. Softv

Non Empty Sets

Bitscan forward

Trailing Zero Count

De Bruijn Multiplication With isolated LS1B

With separated LS1E

Matt Taylor's Folding trick

Walter Faxon's magic Bitso

Bitscan by Modulo

Divide and Conquer

Double conversion of LS1B

Index of LS1B by Poncount

Ritscan reverse

Divide and Conquer

Tribute to Frank Zappa

De Bruiin Multiplication Double conversion

Leading Zero Count

Bitscan versus Zero Count

Bitscan with Reset

Generalized Bitscan

Processor Instructions for Bitscans

Emulating Intrinsics

Intrinsics versus asm

Bsf/Bsr x86-64 Timings

Bsf/Bsr behavior with zero source

ARM

Engine Samples

See also Publications

Forum Posts

1996 ...

2000 ...

2005 ... 2010 ...

2015 ... External Links

References What links here?

## Hardware vs. Software

For recent x86-64 architectures like Core 2 duo and K10 , one should use the Processor Instructions for Bitscans via intrinsics or inline assembly, see x86-64 timing, P4 and K8 have rather slow bitscan-instructions. K8 uses so called vector path instructions [2] with 9 or 11 cycles latency, even blocking other processor resources. For these processors, specially K8 with already fast multiplication, the De Bruijn Multiplication (64-bit mode) or Matt Taylor's Folded 32-bit Multiplication (32-bit mode) might be the right choice. Other routines mentioned might be advantageous on certain architectures, specially with slow integer multiplications.

## Non Empty Sets

Bitscan is most often used in serializing bitboards, and is therefor - due to a leading while-condition - not called with empty sets. Until stated otherwise, most mentioned bitscan-routines in CIC++ have the same prototype and assume none empty sets as actual parameter.

## Bitscan forward

A bitscan forward is used to find the index of the least significant 1 bit (LS1B)

## Trailing Zero Count

Bitscan forward is identical with a Trailing Zero Count for none empty sets, possibly available as machine instruction on some architectures, for instance the x86-64 bit-manipulation expansion set BMI1.

## De Bruiin Multiplication

The **De Bruijn** bitscan was devised in 1997, according to Donald Knuth [3] by Martin Läuter, and independently by Charles Leiserson, Harald Prokop and Keith H. Randall a few month later [4] [5], to determine the LS1B index by minimal perfect hashing. De Bruijn sequences were named after the Dutch mathematician Nicolaas de Bruijn. Interestingly sequences with the binary alphabet were already investigated by the French mathematician Camille Flye Sainte-Marie in 1894, but later "forgotten" and re-investigated and generalized by De Bruijn and Tanja van Ardenne-Ehrenfest half a century later 10.

A 64-bit De Bruijn sequence contains 64-overlapped unique 6-bit sequences, thus a circle of 64 bits, where five leading zeros overlap five hidden "trailing" zeros. There are 226 = 67108864 odd sequences with 6 leading binary zeros and 226 even sequences with 5 leading binary zeros, which may be calculated from the odd ones by shifting left one.

A multiplication with a power of two value (the isolated LS1B) acts like a left shift by it's exponent. Thus, if we multiply a 64-bit De Bruijn sequence with the isolated LS1B, we get a unique six bit subsequence inside the most significant bits. To obtain the bit-index we need to extract these upper six bits by shifting right the product, to lookup an array

```
const int index64[64] = {
   0, 1, 48, 2, 57, 49, 28, 3, 61, 58, 50, 42, 38, 29, 17, 4,
   62, 55, 59, 36, 53, 51, 43, 22,
   45, 39, 33, 30, 24, 18, 12,
   63, 47, 56, 27, 60, 41, 37, 16,
   54, 35, 52, 21, 44, 32, 23, 11,
   46, 26, 40, 15, 34, 20, 31, 10, 25, 14, 19, 9, 13, 8, 7, 6
* @author Martin Läuter (1997)
            Charles E. Leiserson
             Harald Prokop
            Keith H. Randall
```

```
* "Using de Bruijn Sequences to Index a 1 in a Computer Word"

* @param bb bitboard to scan

* "Params "Mirio fine file list say farewell... Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wikispaces.com)

* @return index (0.63) of Least significant one bit

*/
int bitScanForward(UG4 bb) {
    const UG4 debruijn64 = CG4(0x03F79d71b4cb0a89);
    assert (bb != 0);
    return index64[((bb & -bb) * debruijn64) >> 58];
}
```

See also how to Generate your "private" De Bruijn Bitscan Routine.

#### With separated LS1B

Instead of the classical LS1B isolation, Kim Walisch proposed the faster xor with the ones' decrement. The separation bb ^ (bb-1) contains all bits set including and below the LS1B. The 2<sup>22</sup> (4,194,304) upper De Bruijn sequences of the 2<sup>26</sup> available leave unique 6-bit indices. Using LS1B separation takes advantage of the x86 lea instruction, which saves the move instruction and unlike negate, has no data dependency on the flag register. Kim reported a 10 to 15 percent faster execution (compilers: g++-4.7 -O2, clang++-3.1 -O2, x86\_64) than the traditional 64-bit De Bruijn bitscan on Intel Nehalem and Sandy Bridge CPUs.

```
const int index64[64] = {
   0, 47, 1, 56, 48, 27, 2, 60, 57, 49, 41, 37, 28, 16, 3, 61,
   54, 58, 35, 52, 50, 42, 21, 44,
   38, 32, 29, 23, 17, 11, 4, 62,
   34, 51, 20, 43, 31, 22, 10, 45,
   25, 39, 14, 33, 19, 30, 9, 24,
   13, 18, 8, 12, 7, 6, 5, 63
* bitScanForward
* @author Kim Walisch (2012)
* @param bb bitboard to scan
 * @precondition bb != 0
* @return index (0..63) of Least significant one bit
int bitScanForward(U64 bb) {
   const U64 debruijn64 = C64(0x03f79d71b4cb0a89);
   assert (bb != 0);
   return index64[((bb ^ (bb-1)) * debruijn64) >> 58];
```

#### Matt Taylor's Folding trick

A 32-bit finedly implementation to find the the bit-index of LS1B by Matt Taylor (I). The xor with the ones' decrement, bb ^ (bb-1) contains all bits set including and below the LS1B. The 32-bit xor-difference of both halves yields either the complement of the upper half, or the lower half otherwise. Some samples:

Even if this folded "LS1B" contains multiple consecutive one-bits, the multiplication is De Bruijn like. There are only two magic 32-bit constants with the combined property of 32- and 64-bit De Bruijn sequences to apply this minimal parfect hashing:

```
const int lsb_64_table[64] =
   63, 30, 3, 32, 59, 14, 11, 33,
   60, 24, 50, 9, 55, 19, 21, 34, 61, 29, 2, 53, 51, 23, 41, 18,
   56, 28, 1, 43, 46, 27, 0, 35,
   62, 31, 58, 4, 5, 49, 54, 6,
15, 52, 12, 40, 7, 42, 45, 16,
   25, 57, 48, 13, 10, 39, 8, 44,
   20, 47, 38, 22, 17, 37, 36, 26
* bitScanForward
* @author Matt Taylor (2003)
   @param bb bitboard to scan
* @precondition bb != 0
   @return index (0..63) of Least significant one bit
int bitScanForward(U64 bb) {
   unsigned int folded;
assert (bb != 0);
   bb ^= bb - 1;
folded = (int) bb ^ (bb >> 32);
   return lsb_64_table[folded * 0x78291ACF >> 26];
```

A slightly modified version may take one x86-register less in 32-bit mode, but calculates bb-1 twice:

```
int bitScanForwardM(BitBoard bb) {
    unsigned int folded;
    assert (bb != 0);
    folded = (int)((bb ^ (bb-1)) >> 32);
    folded ^= (int)( bb ^ (bb-1)); // lea
    return lsb_64_table[folded * 0x78291ACF >> 26];
}
```

with this VC6 generated x86 assembly to compare:

```
bitScanForward PROC NEAR

mov ecx, DWORD PTR _bb$[esp-4]

mov eax, DWORD PTR _bb$[esp]

mov eax, DWORD PTR _bb$[esp]

mov edx, ecx

mov edx, ecx

push esi

bitScanForwardM PROC NEAR

mov eax, DWORD PTR _bb$[esp-4]

mov ecx, eax

add ecx, -1

mov ecx, DMORD PTR _bb$[esp]
```

```
add edx. -1
                                                   mov edx, ecx
   mov esi, eax
                                                   adc edx, -1
 📭 ወቅ 🖈 lt% stimer for us to say farewell... Regretfully, wê%e ተዘቋ፞ኘው የfex tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wikispaces.com) xor ecx, edx lea ecx, DWORD PTR [eax-1]
   xor eax, esi
                                                   xor edx, ecx
   pop esi
    xor eax, ecx
                                                   xor edx, eax
   imul eax, 78291acfH
                                                   imul edx, 78291acfH
   shr eax, 26
                                                   shr edx, 26
   mov eax, DWORD PTR _lsb_64_table[eax*4] mov eax, DWORD PTR _lsb_64_table[edx*4]
bitScanForward ENDP
                                               bitScanForward ENDP
```

## Walter Faxon's magic Bitscan

Walter Faxon's 32-bit friendly magic bitscan [8] uses a fast none minimal perfect hashing function

```
const char LSB_64_table[154] =
#define __ 0
   22,__,_,30,__,38,18,__,16,15,17,__,46, 9,19, 8, 7,10, 0, 63, 1,56,55,57, 2,11,__,58, __,20,_, 3,__,59,__,_,
  28, __, 26, __, __, 53, __, __, 27, __, 35, __, 52, __, 26, __, 43, 34, 25, 23, 24, 33, 31, 32, 42, 39, 40, 51, 41, 14, __, 49, 47, 48, __, 50, 6, __, 62, __, __, 54
#undef __
* bitScanForward
* @author Walter Faxon, slightly modified
* @param bb bitboard to scan
* @precondition bb != 0
* @return index (0..63) of Least significant one bit
int bitScanForward(U64 bb)
   assert(bb);
   bb ^= bb - 1;
   t32 = (int)bb ^ (int)(bb >> 32);
   t32 ^= 0x01C5FC81;
   t32 += t32 >> 16;
t32 -= (t32 >> 8) + 51;
   return LSB_64_table [t32 & 255]; // 0..63
```

A slightly modified version may take one  $\underline{x86}$ -register less in 32-bit mode, but calculates bb-1 twice:

```
int bitScanForward(U64 bb)
{
  int t32 = 0x01C5FC81;
  assert(bb);
  t32 ^= (int)((bb ^ (bb-1)) >> 32);
  t32 ^= (int)( (bb ^ (bb-1)); // lea
  t32 += t32 >> 16;
  t32 -= (t32 >> 8) + 51;
  return LSB_64_table [t32 & 255];
}
```

The initial LS1B separation by bb ^ (bb-1) and folding is equivalent to Matt's,

ls1b	bb ^ (bb-1)	folded
63	0xffffffffffffffff	0x00000000
62	0x7fffffffffffffffff	0x80000000
59	0x0ffffffffffffffff	0xf0000000
32	0x00000001ffffffff	0xfffffffe
31	0x00000000fffffffff	0xffffffff
30	0x000000007fffffff	0x7fffffff
0	0x000000000000000001	0x00000001

while Walter originally resets the LS1B, yielding in a cyclic index wrap:

## Bitscan by Modulo

Another idea is to apply a modulo (remainder of a division) operation of the isolated LS1B by the prime number 67 1101. The remainder 0.66 can be used to perfectly hash the bit-index table. Three gaps are 0, 17, and 34, so the mod 67 can make a branchless trailing zero count:

Bit-Index	Bitboard	mod 67
-	0x0000000000000000	0
0	0x0000000000000001	1
1	0x00000000000000002	2
2	0x0000000000000004	4
3	0x000000000000000	8
4	0x0000000000000010	16
5	0x000000000000000000000000000000000000	32
6	0x0000000000000040	64
7	0x000000000000000000000000000000000000	61
8	0x0000000000000100	55
9	0x000000000000000000000000000000000000	43
10	0x0000000000000400	19
11	0x000000000000000000000000000000000000	38
12	0x000000000001000	9
13	0x0000000000002000	18
14	0x0000000000004000	36
15	0x000000000000000000000000000000000000	5

```
16
           0x0000000000010000
                                                                         10
             0x0000000000020000
                                                                        20

    ★ 0%90999909099099 farev99. Tracev99. Tracev99. Tracev999. Tracev999. Tracev999. Tracev9999. Tracev9999. Tracev9999. Tracev9999. Tracev9999. Tracev9999. Tracev99999. Tracev9999. Tracev99999. Tracev99999. Tracev99999. Tracev99999. Tracev9999. Tracev99999. Tracev9999. Tracev999. Tracev9999. Tracev9999. Tracev9999. Tracev9999. Tracev999. Tracev999. Tra
 19
             0x0000000000080000
                                                                         13
20
              0x000000000100000
                                                                        26
21
             0x0000000000200000
22
             0x0000000000400000
                                                                        37
23
             0x00000000008000000
24
              0x000000001000000
25
             0x00000000002000000
                                                                        28
26
             0x0000000004000000
                                                                        56
27
              0x0000000008000000
28
             0x0000000010000000
                                                                        23
29
             0x0000000020000000
                                                                        46
30
              0x0000000040000000
31
             0x0000000080000000
                                                                        50
             0x0000000100000000
32
                                                                        33
33
              0x0000000200000000
34
             0x0000000400000000
                                                                        65
             0x0000000800000000
35
                                                                        63
              0x0000001000000000
37
             0x0000002000000000
                                                                        51
             0x0000004000000000
38
                                                                        35
39
40
             0x0000010000000000
                                                                           6
41
             0x0000020000000000
                                                                         12
42
43
             0x0000080000000000
                                                                        48
44
             0x00001000000000000
                                                                        29
45
46
             0x0000400000000000
                                                                        49
47
             0x0000800000000000
                                                                        31
              0x0001000000000000
49
             0x0002000000000000
                                                                        57
50
             0x00040000000000000
                                                                        47
              0x000800000000000
52
             0x00100000000000000
                                                                        54
53
             0x00200000000000000
                                                                        41
              0x0040000000000000
55
             0x0080000000000000
                                                                        30
56
             0x01000000000000000
                                                                        60
57
              0x0200000000000000
                                                                        53
58
             0x04000000000000000
                                                                        39
59
             0x0800000000000000
                                                                         11
60
             0x1000000000000000
                                                                        22
61
             0x20000000000000000
                                                                        44
62
             0x40000000000000000
                                                                        21
63
              0x8000000000000000
```

```
* trailingZeroCount

* @param bb bitboard to scan

* @return index (0..63) of least significant one bit, 64 if bb is zero

*/
int trailingZeroCount(U64 bb) {
    static const int lookup67[67+1] = {
        64,       0,       1,       39,       2,       15,       40,       23,
        3,       12,       16,       59,       41,       19,       24,       54,
        4,       -1,       13,       10,       17,       62,       60,       28,
        42,       30,       20,       51,       25,       44,       55,       47,
        5,       32,       -1,       38,       14,       22,       11,       58,
        18,       53,       63,       9,       61,       27,       29,       58,
        43,       46,       31,       37,       21,       57,       52,       8,
        26,       49,       45,       36,       56,       7,       48,       35,
        6,       34,       33,       -1 };
    return lookup67[(bb & -bb) % 67];
}
```

Since div/mod is an expensive instruction, a modulo by a constant is likely replaced by reciprocal fixed point multiplication to get the quotient and a second multiplication and difference to get the remainder. Compared with De Bruijn multiplication it is still too slow.

## Divide and Conquer

This is a broad group of bitscans that test in succession, like the trailing zero count based on Reinhard Scharnag's proposal [11]:

```
* trailingZeroCount
    Like bitScanForward for none empty sets
* @author Reinhard Scharnagl
 * @param bb bitboard to scan
* @return index (0..64)
unsigned char lsbRS[256] = {
    8, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
     5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
        0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
int trailingZeroCount(U64 b) {
  unsigned buf;
  int acc = 0:
```

What about direct calculation? On x86 this is a chain of test, set and lea instructions:

#### Double conversion of LS1B

Assuming 64-bit doubles and little-endian structure (not portable). We convert the isolated LS1B to a double and interprete the exponent:

## Index of LS1B by Popcount

If we have a fast population-count instruction, we can count the trailing zeros of LS1B after subtracting one

```
// precondition bb != 0
int bitScanForward(U64 bb) {
   assert (bb != 0);
   return popCount( (bb & -bb) - 1 );
}
```

## Ritscan reverse

A bitscan **reverse** is used to find the index of the **most** significant 1 bit (MS1B). For non empty sets it is equivalent to floor of the base-2 logarithm. MS1B isolalation or separation is more expensive than LS1B isolalation or separation, due to the LS1B related Two's complement tricks are not applicable. However, beside Divide and Conquer and Double conversion, Bitscan reverse with MS1B separation is mentioned.

# Divide and Conquer

As introduced by Eugene Nalimov in 2000, for an IA-64 version of Crafty [12] [13]

```
/**
    bitScanReverse
    @author Eugene Nalimov
    @param bb bitboard to scan
    @return index (0..63) of most significant one bit
    //
int bitScanReverse(U64 bb)
{
    int result = 0;
    if (bb > 0xFFFFFFFFF) {
        bb >>= 32;
        result = 32;
    }
    if (bb > 0xFFFFFF {
        bb >>= 16;
        result += 16;
    }
    if (bb > 0xFFF) {
        bb >>= 8;
        result += 8;
    }
    return result + mslbTable[bb];
}
```

TriButs to Fnank Zappay farewell... Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wikispaces.com)

A branchless and little bit obfuscated version of the devide and conquer bitScanReverse with in-register-lookup [14] - as tribute to Frank Zappa with identifiers from Freak Out! (1966), Hot Rats (1969), Waka/Jawaka (1972), Sofa (1975), One Size Fits All (1975), Sheik Yerbouti (1979), and Jazz from Hell (1986):

```
typedef unsigned __int64 OneSizeFits;
typedef unsigned int HotRats;
const HotRats s
                         = 0:
const HotRats heik = 457;
const HotRats y
const HotRats e
                         = 1;
= 2;
const HotRats r
                         = 4;
const HotRats b
const HotRats o
                         = 5;
const HotRats u
                        = 8;
const HotRats t
                         = 16;
                         = 32;
const HotRats i
const HotRats ka = (1 << 4)-1;
const HotRats waka = (1 << 8)-1;
const HotRats jawaka = (1<<16)-1;
const HotRats jazzFromHell = 0-(16*3*heik);
HotRats freakOut(OneSizeFits all) {
   HotRats so, fa;
   fa = (HotRats)(all >> i);
so = (fa!=s) << o;
   fa ^= (HotRats) all & (fa!=s)-y;
   so ^= (jawaka < fa) << b;
fa >>= (jawaka < fa) << b;
so ^= ( waka - fa) >> t & u;
   fa >>= ( waka - fa) >> t & u;

so ^= ( ka - fa) >> u & b;

fa >>= ( ka - fa) >> u & b;
   so ^= jazzFromHell >> e*fa & r;
   return so;
```

## De Bruijn Multiplication

While the <u>fribute</u> to <u>Frank Zappa</u> is quite 32-bit friendly [15], <u>Kim Walisch</u> suggested to use the <u>parallel prefix fill</u> for a <u>MS1B</u> separation with the same <u>De Bruijn</u> multiplication and lookup as in his <u>bitScanForward</u> routine with <u>separated LS1B</u>, with less instructions in 64-bit mode. A log base 2 method was already devised by Eric Cole on January 8, 2006, and shaved off rounded up to one less than the next power of 2 by Mark Dickinson [15] on December 10, 2009, as published in Sean Eron Anderson's *Bit Twiddling Hacks* for 32-bit integers [17].

```
const int index64[64] = {
   0, 47, 1, 56, 48, 27, 2, 60, 57, 49, 41, 37, 28, 16, 3, 61,
   54, 58, 35, 52, 50, 42, 21, 44,
   38, 32, 29, 23, 17, 11, 4, 62,
   46, 55, 26, 59, 40, 36, 15, 53,
   34, 51, 20, 43, 31, 22, 10, 45,
  25, 39, 14, 33, 19, 30, 9, 24, 13, 18, 8, 12, 7, 6, 5, 63
* bitScanReverse
* @authors Kim Walisch, Mark Dickinson
* @param bb bitboard to scan
   @precondition bb != 0
* @return index (0..63) of most significant one bit
int bitScanReverse(U64 bb) {
   const U64 debruijn64 = C64(0x03f79d71b4cb0a89);
   assert (bb != 0):
   bb |= bb >> 1;
   bb |= bb >> 2;
   bb |= bb >> 4;
   bb |= bb >> 8;
   bb |= bb >> 16;
   return index64[(bb * debruijn64) >> 58];
```

## Double conversion

Assuming 64-bit doubles and little-endian structure (not portable). Conversion to a double, interpreting the exponent. To avoid possible rounding errors, some lower bits may be cleared.

## Leading Zero Count

Some processors have a fast leading zero count instruction. The Motorola 68020 has a bit field find first one instruction (BFFFO), which actually performs an up to 32-bit Leading Zero Count 119 . x86-64 AMD K10 has Izcnt as part of the SSE4a extension 119 120 , BM11 has Izcnt as well, while AVX-512CD even features leading zero count on vectors of eight bitbaords.

One can replace bitScanReverse of non empty sets by leading ZeroCount xor 63. Like trailing zero count, it returns 64 for empty sets, and might therefor save the leading condition in some applications

Bitecant views its zeroa commell... Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wikispaces.com)

While the presented bitscan routines are suited to work only on none empty sets and return a value-range from 0 to 63 as bit-index, leading or trailing zero-count instructions or routines leave 64 for empty sets. Zero-counting has a immanent property of dealing correctly with empty sets - while it likely takes a conditional branch to implement this semantic in bit-scanning.

```
int trailingZeroCount(U64 bb) {
    if ( bb )
        return bitScanForward(bb);
    return 64;
}

int leadingZeroCount(U64 bb) {
    if ( bb )
        return bitScanReverse(bb) ^ 63;
    return 64;
}
```

## Bitscan with Reset

While traversing sets, one may combine bitscanning with reset found bit. That implies passing the bitboard per reference or pointer, and tends to confuse compilers to keep all inside registers inside a typical serialization loop [21].

```
int bitScanForwardWithReset(U64 &bb) { // also called dropForward
  int idx = bitScanForward(bb);
  bb &= bb - 1; // reset bit outside
  return idx;
}
```

#### Generalized Bitscan

This generalized bitscan uses a boolean parameter to scan reverse or forward. It relies on bitScanReverse, but conditionally masks the <u>LS1B</u> in case of scanning forward. It might be used in the <u>classical approach</u> to get positive or negative ray directions with one generalized routine.

```
* generalized bitScan

* @author Gerd Isenberg

* @param bb bitboard to scan

* @precondition bb != 0

* @param reverse, true bitScanReverse, false bitScanForward

* @return index (0.63) of least/most significant one bit

*/

int bitScan(U64 bb, bool reverse) {
    U64 rMask;
    assert (bb != 0);
    rMask = -(U64)reverse;
    bb &= -bb | rMask;
    return bitScanReverse(bb);
}
```

## **Processor Instructions for Bitscans**

# x86

x86-64 processors have bitscan instructions and can be accessed with compilers today through either inline assembly or compiler intrinsics. For the Microsoft/Intel C compiler, the intrinsics can be accessed by including and using the instructions \_BitScanForward64 [22] , \_BitScanReverse64 [23] or \_Izcn164 [24] .

```
unsigned char_BitScanForward64(unsigned long * Index, unsigned __int64 Mask);
unsigned char_BitScanReverse64(unsigned long * Index, unsigned __int64 Mask);
unsigned __int64 __lzcnt64(unsigned __int64 value); // AMD K10 only see CPUID
```

Linux provides library functions [25], find first bit set (ffsII) in a word leaves an index of 1..64, and zero of no bit is set [25]. GCC 4.4.5 further has the Built-in Function \_builtin\_ffsII for finding the least significant one bit, \_builtin\_ctzII for trailing, and \_builtin\_ctzII for leading zero count [27]:

```
/* Returns one plus the index of the Least significant 1-bit of x, or if x is zero, returns zero */
int _builtin_ffsll (unsigned long long);

/* Returns the number of trailing \theta-bits in x, starting at the least significant bit position.

If x is \theta, the result is undefined */
int _builtin_ctzll (unsigned long long);

/* Returns the number of leading \theta-bits in x, starting at the most significant bit position.

If x is \theta, the result is undefined */
int _builtin_ctzll (unsigned long long);
```

## **Emulating Intrinsics**

For the GNU C compiler, the intrinsics can be emulated with inline assembly [28]

```
//These processor instructions work only for 64-bit processors
#ifdef _MSC_VER
   #include <intrin.h>
   #ifdef _WIN64
       #pragma intrinsic(_BitScanForward64)
#pragma intrinsic(_BitScanReverse64)
       #define USING_INTRINSICS
   #endif
#elif defined(__GNUC__) && defined(__LP64__)
   U64 Ret;
       __asm__
          "bsfq %[Mask], %[Ret]"
          :[Ret] "=r" (Ret)
          :[Mask] "mr" (Mask)
       *Index = (unsigned long)Ret;
       return Mask?1:0
   static INLINE unsigned char _BitScanReverse64(unsigned long* Index, U64 Mask)
```

```
U64 Ret;
__asm__

■ *It's time for us to say farewell... Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wikispaces.com)

"bsrq *{Mask}, *{Ret}"

: [Ret] "=r" (Ret)
: [Rask] "mr" (Mask)
);

*Index = (unsigned long)Ret;
return Mask?1:0;
}
#define USING_INTRINSICS
#endif
```

#### Intrinsics versus asm

Alternatively, rather than to emulate the intrinsics one might use the standard prototype, by using intrinsics or inline assembly, for GCC [29]:

```
#ifdef USE_X86INTRINSICS
#include <intrin.h>
#pragma intrinsic(_BitScanForward64)
#pragma intrinsic(_BitScanReverse64)
* bitScanForward
* @param bb bitboard to scan
 * @precondition bb != 0
* @return index (0..63) of least significant one bit
int bitScanForward(U64 x) {
   unsigned long index;
assert (x != 0);
   _BitScanForward64(&index, x);
return (int) index;
* bitScanReverse
* @param bb bitboard to scan
 * @precondition bb != 0
* @return index (0..63) of most significant one bit
int bitScanReverse(U64 x) {
   unsigned long index;
assert (x != 0);
   _BitScanReverse64(&index, x);
return (int) index;
#else
* bitScanForward
 * @param bb bitboard to scan
   @precondition bb != 0
* @return index (0..63) of least significant one bit
int bitScanForward(U64 x) {
   assert (x != 0);
asm ("bsfq %0, %0" : "=r" (x) : "0" (x));
return (int) x;
* bitScanReverse
* @param bb bitboard to scan
* @precondition bb != 0
* @return index (0..63) of most significant one bit
int bitScanReverse(U64 x) {
   assert (x != 0);
asm ("bsrl %0, %0" : "=r" (x) : "0" (x));
   return (int) x;
#endif
```

# Bsf/Bsr x86-64 Timings

The instruction latency and reciprocal throughput [30] heavily differs between various x86-64 architectures:

Architecture Stepping	Instruction(s)	Latency / Cycles	Reciprocal Throughput
AMD			
K8 [31]	BSF reg16/32/64, mreg16/32/64	Vector Path 8/8/9	8/8/9
	BSR reg16/32/64, mreg16/32/64	Vector Path 11	11
K10 [32]	BSF reg, reg	Vector Path 4	4
	BSR reg, reg	Vector Path 4	4
	LZCNT reg, reg	Direct Path single 2	1
Intel [33]			
<u>ATOM</u>	BSF/BSR	16	15
NetBurst 0F_3H	BSF/BSR	16	i 4
NetBurst 0F_2H	BSF/BSR	8	3 2
Core 06_0EH	BSF/BSR	2	1
65 nm Intel Core 06_0FH	BSF/BSR	2	1
Enhanced Intel Core 06_17H	BSF/BSR	1	1
Enhanced Intel Core 06_1DH	BSF/BSR	1	1
Nehalem 06_1AH	BSF/BSR	3	1
Sandy Bridge	BSF/BSR	3	1
Ivy Bridge	LZCNT	3	1
Haswell [34]	TZCNT	3	1

# Bsf/Bsr behavior with zero source

Intel and AMD specify different behavior. In praxis there seems no difference so far. However, as long as Intel docs explicitly state content undefined, it is recommend to don't rely on a pre-initialized content of that target register, if the source is zero.

- Intel:If the content of the source operand is 0, the content of the destination operand is undefined. 35
- AMD: If the second operand contains 0, the instruction sets ZF to 1 and does not change the contents of the destination register.

ARM has CLZ (Count Leading Zeros) instruction for 32-bit integers. ARM instruction is available in ARMv5 and above, 32-bit Thumb instruction is available in ARMv6T2 and ARMv7. [32], the C-intrinsic is called \_builtin\_clz [38]

● x It's time for us to say farewell... Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wikispaces.com)

#### **Engine Samples**

- · BitScan in Amundsen
- BitScan in Chess 0.5
- BitScan in CookieCat
- BitScan in Crafty (23.5)
- BitScan in Gibbo
- BitScan in Gk
- BitScan in Hea
- BitScan in Kurt BitScan in Murka
- BitScan in Prophet
- BitScan in RedQu
- BitScan in Spector

#### See also

- BITSCAN, a C++ library for bitstrings by Pablo San Segundo
- Bit-Twiddling
- De Bruiin Sec
- Java-Bitscan
- Population Count

## **Publications**

- Alan Turing (1949). Alan Turing's Manual for the Ferranti Mk. ! . transcribed in 2000 by Robert Thau , pdf from The Computer History Museum, 9.4 The position of the most significant digit » Ferranti Mark 1.
- Charles E, Leiserson, Harald Prokop and Keith H, Randall (1998). Using de Bruijn Sequences to Index a 1 in a Computer Word, pdf
   Pablo San Segundo, Ramón Galán (2005). Bitboards: A New Approach. AIA 2005
- Donald Knuth (2009). The Art of Computer Programming. , Volume 4, Fascicle 1: Bitwise tricks & techniques, as Pre-Fascicle 1a postscript. , p. 10
- Andreas Stiller (2013). Spezialkommando Bits setzen, abfragen, scannen und mehr . c't Magazin für Computertechnik 7/2013, p. 186 (German)

#### Forum Posts

#### 1996

- Bitboards: speeding up FirstOne by Laurent Desnogues, rgcc, April 10, 1996 » Othello
- bitboard 2<sup>h</sup>i mod 67 is unique by Stefan Plenkner, rgcc, August 6, 1996
- bitboard 2<sup>h</sup> mod 67 is unique by Stefan Plenkner, rgcc, August 7, 1996
- bitboard 2<sup>A</sup>i mod 67 is unique by Joël Rivat, rgcc, September 2, 1996
- Question to Bob: Crafty, Alpha and FindBit() by Guido Schimmels, CCC, June 05, 1998
- To Nalimov and other programmers about BSF/BSR in VC by Dezhi Zhao, CCC, January 16, 1999

#### 2000

- Re: TASM 5.0 versus BSF by Frans Morsch, comp.lang,asm.x86 , March 28, 2000
   Will the Itanium have a BSF or BSR instruction? by Larry Griffiths, CCC, August 15, 2000
  - Re: Will the Itanium have a BSF or BSR instruction? by Eugene Nalimov, CCC, August 16, 2000
- Binary question by Severi Salminen, CCC, October 19, 2000
- Bitboards and Piece Lists by Dann Corbit, CCC, June 14, 2001
  FirstBit() in assembler by David Rasmussen, CCC, January 13, 2002
- Reply from Intel about BSF/BSR by Severi Salminen, CCC, January 31, 2002

  "Using de Bruijn Sequences to Index a 1 in a Computer Word" by Oliver Roese, CCC, February 08, 2002
- Another hacky method for bitboard bit extraction by Walter Faxon, CCC, November 17, 2002 Modulo verus BitScan and MMX-PopCount by Gerd Isenberg, CCC, November 29, 2002
- Fast 3DNow! BitScan by Gerd Isenberg, CCC, December 01, 2002
- Bitscan Conclusions by Matt Taylor, CCC, January 05, 2003
- Bitscan by Matt Taylor, CCC, February 11, 2003
- FirstOne for Linux by Sune Fischer, CCC, March 29, 2003
- Bit magic by Matt Taylor, comp.lang.asm.x86 , June 26, 2003
- Re: De Bruijn Sequence Generator by Dieter Bürssner, CCC, December 30, 2003 » De Bruijn Sequence Generator Determining location of LSB/MSB by Renze Steenhuisen, CCC, February 09, 2004
- Nalimov: bsf/bsr intrinsics implementation still not optimal by Dezhi Zhao, CCC, September 22, 2004
- Re: Nalimov: bsf/bsr intrinsics implementation still not optimal by Eugene Nalimov, CCC, September 23, 2004

## 2005 ...

- A data point for PowerPC bitboard program authors by Steven Edwards, CCC. May 09, 2005 » PowerPC
   Best BitBoard LSB funktion? by Reinhard Scharnagl, Winboard Programming Forum, July 20, 2005
- Fastest bitboard compress routine when you can't use ASM by mambofish, CCC, May 31, 2007
   Bit twiddling question, part 2; arbitrary bitscan order by Zach Wegner, CCC, August 11, 2009
- 32 bit versions for bitscan64 by Michael Hoffmann, CCC, August 21, 2009
- 64-bit intrinsic performance by Nathan Thom, CCC, October 27, 2009
- Bit Scan (equivalent to ASM instructions bsr and bsf) by Pascal Georges, CCC. December 24, 2009

## 2010 ...

- bitScanReverse32 by Luca Hemmerich, CCC, January 25, 2010
- Introduction and (hopefully) contribution bitboard methods by Alcides Schulz, CCC, June 03, 2011 » Population Count
   Leading Zero Count Question by Matthew R. Brades, CCC, September 16, 2012
- Optimizing bitboards for ARM by Martin Sedlak, CCC, November 17, 2012
- Symmetric move generation using bitboards by Lasse Hansen, CCC, December 20, 2014

  Stockfish 32-bit and hardware instructions on MSVC++ by Syed Fahad, CCC, December 30, 2014 » Stockfish, BitScan, Population Count

## 2015 ...

- Fun with De Bruin by Henk van den Belt, CCC, August 27, 2015
- Re: Linux Version of Maverick 1.5 by Michael Dvorkin, CCC, November 12, 2015 » OS X, Maverick
- . syzygy users (and Ronald) by Robert Hyatt, CCC, September 29, 2016 » Population Count

## **External Links**

- Find first set from Wikipedia
- The Aggregate Magic Algorithms by Hank Dietz
- Bit Twiddling Hacks by Sean Eron Anderson
- An Efficient Bit-Reversal Sorting Algorithm for the Fast Fourier Transform by Jennifer Elaan , January 16, 2005
- Efficient bit scan mechanism United States Patent 6172623 from FreePatentsOnline.com

#### References

- 1. ^ Picture gallery "Back in Holland 1941 1954" from The Official M.C. Escher Website
  2. ^ Chip Architect: Detailed Architecture of AMD's Opteron 1.3 A third class of Instructions by Hans de Vries
- 3. ^ Donald Knuth (2009). The Art of Computer Programming , Volume 4, Fascicle 1: Bitwise tricks & techniques, as Pre-Fascicle 1a postscript , p 10 4. ^ Charles E. Leiserson, Harald Prokop and Keith H. Randall (1998). Using de Bruijn Sequences to Index a 1 in a Computer Word, pdf
- 5. ^ "Using de Bruijn Sequ ences to Index a 1 in a Computer Word" discussion in CCC, February 08, 2002
- 6. N. G. de Bruijn (1975). Acknowledgement of priority to C. Flye Sainte-Marie on the counting of circular arrangements of 2n zeros and ones that show each n-letter word exactly once. Technical Report, Technische Hogeschool
- 7. A Bit magic by Matt Taylor, comp.lang.asm.x86 , June 26, 2003
- 8. <u>Another hacky method for bitboard bit extraction</u> by Walter Faxon, CCC, November 17, 2002
- 9. <u>^ bitboard 2^i mod 67 is unique</u> by <u>Stefan Plenkner, rgcc</u>, August 6, 1996
  10. <u>^ Pablo San Segundo</u>, <u>Ramón Galán</u> (**2005**). <u>Bitboards: A New Approach</u> . <u>AIA 2005</u>
- 11. <u>^ Best BitBoard LSB funktion?</u> by Reinhard Scharnagl, Winboard Programming Forum, July 20, 2005
  12. <u>^ Re: Will the Itanium have a BSF or BSR instruction?</u> by <u>Eugene Nalimov, CCC</u>, August 16, 2000
- 12. Are with the national rave a BSF of BSF instruction? by <u>suggere Nation</u>, <u>CCC</u>, Adgust to, 13. Are with the national rave a BSF of BSF instruction? by <u>Stef Luiten</u>, <u>CCC</u>, Adgust to, 14. A just another reverse bitscan by <u>Gerd Isenberg</u>, <u>CCC</u>, December 22, 2005

  15. A final version homage to FZ by <u>Gerd Isenberg</u>, <u>CCC</u>, December 23, 2005

  16. A <u>EuroPython 2012</u>: Florence, July 2–8 | Mark Dickinson

- 17. A Find the log base 2 of an N-bit integer in O(lg(N)) operations with multiply and lookup from Bit Twiddling Hacks by Sean Eron Anderson
- 18. ^ 68020 Bit Field Instructions
- 19. ^ SSE4a from Wikipedia
- 20. ^ Izcnt16, Izcnt, Izcnt64 Visual C++ Language Reference
- 21. A Bitscan by Matt Taylor, CCC, February 11, 2003
- 22. ^ BitScanForward, BitScanForward64 Visual C++ Language Reference
  23. ^ BitScanReverse, BitScanReverse64 Visual C++ Language Reference
- 24. <u>^ Izont16, Izont, Izont64</u> Visual C++ Language Reference 25. <u>^ Section 3: library functions Linux man pages</u>

- 26. ^ ffsll(3): find first bit set in word Linux man page
  27. ^ Other Builtins Using the GNU Compiler Collection (GCC)
- 28. A Re: Nalimov: bsf/bsr intrinsics implementation still not optimal by Eugene Nalimov, CCC, September 23, 2004
- 29. <u>^ Matters Computational ideas, algorithms, source code</u> (pdf) Ideas and Source Code by <u>Jörg Arndt</u>
- 30. A Instruction tables, Lists of instruction latencies, throughputs and microoperation breakdowns for Intel and AMD CPU's (pdf) by Agner Fog
- 31. ^ Software Optimization Guide for AMD64 Processors
- 32. ^ Software Optimization Guide for AMD Family 10h and 12h Proce
- 33. ^ Intel 64 and IA32 Architectures Optimization Reference Manual
- 34. A Haswell Instructions Latency
- 35. ^ Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M (pdf) BSF—Bit Scan Forward 3-87
  36. ^ AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions (pdf) Bit Scan Forward pg. 111
- 37.  $\underline{\land}$  ARM Information Center > General data processing instructions > CLZ
- 38. ^ ARM Information Center > Instruction intrinsics > builtin clz 39. ^ Other Builtins - Using the GNU Compiler Collection (GCC)
- 40. A Bit Scan (equivalent to ASM instructions bsr and bsf) by Pascal Georges, CCC, December 24, 2009

# What links here?

Page	Date Edited
68020	Jun 7, 2016
<u>Alcides Schulz</u>	Mar 29, 2018
Algorithms	May 5, 2017
Amundsen	Sep 3, 2013
Аггау.	Dec 1, 2016
Attack and Defend Maps	Nov 5, 2016
AVX-512	Aug 8, 2017
Backtracking	Dec 16, 2017
Beowulf	May 5, 2017
Bison	Sep 29, 2016
<u>Bit</u>	Oct 29, 2012
Bit-Twiddling	Nov 6, 2017
<u>Bitboard Serialization</u>	Dec 24, 2014
<u>Bitboards</u>	Nov 14, 2017
<u>Bitfoot</u>	Sep 8, 2015
<u>BitScan</u>	Sep 10, 2017
Blockage Detection	Oct 19, 2017
BMI1	Mar 24, 2014
BMI2	Mar 6, 2018
Bobcat	Jun 27, 2017
Brainless	Jun 24, 2017
Cassandre	Jul 5, 2013
<u>Charles Leiserson</u>	Oct 12, 2016
CHEOPS	Apr 18, 2015
<u>Chess 0.5</u>	Nov 20, 2016
Chezzz	Jan 20, 2013
Classical Approach	Jan 28, 2018
Congruent Modulo Bitboards	Jun 26, 2013

Company         Company <t< th=""><th>Page</th><th>Date Edited</th></t<>	Page	Date Edited
EachOne AD, 2017Cligati100 y 100Cligati100 y 100Cligati </td <td>Cokie Cat: s time for us to say farewell Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wik</td> <td>Nov 15, 2016 ispaces.com)</td>	Cokie Cat: s time for us to say farewell Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wik	Nov 15, 2016 ispaces.com)
CasesAssert 1985Cases AllerinaNe. 2985Cases AllerinaNe.		
Data billionsPer 3 2016Calistificanisma(0+18) 70Calistificanisma(0+18) 70Calistificanisma(0+18) 70Calistificanisma(0+18) 70Calistificanisma(0+18) 70Calistificanisma(0+18) 70Calistificanisma(0+18) 70Calistificanisma(0+18) 70Calistificanisma(0+18) 70Calistificanisma(0+28) 70Cali	<u>Cray-1</u>	Dec 25, 2017
Case Salamanean         10-16 2017           Case Salamanean         10-16 2018           Case Salamanean         10-17 2018           Case Salamanean		
Carbonis Sequence Connector         Sept. 2009           Chilling Sequence Connector         Cont. 2009           Chilling Sequence Connector         Aug 82, 2017           Chilling Sequence Connector         Aug 82, 2017           Chilling Sequence Connector         Cont. 2009           Children Connector         Aug 82, 2016           Children Connector         Aug 82, 2016           Claims Connector Connector         Aug 72, 2019           Claims Conn		
De Polit Signame dineries (Del 1908)         May 1.0016           Calification         Aug 1.0016           Calification         Aug 2.007           Clause (Statistics)         Aug 2.001           Clause (Statistics)         Most 2.001           Clause (Statistics)         Most 2.001           Clause (Statistics)         Most 2.001           Control Musica         Most 2.001           California         Most 2.001		
DEL Maria         Moy 5,7915           Obserte         Moy 5,7915           Obserte         Moy 5,7915           Obserte         Moy 5,7916           Obserte         Moy 5,7916           Claman         Moy		
DescriptionsMay 25, 2019DescriptionsAny 25, 2019DescriptionsProbabilityDescriptionsMay 25, 2019DescriptionsMay 25, 2019Entered Research Statisty ProceduresMay 25, 2019Entered Research Statisty ProceduresMay 25, 2019Entered Research Statisty ProceduresMay 27, 2018Entered Research Statisty ProceduresMay 22, 2019EstatMay 22, 2019EstatMay 22, 2019EstatMay 23, 2019CalcalAny 23, 2019 <t< td=""><td></td><td></td></t<>		
Date of the Control         Control <td></td> <td></td>		
Date         May 90, 1916           Excention of Altricy Photo Alexis         Key 2, 2018           Excention Allering         May 2, 2018           Excention Allering         Concert Allering         Concert Allering           Excention Allering         App 2, 2013         Concert Allering           Excent Allering         App 2, 2013         Concert Allering           Excent Allering         App 2, 2013         Concert Allering           Excent Allering         App 2, 2013         Concert Allering		Dec 1, 2016
Escotion Motions         No. 6, 2016           Excotion Motions         Leep 2, 2016           Escotion Motions         Leep 2, 2016           Escotion Motions         Leep 2, 2017           Escotion Motions         Leep 2, 2017           Escotion School Control         Leep 2, 2018           Escotion School Control         Leep 2, 2019           Escotion School Control         Leep 2, 2017	<u>Djinn</u>	Feb 8, 2016
Exercite Ministra         Med 27.219           Express Ministra         Dec 27.007           Existra         Anne 2.009           Existra         One 22.007	Double Country	May 30, 2016
Expend Marchand         Dec 22, 2016           Examed Man, 1         Jun 2, 2016           Elace         Cec 2, 2017           Blass         Revol 11, 2016           Control         Revol 12, 2016           Control         Revol 12, 2016           Called Control         Pack 22, 2014           Called Control         Out, 2017           Called Control         Apr 22, 2013           Called Control         Apr 22, 2013           Elace Honor         Apr 22, 2013           User Honor         Apr 22, 2013           Heart Propose         Apr 22, 2014	Efficient Generation of Sliding Piece Attacks	Nov 5, 2016
### 1985年		
Elab         Des 22 2017           Dial         No. 1, 1016           Commit         No. 1, 1016           Commit         April 2, 1018           Commit         Sept. 2, 2018           Classical Contrains         Sept. 2, 2018           Classical Contrains         Classical Contrains           Classical Contrains         April 2, 2017           Classical Contrains         April 2, 2017           Usual Date         April 2, 2017           Usual Date         No. 29, 2017           Usual Date         April 2, 2014           Usual Date         April 2, 2014           Usual Date         April 2, 2014           Usual Date         April 2, 2017		
Basel         Mov 11, 2016           Gardel         Jan 2, 1016           Gardel         Pee 25, 2016           Garden Schinards         Pee 25, 2014           Garden Schinards         Cert 2, 2017           Garden Schinards         Apr 22, 2013           BaskerBille         Apr 22, 2013           Basker Product         Apr 22, 2017           Basker Product         Jan 1, 2018           Basker Product         Pee 14, 2014           Basker Product         Apr 22, 2015           Basker Product         Apr 22, 2017           Basker Product         Apr 22, 2014           Basker Product		
Scientis         Septimization         19.1 (2014)           Classica         Dec. 22, 2014           Classica         Cer. 22, 2014           Classica Schreiner         Cer. 22, 2014           Classica Schreiner         Apr 22, 2013           Education Schreiner         Apr 22, 2013           Education Schreiner         Apr 22, 2013           Hand Problem         Jun 1, 2018           Hand Problem         Jun 1, 2018           Hand Problem         Apr 22, 2014           Hall         Apr 22, 2015           Hand         Apr 22, 2015           Hand         Apr 22, 2015           Hand         Apr 22, 2017           Hall         Apr 22, 2017           Hall         Apr 22, 2017           Hall         Apr 22, 2013           Hall         Apr 22, 2014		
Schemal Matheu Orantom         Peb 25, 2018           Gilboon         0ct 2, 2017           Octs         0ct 9, 2017           Schemach         Apr 22, 2013           Halkaselith         Apr 22, 2013           Halkaselith         Nove 22, 2017           Halkaselith         Apr 22, 2018           Halkaselith         Apr 22, 2017           James Halkasel         Jul 2, 1017           Schill, Rasell         Jul 2, 1017           Schill, Rasell         Apr 10, 2017           Schill, Rasell         Apr 10, 2017           Schill, Park         Peb 22, 2018           Schill, Park         Peb 22, 2018           Schill, Park         Peb 22, 2018		
District         Obe 23, 2014           CR         Oct 9, 2017           Cable Schirmeis         Apr 2, 2013           Habscellith         Apr 22, 2013           Habscellith         Apr 22, 2014           Harris Praise         Apr 22, 2015           Harris Praise         Feb 1, 2014           Harry Class         Feb 7, 2016           Brill         Feb 7, 2016           Idel         Aug 29, 2015           Idel         Aug 29, 2015           Identified         Aug 29, 2017           Identified         Aug 29, 2017           Identified         Aug 12, 2014           Identified         Aug 12, 2014           Identified         Aug 12, 201		
Gist         Oct 9, 2017           Guito Schirmatis         Apr 20, 2013           Hashinghillis         Apr 20, 2013           Hashinghillis         Nov 20, 2017           Hashinghillis         Nov 20, 2017           Hashinghillis         Apr 20, 2014           Hashinghillis         Fab 1, 2014           Bill         Apr 20, 2015           Bill         Apr 20, 2015           Bill         Apr 20, 2015           Barrier         Apr 20, 2015           Barrier         Apr 20, 2015           Barrier         Apr 20, 2015           Barrier         Apr 20, 2014           Barrier         Apr 20, 2017           Barrier         Apr 20, 2014	•	
Godde Schimmein         Apr 20, 2013           Habbanelliting         Apr 20, 2016           Habbanelliting         Apr 20, 2017           Heard Facion         Apr 20, 2018           Heard Table         Jun 1, 2018           Heard Chaes         Feb 1, 2014           Bill         Feb 1, 2016           Bill         App 29, 2015           Berling         App 29, 2017           Berling         App 20, 2017           Select         App 20, 2017           Select         App 20, 2017           Select         App 20, 2017           Select         App 20, 2013           Select         App 20, 2014           Select <td></td> <td></td>		
March Prinzico         Any 26, 2016           Francis Prinzico         Nov. 20, 2017           Base Prinzico         Nov. 20, 2017           Biana Chicas         Feb. 14, 2014           Biana Chicas         Feb. 14, 2014           Biana Chicas         Aug. 20, 2015           Biana Chicas         Aug. 20, 2015           Biana Chicas         Aug. 20, 2015           Biana Chicas         Aug. 20, 2017           Jan Renzo Steenhulen         Aug. 20, 2017           Jan Renzo Steenhulen         Aug. 20, 2017           Jan Bernard         Aug. 20, 2017           Jane Life Chicas         Aug. 20, 2017           Kenth L. Facada         Aug. 20, 2013           Kenth L. Facada         Aug. 20, 2014           Kenth L. Facada         Aug. 20, 2014           Kenth Parken         Aug. 20, 2014           Kell Parken         Aug. 20, 2014 <td></td> <td></td>		
Haben Tables         Jun 1, 2018           Haben Chases         Feb 1, 4, 2014           Iff         Feb 1, 4, 2014           Iff         Feb 7, 2016           Intill         Aug 20, 2015           Intill         Aug 20, 2015           Jam Ranna Steerhuisen         Aug 20, 2017           Jam Bilasca         Man 15, 2014           Joel Birken         Jul 21, 2017           Sober IT         Sep 16, 2017           Kim LH Jamoid         Jul 10, 2017           Kim Vallech         Aug 10, 2017           Kim Vallech         Aug 10, 2017           Kim Quality         Sep 4, 2013           Kim Quality         Sep 2, 2015           Kim Limb         Feb 22, 2015           Kim Limb         Feb 22, 2015           Kim Limb         Feb 22, 2016           Kim Limb         Feb 22, 2016           Kim Limb         Feb 22, 2016           Kim Limb         Feb 22, 2014           Kim Limb         May 8, 2017           Limb         Wing 2, 2014           Matterneticle         May 8, 2017	<u>Hakkapeliitta</u>	Apr 26, 2016
Heary Chase         Feb 14, 2014           tits         Feb 12, 2016           tits         Feb 2, 2015           lateliam         Aug 29, 2015           lateliam         Aug 29, 2015           Jan Record Steenhalsen         Sep 10, 2017           Janne Bitscord         Jun 15, 2014           John Bitscord         Jul 21, 2017           Joher IT         Sep 16, 2017           Keith H. Randal         Jun 16, 2015           Kon Malmich         Aug 10, 2017           Kon Dettere         Nov 15, 2013           Kon Dettere         Nov 15, 2013           Konn Livya Velant         Feb 23, 2015           Konditiva Velant         Feb 23, 2015           Kurd         Aug 20, 2014           Mark 20, 2014         Aug 20, 2014           Mark 20, 2014         Aug 20, 2014 <td>Harald Prokop</td> <td>Nov 29, 2017</td>	Harald Prokop	Nov 29, 2017
Iring         Feb. 7, 2016           Iring         Aug 29, 2015           Bank         Aug 29, 2015           Bank Repres Steenhuisen         Sep 10, 2017           Jave-Bitscan         Mar 15, 2014           Joel Hora         Jun 16, 2015           Steber IT         Sep 16, 2017           Kein H. Randell         Jun 16, 2015           Kein Wallisch         Aug 10, 2017           Kein Spatien         Aug 10, 2017           Keinge Patien         Nov 15, 2013           Kein Wallisch         Heb 23, 2015           Keinger Patien         Peb 23, 2015           Kongel Ming         Feb 22, 2015           Kein Wall         Mary 15, 2013           Kein Steel         Aug 10, 2014           Leils         Aug 20, 2014           Leils         Apr 20, 2014           Leils         Aug 20, 2017           Leils Wing         De 28, 2017           March Seduk         Mar 25, 2016           March Seduk         Mar 25, 2016           Martin Seduk         Jun 9, 2017           Michael Schwi	Hash Table	Jan 1, 2018
Intel         Aug 29, 2015           Bankum         Aug 29, 2016           Jan Renze Steenhuisen         Sep 10, 2017           Jan Renze Steenhuisen         Mar 15, 2014           Jacker IT         Sep 16, 2017           John Call         Aug 10, 2017           Keith H. Bandell         Aug 10, 2017           Kinn Delktern         Sep 4, 2013           Kinn Jacker IT         Peb 22, 2015           Koundrivu Velar         Feb 22, 2015           Koundrivu Velar         Peb 22, 2014           Leal         Mar 19, 2017           Lines Hermerich         Jun 19, 2017           Mac 28         Mar 19, 2017           Mac 29         Mar 19, 2014           Marin Taylor         Feb 22, 2018           Marin Taylor         Feb 22, 2018           Markham Bradela         Jun 9, 2017           Michael Charchira         Mor 19, 2014           Michael Charchira	<u>HeavyChess</u>	Feb 14, 2014
Itaniam         Aug 29, 2015           Jan Renze Sterenhuisen         Sep 10, 2017           Javo-Bitscan         Mar 15, 2014           Jeurs Bitzen         Jul 21, 2017           Jose Bitzen         Jul 21, 2017           Jose Li TY         Sep 16, 2017           Keath H. Randall         Jul 16, 2015           Kinn Yalisch         Aug 10, 2017           Kinn Se Kinns         Sep 4, 2013           Kinn Se Kinns         Sep 4, 2013           King Pattern         Peb 23, 2015           King Pattern         Feb 23, 2015           Kund Liver         Feb 22, 2013           Kurit         Feb 22, 2013           Kurit         Jul 19, 2017           Lelle         Peb 28, 2014           Lelle         Jul 19, 2017           Mac DS         Mar 26, 2016           Martin Sciolik         Mer 26, 2016           Martin Sciolik         De 28, 2017           Mathematician         De 28, 2017           Mathematician         Jul 19, 2017           Methale Divortin         Jul 19, 2017           Methale Divortin         Jul 19, 2017           Methale Divortin         Oct 30, 2013           Multinution         Oct 30, 2013      <	<u>lfit</u>	Feb 7, 2016
Jane Benze Steenhuisen         Sep 10, 2017           Jave Stäcen         Mar 15, 2014           Jedt Rivers         Jul 21, 2017           Joster IT         Sep 16, 2017           Keith H. Randall         Jun 16, 2015           Kim Wallisch         Aug 10, 2017           King 24 Kinng         Sep 4, 2013           King 24 Rivers         Nov 15, 2013           King 24 Rivers         Feb 23, 2015           Kong Pattern         Feb 23, 2015           Kong Pattern         Feb 22, 2013           Kong Le Rivers         Feb 22, 2013           Kurt         Aug 10, 2014           Liels         May 20, 2014           Liels         May 20, 2014           Liels Wing         Oct 26, 2017           Liels Heimmerlich         Jun 19, 2017           Mac S         Mar 25, 2016           Mac S         Re 22, 2013           Martin Sediak         Mar 25, 2016           Martin Sediak         Jun 9, 2017	<u>Intel</u>	Aug 29, 2015
Jave-Bitscan         Mar 15, 2014           Jose Fixed         Jul 21, 2017           Joker IT         Sep 16, 2017           Keith H. Randell         Jun 16, 2015           Keith H. Randell         Aug 10, 2017           Kino of Kings         Sep 4, 2013           King Palletin         Nov 15, 2013           King Palletin         Feb 22, 2015           King Libertan         Feb 22, 2015           Kur         Apr 20, 2014           Leila         May 8, 2017           Leile Wing         Oct 28, 2017           Liuca Hammarich         Mar 25, 2016           Marcin Sediak         Dec 25, 2017           Martin Sediak         Dec 26, 2018 <td></td> <td></td>		
Joel Rivat         Jul 21, 2017           Joker IT         Sep 16, 2017           Keith H. Randall         Jun 16, 2015           Kim Wallach         Aug 10, 2017           Kim Of Kinya         Sep 4, 2013           King Pattern         Nov 15, 2013           King Pattern         Feb 23, 2015           Kundinya Vahari         Feb 2, 2013           Kur         Feb 2, 2013           Kur         Aug 20, 2014           Leila         May 8, 2017           Little Wing         Oct 26, 2017           Luca Hemmerich         Jun 19, 2017           Macu CS         Mar 25, 2016           Martin Sediak         Dec 25, 2017           Matthewark I Brades         Feb 2, 2018           Matthewar I, Brades         Feb 17, 2013           Matthewar I, Brades         Jun 9, 2017           Matthewar I, Brades         Jun 9, 2017           Matthewar I, Brades         Jun 9, 2017           Matthewar I, Brades         Oct 30, 2013           Matthewar I, Brades         Oct 30, 2013           Matthewar I, Brades         Oct 30, 2013           Markes         Shura-Bura         Oct 30, 2013           Markes         Shura-Bura         Oct 30, 2013      <		
Johen IT         Sep 16, 2017           Keith H. Randall         Jun 16, 2015           Kim Vallaich         Aug 10, 2017           King at Kings         Sep 4, 2013           King Pattern         Nov 15, 2013           Koudinya Veluri         Feb 22, 2015           Koudinya Veluri         Feb 2, 2013           Kurt         Apr 20, 2014           Leila         May 8, 2017           Litte Wing         Oct 26, 2017           Litte Hammerich         Jun 19, 2017           Mac DS         Mar 25, 2016           Martin Sediak         Dec 25, 2017           Mathematisian         Feb 28, 2018           Matt Taylor         Feb 28, 2018           Matt Taylor         Feb 28, 2018           Matt Taylor         Jun 9, 2017           Mathewa R. Brades         Jun 9, 2017           Michael Doorkin         Jan 8, 2016           Mikhall R. Shura-Bura         Oct 30, 2013           Murka         Nov 11, 2016           Nicolas de Brüln         Feb 3, 2015           Noragériace         Nov 23, 2014           Obstruction Difference         May 27, 2016		
Keith H. Randall         Jun 16, 2015           Kim Walisch         Aug 10, 2017           King Cattern         Nov 15, 2013           Kolgel Ratern         Nov 15, 2013           Koundinya Veluri         Feb 2, 2013           Kurt         Apr 20, 2014           Leita         May 8, 2017           Little Wing         Oct 26, 2017           Luca Hemmerich         Mar 25, 2016           Martin Sediak         Mer 25, 2016           Martin Sediak         Dec 25, 2017           Mathematician         Feb 28, 2018           Mathraylor         Feb 17, 2013           Mathraylor         Feb 17, 2013           Mathrew R. Brades         Jun 9, 2017           Michael Dorrkin         Jan 8, 2016           Mikhail R. Shura-Bura         Oct 30, 2013           Murka         Nov 11, 2016           Nicolasa de Brujn         Feb 2, 2015           Nicolasa de Brujn         Feb 2, 2015           Nicolasa de Brujn         Nov 23, 2014           Obstruction Difference         May 27, 2016		
Kim Walisch       Aug 10, 2017         King of Kings       Sep 4, 2013         King Pattern       Nov 15, 2013         Kinght Pattern       Feb 23, 2015         Koundinya Yakuri       Feb 2, 2013         Kurt       Apr 20, 2014         Leils       May 8, 2017         Little Wing       Oct 26, 2017         Luca Hemmerich       Jun 19, 2017         Mac, OS       Mar 25, 2016         Martin Sedlak       Dec 25, 2017         Mathematician       Feb 28, 2018         Matt Taylor       Feb 17, 2013         Mattlaryor       Feb 17, 2013         Mattlew R. Brades       Jun 9, 2017         Michael Dvotkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolaes de Brujn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016		
Kins of Kings         Sep 4, 2013           King Pattern         Nov 15, 2013           Knight Pattern         Feb 23, 2015           Koundinya Veluri         Feb 2, 2013           Kurt         Apr 20, 2014           Leila         May 8, 2017           Little Wing         Oct 26, 2017           Luca Hemmerich         Jun 19, 2017           Mac OS         Mar 25, 2016           Martin Sedlak         Dec 25, 2017           Matthematician         Feb 28, 2018           Matt Taylor         Feb 17, 2013           Matthem R. Brades         Jun 9, 2017           Michael Dvorkin         Jan 8, 2016           Michael Dvorkin         Oct 30, 2013           Mukha         Nov 11, 2016           Nathan Thom         Mar 15, 2013           Nicolaas de Bruijn         Feb 3, 2015           Nora Grace         Nov 23, 2014           Obstruction Difference         May 27, 2016		
King-Pattern       Nov 15, 2013         Knight Pattern       Feb 23, 2015         Koundinya Veluri       Feb 2, 2013         Kurt       Apr 20, 2014         Leila       May 8, 2017         Little Wing       Oct 26, 2017         Luca Hemmerich       Jun 19, 2017         Mac OS       Mar 25, 2016         Martin Sedlak       Dec 25, 2017         Mathematician       Feb 28, 2018         Matti Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael Dorckin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Mulka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolaas de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016		
Koundinya Veluri       Feb 2, 2013         Kurt       Apr 20, 2014         Leila       May 8, 2017         Little Wing       Oct 26, 2017         Luca Hemmerich       Jun 19, 2017         Mac OS       Mar 25, 2016         Martin Sediak       Dec 25, 2017         Mathematician       Feb 28, 2018         Matt Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolaas de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016		
Kurt       Apr 20, 2014         Leila       May 8, 2017         Little Wing       Oct 26, 2017         Luca Hemmerich       Jun 19, 2017         Mac OS       Mar 25, 2016         Martin Sedlak       Dec 25, 2017         Mathematician       Feb 28, 2018         Matt Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael Dvorkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolaas de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016	Knight Pattern	Feb 23, 2015
Leila       May 8, 2017         Little Wing       Oct 26, 2017         Luca Hemmerich       Jun 19, 2017         Mac OS       Mar 25, 2016         Martin Sedlak       Dec 25, 2017         Mathematician       Feb 28, 2018         Matt Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael Dvorkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolaas de Brujn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016	Koundinya Veluri	Feb 2, 2013
Little Wing       Oct 26, 2017         Luca Hemmerich       Jun 19, 2017         Mac OS       Mar 25, 2016         Martin Sedlak       Dec 25, 2017         Mathematician       Feb 28, 2018         Matl Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael Dvorkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolass de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016	Kurt	Apr 20, 2014
Luca Hemmerich       Jun 19, 2017         Mac OS       Mar 25, 2016         Martin Sediak       Dec 25, 2017         Mathematician       Feb 28, 2018         Matt Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael Dvorkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolass de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016	<u>Leila</u>	May 8, 2017
Mac OS       Mar 25, 2016         Matrin Sediak       Dec 25, 2017         Mathematician       Feb 28, 2018         Matt Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael Dvorkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolass de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016		
Martin Sediak         Dec 25, 2017           Mathematician         Feb 28, 2018           Matt Taylor         Feb 17, 2013           Matthew R. Brades         Jun 9, 2017           Michael Dvorkin         Jan 8, 2016           Mikhail R. Shura-Bura         Oct 30, 2013           Murka         Nov 11, 2016           Nathan Thom         Mar 15, 2013           Nicolass de Bruijn         Feb 3, 2015           NoraGrace         Nov 23, 2014           Obstruction Difference         May 27, 2016		
Mathematician       Feb 28, 2018         Matt Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael Dvorkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolaas de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016		
Matt Taylor       Feb 17, 2013         Matthew R. Brades       Jun 9, 2017         Michael Dvorkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolaas de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016		
Matthew R. Brades       Jun 9, 2017         Michael Dvorkin       Jan 8, 2016         Mikhail R. Shura-Bura       Oct 30, 2013         Murka       Nov 11, 2016         Nathan Thom       Mar 15, 2013         Nicolaas de Bruijn       Feb 3, 2015         NoraGrace       Nov 23, 2014         Obstruction Difference       May 27, 2016		
Michael Dvorkin         Jan 8, 2016           Mikhail R. Shura-Bura         Oct 30, 2013           Murka         Nov 11, 2016           Nathan Thom         Mar 15, 2013           Nicolaas de Bruijn         Feb 3, 2015           NoraGrace         Nov 23, 2014           Obstruction Difference         May 27, 2016		
Mikhail R. Shura-Bura         Oct 30, 2013           Murka         Nov 11, 2016           Nathan Thom         Mar 15, 2013           Nicolass de Bruijn         Feb 3, 2015           NoraGrace         Nov 23, 2014           Obstruction Difference         May 27, 2016		
Murka         Nov 11, 2016           Nathan Thom         Mar 15, 2013           Nicolaas de Bruijn         Feb 3, 2015           NoraGrace         Nov 23, 2014           Obstruction Difference         May 27, 2016		
Nicolaas de Bruijn         Feb 3, 2015           NoraGrace         Nov 23, 2014           Obstruction Difference         May 27, 2016		
NoraGrace         Nov 23, 2014           Obstruction Difference         May 27, 2016	Nathan Thom	Mar 15, 2013
Obstruction Difference May 27, 2016	Nicolaas de Bruijn	Feb 3, 2015
	NoraGrace	Nov 23, 2014
Othello	Obstruction Difference	May 27, 2016
		Jan 4, 2018
Pablo San Segundo Feb 25, 2015		
Paladin Jan 29, 2017		
Pascal Georges Jun 1, 2014		
PIC Microcontroller         Oct 27, 2017           Pieces versus Directions         Oct 6, 2016		
Pieces versus Directions         Oct 6, 2016           Population Count         Sep 3, 2017		
PowerPC Oct 6, 2017		
Prophet Sep 30, 2017		
Ramón Galán Feb 25, 2015		
RedQueen Nov 13, 2017		
Reinhard Scharnagl		
Reverse Bitboards Aug 29, 2015		
Robert Hyatt Dec 25, 2017	Robert Hyatt	Dec 25, 2017
Robocide May 11, 2016	Robocide	May 11, 2016
SBAMG Dec 4, 2016	<u>SBAMG</u>	Dec 4, 2016

Page	Date Edited
Scoping It's time for us to say farewell Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wi	Mar 28, 2018, sispaces.com)
Searcher	Sep 26, 2016
Senpai	Nov 10, 2017
Severi Salminen	May 7, 2015
Shifted Bitboards	Jan 9, 2011
Sliding Piece Attacks	May 27, 2016
Space-Time Tradeoff	Jun 17, 2015
Spector	Nov 11, 2016
SSE4	Jun 5, 2016
<u>Stef Luijten</u>	Apr 26, 2015
Stefan Plenkner	Dec 31, 2015
Stockfish	Mar 10, 2018
<u>Sune Fischer</u>	Apr 7, 2014
Sungarus	Apr 11, 2014
Syed Fahad	Jan 1, 2017
Thor's Hammer	Nov 23, 2013
Traversing Subsets of a Set	Oct 14, 2016
Tucano	Dec 16, 2017
Tunguska	Sep 16, 2017
<u>Vadim Demichev</u>	Jul 26, 2013
Vice	Mar 8, 2016
Walter Faxon	Mar 6, 2015
Warrior	Feb 23, 2015
Wasp	Nov 24, 2017
X-ray Attacks (Bitboards)	Mar 31, 2015
<u>x86</u>	Jan 4, 2018
<u>x86-64</u>	Mar 6, 2018
x86-64 Instructions to Include	Feb 12, 2011
Zurichess	Mar 12, 2018

Up one Level



bitscan broken on wiki page (Walter Faxon bitscan) ew/EricMullins) Nov 5, 2009

I recently got interested in bitscans due to porting RobboLito to my router. There weren't any scans in the source that worked, so I found my way

In testing various versions, I discovered the Walter Faxon bitscan doesn't return correct results, at least the version presented here. I kind of left it alone after that but got re-interested in it and found my way to this page

 $http://www.stmintz.com/ccc/index.php?id=265635 \ (http://www.stmintz.com/ccc/index.php?id=265635)$ 

That version gave correct results, and was quite fast in my testing. Scrutinizing the differences, I determined the comment from that page about ommitting a line to retain the LSB was wrong. That line \*must\* be included to give correct results. It may be possible to save cyles by changing the routine to preserve the LSB, but the line cannot simply be removed without causing an incorrect result.

Also, the version at the above link uses unsigned char for a reason. That typedef/cast was necessary in my tests.

(https://www.wikispaces.com/user/view/GerdIsenberg) GerdIsenberg (https://www.wikispaces.com/user/view/GerdIsenberg) Nov 5, 2009
Thanks Eric for pointing that out. I will be corrected immediately.

(https://www.wikispaces.com/user/view/GerdIsenberg) GerdIsenberg (https://www.wikispaces.com/user/view/GerdIsenberg) Nov 6, 2009

We can still omit the line to retain the LSB: bb ^ (bb-1)

results in all bits below LS1B including it. While bb = (bb-1) ^ (bb & (bb-1)); results in all bits below LS1B excluding LS1B, thus only a cyclic index translation, therefor the decremented indices in LSB\_64\_table where 0 became 63.

(https://www.wikispaces.com/user/view/Pradu)

Asserts Pradu (https://www.wikispaces.com/user/view/Pradu) Jun 4, 2008

Would it be better to place asserts after variable declarations? This way the routines would also compile with C compilers.

[https://www.wikispaces.com/user/view/GerdIsenberg) GerdIsenberg (https://www.wikispaces.com/user/view/GerdIsenberg) Jun 5, 2008 Ok, feel free to change it.

(https://www.wikispaces.com/user/view/Pradu)



Walter Faxon bitscan

Pradu (https://www.wikispaces.com/user/view/Pradu) Jun 2, 2008

t64 undeclared



[https://www.wikispaces.com/user/view/GerdIsenberg] GerdIsenberg (https://www.wikispaces.com/user/view/GerdIsenberg) Jun 4, 2008 oups should be bb

Hilfe - Über - Preisliste - Privatsphäre - Bedingungen - Unterstützung - Höherstufen

Contributions to https://chessprogramming.wikispaces.com/ are licensed under a Creative Commons Attribution Share-Alike 3.0 License. ((x)) Vosans

V It's time for us to say farewell... Regretfully, we've made the tough decision to close Wikispaces. Find out why, and what will happen, here (http://blog.wikispace83667) not contributed by visitors are Copyright 2018 Tanglent LLC

TES: The largest network of teachers in the world