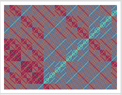
Population Count (/Population+Count)


Visualization of Hamming distance [8]

Population count determines the cardinality of a bitboard, also called Hamming weight or sideways sum [1]. How many one bits exists in a 64-bit computer word? In computer chess, population count is used to evaluate the mobility of pieces from their attack sets, as already applied in Chess 4.6 on the CDC 6600 and CDC Cyber.

Future or recent x86-64 processors (AMD K10 - SSE4a, Intel Nehalem - SSE4.2) provide a 64-bit popcount instruction [2], available via C++ compiler intrinsic [3][4][5][6] or inline assembly [7]. Despite different Intrinsic prototypes (_mm_popcnt_u64 vs. popcnt64), Intel and AMD popcnt instructions are binary compatible, have same encoding (F3 [REX] 0F B8 /r), and both require bit 23 set in RCX of the CPUID function 0000_0001h.

Table of Contents

Code samples in C / C++, see Defining Bitboards

Recurrence Relation

The recursive recurrence relation of population counts can be transformed to iteration as well, but lacks an arithmetical sum-formula:

$$popcnt(0) = 0$$
$$popcnt(n) = popcnt(n \div 2) + (n \mod 2)$$

However, it is helpful to initialize a lookup table, i.e. for bytes:

```
unsigned char popCountOfByte256[];

void initpopCountOfByte256()
{
   popCountOfByte256[0] = 0;
   for (int i = 1; i < 256; i++)
      popCountOfByte256[i] = popCountOfByte256[i / 2] + (i & 1);
}
```

Empty or Single?

Often one has to deal with sparsely populated or even empty bitboards. To determine whether a bitboard is empty or a single populated power of two value, one may use simple boolean statements rather than a complete population count.

Empty Bitboards

To test a bitboard is empty, one compares it with zero, or use the logical not operator:

```
if ( x == 0 ) -> bitboard is empty
if ( !x )     -> bitboard is empty
```

The inverse condition (not empty) is of course

```
if ( x != 0 ) -> bitboard is not empty
if ( x )      -> bitboard is not empty
```

Single Populated Bitboards

If the bitboard is not empty, we can extract the LS1B to look whether it is equal with the bitboard. Alternatively and faster, we can reset the LS1B to look whether the bitboard becomes empty.

```
if ( x != 0 && (x & (x-1)) == 0 ) -> population count is one, power of two value
```

One can skip the leading x != 0 condition to test popcount <= 1:

```
if ( (x & (x-1)) == 0 ) -> population count is less or equal than one
```

Again the inverse relation tests, whether a bitboard has more than one bit set:

```
if ( x & (x-1) ) -> population count is greater than one
```

An alternative approach to determine single populated sets, aka power of two values is based on Inclusive LS1B separation divided by two equals the ones' decrement [9]:

```
if ( ((x ^ (x-1)) >> 1) == (x-1) ) -> population count is one, power of two value
```

Loop-Approaches

Too Slow

Brute force adding all 64-bits

```
int popCount (U64 x) {
   int count = 0;
   for (int i = 0; i < 64; i++, x >>= 1)
      count += (int)x & 1;
   return count;
}
```

Of course, this is a slow algorithm, which might be improved by testing x not empty rather than i < 64. But unrolled in parallel prefix manner it already reminds on this one.

Brian Kernighan's way

Consecutively reset LS1B in a loop body and counting loop cycles until the bitset becomes empty. Brian Kernighan [10] mentioned the trick in his and Ritchie's book The C Programming Language, 2nd Edition 1988, exercise 2-9. However, the method was first published in 1960 by Peter Wegner [11], discovered independently by Derrick Henry Lehmer, published in 1964 [12]:

```
int popCount (U64 x) {
   int count = 0;
   while (x) {
       count++;
       x &= x - 1; // reset LS1B
   }
   return count;
}
```

Despite branches - this is still one of the fastest approaches for sparsely populated bitboards. Of course the more bits that are set, the longer it takes.

Lookup

Of course we can not use the whole bitboard as index to a lookup table - since it's size would be 18,446,744,073,709,551,616 bytes! If it is about the population count of a byte, we can simply construct a table lookup with 256 elements. For a bitboard that takes eight byte lookups we can add together:

```
unsigned char popCountOfByte256[];

void initpopCountOfByte256()
{
   popCountOfByte256[0] = 0;
   for (int i = 1; i < 256; i++)
      popCountOfByte256[i] = popCountOfByte256[i / 2] + (i & 1);
}

int popCount (U64 x) {
   return popCountOfByte256[ x        & 0xff] +
          popCountOfByte256[(x >>  8) & 0xff] +
          popCountOfByte256[(x >> 16) & 0xff] +
          popCountOfByte256[(x >> 24) & 0xff] +
          popCountOfByte256[(x >> 32) & 0xff] +
          popCountOfByte256[(x >> 40) & 0xff] +
          popCountOfByte256[(x >> 48) & 0xff] +
          popCountOfByte256[ x >> 56];
}
```

Looks quite expensive - one may use four 16-bit word-lookups with a pre-calculated 64KByte table though, but that pollutes the memory caches quite a bit. One can also treat the bitboard as array of bytes or words in memory, since endian issues don't care here - that safes all the shifts and 'ands', but has to read byte for byte from memory.

```
int popCount (U64 x) {
   unsigned char * p = (unsigned char *) &x;
   return popCountOfByte256[p[0]] +
          popCountOfByte256[p[1]] +
          popCountOfByte256[p[2]] +
          popCountOfByte256[p[3]] +
          popCountOfByte256[p[4]] +
          popCountOfByte256[p[5]] +
          popCountOfByte256[p[6]] +
          popCountOfByte256[p[7]];
}
```

SWAR-Popcount

The divide and conquer SWAR-approach deals with counting bits of duos, to aggregate the duo-counts to nibbles and bytes inside one 64-bit register in parallel, to finally sum all bytes together. According to Donald Knuth [13], a parallel population count routine was already introduced in 1957 due to Donald B. Gillies and Jeffrey C. P. Miller in the first textbook on programming, second edition: *The Preparation of Programs for an Electronic Digital Computer*, by Maurice Wilkes, David Wheeler and Stanley Gill, pages 191–193 [14] [15].

Counting Duo-Bits

A bit-duo (two neighboring bits) can be interpreted with bit 0 = a, and bit 1 = b as

$$duo := 2b + a$$

The duo population is

$$popcnt(duo) := b + a$$

which can be archived by

$$(2b + a) - (2b + a) \div 2$$

or

$$(2b + a) - b$$

The bit-duo has up to four states with population count from zero to two as demonstrated in following table with binary digits:

x	-	x div 2	→	popcnt(x)
00	-	00	→	00
01	-	00	→	01
10	-	01	→	01
11	-	01	→	10

Only the lower bit is needed from x div 2 - and one don't has to worry about borrows from neighboring duos. SWAR-wise, one needs to clear all "even" bits of the div 2 subtrahend to perform a 64-bit subtraction of all 32 duos:

```
x = x - ((x >> 1) & 0x5555555555555555);
```

Note that the popcount-result of the bit-duos still takes two bits.

Counting Nibble-Bits

The next step is to add the duo-counts to populations of four neighboring bits, the 16 nibble-counts, which may range from zero to four. SWAR-wise it is done by masking odd and even duo-counts to add them together:

```
 x = (x & 0x3333333333333333) + ((x >> 2) & 0x3333333333333333);
```

Note that the popcount-result of the nibbles takes only three bits, since 100B is the maximum population (of the nibble 1111B).

Byte-Counts

You already got the idea? Now it is about to get the byte-populations from two nibble-populations. Maximum byte-population of 1000B only takes four bits, so it is safe to mask all those four bits of the sum, rather than to mask the summands:

```
x = (x + (x >> 4)) & 0x0f0f0f0f0f0f0f0f;
```

Adding the Byte-Counts

Parallel Prefix Adds

We may continue with mask-less parallel prefix SWAR-adds for byte-counts, word-counts and finally double-word-counts, to mask the least significant 8 (or 7) bits for final result in the 0..64 range:

```
x += (x >>  8);
x += (x >> 16);
x += (x >> 32);
return x & 255;
```

Multiplication

With todays fast 64-bit multiplication one can multiply the vector of 8-byte-counts with 0x0101010101010101 to get the final result in the most significant byte, which is then shifted right:

```
x = (x * 0x0101010101010101) >> 56;
```

Casting out

Interestingly, there is another approach to add the bytes together. As demonstrated with decimal digits (base 10) and Casting out nines [16], casting out by modulo base minus one is equivalent to taking the digit sum , which might be applied here with low range 0..8 "base 256" digits:

```
x = x % 255;
```

However, since division and modulo are usually slow instructions and modulo by constant is likely replaced by reciprocal multiplication anyway by the compiler, the multiplication by 0x0101010101010101 aka the 2-adic fraction -1/255 is the preferred method.

The PopCount routine

The Constants

Putting all together, the various SWAR-Masks and factors as defined by Donald Knuth as 2-adic fractions [17]:

```
const U64 k1 = C64(0x5555555555555555); /*  -1/3   */
const U64 k2 = C64(0x3333333333333333); /*  -1/5   */
const U64 k4 = C64(0x0f0f0f0f0f0f0f0f); /*  -1/17  */
const U64 kf = C64(0x0101010101010101); /*  -1/255 */
```

represented as bitboards:

```
k1  -1/3            k2  -1/5            k4  -1/17           kf  -1/255
0x5555555555555555  0x3333333333333333  0x0f0f0f0f0f0f0f0f  0x0101010101010101
1 . 1 . 1 . 1 .     1 1 . . 1 1 . .     1 1 1 1 . . . .     1 . . . . . . .
1 . 1 . 1 . 1 .     1 1 . . 1 1 . .     1 1 1 1 . . . .     1 . . . . . . .
1 . 1 . 1 . 1 .     1 1 . . 1 1 . .     1 1 1 1 . . . .     1 . . . . . . .
1 . 1 . 1 . 1 .     1 1 . . 1 1 . .     1 1 1 1 . . . .     1 . . . . . . .
1 . 1 . 1 . 1 .     1 1 . . 1 1 . .     1 1 1 1 . . . .     1 . . . . . . .
1 . 1 . 1 . 1 .     1 1 . . 1 1 . .     1 1 1 1 . . . .     1 . . . . . . .
1 . 1 . 1 . 1 .     1 1 . . 1 1 . .     1 1 1 1 . . . .     1 . . . . . . .
1 . 1 . 1 . 1 .     1 1 . . 1 1 . .     1 1 1 1 . . . .     1 . . . . . . .
```

popCount

This is how the complete routine looks in C:

```
int popCount (U64 x) {
    x =  x       - ((x >> 1)  & k1); /* put count of each 2 bits into those 2 bits */
    x = (x & k2) + ((x >> 2)  & k2); /* put count of each 4 bits into those 4 bits */
    x = (x       +  (x >> 4)) & k4 ; /* put count of each 8 bits into those 8 bits */
    x = (x * kf) >> 56; /* returns 8 most significant bits of x + (x<<8) + (x<<16) + (x<<24) + ...  */
    return (int) x;
}
```

Advantage: no branches, no memory lookups, constant runtime - independent from population
Drawback: dependency chain, not much parallel speedup

slowmul_popCount

And as stated before, for computers with relatively slower multiplication, the addition can be done manually:

```
int slowmul_popCount (U64 x) {
    x =  x       - ((x >> 1)  & k1); /* put count of each 2 bits into those 2 bits */
    x = (x & k2) + ((x >> 2)  & k2); /* put count of each 4 bits into those 4 bits */
```

```
   x = (x       +  (x >> 4)) & k4 ; /* put count of each 8 bits into those 8 bits */
   x += x >>  8;  /* put count of each 16 bits into their lowest 8 bits */
   x += x >> 16;  /* put count of each 32 bits into their lowest 8 bits */
   x += x >> 32;  /* put count of the final 64 bits into the lowest 8 bits */
   return (int) x & 255;
}
```

For likely sparsely populated bitboards, the loop-wise Brian Kernighan's way might be the faster one.

HAKMEM 169

A similar technique was proposed by Bill Gosper et al. from Massachusetts Institute of Technology, as published 1972 in Memo 239 (HAKMEM) [18] [19], to add bit-trio- rather than duo populations consecutively, and the 32 bit version relies on casting out 63. Note that the constants in the code below have octal (base-8) digits, originally written in Assembly for the PDP-6 [20]. An expanded 64-bit version, casting out 511 or 4095, is slightly less efficient than the binary SWAR version above.

```
int hakmem169_32(unsigned int x) {
   x = x  - ((x >> 1)  & 033333333333)
          - ((x >> 2)  & 011111111111);
   x = (x +  (x >> 3)) & 030707070707 ;
   return x % 63; /* casting out 63 */
}
```

Miscellaneous

Cardinality of Multiple Sets

If we like to count arrays of sets, we can reduce 2^N-1 popcounts to N popcounts, by applying the odd-major-trick. For three sets to count we safe one, with five additional cheap instructions.

```
  odd   =  (x ^ y)  ^ z;
  major = ((x ^ y ) & z) | (x & y);

  popCount(x) + popCount(y) + popCount(z) == 2*popCount(major) + popCount(odd)
```

The combined popCount3 likely gains more parallel speedup, since there are two independent chains to calculate. Possible Application is to pass the union of both bishops (since usually bishops cover disjoint sets due to different square colors) plus the up to two knight move-target sets.

```
// return popCount(x) + popCount(y) + popCount(z)
int popCount3 (U64 x, U64 y, U64 z) {
    U64 maj = ((x ^ y ) & z) | (x & y);
    U64 odd = ((x ^ y ) ^ z);
    maj =  maj - ((maj >> 1) & k1 );
    odd =  odd - ((odd >> 1) & k1 );
    maj = (maj & k2) + ((maj >> 2) & k2);
    odd = (odd & k2) + ((odd >> 2) & k2);
    maj = (maj + (maj >> 4)) & k4;
    odd = (odd + (odd >> 4)) & k4;
    odd = ((maj + maj + odd) * kf ) >> 56;
    return (int) odd;
}
```

Odd and Major 7-15

Of course - reducing seven popcount to three, or even 15 popcounts to four sounds even more promising.
For N = 2^n - 1 it takes N - n odd-major pairs. Thus one add-major pair - five instructions - per saved popCount.

That is 7 - 3 = 4 pairs:

```
one1,two1 := oddMaj(x1,x2,x3)
one2,two2 := oddMaj(x4,x5,x6)
ones,two3 := oddMaj(x7,one1,one2)
twos,four := oddMaj(two1,two2,two3)
```

Or 15 - 4 = 11 pairs:

```
one1,two1  := oddMaj(x1,x2,x3)
one2,two2  := oddMaj(x4,x5,x6)
one3,two3  := oddMaj(x7,x8,x9)
one4,two4  := oddMaj(x10,x11,x12)
one5,two5  := oddMaj(x13,x14,x15)
one6,two6  := oddMaj(one1,one2,one3)
ones,two7  := oddMaj(one4,one5,one6)
two8,four1 := oddMaj(two1,two2,two3)
two9,four2 := oddMaj(two4,two5,two6)
twos,four3 := oddMaj(two7,two8,two9)
four,eight := oddMaj(four1,four2,four3)
```

Odd and Major Digit Counts

Odd-Major is probably also useful to determine digit count sets of attacks or other stuff:

```
U64 odd(U64 x, U64 y, U64 z) {return x^y^z;}
U64 maj(U64 x, U64 y, U64 z) {return ((x^y)&z)|(x&y);}

void attackCounts(U64 t[3], const U64 s[7]) {
   one1 = odd(s[0], s[1], s[2]);
   two1 = maj(s[0], s[1], s[2]);
   one2 = odd(s[3], s[4], s[5]);
   two2 = maj(s[3], s[4], s[5]);
   t[0] = odd(s[6], one1, one2);
   two3 = maj(s[6], one1, one2);
   t[1] = odd(two1, two2, two3);
   t[2] = maj(two1, two2, two3);
}
```

with following semantics:

```
exactly7attacks :=   t[2] &  t[1] &  t[0]
exactly6attacks :=   t[2] &  t[1] & ~t[0]
exactly5attacks :=   t[2] & ~t[1] &  t[0]
exactly4attacks :=   t[2] & ~t[1] & ~t[0]
exactly3attacks :=  ~t[2] &  t[1] &  t[0]
exactly2attacks :=  ~t[2] &  t[1] & ~t[0]
exactly1attack  :=  ~t[2] & ~t[1] &  t[0]
```

```
noAttack        :=  ~t[2] & ~t[1] & ~t[0]

atLeast4attacks :=  t[2]
atLeast2attacks := atLeast4attacks | t[1]
atLeast1attack  := atLeast2attacks | t[0]
noAttack        := ~atLeast1attack
exactly1attack  :=  atLeast1attack  ^ atLeast2attacks
```

Popcount as log2 of LS1B
Assuming an architecture has a fast popcount-instruction (but no bitscan). One can isolate the LS1B, decrement it and count the remaining trailing "ones" to perform the logarithm dualis:

```
log2(LS1B) = popCount( LS1B - 1 );
bitIndexOfLS1B(x) = popCount( (x & -x) - 1 );
```

For instance, LS1B is 2^44, decrementing leaves a below LSB1 mask with exactly 44 bits set:

```
0x0000100000000000   0x00000FFFFFFFFFFF
. . . . . . . .      . . . . . . . .
. . . . . . . .      . . . . . . . .
. . . . 1 . . .      1 1 1 1 . . . .
. . . . . . . .      1 1 1 1 1 1 1 1
. . . . . . . .      1 1 1 1 1 1 1 1
. . . . . . . .      1 1 1 1 1 1 1 1
. . . . . . . .      1 1 1 1 1 1 1 1
. . . . . . . .      1 1 1 1 1 1 1 1
```

Hamming Distance
The hamming distance of two words is defined as the number of corresponding different bits.

```
int hammingDistance (U64 a, U64 b) {return popcnt( a ^ b);}
```

Hamming distance greater than one or two is an important property of codes to detect or even correct one-bit errors.

The minimum and average hamming distance over all Zobrist keys was considered as "quality"-measure of the keys. However, as long the minimum hamming distance is greater zero, linear independence (that is a small subset of all keys doesn't xor to zero), is much more important than hamming distance [21]. Maximizing the minimal hamming distance leads to very poor Zobrist keys [22].

Weighted PopCount
For a SIMD-wise kind of weighted population count, see the SSE2 dot-product.

Pre-calculated Mobility
Similar to Attacks by Occupancy Lookup to determine attack sets of sliding pieces, we may use pre-calculated population count or even center-weighted population count as a rough estimate on piece mobility [23]. It may not consider subsets of let say safe target squares.

Piece Attacks Count
As pointed out by Marco Costalba [24] [25], specialized routines to count the population (Mobility) of attack sets of king, knight and line-wise sub-sets of sliding pieces can be done more efficiently than the general SWAR-Popcount. This is similar to Flipping Mirroring and Rotating the whole bitboard versus Rank, File and Diagonal, and is based on mapping the up to eight scattered occupied bits to one byte, to perform a single byte lookup. For various mapping techniques, see:

- Hashing Multiple Bits from Bitboard Serialization
- Rank, File and Diagonal from Flipping Mirroring and Rotating
- Occupancy of any Line

Popcount in Hardware

- Ferranti Mark 1
- CDC 6600
- CDC Cyber
- SSE4.2, Intel x86, x86-64
- SSE4a, AMD x86, x86-64

See also

- Assembly Popcounts
- Bit-Twiddling
- Greater One Sets from General Setwise Operations
- libpopcnt by Kim Walisch
- MMX Popcount
- Mobility in Chess 4.6 on the CDC Cyber
- SIMD and SWAR Techniques
- SSE2 Population Count
- SSSE3 Population Count

Publications

1949 ...

- Alan Turing (**1949**). *Alan Turing's Manual for the Ferranti Mk. I* . transcribed in 2000 by Robert Thau , pdf from The Computer History Museum, 9.4 The position of the most significant digit » Ferranti Mark 1
- Maurice Wilkes, David Wheeler, Stanley Gill (**1957**). *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley Press; 2nd edition, amazon.com , Donald B. Gillies and Jeffrey C. P. Miller on SWAR-Popcount, pages 191–193
- Peter Wegner (**1960**). *A technique for counting ones in a binary computer*. Communications of the ACM, Volume 3, 1960
- David A. Wagner, Steven M. Bellovin (**1994**). *A Programmable Plaintext Recognizer* . [26]

2000 ...

- Simon Y. Berkovich , Gennadi M. Lapir, Marilyn Mack (**2000**). *A Bit-Counting Algorithm Using the Frequency Division Principle* . Software, Practice and Experience Vol. 30, No. 14, 2000, pp. 1531-1540
- Eyas El-Qawasmeh (**2001**). *Beating the Popcount* . International Journal of Information Technology, Singapore, Vol. 9. No. 1
- Henry S. Warren, Jr. (**2002**). *Hacker's Delight* . Addison-Wesley Professional
- Eyas El-Qawasmeh , Wafa'a Al-Qarqaz (**2006**). *Reducing Lookup Table Size used for Bit-Counting Algorithm*. Computer Science Dept. Jordan University of Science and Technology , pdf
- Henry S. Warren, Jr. (**2007**). *The Quest for an Accelerared Population Count*. in Andy Oram & Greg Wilson (eds.) (**2007**). *Beautiful code: Leading Programmers Explain How They Think*. O'Reilly , amazon.com
- Donald Knuth (**2009**). *The Art of Computer Programming* , *Volume 4, Fascicle 1: Bitwise tricks & techniques*, as Pre-Fascicle 1a postscript , Sideways addition, pp 11

2010 ...

- Henry S. Warren, Jr. (**2012**). *Hacker's Delight, 2nd Edition* . Addison-Wesley Professional , More coverage of population count and counting leading zeros, Array population count
- Andreas Stiller (**2013**). *Spezialkommando - Intrinsic __popcnt() zählt die Einsen* . c't Magazin für Computertechnik 5/2013, p. 180 (German)
- Wojciech Muła, Nathan Kurz , Daniel Lemire (**2016**). *Faster Population Counts Using AVX2 Instructions*. arXiv:1611.07612 [27] » AVX2, AVX-512

Postings

1998 ...

- Bean counters Part1 by Peter Fendrich, CCC, August 19, 1998
- Bean counters Part2 by Peter Fendrich, CCC, August 19, 1998
- Countbits() Function by Roberto Waldteufel, CCC, January 03, 1999
- Sideways Add / Population Count by Jitze Couperus , Steve Bellovin and Axel H. Horns , cryptography@c2.net , January 28, 1999 » CDC 6600 [28]

2000 ...

- fast bit counting by Flemming Rodler, CCC, April 19, 2000
- Bit counting revisited by Flemming Rodler, CCC, April 19, 2000
- PowerPC BitCounting Functions Speed by William Bryant, CCC, April 20, 2000 » PowerPC
- Counting the number of bits in a 32-bit word by George Marsaglia, comp.lang.c , December 7, 2000
- Re: Chezzz 1.0.1 - problem solved - for David Rasmussen by David Rasmussen, CCC, February 05, 2003 [29]
- Counting bits by Andreas Herrmann, CCC, April 17, 2003
- Hamming distance and lower hash table indexing by Tom Likens, CCC, September 02, 2003
- PopCount optimization by milix, CCC, March 11, 2004

2005 ...

- Population count in SSE2, again by James Van Buskirk, comp.lang.asm.x86 , April 12, 2008
- core2 popcnt by Frank Phillips, CCC, February 13, 2009
- Piece attacks count by Marco Costalba, CCC, May 18, 2009 » Attack and Defend Maps
- Bit twiddlement question: greater of two popcounts by Zach Wegner, CCC, August 06, 2009

2010 ...

- Stockfish POPCNT support with gcc by Marco Costalba, CCC, January 31, 2010
- Yet another handmade POPCNT by hopcode, comp.lang.asm.x86 , January 05, 2011
- A brief history of the popcnt instruction by Steven Edwards, CCC, March 22, 2011
- Introduction and (hopefully) contribution - bitboard methods by Alcides Schulz, CCC, June 03, 2011 » BitScan
- using Popcount and Prefetch with SSE4 hardware support by Engin Üstün, CCC, May 19, 2012 » Memory, SSE4
- 64 bits for 64 squares ? by Thomas Petzke, mACE Chess , April 28, 2013
- Stockfish 32-bit and hardware instructions on MSVC++ by Syed Fahad, CCC, December 30, 2014 » Stockfish, BitScan, Population Count

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 » BitScan

External Links

- Hamming weight from Wikipedia
- Population count (POPCNT) - CompArch
- Crazy On Tap - Secret Opcodes [30]
- Blender: POPCNT for counting bits
- HAKMEM - ITEM 169 To count the ones in a PDP-6/10 word (in order of one-ups-manship: Gosper, Mann, Lenard, [Root and Mann]) [31]
  HAKMEMC -- HAKMEM Programming hacks in C by Alan Mycroft
- popcount C samples from Henry S. Warren, Jr. (**2002, 2012**). *Hacker's Delight*. Addison-Wesley
- The Aggregate Magic Algorithms -Population Count (Ones Count) by Hank Dietz
- Counting bits set from Bit Twiddling Hacks by Sean Eron Anderson
- Optimising Bit Counting using Iterative, Data-Driven Development from Necessary and Sufficient by Damien Wintour
- Count bits set in parallel a.k.a. Population Count from the bit twiddler by Stephan Brumme
- Benchmarking CRC32 and PopCnt instructions - strchr.com by Peter Kankowski
- SSSE3: fast popcount by Wojciech Muła, May 24, 2008 » SSSE3
- Speeding up bit-parallel population count by Wojciech Muła, April 13, 2015
- Population count using XOP instructions by Wojciech Muła, December 16, 2016 » XOP
- GitHub - WojciechMula/sse-popcount: SIMD (SSE) population count by Wojciech Muła
- GitHub - kimwalisch/libpopcnt: Fast C/C++ bit population count library » libpopcnt by Kim Walisch
- Census from Wikipedia
- John Abercrombie 4tet - One, one, one + Spring song, Subway , Cologne , April 12, 1999, 3sat broadcast [32], YouTube Video
  John Abercrombie, Bobo Stenson , Lars Danielsson, Jon Christensen

John Abercrombie Quartet - Köln (Cologne), Germany, 1999-04-12

References

1. ^ Cryptography is also a significant application of the /R function symbol, which counts the number of one bits in a word; Turing refers to this as the "sideways adder" in his quick-reference summary. from Alan Turing (**1949**). *Alan Turing's Manual for the Ferranti Mk. I* . transcribed in 2000 by Robert Thau , pdf from The Computer History Museum, 9.4 The position of the most significant digit » Ferranti Mark 1
2. ^ Extending the World's Most Popular Processor Architecture by R.M. Ramanathan, Intel, covers SSE4 and popcnt
3. ^ _mm_popcnt_u64
4. ^ _popcnt16, _popcnt, _popcnt64 C-Intrinsic MSDN Library
5. ^ Miscellaneous Intrinsic
6. ^ __builtin_popcountll GCC Intrinsic
7. ^ Stockfish POPCNT support with gcc by Marco Costalba, CCC, January 31, 2010
8. ^ Color of each pixel is Hamming distance between the binary representations of its x and y coordinates, modulo 16, in the 16-color system, by Josiedraus, June 8, 2007, Richard Hamming from Wikipedia
9. ^ Matters Computational - ideas, algorithms, source code (pdf) Ideas and Source Code by Jörg Arndt, 1.7 Functions related to the base-2 logarithm, function one_bit_q(), pp 18
10. ^ Counting bits set, Brian Kernighan's way from Bit Twiddling Hacks by Sean Eron Anderson
11. ^ Peter Wegner (**1960**). *A technique for counting ones in a binary computer*. Communications of the ACM, Volume 3, 1960
12. ^ Edwin Ford Beckenbach (editor) (**1964**). *Applied combinatorial mathematics* , John Wiley
13. ^ Donald Knuth (**2009**). *The Art of Computer Programming* , *Volume 4, Fascicle 1: Bitwise tricks & techniques*, as Pre-Fascicle 1a postscript , Sideways addition, p 11
14. ^ Maurice Wilkes, David Wheeler, Stanley Gill (**1951**). *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley Press; 1st edition, amazon.com ; 2nd edition 1957, amazon.com
15. ^ Electronic Delay Storage Automatic Calculator from Wikipedia
16. ^ Casting Out Nines by Bill the Lizard , June 13, 2009
17. ^ Donald Knuth (**2009**). *The Art of Computer Programming* , *Volume 4, Fascicle 1: Bitwise tricks & techniques*, as Pre-Fascicle 1a postscript , p 9
18. ^ HAKMEM - ITEM 169 To count the ones in a PDP-6/10 word (in order of one-ups-manship: Gosper, Mann, Lenard, [Root and Mann])
19. ^ HAKMEMC -- HAKMEM Programming hacks in C by Alan Mycroft
20. ^ HAKMEM 169 for PDP-6/PDP-10 36-bit words
21. ^ Re: About random numbers and hashing by Sven Reichard, CCC, December 05, 2001
22. ^ Zobrist key random numbers by Robert Hyatt from CCC, January 21, 2009
23. ^ Magic and precomputation by Onno Garms from Winboard Programming Forum, September 23, 2007
24. ^ fast mobility count through hashing by Marco Costalba from CCC, May 09, 2009
25. ^ Piece attacks count by Marco Costalba from CCC, May 18, 2009
26. ^ Sideways Add / Population Count by Jitze Couperus and Steve Bellovin et al., cryptography@c2.net , January 28, 1999
27. ^ sse-popcount/popcnt-avx512-harley-seal.cpp at master · WojciechMula/sse-popcount · GitHub
28. ^ David A. Wagner, Steven M. Bellovin (**1994**). *A Programmable Plaintext Recognizer* .
29. ^ AMD Athlon Processor x86 Code Optimization Guide (pdf) Efficient 64-Bit Population Count Using MMX™ Instructions Page 184
30. ^ National Security Agency from Wikipedia
31. ^ Michael Beeler, Bill Gosper and Rich Schroeppel (**1972**). *HAKMEM* , Memo 239, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, made Web-available by Henry Baker
32. ^ Ali Haurand from Wikipedia.de (German)

What links here?

Page	Date Edited

Page	Date Edited
Alcides Schulz	Mar 29, 2018
Algorithms	May 5, 2017
Amundsen	Sep 3, 2013
Anastasios Milikas	Nov 29, 2017
Andreas Herrmann	Nov 7, 2014
Assembly	Sep 3, 2017
Asterisk	Mar 24, 2016
Attack and Defend Maps	Nov 5, 2016
AVX-512	Aug 8, 2017
AVX2	Aug 8, 2017
Beaches	Jun 10, 2017
Best Magics so far	Sep 17, 2017
Bill Gosper	Aug 20, 2014
Bison	Sep 29, 2016
Bit-Twiddling	Nov 6, 2017
Bitboard Serialization	Dec 24, 2014
Bitboards	Nov 14, 2017
BitScan	Sep 10, 2017
BlackMamba	Nov 26, 2016
Blockage Detection	Oct 19, 2017
Bobcat	Jun 27, 2017
Bouquet	May 17, 2016
Brainless	Jun 24, 2017
Cassandre	Jul 5, 2013
CDC 6600	Oct 8, 2014
CDC Cyber	Dec 22, 2017
Cheng	Jul 30, 2017
Chess 0.5	Nov 20, 2016
Chezzz	Jan 20, 2013
Congruent Modulo Bitboards	Jun 26, 2013
CookieCat	Nov 15, 2016
Cray-1	Dec 25, 2017
David Rasmussen	Dec 16, 2017
DEC Alpha	Aug 15, 2015
DirGolem	Jun 5, 2016
Dispersion and Distortion	Nov 11, 2017
Djinn	Feb 8, 2016
Double and Triple (Bitboards)	May 12, 2016
Engin Üstün	Jan 21, 2018
Ferranti Mark 1	Jun 2, 2015
Fizbo	Dec 22, 2017
Frank Phillips	Sep 26, 2016
General Setwise Operations	Feb 25, 2018
Georg von Zimmermann	May 29, 2017
Gk	Oct 9, 2017
Hakkapeliitta	Apr 26, 2016
HeavyChess	Feb 14, 2014
Henry S. Warren, Jr.	Oct 14, 2016
Ifrit	Feb 7, 2016
Itanium	Aug 29, 2015
Joker IT	Sep 16, 2017
Kim Walisch	Aug 10, 2017
Leila	May 8, 2017
Little Wing	Oct 26, 2017
Mac OS	Mar 25, 2016
Marco Costalba	Feb 28, 2018
Material Tables	May 5, 2017
Mathematician	Feb 28, 2018
Michael Dvorkin	Jan 8, 2016
Mikhail R. Shura-Bura	Oct 30, 2013
MMX	Jun 5, 2016
Mobility	Jan 17, 2018
Nebula	Feb 7, 2015
Nibble	Jan 25, 2015
Paladin	Jan 29, 2017
Parallel Prefix Algorithms	Jun 22, 2016
Pawn Islands (Bitboards)	May 4, 2017
PDP-10	Jan 19, 2018
PDP-6	Jan 19, 2018
Peter Fendrich	May 19, 2017
Population Count	Sep 3, 2017
PowerPC	Oct 6, 2017
Prophet	Sep 30, 2017

Page	Date Edited
Recursion	Nov 18, 2017
RedQueen	Nov 13, 2017
Robert Hyatt	Dec 25, 2017
Robocide	May 11, 2016
Senpai	Nov 10, 2017
SIMD and SWAR Techniques	Jun 27, 2017
Simona Tancig	Nov 7, 2012
Space-Time Tradeoff	Jun 17, 2015
Spector	Nov 11, 2016
SSE2	Feb 27, 2018
SSE4	Jun 5, 2016
SSSE3	Aug 8, 2017
Steven Edwards	Aug 26, 2017
Steven M. Bellovin	Dec 14, 2017
Stockfish	Mar 10, 2018
Sungorus	Apr 11, 2014
Syed Fahad	Jan 1, 2017
Texel	Oct 9, 2017
Tornado	Dec 14, 2017
Transposition Table	Feb 19, 2018
Tucano	Dec 16, 2017
Vadim Demichev	Jul 26, 2013
Vice	Mar 8, 2016
Warrior	Feb 23, 2015
Wasp	Nov 24, 2017
Wojciech Muła	Aug 10, 2017
x86-64	Mar 6, 2018
x86-64 Instructions to Include	Feb 12, 2011
Zobrist Hashing	Jan 22, 2018

Up one Level

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

Odd-Major Trick
Pradu (https://www.wikispaces.com/user/view/Pradu) Nov 14, 2007

Very nice trick! Should speedup Buzz a bit with my lousy single-parameter popcounts. :)

(https://www.wikispaces.com/user/view/GerdIsenberg) GerdIsenberg (https://www.wikispaces.com/user/view/GerdIsenberg) *Nov 14, 2007*
Hi Pradu,

I accidently unmade your last changes, while adding an anchor.
Still not too familar with this wiki-stuff.
Can you please retore it again?

Sorry,
Gerd

(https://www.wikispaces.com/user/view/Pradu) Pradu (https://www.wikispaces.com/user/view/Pradu) *Nov 14, 2007*
Ok, I'll have it done in a few minutes.