中道天狗 - *NAKAMICHI 'Tengu' a.k.a. 'Skydog'*

Meteoric LZSS decompressor: www.sanmayce.com/Nakamichi/index.html

---

My dream to sidekick English language usage by offering 100% FREE rip/search/decompress tools continues.
Sadly, the status quo is ugly, nearly 2,200,000,000 people use English and what FREE/REAL support do they have?! Had I been an alien assessing the situation on Earth I would have said - 'WHAT A SHITTY STALEMATE.'
As far as I can see the Earth is plagued by blood-sucking hypocrites who have built many a sect disguised as EDUCATIONAL. One superb song goes: "… THE VIBE IS WRONG … YOU KEEP YOUR **LOVE LOCKDOWN** YOU LOSE …"
The pimps that sell EDUCATION make me sick.
Early this year while writing 'Gallowwalker' I had been obsessed by one notion 'coming operational of a sidekick/fulcrum' - a visual tool for textual support offering an easy-to-use superfast word/phrase searching/suggesting/checking.

...
You know, I really don't know much about my mother.
I remember her drinking a lot and always angry and fighting.
I knew she had dreams of becoming a schoolteacher.
But then she met my father...
well, the man I was told was my father.
The fast-talking, cool-dressing pimp who I always credited with changing the path of my mother's life.
And before long, she was caught up in the street life.
But she paid the heavy toll because at heart she really wasn't that girl at all.
So she drank to cover up the pain.
...
You know, I have, like, a 9th grade education from a really, really fucked up, bad school, right?
And all my kids go to private schools and Ivy League schools, right?
And every now and then I text them and I see how they're doing, and they call me back... text me back and say,
"Daddy, this is not how you spell my name, Daddy.
And you don't spell 'birthday' like this, Dad.
I can't believe you can't spell 'birthday'."
...
/From the 'Mike Tyson - Undisputed Truth'/

I won't be surprised at all if 1,200,000,000 out of 2,200,000,000 would misspell 'birthday'. The shame lies not in people erring but in JOIN-FORCES behind all this SICKNESS - keeping LITERACY out of reach.
Free access to English texts is quite as the right for happiness for EVERYONE as stated in constitution, that's right.
I see no genuine will whatsoever to help the fellow man, only business and "PROFESSIONALISM".
The old term 'men of letters' has lost its sacred power in our days, most of the teachers/writers are simply 'moneymakers'.
Or, as one popular song goes 'She don't believe in shooting-stars but she believe in shoes-and-cars'.
My point, this nasty sickness if not ended will spread as in 'RESIDENT EVIL' hurting ocean of souls.
Old thinking has to go, the tomorrow that never comes, in my view, is now.
The topic is vast, but as *Babaji* says 'One spark is enough.'
The want (not merely need) for more purer English texts corpus resulted in 'Autobiography_411-ebooks_Collection.tar' corpus:

```
273,401,856 Autobiography_411-ebooks_Collection.tar
117,126,206 Autobiography_411-ebooks_Collection.tar.lz4                 ! -9 !
115,558,538 Autobiography_411-ebooks_Collection.tar.v1.2_19.lzt         ! -19 !
113,511,873 Autobiography_411-ebooks_Collection.tar.Z
107,237,997 Autobiography_411-ebooks_Collection.tar.Tengu.Nakamichi
102,514,628 Autobiography_411-ebooks_Collection.tar.lzh                 / LHA32 version 2.67 /
101,966,054 Autobiography_411-ebooks_Collection.tar.method1.zpaq        / zpaq 6.50 /
 97,569,200 Autobiography_411-ebooks_Collection.tar.zip                 ! -tzip -mx9 !
 87,439,698 Autobiography_411-ebooks_Collection.tar.method2.zpaq        / zpaq 6.50 /
 82,724,804 Autobiography_411-ebooks_Collection.tar.sr2
 77,194,175 Autobiography_411-ebooks_Collection.tar.lzhds.nz            ! -cD !
 75,388,190 Autobiography_411-ebooks_Collection.tar.ST3_block256.bsc    ! -m0 -Tt -b256 !
 69,832,119 Autobiography_411-ebooks_Collection.tar.7z                  ! -t7z -mx9 !
 69,398,819 Autobiography_411-ebooks_Collection.tar.v1.2_39_block256.lzt ! -39 -b256 -p1 !
 67,946,358 Autobiography_411-ebooks_Collection.tar.v1.2_49_block256.lzt ! -49 -b256 -p1 !
 66,517,316 Autobiography_411-ebooks_Collection.tar.order04.PPMonstr    ! -m1024 -o4 !
 65,416,196 Autobiography_411-ebooks_Collection.tar.ST4_block256.bsc    ! -m1 -Tt -b256 !
 61,018,244 Autobiography_411-ebooks_Collection.tar.ST5_block256.bsc    ! -m2 -Tt -b256 !
 59,401,801 Autobiography_411-ebooks_Collection.tar.method3.zpaq        / zpaq 6.50 /
 58,736,456 Autobiography_411-ebooks_Collection.tar.order06.PPMonstr    ! -m1024 -o6 !
 58,266,061 Autobiography_411-ebooks_Collection.tar.tangelo             / Version 2.3 /
 57,962,141 Autobiography_411-ebooks_Collection.tar.method4.zpaq        / zpaq 6.50 /
 56,862,830 Autobiography_411-ebooks_Collection.tar.bbb                 ! cfm128 !
 55,962,342 Autobiography_411-ebooks_Collection.tar.order08.PPMonstr    ! -m1024 -o8 !
 55,464,039 Autobiography_411-ebooks_Collection.tar.BWT_block256.bsc    ! -m3 -Tt -b256 !
 55,117,057 Autobiography_411-ebooks_Collection.tar.method5.zpaq        / zpaq 6.50 /
 55,011,526 Autobiography_411-ebooks_Collection.tar.order32.PPMonstr    ! -m1024 -o32 !
 54,635,921 Autobiography_411-ebooks_Collection.tar.order16.PPMonstr    ! -m1024 -o16 !
 52,631,060 Autobiography_411-ebooks_Collection.tar.cm.nz               ! -cc !
 48,194,287 Autobiography_411-ebooks_Collection.tar.paq8hp12            ! -8 !
```

```
D:\_KAZE\DDETT>Nakamichi_Tengu_GP_64bit.exe Autobiography_411-ebooks_Collection.tar.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2
enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 107237997 bytes ...
RAM-to-RAM performance: 576 MB/s.
Memory pool starting address: 0000000006BB0080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 280989 clocks or 1.866MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 30%

D:\_KAZE\DDETT>lz4 -9 -Sx -b -T1 Autobiography_411-ebooks_Collection.tar
Nb of threads = 1 ; Compression Level = 9
Autobiography_4 : 273401856 -> 117125927 ( 42.84%),   10.2 MB/s ,  710.8 MB/s

D:\_KAZE\DDETT>Yappy_32bit.exe Autobiography_411-ebooks_Collection.tar 65536 999
YAPPY: [b 64K] bytes 273401856 -> 158222198  57.9%  comp  27.3 MB/s  uncomp 437.5 MB/s

D:\_KAZE\DDETT>Yappy_32bit.exe Autobiography_411-ebooks_Collection.tar 1048576 999
YAPPY: [b 1024K] bytes 273401856 -> 156590246  57.3%  comp  27.0 MB/s  uncomp 434.6 MB/s
```

Above benchmark is for my laptop with T7500 2200MHz, the goal is to open/traverse 'transparently' i.e. 'on the fly' 2,000,000+ ebooks, not to speak of epapers/emagazines. Too many times I see empty/ununderstanding eyes while I mention 'greed-for-speed' instead of 'need-for-speed', see for yourself: Pure text (EPUB2TXT): 411 ebooks ~ 260MB, then roughly 2,000,000/411x260 = 1,265,206MB or 1.2TB!
Now, 273,401,856:107,237,997 = 2.5:1 i.e. 1.2TB:491GB, if we have 512GB m.2 SSD with 1GB/s read and Haswell (it offers 1500+MB/s decompression speed with 'Tengu') then we are nearing one good scenario with potential.

My first attempt ('DDETT' corpus) to offer solid ground for benchmarking is good enough, however not purely bookish. This textual corpus, built from contemporary English style prose, having unique 411 files fits ultrawell in heavy English texts benchmarking. The corpus contains the TXT format of EPUB sources, 'calibre' by *Kovid Goyal* was used as convertor, great tool! Some 84 out of 411 covers are shown at right.

2014-Oct-07, *Machinely yours 'Kaze'.*

中道天狗 ~ *NAKAMICHI 'Tengu' a.k.a. 'Skydog'*

Meteoric LZSS decompressor: www.sanmayce.com/Nakamichi/index.html

Final speed showdown on laptop with Core 2 Q9550s 2.83GHz:

```
10/05/2014  12:54 AM    273,401,856 Autobiography_411-ebooks_Collection.tar
10/08/2014  05:43 PM    117,126,206 Autobiography_411-ebooks_Collection.tar.lz4
10/08/2014  06:10 PM    115,586,097 Autobiography_411-ebooks_Collection.tar.lzt
10/05/2014  09:12 AM    107,237,997 Autobiography_411-ebooks_Collection.tar.Nakamichi
10/08/2014  06:37 PM  1,082,907,648 DDETT-Definitive_Decompression_English_Texts_Torture.tar
10/08/2014  06:39 PM    386,308,041 DDETT-Definitive_Decompression_English_Texts_Torture.tar.lz4
10/08/2014  06:48 PM    380,665,413 DDETT-Definitive_Decompression_English_Texts_Torture.tar.lzt
10/04/2014  11:00 PM    359,115,942 DDETT-Definitive_Decompression_English_Texts_Torture.tar.Nakamichi
10/05/2014  01:16 AM     42,874,368 Dean_Koontz_71-ebooks_Collection.tar
10/08/2014  05:43 PM     18,457,740 Dean_Koontz_71-ebooks_Collection.tar.lz4
10/08/2014  06:33 PM     18,189,111 Dean_Koontz_71-ebooks_Collection.tar.lzt
10/05/2014  10:23 AM     16,529,379 Dean_Koontz_71-ebooks_Collection.tar.Nakamichi
10/03/2014  01:15 PM  1,000,000,000 enwik9
10/03/2014  01:40 PM    374,905,569 enwik9.lz4
10/03/2014  01:48 PM    371,915,871 enwik9.lzt
10/07/2014  11:35 AM    376,138,043 enwik9.Nakamichi
10/05/2014  12:59 AM    811,308,544 For_Dummies_978-ebooks_Collection.tar
10/08/2014  05:43 PM    285,866,635 For_Dummies_978-ebooks_Collection.tar.lz4
10/08/2014  05:52 PM    281,833,812 For_Dummies_978-ebooks_Collection.tar.lzt
10/06/2014  07:33 AM    269,699,818 For_Dummies_978-ebooks_Collection.tar.Nakamichi
08/05/2014  03:33 AM    846,351,894 Kazahana_on.PAGODA-order-5.txt
10/08/2014  05:42 PM    108,865,183 Kazahana_on.PAGODA-order-5.txt.lz4
10/08/2014  06:05 PM    107,077,360 Kazahana_on.PAGODA-order-5.txt.lzt
10/07/2014  08:08 PM    140,042,742 Kazahana_on.PAGODA-order-5.txt.Nakamichi
10/04/2014  07:30 PM     74,104,320 Zombie_200-ebooks_Collection.tar
10/08/2014  05:41 PM     30,569,090 Zombie_200-ebooks_Collection.tar.lz4
10/08/2014  06:35 PM     30,099,147 Zombie_200-ebooks_Collection.tar.lzt
10/05/2014  01:42 PM     27,645,384 Zombie_200-ebooks_Collection.tar.Nakamichi
```

Grrrbl!

Grrrbl!

```
C:\Tengu>lz4 -9 -Sx -b -T1 For_Dummies_978-ebooks_Collection.tar
Nb of threads = 1 ; Compression Level = 9
For_Dummies_978 : 811308544 -> 285865844 ( 35.24%),   14.6 MB/s , 968.9 MB/s
C:\Tengu>Nakamichi_Tengu_GP_64bit.exe For_Dummies_978-ebooks_Collection.tar.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2
enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 269699818 bytes ...
RAM-to-RAM performance: 896 MB/s.
Memory pool starting address: 0000000010630080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 190134 clocks or 2.757MB per
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 32%
```

3.0:1 compression ratio

```
C:\Tengu>lz4 -9 -Sx -b -T1 Autobiography_411-ebooks_Collection.tar
Nb of threads = 1 ; Compression Level = 9
Autobiography_4 : 273401856 -> 117125927 ( 42.84%),   12.7 MB/s , 908.7 MB/s
C:\Tengu>Nakamichi_Tengu_GP_64bit.exe Autobiography_411-ebooks_Collection.tar.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2
enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 107237997 bytes ...
RAM-to-RAM performance: 768 MB/s.
Memory pool starting address: 0000000006A40080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 190040 clocks or 2.759MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 27%
```

```
C:\Tengu>lz4 -9 -Sx -b -T1 Kazahana_on.PAGODA-order-5.txt
Nb of threads = 1 ; Compression Level = 9
Kazahana_on.PAG : 846351894 -> 108864360 ( 12.86%),    9.9 MB/s , 1584.2 MB/s
C:\Tengu>Nakamichi_Tengu_GP_64bit.exe Kazahana_on.PAGODA-order-5.txt.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2
enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 140042742 bytes ...
RAM-to-RAM performance: 1216 MB/s.
Memory pool starting address: 0000000008A70080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 190088 clocks or 2.758MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 44%
```

```
C:\Tengu>lz4 -9 -Sx -b -T1 Dean_Koontz_71-ebooks_Collection.tar
Nb of threads = 1 ; Compression Level = 9
Dean_Koontz_71- :  42874368 ->  18457681 ( 43.05%),   12.3 MB/s , 902.0 MB/s
C:\Tengu>Nakamichi_Tengu_GP_64bit.exe Dean_Koontz_71-ebooks_Collection.tar.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2
enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 16529379 bytes ...
RAM-to-RAM performance: 832 MB/s.
Memory pool starting address: 00000000014F0080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 189463 clocks or 2.767MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 30%
```

```
C:\Tengu>lz4 -9 -Sx -b -T1 enwik9
Nb of threads = 1 ; Compression Level = 9
Not enough memory for 'enwik9' full size; testing 896 MB only...
enwik9          : 939524096 -> 350753087 ( 37.33%),   20.9 MB/s , 1026.4 MB/s
C:\Tengu>Nakamichi_Tengu_GP_64bit.exe enwik9.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2
enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 376138043 bytes ...
RAM-to-RAM performance: 768 MB/s.
Memory pool starting address: 0000000016C30080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 189697 clocks or 2.764MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 27%
```

```
C:\Tengu>lz4 -9 -Sx -b -T1 DDETT-Definitive_Decompression_English_Texts_Torture.tar
Nb of threads = 1 ; Compression Level = 9
Not enough memory for 'DDETT-Definitive_Decompression_English_Texts_Torture.tar' full size; testing 896 MB
only...
DDETT-Definitiv : 939524096 -> 329287352 ( 35.05%),   14.4 MB/s , 982.1 MB/s
C:\Tengu>Nakamichi_Tengu_GP_64bit.exe DDETT-Definitive_Decompression_English_Texts_Torture.tar.Nakamichi
/report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2
enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 359115942 bytes ...
RAM-to-RAM performance: 896 MB/s.
Memory pool starting address: 0000000015A80080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 189432 clocks or 2.768MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 32%
```

Note1: All (except 'DDETT' corpus) .TAR files are made of converted EPUB-to-TXT files.
Note2: At right, 78 out of 978 covers are given, the richest 'DUMMIES' collection so far.
Note3: Obviously 'DDETT' corpus serves well and stands up for its name - same speeds as 'DUMMIES' corpus.

Bottomline: Excellent work by *Yann*, no doubt about it.

2014-Oct-09, *Machinely yours 'Kaze'.*

中道落花 ~ *NAKAMICHI 'Rakka' a.k.a. 'Fallenflower'*

Blossomed LZSS decompressor: www.sanmayce.com/Nakamichi/index.html*

* 1. To come into flower; bloom. 2. To develop; flourish: *The child blossomed into a beauty.* /HERITAGE/

```
unsigned int Decompress_Rakka (char* ret, char* src, unsigned int srcSize) {
            char* retLOCAL = ret; char* srcLOCAL = src; char* srcEndLOCAL = src+srcSize;
            unsigned int DWORDtrio; unsigned int DWORDtrioDumbo;
            while (srcLOCAL < srcEndLOCAL) {
                        DWORDtrio = *(unsigned int*)srcLOCAL;
// |LL|OO|xxxx|xxxxxxxx|xxxxxx|xx|xxxxxx|xx|
// [1bit          16bit]    24bit]    32bit]
// LL = 00b means Long MatchLength, 32>>LL or 32
// LL = 01b means Long MatchLength, 32>>LL or 16
// LL = 10b means Long MatchLength, 32>>LL or 8
// LL = 11b means Long MatchLength, 32>>LL or 4
// LL/OO = 00b/11b means Literal
// LL/OO = 01b/11b UNUSED
// LL/OO = 11b/00b UNUSED
// Four bytes long Sliding Window is limited to 2MB!
// OO = 00b MatchOffset, 0xFFFFFFFF>>OO, 4 bytes long i.e. Sliding Window is 4*8-LL-OO=4*8-4=28 or 256MB
// OO = 01b MatchOffset, 0xFFFFFFFF>>OO, 3 bytes long i.e. Sliding Window is 3*8-LL-OO=3*8-4=20 or 1MB
// OO = 10b MatchOffset, 0xFFFFFFFF>>OO, 2 bytes long i.e. Sliding Window is 2*8-LL-OO=2*8-4=12 or 4KB
// OO = 11b MatchOffset, 0xFFFFFFFF>>OO, 1 byte long  i.e. Sliding Window is 1*8-LL-OO=1*8-4=4 or 16B
                        if ( (DWORDtrio & 0x0F) == 0x0C ) {
                                                #ifndef _N_YMM
memcpy(retLOCAL, (const char *)( (uint64_t)(srcLOCAL+1) ), 16);
                                                #endif
                                                #ifdef _N_YMM
SlowCopy128bit( (const char *)( (uint64_t)(srcLOCAL+1) ), retLOCAL );
                                                #endif
            retLOCAL+= ((DWORDtrio & 0xFF)>>4);
            srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1;
                        } else {
                        DWORDtrioDumbo = ((DWORDtrio & 0x0C)>>2)<<3; // To avoid 'LEA'
                                                #ifndef _N_YMM
memcpy(retLOCAL, (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), 16);
                                                #endif
                                                #ifdef _N_YMM
SlowCopy128bit( (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), retLOCAL );
                                                #endif
                        if ( (DWORDtrio & 0x03) == 0 ) {
                                                #ifndef _N_YMM
memcpy(16+ retLOCAL, (const char *)( 16+ (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), 16);
                                                #endif
                                                #ifdef _N_YMM
SlowCopy128bit( (const char *)( 16+ (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), 16+ retLOCAL );
                                                #endif
                        }
            retLOCAL+= Min_Match_Length>>(DWORDtrio&0x03);
            srcLOCAL+= 4-(DWORDtrioDumbo>>3); // 8*0, 8*1, 8*2, 8*3
                        }
            }
            return (unsigned int)(retLOCAL - ret);
}
```

Results on Core2 T7500 2200MHz:

```
D:\Nakamichi_Rakka>Nakamichi_Rakka_YMMless.exe enwik8.Nakamichi /report
Nakamichi 'Rakka', written by Kaze, based on Nobuo Ito's LZSS source, babealicious
suggestion by m^2 enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 41699410 bytes ...
RAM-to-RAM performance: 448 MB/s.
Memory pool starting address: 0000000002C90080 ... 64 byte aligned, OK
Copying a 256MB block 1024 times i.e. 256GB READ + 256GB WRITTEN ...
memcpy(): (256MB block); 262144MB copied in 139450 clocks or 1.880MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 23%
```

```
; 'Rakka' decompression loop, 99-1e+2=125 bytes long:
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications
; Version 12.1.1.258 Build 20111";
; mark_description "-O3 -QxSSE2 -D_N_YMM -FAcs";
.B8.3::
  0001e 8b 2a           mov ebp, DWORD PTR [rdx]
  00020 89 e8           mov eax, ebp
  00022 83 e0 0f        and eax, 15
  00025 83 f8 0c        cmp eax, 12
  00028 75 1b           jne .B8.5
.B8.4::
  0002a 40 0f b6 ed     movzx ebp, bpl
  0002e f3 0f 6f 42 01  movdqu xmm0, XMMWORD PTR [1+rdx]
  00033 c1 ed 04        shr ebp, 4
  00036 f3 41 0f 7f 02  movdqu XMMWORD PTR [r10], xmm0
  0003b 4c 03 d5        add r10, rbp
  0003e ff c5           inc ebp
  00040 48 03 d5        add rdx, rbp
  00043 eb 51           jmp .B8.8
.B8.5::
  00045 89 e8           mov eax, ebp
  00047 41 bb ff ff ff
        ff              mov r11d, -1
  0004d 83 e0 0c        and eax, 12
  00050 03 c0           add eax, eax
  00052 89 c1           mov ecx, eax
  00054 41 d3 eb        shr r11d, cl
  00057 44 23 dd        and r11d, ebp
  0005a 41 c1 eb 04     shr r11d, 4
  0005e 49 f7 db        neg r11
  00061 4d 03 da        add r11, r10
  00064 83 e5 03        and ebp, 3
  00067 f3 41 0f 6f 03  movdqu xmm0, XMMWORD PTR [r11]
  0006c f3 41 0f 7f 02  movdqu XMMWORD PTR [r10], xmm0
  00071 75 0c           jne .B8.7
.B8.6::
  00073 f3 41 0f 6f 43
        10              movdqu xmm0, XMMWORD PTR [16+r11]
  00079 f3 41 0f 7f 42
        10              movdqu XMMWORD PTR [16+r10], xmm0
.B8.7::
  0007f c1 e8 03        shr eax, 3
  00082 89 e9           mov ecx, ebp
  00084 bd 20 00 00 00  mov ebp, 32
  00089 f7 d8           neg eax
  0008b d3 ed           shr ebp, cl
  0008d 83 c0 04        add eax, 4
  00090 4c 03 d5        add r10, rbp
  00093 48 03 d0        add rdx, rax
.B8.8::
  00096 49 3b d0        cmp rdx, r8
  00099 72 83           jb .B8.3
```

```
; 'Rakka' decompression loop, b1-1e+6=153 bytes long:
; mark_description "-O3 -QxSSE2 -D_N_XMM -FAcs";
.B7.3::
  0001e 44 8b 1a        mov r11d, DWORD PTR [rdx]
  00021 44 89 dd        mov ebp, r11d
  00024 83 e5 0f        and ebp, 15
  00027 83 fd 0c        cmp ebp, 12
  0002a 75 22           jne .B7.6
.B7.4::
  0002c 48 8b 6a 01     mov rbp, QWORD PTR [1+rdx]
  00030 49 89 29        mov QWORD PTR [r9], rbp
  00033 4c 8b 52 09     mov r10, QWORD PTR [9+rdx]
  00037 4d 89 51 08     mov QWORD PTR [8+r9], r10
.B7.5::
  0003b 45 0f b6 db     movzx r11d, r11b
  0003f 41 c1 eb 04     shr r11d, 4
  00043 4d 03 cb        add r9, r11
  00046 41 ff c3        inc r11d
  00049 49 03 d3        add rdx, r11
  0004c eb 60           jmp .B7.10
.B7.6::
  0004e 45 89 da        mov r10d, r11d
  00051 bd ff ff ff ff  mov ebp, -1
  00056 41 83 e2 0c     and r10d, 12
  0005a 45 03 d2        add r10d, r10d
  0005d 44 89 d1        mov ecx, r10d
  00060 d3 ed           shr ebp, cl
  00062 41 23 eb        and ebp, r11d
  00065 c1 ed 04        shr ebp, 4
  00068 48 f7 dd        neg rbp
  0006b 49 03 e9        add rbp, r9
  0006e 48 8b 4d 00     mov rcx, QWORD PTR [rbp]
  00072 49 89 09        mov QWORD PTR [r9], rcx
  00075 48 8b 4d 08     mov rcx, QWORD PTR [8+rbp]
  00079 49 89 49 08     mov QWORD PTR [8+r9], rcx
.B7.7::
  0007d 41 83 e3 03     and r11d, 3
  00081 75 10           jne .B7.9
.B7.8::
  00083 48 8b 4d 10     mov rcx, QWORD PTR [16+rbp]
  00087 49 89 49 10     mov QWORD PTR [16+r9], rcx
  0008b 48 8b 6d 18     mov rbp, QWORD PTR [24+rbp]
  0008f 49 89 69 18     mov QWORD PTR [24+r9], rbp
.B7.9::
  00093 41 c1 ea 03     shr r10d, 3
  00097 44 89 d9        mov ecx, r11d
  0009a bd 20 00 00 00  mov ebp, 32
  0009f 41 f7 da        neg r10d
  000a2 d3 ed           shr ebp, cl
  000a4 41 83 c2 04     add r10d, 4
  000a8 4c 03 cd        add r9, rbp
  000ab 49 03 d2        add rdx, r10
.B7.10::
  000ae 49 3b d0        cmp rdx, r8
  000b1 0f 82 67 ff ff
        ff              jb .B7.3
```

Nakamichi '**Rakka**' vs **lzturbo** 1.2

```
09/08/2014  07:42 AM           152,089 alice29.txt
09/12/2014  06:28 PM            75,834 alice29.txt.Rakka.Nakamichi
AM                              63,013 alice29.txt.v1.2_19.lzt
AM                              50,668 alice29.txt.v1.2_39.lzt
AM                           3,153,408 CalgaryCorpus.tar
PM                           1,315,396 CalgaryCorpus.tar.Rakka.Nakamichi
AM                           1,186,476 CalgaryCorpus.tar.v1.2_19.lzt
AM                             886,634 CalgaryCorpus.tar.v1.2_39.lzt
AM                          10,192,446 dickens
PM                           4,173,755 dickens.Rakka.Nakamichi
AM                           4,377,543 dickens.v1.2_19.lzt
AM                           2,861,165 dickens.v1.2_39.lzt
AM                         100,000,000 enwik8
AM                          41,699,410 enwik8.Rakka.Nakamichi
AM                          41,924,186 enwik8.v1.2_19.lzt
AM                          25,330,833 enwik8.v1.2_39.lzt
AM                          20,617,071 Large_traffic_log_file_of_a_popular_website_fp.log
AM                           1,901,571 Large_traffic_log_file_of_a_popular_website_fp.log.Rakka.Nakamichi
AM                           1,554,849 Large_traffic_log_file_of_a_popular_website_fp.log.v1.2_19.lzt
AM                             732,244 Large_traffic_log_file_of_a_popular_website_fp.log.v1.2_39.lzt
AM                           3,903,143 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd
AM                           1,335,948 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.Rakka.Nakamichi
AM                           1,516,168 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.v1.2_19.lzt
PM                             846,634 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.v1.2_39.lzt
AM                         206,908,949 OSHO.TXT
AM                          66,989,899 OSHO.TXT.Rakka.Nakamichi
AM                          70,050,926 OSHO.TXT.v1.2_19.lzt
PM                          40,006,461 osho.txt.v1.2_39.lzt
AM                           3,984,896 Quran.tar
AM                           1,151,334 Quran.tar.Rakka.Nakamichi
AM                           1,168,422 Quran.tar.v1.2_19.lzt
PM                             753,602 Quran.tar.v1.2_39.lzt
AM                           4,999,168 Sahih_Bukhari.tar
AM                           1,454,483 Sahih_Bukhari.tar.Rakka.Nakamichi
AM                           1,490,821 Sahih_Bukhari.tar.v1.2_19.lzt
PM                             935,699 Sahih_Bukhari.tar.v1.2_39.lzt
AM                           5,582,655 shaks12.txt
PM                           2,280,595 shaks12.txt.Rakka.Nakamichi
AM                           2,292,406 shaks12.txt.v1.2_19.lzt
PM                           1,584,881 shaks12.txt.v1.2_39.lzt
AM                         211,948,544 silesia.tar
PM                          80,315,189 silesia.tar.Rakka.Nakamichi
AM                          77,052,909 silesia.tar.v1.2_19.lzt
PM                          51,704,548 silesia.tar.v1.2_39.lzt
AM                           4,612,608 Webster_Bible.tar
AM                           1,638,402 Webster_Bible.tar.Rakka.Nakamichi
AM                           1,656,134 Webster_Bible.tar.v1.2_19.lzt
PM                           1,122,782 Webster_Bible.tar.v1.2_39.lzt
AM                          28,762,624 _Agatha.Christie.Complete.Work.74.books.tar
AM                          11,697,407 _Agatha.Christie.Complete.Work.74.books.tar.Rakka.Nakamichi
AM                          11,709,136 _Agatha.Christie.Complete.Work.74.books.tar.v1.2_19.lzt
PM                           7,591,061 _Agatha.Christie.Complete.Work.74.books.tar.v1.2_39.lzt
```

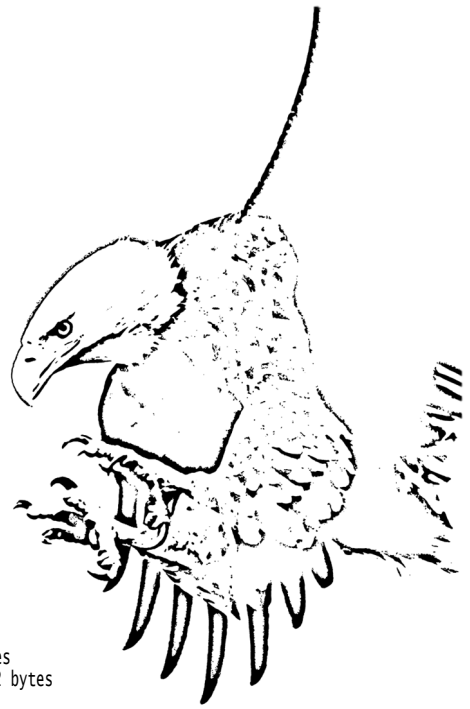
Grmbl!

Grmbl!

Grmbl!

Grmbl!

中道鷲 - NAKAMICHI 'WASHI'



```
unsigned int Decompress(char* ret, char* src, unsigned int srcSize){
              unsigned int srcIndex=0;
              unsigned int retIndex=0;
              unsigned int DWORDtrio;
              unsigned int Flag;
              uint64_t FlagMASK; //=       0xFFFFFFFFFFFFFFFF;
              uint64_t FlagMASKnegated; //=0x0000000000000000;
              while(srcIndex < srcSize){
                            DWORDtrio = *(unsigned int*)&src[srcIndex];
// |1stLSB   |2ndLSB  |3rdLSB   |
// -----------------------------
// |xxxxx|TTT|xxxxxxxF|xxxxxx|LL|
// -----------------------------
// [1bit                  24bit]
// LL = 0 means MatchLength (32>>LL) or 32
// LL = 1 means MatchLength (32>>LL) or 16
// LL = 2 means MatchLength (32>>LL) or 8
// LL = 3 means MatchLength (32>>LL) or 4
// TO-DO: F = 1 means MatchOffset 3 bytes long
// TO-DO: F = 0 means MatchOffset 2 bytes long
                            Flag=!(DWORDtrio & 0xE0);
                            // In here Flag=0|1
                            FlagMASKnegated= Flag - 1; // -1|0
                            FlagMASK= ~FlagMASKnegated;
                            // DWORDtrio&(0xFFFFFF>>((DWORDtrio&0x8000)>>12)) // shift by 0/8 for 3/2 bytes
                            // DWORDtrio&(0xFFFFFF>>(((DWORDtrio&0xFFFF)>>15)<<3)) // shift by 0/8 for 3/2 bytes
//                                                  #ifdef _N_XMM
//SlowCopy128bit( (const char *)( ((uint64_t)(src+srcIndex+1+16*(0))&FlagMASK) + ((uint64_t)(ret+retIndex-(DWORDtrio&0x3FFFFF))&FlagMASKnegated) ), (ret+retIndex+16*(0)));
//SlowCopy128bit( (const char *)( ((uint64_t)(src+srcIndex+1+16*(1))&FlagMASK) + ((uint64_t)(ret+retIndex-(DWORDtrio&0x3FFFFF)+16*(1))&FlagMASKnegated) ),
(ret+retIndex+16*(1)));
//                                                  #endif
                                                    #ifndef _N_YMM
memcpy((ret+retIndex+16*(0)), (const char *)( ((uint64_t)(src+srcIndex+1)&FlagMASK) + ((uint64_t)(ret+retIndex-(DWORDtrio&0x3FFFFF))&FlagMASKnegated) ), 32);
                                                    #endif
                                                    #ifdef _N_YMM
SlowCopy256bit( (const char *)( ((uint64_t)(src+srcIndex+1)&FlagMASK) + ((uint64_t)(ret+retIndex-(DWORDtrio&0x3FFFFF))&FlagMASKnegated) ), (ret+retIndex+16*(0)));
                                                    #endif
                            srcIndex+= ((uint64_t)((DWORDtrio & 0xFF)+1)&FlagMASK) + ((uint64_t)(3)&FlagMASKnegated) ;
                            retIndex+= ((uint64_t)((DWORDtrio & 0xFF))&FlagMASK) +   ((uint64_t)(Min_Match_Length>>((DWORDtrio&0xFFFFFF)>>22))&FlagMASKnegated) ;
              }
              return retIndex;
}
```

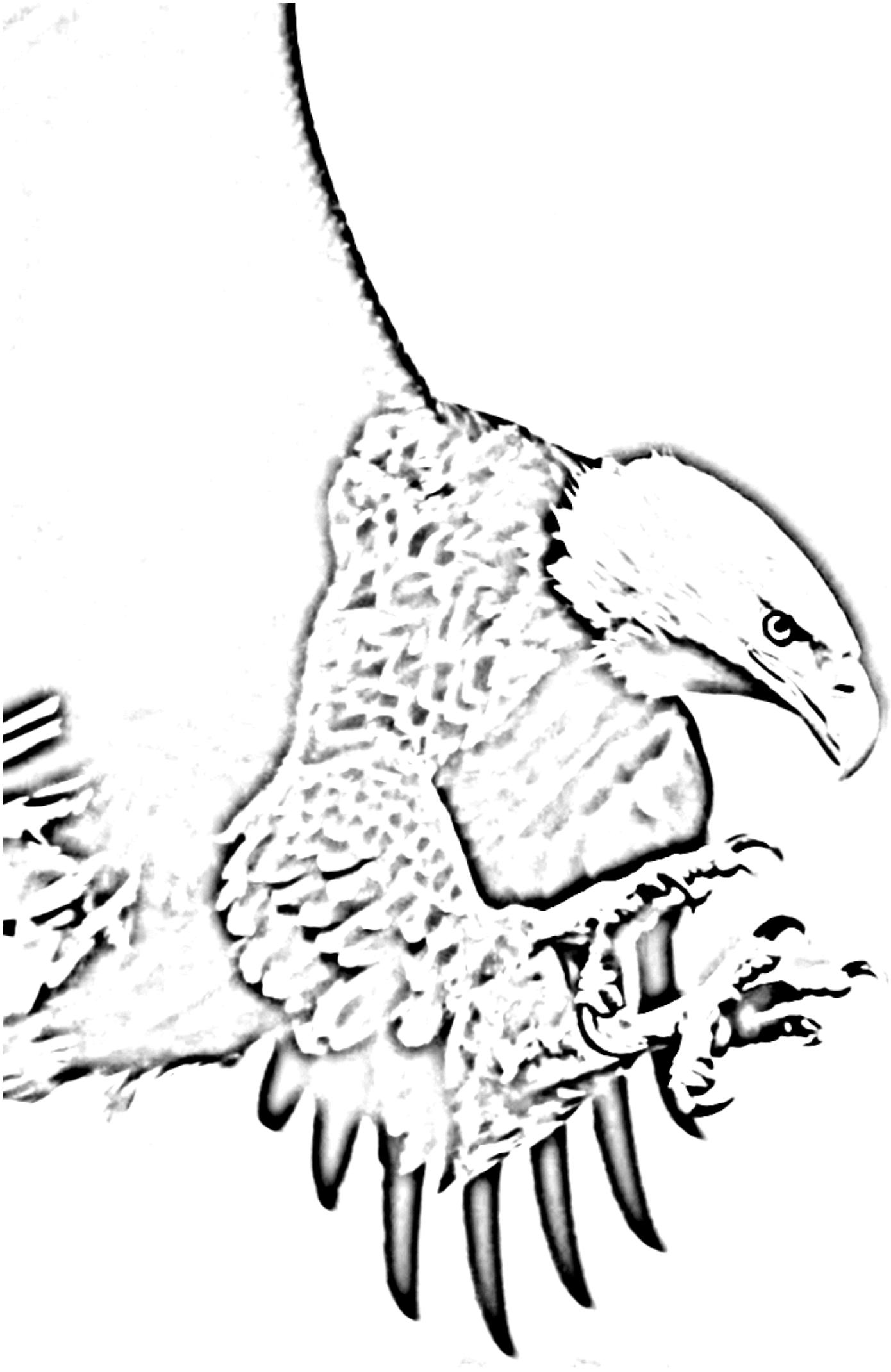
```
; 'Washi' decompression loop, be-40+2=128 bytes long:
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 12.1.1.258 Build 20111";
; mark_description "-O3 -QxSSE2 -D_N_YMM -FAcs";

.B8.3::
  00040 42 8b 0c 12      mov ecx, DWORD PTR [rdx+r10]
  00044 33 ff            xor edi, edi
  00046 f7 c1 e0 00 00
        00               test ecx, 224
  0004c 0f 44 f8         cmove edi, eax
  0004f 49 89 cc         mov r12, rcx
  00052 ff cf            dec edi
  00054 49 81 e4 ff ff
        3f 00            and r12, 4194303
  0005b 49 f7 dc         neg r12
  0005e 48 89 fe         mov rsi, rdi
  00061 4d 03 e1         add r12, r9
  00064 48 f7 d6         not rsi
  00067 4e 8d 74 12 01   lea r14, QWORD PTR [1+rdx+r10]
  0006c 4d 03 e3         add r12, r11
  0006f 4c 23 f6         and r14, rsi
  00072 4c 23 e7         and r12, rdi
  00075 0f b6 d9         movzx ebx, cl
  00078 ff c3            inc ebx
  0007a c4 81 7e 6f 04
        34               vmovdqu ymm0, YMMWORD PTR [r12+r14]
  00080 49 89 fc         mov r12, rdi
  00083 48 23 de         and rbx, rsi
  00086 49 83 e4 03      and r12, 3
  0008a 49 03 dc         add rbx, r12
  0008d 49 03 da         add rbx, r10
  00090 41 89 da         mov r10d, ebx
  00093 0f b6 d9         movzx ebx, cl
  00096 81 e1 ff ff ff
        00               and ecx, 16777215
  0009c c1 e9 16         shr ecx, 22
  0009f 48 23 de         and rbx, rsi
  000a2 be 20 00 00 00   mov esi, 32
  000a7 d3 ee            shr esi, cl
  000a9 48 23 f7         and rsi, rdi
  000ac 48 03 de         add rbx, rsi
  000af 49 03 db         add rbx, r11
  000b2 c4 81 7e 7f 04
        19               vmovdqu YMMWORD PTR [r9+r11], ymm0
  000b8 45 3b d0         cmp r10d, r8d
  000bb 41 89 db         mov r11d, ebx
  000be 72 80            jb .B8.3
```

http://www.sanmayce.com/Nakamichi

- *Superfast Branchless LZSS decompressor*
- *Licenseless i.e. 100% FREE*
- *Written by machinely yours Kaze*

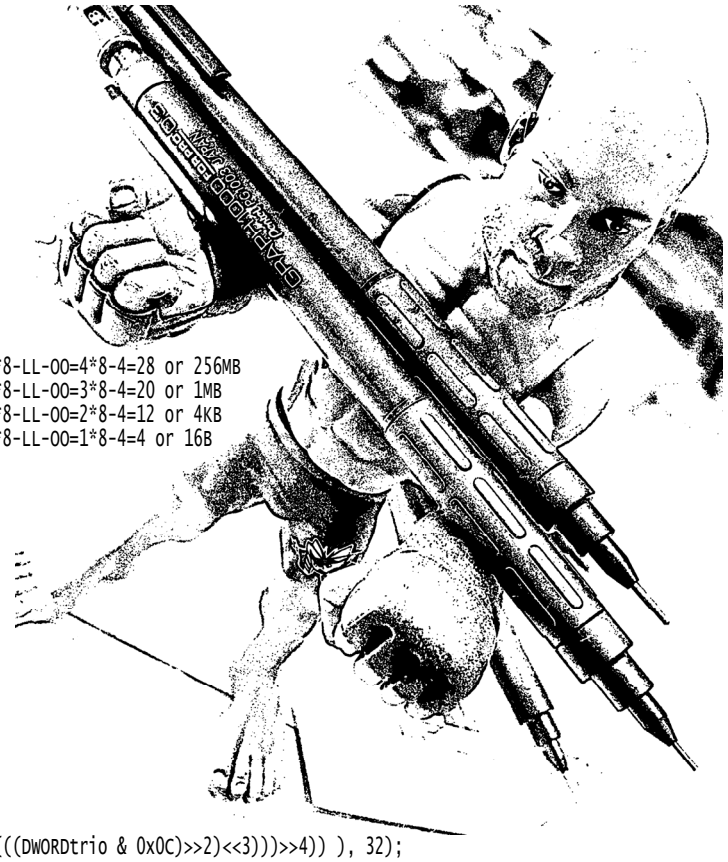
compressor \ filedataset	alice29.txt	CalgaryCorpus.tar	shaks12.txt	dickens	enwik8
UNCOMPRESSED	152,089	3,153,408	5,582,655	10,192,446	100,000,000
Nakamichi 'Kaidanji' (64KB)	092,285 / 0328	1,862,449 / 011838	3,391,657 / 006799	06,387,079 / 014977	063,430,147 / 0283161 / 1014 MB/s
Nakamichi 'Washi' (4MB)	088,897 / 0006	1,484,221 / 000966	2,384,536 / 000011	04,261,276 / 000012	042,714,346 / 0000232 / 435+ MB/s
7z's gz, Ultra Deflate32	051,707	0,980,026	1,934,787	03,681,828	035,102,891
7z's zip, Ultra Deflate64	050,051	0,945,849	1,834,240	03,508,645	033,757,921
TANGELO 2.3	039,160	0,710,066	1,236,021	02,279,659	020,921,619
LZ4 v1.4, -9	063,705	1,195,853	2,315,036	04,442,992	042,283,904 / 2186.9 MB/s
Yappy, 8192 10000	087,965	1,654,203	3,337,964	06,374,780	057,701,807 / 698.7 MB/s
Yappy, 1048576 10000	080,353	1,530,823	3,091,493	05,850,648	053,687,370 / 679.4 MB/s

Note: Core 2 Q9550s @2.83GHz was used (4 cores/threads), since Q9550s doesn't support YMM 'Washi' used the slow 'memcpy' thus 435+ MB/s.
In its native mode (AVX) the result should be MUCH better.



中道黒鉛 - *NAKAMICHI 'Kokuen' a.k.a. 'Blacklead'*

Diamondiferous (bearing/yielding diamonds) LZSS decompressor: www.sanmayce.com/Nakamichi/index.html



```
unsigned int Decompress_Kokuen (char* ret, char* src, unsigned int srcSize) {
            char* retLOCAL = ret; char* srcLOCAL = src;
            char* srcEndLOCAL = src+srcSize;
            unsigned int DWORDtrio;
            while (srcLOCAL < srcEndLOCAL) {
                        DWORDtrio = *(unsigned int*)srcLOCAL;
// ---------------------------------------
// |LL|OO|xxxx|xxxxxxxx|xxxxxx|xx|xxxxxx|xx|
// ---------------------------------------
// [1bit          16bit]    24bit]    32bit]
// LL = 00b means Long MatchLength, 32>>LL or 32
// LL = 01b means Long MatchLength, 32>>LL or 16
// LL = 10b means Long MatchLength, 32>>LL or 8
// LL = 11b means Long MatchLength, 32>>LL or 4
// OO = 00b MatchOffset, 0xFFFFFFFF>>OO, 4 bytes long i.e. Sliding Window is 4*8-LL-OO=4*8-4=28 or 256MB
// OO = 01b MatchOffset, 0xFFFFFFFF>>OO, 3 bytes long i.e. Sliding Window is 3*8-LL-OO=3*8-4=20 or 1MB
// OO = 10b MatchOffset, 0xFFFFFFFF>>OO, 2 bytes long i.e. Sliding Window is 2*8-LL-OO=2*8-4=12 or 4KB
// OO = 11b MatchOffset, 0xFFFFFFFF>>OO, 1 byte long  i.e. Sliding Window is 1*8-LL-OO=1*8-4=4 or 16B
// LL/OO = 00b/11b means Literal
// LL/OO = 01b/11b UNUSED
// LL/OO = 10b/00b UNUSED
// LL/OO = 11b/00b UNUSED
// The 'if' below involves 'cmp' - not pretty!
                        if ( (DWORDtrio & 0x0F) == 0x0C ) {
                                                #ifndef _N_YMM
memcpy(retLOCAL, (const char *)( (uint64_t)(srcLOCAL+1) ), 16);
                                                #endif
                                                #ifdef _N_YMM
SlowCopy128bit( (const char *)( (uint64_t)(srcLOCAL+1) ), retLOCAL );
                                                #endif
                        retLOCAL+= ((DWORDtrio & 0xFF)>>4);
                        srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1;
                        } else {
                                                #ifndef _N_YMM
memcpy(retLOCAL, (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>(((DWORDtrio & 0x0C)>>2)<<3)))>>4)) ), 32);
                                                #endif
                                                #ifdef _N_YMM
SlowCopy256bit( (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>(((DWORDtrio & 0x0C)>>2)<<3)))>>4)) ), retLOCAL );
                                                #endif
                        srcLOCAL+= 4-((DWORDtrio & 0x0C)>>2);
                        retLOCAL+= Min_Match_Length>>(DWORDtrio&0x03);
                        }
            }
            return (unsigned int)(retLOCAL - ret);
}
```

```
; 'Kokuen' decompression loop, 86-1a+2=110 bytes long:
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64,
; Version 12.1.1.258 Build 20111";
; mark_description "-O3 -QxSSE2 -D_N_YMM -FAcs";

.B8.3::
  0001a 8b 2a           mov ebp, DWORD PTR [rdx]
  0001c 89 e8           mov eax, ebp
  0001e 83 e0 0f        and eax, 15
  00021 83 f8 0c        cmp eax, 12
  00024 75 1b           jne .B8.5
.B8.4::
  00026 40 0f b6 ed     movzx ebp, bpl
  0002a f3 0f 6f 42 01  movdqu xmm0, XMMWORD PTR [1+rdx]
  0002f c1 ed 04        shr ebp, 4
  00032 f3 41 0f 7f 02  movdqu XMMWORD PTR [r10], xmm0
  00037 4c 03 d5        add r10, rbp
  0003a ff c5           inc ebp
  0003c 48 03 d5        add rdx, rbp
  0003f eb 42           jmp .B8.6
.B8.5::
  00041 89 e8           mov eax, ebp
  00043 41 bb ff ff ff
        ff              mov r11d, -1
  00049 83 e0 0c        and eax, 12
  0004c 8d 0c 00        lea ecx, DWORD PTR [rax+rax]
  0004f c1 e8 02        shr eax, 2
  00052 41 d3 eb        shr r11d, cl
  00055 f7 d8           neg eax
  00057 44 23 dd        and r11d, ebp
  0005a 83 e5 03        and ebp, 3
  0005d 41 c1 eb 04     shr r11d, 4
  00061 89 e9           mov ecx, ebp
  00063 bd 20 00 00 00  mov ebp, 32
  00068 49 f7 db        neg r11
  0006b 83 c0 04        add eax, 4
  0006e 4d 03 da        add r11, r10
  00071 d3 ed           shr ebp, cl
  00073 48 03 d0        add rdx, rax
  00076 c4 c1 7e 6f 03  vmovdqu ymm0, YMMWORD PTR [r11]
  0007b c4 c1 7e 7f 02  vmovdqu YMMWORD PTR [r10], ymm0
  00080 4c 03 d5        add r10, rbp
.B8.6::
  00083 49 3b d0        cmp rdx, r8
  00086 72 92           jb .B8.3
```

Schocker!
On Haswell 4x8 is 200+MB/s faster than 1x32 transfer:

Nakamichi_Kokuen_**YMMless**_64bit.cod:

```
...
  0006c 48 8b 4d 00     mov rcx, QWORD PTR [rbp]
  00070 49 89 09        mov QWORD PTR [r9], rcx
  00073 48 8b 4d 08     mov rcx, QWORD PTR [8+rbp]
  00077 49 89 49 08     mov QWORD PTR [8+r9], rcx
  0007b 48 8b 4d 10     mov rcx, QWORD PTR [16+rbp]
  0007f 49 89 49 10     mov QWORD PTR [16+r9], rcx
  00083 48 8b 6d 18     mov rbp, QWORD PTR [24+rbp]
  00087 49 89 69 18     mov QWORD PTR [24+r9], rbp
...
```

Nakamichi_Kokuen_**YMM**_64bit.cod:

```
...
  00076 c4 c1 7e 6f 03  vmovdqu ymm0, YMMWORD PTR [r11]
  0007b c4 c1 7e 7f 02  vmovdqu YMMWORD PTR [r10], ymm0
...
```

All the moves were within 1MB Sliding Window!

Grmbl!

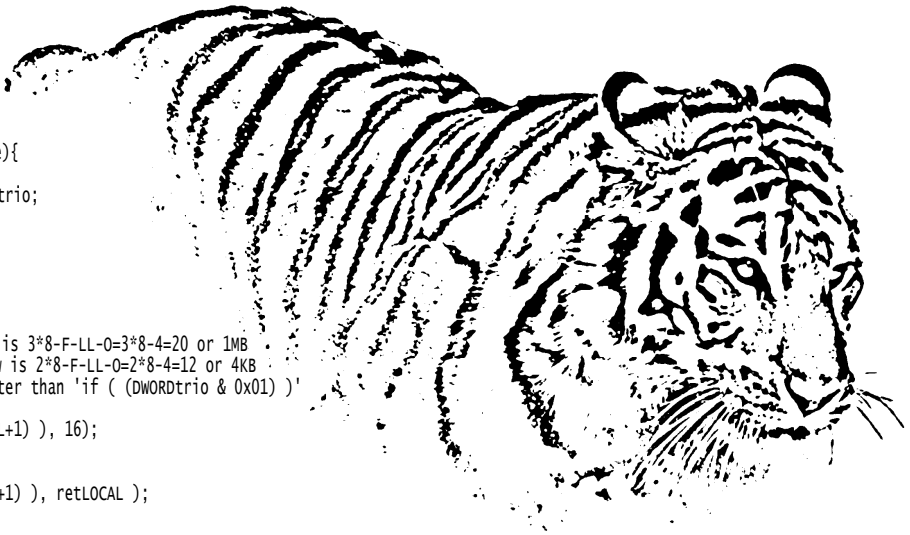
Nakamichi '**Kokuen**' vs **lzturbo** 1.2

```
09/08/2014  07:42 AM           152,089 alice29.txt
09/10/2014  08:21 AM            75,834 alice29.txt.Kokuen.Nakamichi
09/08/2014  11:33 AM            63,013 alice29.txt.v1.2_19.lzt
09/08/2014  11:55 AM            50,668 alice29.txt.v1.2_39.lzt
09/08/2014  07:42 AM         3,153,408 CalgaryCorpus.tar
09/10/2014  08:36 AM         1,318,589 CalgaryCorpus.tar.Kokuen.Nakamichi
09/08/2014  11:33 AM         1,186,476 CalgaryCorpus.tar.v1.2_19.lzt
09/08/2014  11:55 AM           886,634 CalgaryCorpus.tar.v1.2_39.lzt
09/08/2014  07:42 AM        10,192,446 dickens
09/10/2014  10:32 AM         4,156,986 dickens.Kokuen.Nakamichi
09/08/2014  11:33 AM         4,377,543 dickens.v1.2_19.lzt
09/08/2014  11:56 AM         2,861,165 dickens.v1.2_39.lzt
09/08/2014  07:42 AM       100,000,000 enwik8
09/10/2014  ??:???,??? ??? ??,???,??? enwik8.Kokuen.Nakamichi
09/08/2014  11:33 AM        41,924,186 enwik8.v1.2_19.lzt
09/08/2014  11:56 AM        25,330,833 enwik8.v1.2_39.lzt
09/08/2014  07:42 AM        20,617,071 Large_traffic_log_file_of_a_popular_website_fp.log
09/10/2014  08:36 AM         1,867,271 Large_traffic_log_file_of_a_popular_website_fp.log.Kokuen.Nakamichi
09/08/2014  11:33 AM         1,554,849 Large_traffic_log_file_of_a_popular_website_fp.log.v1.2_19.lzt
09/08/2014  11:55 AM           732,244 Large_traffic_log_file_of_a_popular_website_fp.log.v1.2_39.lzt
09/08/2014  07:42 AM         3,903,143 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd
09/10/2014  08:36 AM         1,341,024 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.Kokuen.Nakamichi
09/08/2014  11:33 AM         1,516,168 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.v1.2_19.lzt
09/08/2014  11:55 AM           846,634 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.v1.2_39.lzt
09/08/2014  07:42 AM       206,908,949 OSHO.TXT
09/10/2014  ??:???,??? ??? OSHO.TXT.Kokuen.Nakamichi
09/08/2014  11:33 AM        70,050,926 OSHO.TXT.v1.2_19.lzt
09/08/2014  11:56 AM        40,006,461 osho.txt.v1.2_39.lzt
09/08/2014  07:42 AM         3,984,896 Quran.tar
09/10/2014  08:36 AM         1,153,566 Quran.tar.Kokuen.Nakamichi
09/08/2014  11:33 AM         1,168,422 Quran.tar.v1.2_19.lzt
09/08/2014  11:55 AM           753,602 Quran.tar.v1.2_39.lzt
09/08/2014  07:42 AM         4,999,168 Sahih_Bukhari.tar
09/10/2014  08:36 AM         1,444,399 Sahih_Bukhari.tar.Kokuen.Nakamichi
09/08/2014  11:33 AM         1,490,821 Sahih_Bukhari.tar.v1.2_19.lzt
09/08/2014  11:55 AM           935,699 Sahih_Bukhari.tar.v1.2_39.lzt
09/08/2014  07:42 AM         5,582,655 shaks12.txt
09/10/2014  08:36 AM         2,292,938 shaks12.txt.Kokuen.Nakamichi
09/08/2014  11:33 AM         2,292,406 shaks12.txt.v1.2_19.lzt
09/08/2014  11:56 AM         1,584,881 shaks12.txt.v1.2_39.lzt
09/08/2014  07:42 AM       211,948,544 silesia.tar
09/10/2014  ??:???,??? ??? silesia.tar.Kokuen.Nakamichi
09/08/2014  11:42 AM        77,052,909 silesia.tar.v1.2_19.lzt
09/08/2014  12:06 PM        51,704,548 silesia.tar.v1.2_39.lzt
07/31/2014  11:01 AM         4,612,608 Webster_Bible.tar
09/10/2014  11:11 AM         1,640,604 Webster_Bible.tar.Kokuen.Nakamichi
09/08/2014  11:36 AM         1,656,134 Webster_Bible.tar.v1.2_19.lzt
09/08/2014  12:00 PM         1,122,782 Webster_Bible.tar.v1.2_39.lzt
07/31/2014  11:01 AM        28,762,624 _Agatha.Christie.Complete.Work.74.books.tar
09/10/2014  03:52 PM        11,298,923 _Agatha.Christie.Complete.Work.74.books.tar.Kokuen.Nakamichi
09/08/2014  11:37 AM        11,709,136 _Agatha.Christie.Complete.Work.74.books.tar.v1.2_19.lzt
09/08/2014  12:01 PM         7,591,061 _Agatha.Christie.Complete.Work.74.books.tar.v1.2_39.lzt
```

中道絹虎 - *NAKAMICHI 'Kinutora' a.k.a. 'Silkentiger'*

Tigerishly Strong&Fast LZSS decompressor: www.sanmayce.com/Nakamichi/index.html

```
05/16/2014  07:22 AM       206,908,949 OSHO.TXT
08/29/2014  02:06 AM        87,532,655 OSHO.TXT.Jiten.Nakamichi
09/03/2014  10:28 AM        69,860,415 OSHO.TXT.Kinutora.Nakamichi
08/08/2014  04:42 PM        66,793,112 OSHO.TXT.Washi.Nakamichi
08/28/2014  04:42 AM        57,600,117 OSHO.TXT.Kinroba.Nakamichi
```



```
unsigned int Decompress(char* ret, char* src, unsigned int srcSize){
            char* retLOCAL = ret; char* srcLOCAL = src;
            char* srcEndLOCAL = src+srcSize; unsigned int DWORDtrio;
            while(srcLOCAL < srcEndLOCAL){
                        DWORDtrio = *(unsigned int*)srcLOCAL;
// |T|LL|O|xxxx|xxxxxxxx|xxxxxx|xx|
// [1bit          16bit]    24bit]
// T = 0 means Literal
// LL = 01b means Long MatchLength, 32>>LL or 16
// O = 0 means Long MatchOffset, 3 bytes long i.e. Sliding Window is 3*8-F-LL-O=3*8-4=20 or 1MB
// O = 1 means Short MatchOffset, 2 bytes long i.e. Sliding Window is 2*8-F-LL-O=2*8-4=12 or 4KB
            if ( (DWORDtrio & 0x01) == 0 ) { // That way is faster than 'if ( (DWORDtrio & 0x01) )'
                                    #ifndef _N_YMM
            memcpy(retLOCAL, (const char *)( (uint64_t)(srcLOCAL+1) ), 16);
                                    #endif
                                    #ifdef _N_YMM
            SlowCopy128bit( (const char *)( (uint64_t)(srcLOCAL+1) ), retLOCAL );
                                    #endif
            //srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1;
            retLOCAL+= ((DWORDtrio & 0xFF)>>4);
            srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1; // Swapping the order of above two lines led to 'LEA' vanishing!
            } else {
                                    #ifndef _N_YMM
            memcpy(retLOCAL, (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFF>>((DWORDtrio & 0x08)<<0)))>>4)) ), 32);
                                    #endif
                                    #ifdef _N_YMM
            SlowCopy256bit( (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFF>>((DWORDtrio & 0x08)<<0)))>>4)) ), retLOCAL );
                                    #endif
            srcLOCAL+= (uint64_t)(3-((DWORDtrio & 0x08)>>3));
            retLOCAL+= (uint64_t)( Min_Match_Length>>((DWORDtrio>>1)&0x03) );
            }
            }
            return (unsigned int)(retLOCAL - ret);
}
```

```
; 'Kinutora' decompression loop, 86-15+2=115 bytes long:
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64,
; Version 12.1.1.258 Build 20111";
; mark_description "-O3 -QxSSE2 -D_N_YMM -FAcs";
.B8.3::
  00015 44 8b 12          mov r10d, DWORD PTR [rdx]
  00018 41 f7 c2 01 00
        00 00             test r10d, 1
  0001f 75 1c             jne .B8.5
.B8.4::
  00021 45 0f b6 d2       movzx r10d, r10b
  00025 f3 0f 6f 42 01    movdqu xmm0, XMMWORD PTR [1+rdx]
  0002a 41 c1 ea 04       shr r10d, 4
  0002e f3 0f 7f 00       movdqu XMMWORD PTR [rax], xmm0
  00032 49 03 c2          add rax, r10
  00035 41 ff c2          inc r10d
  00038 49 03 d2          add rdx, r10
  0003b eb 46             jmp .B8.6
.B8.5::
  0003d 44 89 d1          mov ecx, r10d
  00040 41 bb ff ff ff
        00                mov r11d, 16777215
  00046 83 e1 08          and ecx, 8
  00049 41 d3 eb          shr r11d, cl
  0004c 45 23 da          and r11d, r10d
  0004f 41 c1 eb 04       shr r11d, 4
  00053 49 f7 db          neg r11
  00056 4c 03 d8          add r11, rax
  00059 c1 e9 03          shr ecx, 3
  0005c f7 d9             neg ecx
  0005e 41 d1 ea          shr r10d, 1
  00061 83 c1 03          add ecx, 3
  00064 c4 c1 7e 6f 03    vmovdqu ymm0, YMMWORD PTR [r11]
  00069 41 83 e2 03       and r10d, 3
  0006d 48 03 d1          add rdx, rcx
  00070 44 89 d1          mov ecx, r10d
  00073 41 ba 20 00 00
        00                mov r10d, 32
  00079 41 d3 ea          shr r10d, cl
  0007c c5 fe 7f 00       vmovdqu YMMWORD PTR [rax], ymm0
  00080 49 03 c2          add rax, r10
.B8.6::
  00083 49 3b d0          cmp rdx, r8
  00086 72 8d             jb .B8.3
```

```
1.0:1     1,082,907,648 DDETT-Definitive_Decompression_English_Texts_Torture.tar
2.8:1       386,308,041 DDETT-Definitive_Decompression_English_Texts_Torture.tar.lz4              ! -9; 2263 MB/s on Core 2 Q9550s 2.83GHz (4 cores) !
2.8:1       380,665,413 DDETT-Definitive_Decompression_English_Texts_Torture.tar.19.lzt           ! -19 !
2.8:1       377,818,677 DDETT-Definitive_Decompression_English_Texts_Torture.tar.Kinutora.Nakamichi  ! 618 MB/s with YMMless executable on Core 2 Q9550s 2.83GHz (4 cores) !
```

中道地天 - *NAKAMICHI 'Jiten'*

Performance on my 'Bonboniera' laptop, Core 2 T7500 2200MHz:

```
05/16/2014  07:22 AM       206,908,949 OSHO.TXT
08/29/2014  02:06 AM        87,532,655 OSHO.TXT.Jiten.Nakamichi
08/08/2014  08:15 PM        71,399,309 OSHO.TXT.lz4
08/29/2014  11:53 AM        70,461,930 OSHO.TXT.Butsuhira.Nakamichi
08/08/2014  08:46 PM        70,067,665 OSHO.TXT.lzt
08/08/2014  04:42 PM        66,793,112 OSHO.TXT.Washi.Nakamichi
08/28/2014  04:42 AM        57,600,117 OSHO.TXT.Kinroba.Nakamichi

D:\_KAZE\Nakamichi_Kaidanji_benchmark\Nakamichi_benchmark\Nakamichi_Jiten_JB>Nakamichi_Jiten_GP.exe OSHO.TXT.Jiten.Nakamichi
Nakamichi 'Jiten', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2 enforced, muffinesque suggestion by Jim Dempsey enforced.
Decompressing 87532655 bytes ...
RAM-to-RAM performance: 664 MB/s.
```

*NAKAMICHI '**Jiten**' - monstricidally fast!*

Stronger&Faster than the devilishly fast Snappy/Yappy.
Written by machinely yours Kaze.
www.sanmayce.com/Nakamichi/Nakamichi_Jiten_JB.zip

```
unsigned int Decompress_Jiten (char* ret, char* src, unsigned int srcSize) {
//unsigned int srcIndex=0; // Dummy me
//unsigned int retIndex=0; // Dummy me
// The muffinesque suggestion by Jim Dempsey enforced:
char* retLOCAL = ret;
char* srcLOCAL = src;
char* srcEndLOCAL = src+srcSize;
unsigned int DWORDtrio;
//while(srcIndex < srcSize){ // Dummy me
while(srcLOCAL < srcEndLOCAL){
              //DWORDtrio = *(unsigned short int*)&src[srcIndex]; // Dummy me
              DWORDtrio = *(unsigned short int*)srcLOCAL;
// |1stLSB   |2ndLSB   |
// ---------------------
// |xxxTT|TTT|xxxxxx|xL| ! For texts it is better to reduce LL to L i.e. 8/4 matches and 32KB window !
// ---------------------
// [1bit          16bit]
// LL = 0 means MatchLength (8>>LL) or 8
// LL = 1 means MatchLength (8>>LL) or 4
              if ( (DWORDtrio & 0xF8) == 0 ) {
                            #ifdef _N_GP
                            //*(uint64_t*)(ret+retIndex+8*(0)) = *(uint64_t*)(src+srcIndex+1+16*(0)); // Dummy me
                            *(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)(srcLOCAL+1+16*(0));
                            #endif
                            //retIndex+= (DWORDtrio & 0xFF); // Dummy me
                            retLOCAL+= (DWORDtrio & 0xFF);
                            //srcIndex+= ((DWORDtrio & 0xFF)+1); // Dummy me
                            srcLOCAL+= ((DWORDtrio & 0xFF)+1);
              } else {
              #ifdef _N_GP
              //*(uint64_t*)(ret+retIndex+8*(0)) = *(uint64_t*)(ret+retIndex-(DWORDtrio&0x7FFF)); // Dummy me
              *(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)(retLOCAL-(DWORDtrio&0x7FFF));
              #endif
              //retIndex+= (Min_Match_Length>>(DWORDtrio>>(23-8))); // Dummy me
              retLOCAL+= (Min_Match_Length>>(DWORDtrio>>(23-8)));
              //srcIndex+= (3-1); // Dummy me
              srcLOCAL+= (3-1);
              }
}
//return retIndex; // Dummy me
return (unsigned int)(retLOCAL - ret);
}
```

```
; 'Jiten' decompression loop, 60-11+2=81 bytes long:
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64,
; Version 12.1.1.258 Build 20111";
; mark_description "-O3 -D_N_GP -FAcs";

.B6.3::
  00011 0f b7 0a        movzx ecx, WORD PTR [rdx]
  00014 f7 c1 f8 00 00
        00              test ecx, 248
  0001a 75 18           jne .B6.5
.B6.4::
  0001c 44 0f b6 d9     movzx r11d, cl
  00020 0f b6 c9        movzx ecx, cl
  00023 ff c1           inc ecx
  00025 4c 8b 52 01     mov r10, QWORD PTR [1+rdx]
  00029 4d 89 11        mov QWORD PTR [r9], r10
  0002c 4d 03 cb        add r9, r11
  0002f 48 03 d1        add rdx, rcx
  00032 eb 29           jmp .B6.6
.B6.5::
  00034 41 89 ca        mov r10d, ecx
  00037 48 83 c2 02     add rdx, 2
  0003b 49 81 e2 ff 7f
        00 00           and r10, 32767
  00042 49 f7 da        neg r10
  00045 4d 03 d1        add r10, r9
  00048 c1 e9 0f        shr ecx, 15
  0004b 4d 8b 1a        mov r11, QWORD PTR [r10]
  0004e 41 ba 08 00 00
        00              mov r10d, 8
  00054 41 d3 ea        shr r10d, cl
  00057 4d 89 19        mov QWORD PTR [r9], r11
  0005a 4d 03 ca        add r9, r10
.B6.6::
  0005d 49 3b d0        cmp rdx, r8
  00060 72 af           jb .B6.3
```

JITEN
a. k. a.
THE-LIGHT-TOUCH

Fastest wildcard search function

```
// Igor Pavlov's recursive variant modified (and converted to iterative) by Kaze, 2013-Nov-28:
//static boolean Wildcard_IP(const char *mask, int maskPos, const char *name, int namePos) {
//int maskLen = maskGLOBALlen - maskPos;
//int nameLen = nameGLOBALlen - namePos;
//if (maskLen == 0)
//        if (nameLen == 0)
//                return true;
//        else
//                return false;
//if (mask[maskPos] == '*') {
//        if (Wildcard_IP(mask, maskPos + 1, name, namePos))
//                return true;
//        if (nameLen == 0)
//                return false;
//        return Wildcard_IP(mask, maskPos, name, namePos + 1);
//} else if (mask[maskPos] == '?') {
//        if (nameLen == 0)
//                return false;
//        return Wildcard_IP(mask, maskPos + 1, name, namePos + 1);
//} else {
//        if (mask[maskPos] != name[namePos])
//                return false;
//        return Wildcard_IP(mask, maskPos + 1, name, namePos + 1);
//}
//}
int WildcardMatch_Iterative_Kaze(const char* mask, const char* name) {
// Revision 1:
/*
const char* maskSTACK;
const char* nameSTACK;
int BacktrackFlag = 0;
Backtrack:
for (nameSTACK = name, maskSTACK = mask; *nameSTACK; ++nameSTACK, ++maskSTACK) {
        if (*maskSTACK == '&') {
                mask = maskSTACK+1;
                if (!*mask) return 1;
                name = nameSTACK;
                BacktrackFlag = -1;
                goto Backtrack;
        }
        //else if (*maskSTACK == '+') {
        //} else {
        else if (*maskSTACK != '+') {
                //if (tolower(*nameSTACK) != tolower(*maskSTACK)) { // These 'tolower's are outrageous, they hurt speed BADLY, in real-world usage both should
have been lowercased outwith the 'for'.
                if (*nameSTACK != *maskSTACK) {
                        if (!BacktrackFlag) return 0;
                        name++;
                        goto Backtrack;
                }
        }
}
while (*maskSTACK == '&') ++maskSTACK;
return (!*maskSTACK);
*/
// Revision 2, 2013-Nov-30:
const char* maskSTACK;
const char* nameSTACK;
//int BacktrackFlag = 0; // No need of it in rev.2
// Here, outside the main/second 'for', in order to avoid branching we need to set the old/obsolete BacktrackFlag i.e. we need first occurrence of '&':
for (name, mask; *name; ++name, ++mask) {
        if (*mask == '&') {
                goto Backtrack;
        }
        //else if (*mask == '+') {
        //} else {
        else if (*mask != '+') {
                if (*name != *mask) {
                        return 0;
                }
        }
}
// We are entering the main/second 'for' with mask pointing to '&' as if BacktrackFlag is already set in the very first iteration at first condition:
Backtrack:
for (nameSTACK = name, maskSTACK = mask; *nameSTACK; ++nameSTACK, ++maskSTACK) {
        if (*maskSTACK == '&') {
                mask = maskSTACK+1;
                if (!*mask) return 1;
                name = nameSTACK;
                //BacktrackFlag = -1;
                goto Backtrack;
        }
        //else if (*maskSTACK == '+') {
        //} else {
        else if (*maskSTACK != '+') {
                //if (tolower(*nameSTACK) != tolower(*maskSTACK)) { // These 'tolower's are outrageous, they hurt speed BADLY, in real-world usage both should
have been lowercased outwith the 'for'.
                if (*nameSTACK != *maskSTACK) {
                        //if (!BacktrackFlag) return 0; // Stupid branching, SLOW!
                        name++;
                        goto Backtrack;
                }
        }
}
while (*maskSTACK == '&') ++maskSTACK;
return (!*maskSTACK);
}
```

/**Чебурашка**: *Да понятно, а, а где я буду жить?/*

Fastest wildcard search function:

- FREE as FREE at www.sanmayce.com

- written by machinely yours Kaze

Looking at all compression corpora scattered throughout INTERNET one thought arises again and again - we can do better.
Mixing different types of data as in SILESIA corpus is good but it says little about English texts as a whole.
Especially when speaking of decompression speed department.
Hardcore English texts torturers like me need the text picture only, and DEFINITIVE on top of that, that is:
- Giving **coherent** compression ratio statistics;
- Giving **MAIN RAM** decompression speed statistics;
- Being big enough in order to escape being trapped within L1/L2/L3 when compressed.

A year ago I saw a division of Intel, India, producing i7s with 30MB cache.
Now, it is time to have compressed corpus far bigger than this, yes?
For example 'enwik8' is simply outdated, 'enwik9' is quite useful but not exactly.
So, after tarring 1GB (1,082,810,446 bytes) of mainstream English texts into one single file we have:

1.0:1	1,082,907,648 DDETT-Definitive_Decompression_English_Texts_Torture.tar	
2.8:1	380,665,413 DDETT-Definitive_Decompression_English_Texts_Torture.tar.19.lzt	! -19 !
4.7:1	227,860,332 DDETT-Definitive_Decompression_English_Texts_Torture.tar.49.lzt	! -49 !
5.0:1	213,816,605 DDETT-Definitive_Decompression_English_Texts_Torture.tar.49_block128MB.lzt	! -49 -b128 !
5.0:1	216,188,730 DDETT-Definitive_Decompression_English_Texts_Torture.tar.7z	! -t7z -mx9 !
5.8:1	186,555,864 DDETT-Definitive_Decompression_English_Texts_Torture.tar.block128MB.bbb	! cfm128 !
3.2:1	328,252,314 DDETT-Definitive_Decompression_English_Texts_Torture.tar.Doboz	
2.0:1	534,389,558 DDETT-Definitive_Decompression_English_Texts_Torture.tar.FastLZ	! -2 !
2.8:1	386,308,041 DDETT-Definitive_Decompression_English_Texts_Torture.tar.lz4	! -9 !
4.4:1	241,416,540 DDETT-Definitive_Decompression_English_Texts_Torture.tar.lzhds.nz	! -cD !
6.8:1	157,378,486 DDETT-Definitive_Decompression_English_Texts_Torture.tar.cm.nz	! -cc !
5.0:1	213,997,220 DDETT-Definitive_Decompression_English_Texts_Torture.tar.order04.PPMonstr	! -m1024 -o4 !
5.7:1	189,072,423 DDETT-Definitive_Decompression_English_Texts_Torture.tar.order06.PPMonstr	! -m1024 -o6 !
6.0:1	179,543,238 DDETT-Definitive_Decompression_English_Texts_Torture.tar.order08.PPMonstr	! -m1024 -o8 !
6.3:1	169,784,214 DDETT-Definitive_Decompression_English_Texts_Torture.tar.order16.PPMonstr	! -m1024 -o16 !
6.3:1	169,634,437 DDETT-Definitive_Decompression_English_Texts_Torture.tar.order32.PPMonstr	! -m1024 -o32 !
2.5:1	427,885,522 DDETT-Definitive_Decompression_English_Texts_Torture.tar.QuickLZ	/ Level 3 /
3.9:1	271,076,720 DDETT-Definitive_Decompression_English_Texts_Torture.tar.sr2	
4.4:1	245,308,541 DDETT-Definitive_Decompression_English_Texts_Torture.tar.ST3_block128MB.bsc	! -m0b128 -Tt !
5.0:1	214,353,318 DDETT-Definitive_Decompression_English_Texts_Torture.tar.ST4_block128MB.bsc	! -m1b128 -Tt !
5.3:1	201,591,671 DDETT-Definitive_Decompression_English_Texts_Torture.tar.ST5_block128MB.bsc	! -m2b128 -Tt !
5.8:1	184,725,469 DDETT-Definitive_Decompression_English_Texts_Torture.tar.BWT_block128MB.bsc	! -m3b128 -Tt !
6.0:1	179,037,137 DDETT-Definitive_Decompression_English_Texts_Torture.tar.tangelo	/ Version 2.3 /
1.4:1	758,789,281 DDETT-Definitive_Decompression_English_Texts_Torture.tar.0001K.Yappy	! 1024 10000 !
1.5:1	683,426,649 DDETT-Definitive_Decompression_English_Texts_Torture.tar.0002K.Yappy	! 2048 10000 !
1.7:1	612,838,178 DDETT-Definitive_Decompression_English_Texts_Torture.tar.0004K.Yappy	! 4096 10000 !
1.9:1	563,557,594 DDETT-Definitive_Decompression_English_Texts_Torture.tar.0008K.Yappy	! 8192 10000 !
2.0:1	538,909,255 DDETT-Definitive_Decompression_English_Texts_Torture.tar.0016K.Yappy	! 16384 10000 !
2.0:1	520,393,884 DDETT-Definitive_Decompression_English_Texts_Torture.tar.0064K.Yappy	! 65536 10000 !
2.0:1	515,767,567 DDETT-Definitive_Decompression_English_Texts_Torture.tar.0256K.Yappy	! 262144 10000 !
2.9:1	370,986,428 DDETT-Definitive_Decompression_English_Texts_Torture.tar.Washi.Nakamichi	
2.7:1	393,403,817 DDETT-Definitive_Decompression_English_Texts_Torture.tar.Z	
3.2:1	336,177,082 DDETT-Definitive_Decompression_English_Texts_Torture.tar.lzh	/ Yoshizaki's 20+ years old LHA /
2.9:1	363,888,267 DDETT-Definitive_Decompression_English_Texts_Torture.tar.Rakka.Nakamichi	
3.0:1	359,115,942 DDETT-Definitive_Decompression_English_Texts_Torture.tar.Tengu.Nakamichi	
3.3:1	319,107,528 DDETT-Definitive_Decompression_English_Texts_Torture.tar.zip	! -tzip -mx9 !

"Falling flowers have feeling..."
/An example from 'INOUYE's Japanese-English Dictionary'/

Fulcrumish Benchmark

For a long time I wanted to have one solid English comparative TEXTUAL [DE]COMPRESSION corpus, so here it is:
http://www.sanmayce.com/Nakamichi/Fallenflower_DDETT-Definitive_Decompression_English_Texts_Torture.tar.Rakka.Nakamichi.7z

And quick showdown against the superfast Yappy and ultrafast LZ4 on laptop with Core 2 Q9550s @2.83GHz:

```
C:\DDETT>Yappy_32bit.exe DDETT-Definitive_Decompression_English_Texts_Torture.tar 1024 10000
YAPPY: [b 1K] bytes 1082907648 -> 758789281  70.1%  comp  49.0 MB/s  uncomp 817.7 MB/s

C:\DDETT>Yappy_32bit.exe DDETT-Definitive_Decompression_English_Texts_Torture.tar 65536 10000
YAPPY: [b 64K] bytes 1082907648 -> 520393884  48.1%  comp  29.9 MB/s  uncomp 668.9 MB/s

C:\DDETT>Yappy_32bit.exe DDETT-Definitive_Decompression_English_Texts_Torture.tar 4194304 10000
YAPPY: [b 4096K] bytes 1082907648 -> 514318888  47.5%  comp  29.5 MB/s  uncomp 668.9 MB/s

C:\DDETT>Nakamichi_Rakka_GP DDETT-Definitive_Decompression_English_Texts_Torture.tar.Rakka.Nakamichi /report
Nakamichi 'Rakka', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2 enforced,
muffinesque suggestion by Jim Dempsey enforced.
Decompressing 363888267 bytes ...
RAM-to-RAM performance: 768 MB/s.
Memory pool starting address: 00000000160D0080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 189697 clocks or 2.764MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 27%

C:\DDETT>Nakamichi_Tengu_GP DDETT-Definitive_Decompression_English_Texts_Torture.tar.Tengu.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2 enforced,
muffinesque suggestion by Jim Dempsey enforced.
Decompressing 359115942 bytes ...
RAM-to-RAM performance: 896 MB/s.
Memory pool starting address: 00000000015B40080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 189495 clocks or 2.767MB per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 32%

C:\DDETT>lz4 -9 -Sx -b -T1 DDETT-Definitive_Decompression_English_Texts_Torture.tar.Rakka
Nb of threads = 1 ; Compression Level = 9
DDETT-Definitiv : 905969664 -> 315340335 ( 34.81%),   14.5 MB/s ,  984.4 MB/s

C:\DDETT>lz4 -9 -Sx -b DDETT-Definitive_Decompression_English_Texts_Torture.tar.Rakka
Nb of threads = 4 ; Compression Level = 9
DDETT-Definitiv : 905969664 -> 315340335 ( 34.81%),   55.7 MB/s , 2250.3 MB/s
```

```
; 'Rakka' decompression loop, 92-1e+2=118 bytes long:
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for
applications running on Intel(R) 64, Version 12.1.1.258 Build
20111";
; mark_description "-O3 -QxSSE2 -D_N_YMM -D_N_prefetch_4096 -FAcs";
.B10.3::
  0001e 0f 18 8a 00 10
        00 00           prefetcht0 BYTE PTR [4096+rdx]
  00025 8b 2a           mov ebp, DWORD PTR [rdx]
  00027 89 e8           mov eax, ebp
  00029 83 e0 0f        and eax, 15
  0002c 83 f8 0c        cmp eax, 12
  0002f 75 1b           jne .B10.5
.B10.4::
  00031 c5 fe 6f 42 01  vmovdqu ymm0, YMMWORD PTR [1+rdx]
  00036 40 0f b6 ed     movzx ebp, bpl
  0003a c1 ed 04        shr ebp, 4
  0003d c4 c1 7e 7f 02  vmovdqu YMMWORD PTR [r10], ymm0
  00042 4c 03 d5        add r10, rbp
  00045 ff c5           inc ebp
  00047 48 03 d5        add rdx, rbp
  0004a eb 43           jmp .B10.6
.B10.5::
  0004c 89 e8           mov eax, ebp
  0004e 41 bb ff ff ff
        ff              mov r11d, -1
  00054 83 e0 0c        and eax, 12
  00057 03 c0           add eax, eax
  00059 89 c1           mov ecx, eax
  0005b 41 d3 eb        shr r11d, cl
  0005e 44 23 dd        and r11d, ebp
  00061 83 e5 03        and ebp, 3
  00064 41 c1 eb 04     shr r11d, 4
  00068 89 e9           mov ecx, ebp
  0006a bd 20 00 00 00  mov ebp, 32
  0006f 49 f7 db        neg r11
  00072 4d 03 da        add r11, r10
  00075 c1 e8 03        shr eax, 3
  00078 f7 d8           neg eax
  0007a d3 ed           shr ebp, cl
  0007c 83 c0 04        add eax, 4
  0007f c4 c1 7e 6f 03  vmovdqu ymm0, YMMWORD PTR [r11]
  00084 c4 c1 7e 7f 02  vmovdqu YMMWORD PTR [r10], ymm0
  00089 4c 03 d5        add r10, rbp
  0008c 48 03 d0        add rdx, rax
.B10.6::
  0008f 49 3b d0        cmp rdx, r8
  00092 72 8a           jb .B10.3
```

<u>Many thanks, for direct help, go to:</u> my brother, *Nobuo Ito*, *m^2*, Prof. *Okumura*, *Yoshizaki*, *Lasse*, *Fantasy*, *Harold*, *Mr.Eiht*, *Igor Pavlov*, *Colin*, *Jim Dempsey* ...

中道落花 - *NAKAMICHI 'Rakka' a.k.a. 'Fallenflower'*

```
unsigned int Decompress (char* ret, char* src, unsigned int srcSize) {
		char* retLOCAL = ret;
		char* srcLOCAL = src;
		char* srcEndLOCAL = src+srcSize;
		unsigned int DWORDtrio;
		unsigned int DWORDtrioDumbo;
		while (srcLOCAL < srcEndLOCAL) {
				DWORDtrio = *(unsigned int*)srcLOCAL;
#ifndef _N_GP
#ifdef _N_prefetch_64
				_mm_prefetch((char*)(srcLOCAL + 64), _MM_HINT_T0);
#endif
#ifdef _N_prefetch_128
				_mm_prefetch((char*)(srcLOCAL + 64*2), _MM_HINT_T0);
#endif
#ifdef _N_prefetch_4096
				_mm_prefetch((char*)(srcLOCAL + 64*64), _MM_HINT_T0);
#endif
#endif
// |1stLSB    |2ndLSB  |3rdLSB   |4thLSB   |
// ----------------------------------------
// |LL|OO|xxxx|xxxxxxxx|xxxxxx|xx|xxxxxx|xx|
// ----------------------------------------
// [1bit          16bit]    24bit]    32bit]
// LL = 00b means Long MatchLength, 32>>LL or 32
// LL = 01b means Long MatchLength, 32>>LL or 16
// LL = 10b means Long MatchLength, 32>>LL or 8
// LL = 11b means Long MatchLength, 32>>LL or 4
// LL/OO = 00b/11b means Literal
// LL/OO = 01b/11b UNUSED
// LL/OO = 11b/00b UNUSED
// OO = 00b MatchOffset, 0xFFFFFFFF>>OO, 4 bytes long i.e. Sliding Window is 4*8-LL-OO=4*8-4=28 or 256MB
// OO = 01b MatchOffset, 0xFFFFFFFF>>OO, 3 bytes long i.e. Sliding Window is 3*8-LL-OO=3*8-4=20 or 1MB
// OO = 10b MatchOffset, 0xFFFFFFFF>>OO, 2 bytes long i.e. Sliding Window is 2*8-LL-OO=2*8-4=12 or 4KB
// OO = 11b MatchOffset, 0xFFFFFFFF>>OO, 1 byte long  i.e. Sliding Window is 1*8-LL-OO=1*8-4=4 or 16B
				if ( (DWORDtrio & 0x0F) == 0x0C ) {
#ifdef _N_GP
				memcpy(retLOCAL, (const char *)( (uint64_t)(srcLOCAL+1) ), 16);
#endif
#ifdef _N_XMM
				SlowCopy128bit( (const char *)( (uint64_t)(srcLOCAL+1) ), retLOCAL );
#endif
#ifdef _N_YMM
//				SlowCopy128bit( (const char *)( (uint64_t)(srcLOCAL+1) ), retLOCAL );
// Intel AVX-512 offers a level of compatibility with AVX that is stronger than prior transitions
// to new widths for SIMD operations. Unlike SSE and AVX that cannot be mixed without *performance penalties*,
// the mixing of AVX and Intel AVX-512 instructions is supported without penalty. AVX registers YMM0-YMM15 map
// into the Intel AVX-512 registers ZMM0-ZMM15, very much like SSE registers map into AVX registers.
// Therefore, in processors with Intel AVX-512 support, AVX and AVX2 instructions operate on the lower 128 or
// 256 bits of the first 16 ZMM registers.
// /'AVX-512 instructions' - James Reinders/
// Intel describes this in the "Intel 64 and IA-32 Architectures Optimization Reference Manual", which can
// be downloaded from this web page:
// http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
// Intel writes, in section 11.3:
// "Initially the processor is in clean state (1), where Intel SSE and Intel AVX instructions are executed
// with no penalty. When a 256-bit Intel AVX instruction is executed, the processor marks that it is in the
// Dirty Upper state (2). While in this state, executing an Intel SSE instruction saves the upper 128 bits of
// all YMM registers and the state changes to Saved Dirty Upper state (3). Next time an Intel AVX instruction
// is executed the upper 128 bits of all YMM registers are restored and the processor is back at state (2).
// These save and restore operations have a high penalty. Frequent execution of these transitions causes
// significant performance loss."
// /'Performance penalty for mixed AVX512 code?'- Russell Van Zandt/
				SlowCopy256bit( (const char *)( (uint64_t)(srcLOCAL+1) ), retLOCAL );
#endif
				retLOCAL+= ((DWORDtrio & 0xFF)>>4);
				srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1;
				} else {
				DWORDtrioDumbo = ((DWORDtrio & 0x0C)>>2)<<3; // To avoid 'LEA'
#ifndef _N_YMM
#ifdef _N_GP
				memcpy(retLOCAL, (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), 16);
#endif
#ifdef _N_XMM
				SlowCopy128bit( (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), retLOCAL );
#endif
				if ( (DWORDtrio & 0x03) == 0 ) {
#ifdef _N_GP
				memcpy(16+ retLOCAL, (const char *)( 16+ (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), 16);
#endif
#ifdef _N_XMM
				SlowCopy128bit( (const char *)( 16+ (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), 16+ retLOCAL );
#endif
				}
#else
				SlowCopy256bit( (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), retLOCAL );
#endif
				retLOCAL+= Min_Match_Length>>(DWORDtrio&0x03);
				srcLOCAL+= 4-(DWORDtrioDumbo>>3); // 8*0, 8*1, 8*2, 8*3
				}
		}
		return (unsigned int)(retLOCAL - ret);
}
```

```
; 'Rakka' decompression loop, b1-1e+6=153 bytes long:
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE
for applications running on Intel(R) 64, Version 12.1.1.258
Build 20111";
; mark_description "-O3 -D_N_GP -FAcs";

.B8.3::
  0001e 44 8b 1a         mov r11d, DWORD PTR [rdx]
  00021 44 89 dd         mov ebp, r11d
  00024 83 e5 0f         and ebp, 15
  00027 83 fd 0c         cmp ebp, 12
  0002a 75 22            jne .B8.6
.B8.4::
  0002c 48 8b 6a 01      mov rbp, QWORD PTR [1+rdx]
  00030 49 89 29         mov QWORD PTR [r9], rbp
  00033 4c 8b 52 09      mov r10, QWORD PTR [9+rdx]
  00037 4d 89 51 08      mov QWORD PTR [8+r9], r10
.B8.5::
  0003b 45 0f b6 db      movzx r11d, r11b
  0003f 41 c1 eb 04      shr r11d, 4
  00043 4d 03 cb         add r9, r11
  00046 41 ff c3         inc r11d
  00049 49 03 d3         add rdx, r11
  0004c eb 60            jmp .B8.10
.B8.6::
  0004e 45 89 da         mov r10d, r11d
  00051 bd ff ff ff ff   mov ebp, -1
  00056 41 83 e2 0c      and r10d, 12
  0005a 45 03 d2         add r10d, r10d
  0005d 44 89 d1         mov ecx, r10d
  00060 d3 ed            shr ebp, cl
  00062 41 23 eb         and ebp, r11d
  00065 c1 ed 04         shr ebp, 4
  00068 48 f7 dd         neg rbp
  0006b 49 03 e9         add rbp, r9
  0006e 48 8b 4d 00      mov rcx, QWORD PTR [rbp]
  00072 49 89 09         mov QWORD PTR [r9], rcx
  00075 48 8b 4d 08      mov rcx, QWORD PTR [8+rbp]
  00079 49 89 49 08      mov QWORD PTR [8+r9], rcx
.B8.7::
  0007d 41 83 e3 03      and r11d, 3
  00081 75 10            jne .B8.9
.B8.8::
  00083 48 8b 4d 10      mov rcx, QWORD PTR [16+rbp]
  00087 49 89 49 10      mov QWORD PTR [16+r9], rcx
  0008b 48 8b 6d 18      mov rbp, QWORD PTR [24+rbp]
  0008f 49 89 69 18      mov QWORD PTR [24+r9], rbp
.B8.9::
  00093 41 c1 ea 03      shr r10d, 3
  00097 44 89 d9         mov ecx, r11d
  0009a bd 20 00 00 00   mov ebp, 32
  0009f 41 f7 da         neg r10d
  000a2 d3 ed            shr ebp, cl
  000a4 41 83 c2 04      add r10d, 4
  000a8 4c 03 cd         add r9, rbp
  000ab 49 03 d2         add rdx, r10
.B8.10::
  000ae 49 3b d0         cmp rdx, r8
  000b1 0f 82 67 ff ff
        ff               jb .B8.3
```

中道天狗 - NAKAMICHI 'Tengu' a.k.a. 'Skydog'

Meteoric LZSS decompressor: www.sanmayce.com/Nakamichi/index.html

1. Of, relating to, or formed by a meteoroid. 2. Of or relating to the earth's atmosphere. 3. Similar to a meteor in speed, brilliance, or brevity: *a meteoric rise to fame.*

```
unsigned int Decompress_Tengu (char* ret, char* src, unsigned int srcSize) {
            char* retLOCAL = ret; char* srcLOCAL = src; char* srcEndLOCAL = src+srcSize;
            unsigned int DWORDtrio; unsigned int DWORDtrioDumbo;
            while (srcLOCAL < srcEndLOCAL) {
                        DWORDtrio = *(unsigned int*)srcLOCAL;
// |OO|LL|xxxx|xxxxxxxx|xxxxxx|xx|
// [1bit         16bit]    24bit]
// LL = 00b means Long MatchLength, (4-LL)<<2 or 16
// LL = 01b means Long MatchLength, (4-LL)<<2 or 12
// LL = 10b means Long MatchLength, (4-LL)<<2 or 8
// LL = 11b means Long MatchLength, (4-LL)<<2 or 4
// OO = 00b means Literal
// OO = 01b MatchOffset, 0xFFFFFFFF>>OO, 3 bytes long i.e. Sliding Window is 3*8-LL-OO=3*8-4=20 or 1MB
// OO = 10b MatchOffset, 0xFFFFFFFF>>OO, 2 bytes long i.e. Sliding Window is 2*8-LL-OO=2*8-4=12 or 4KB
// OO = 11b MatchOffset, 0xFFFFFFFF>>OO, 1 byte long  i.e. Sliding Window is 1*8-LL-OO=1*8-4=4 or 16B
                        if ( (DWORDtrio & 0x03) == 0x00 ) {
                                                #ifdef _N_GP
memcpy(retLOCAL, (const char *)( (uint64_t)(srcLOCAL+1) ), 16);
                                                #endif
                                                #ifdef _N_XMM
SlowCopy128bit( (const char *)( (uint64_t)(srcLOCAL+1) ), retLOCAL );
                                                #endif
                        retLOCAL+= ((DWORDtrio & 0xFF)>>4);
                        srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1;
                        } else {
                        DWORDtrioDumbo = (DWORDtrio & 0x03)<<3; // To avoid 'LEA'
                                                #ifdef _N_GP
memcpy(retLOCAL, (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), 16);
                                                #endif
                                                #ifdef _N_XMM
SlowCopy128bit( (const char *)( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), retLOCAL );
                                                #endif
                        retLOCAL+= 16-(DWORDtrio&0x0C);
                        srcLOCAL+= 4-(DWORDtrioDumbo>>3);
                        }
            }
            return (unsigned int)(retLOCAL - ret);
}
```

```
; 'Tengu' decompression loop, 7f-1a+2=103 bytes long:
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64,
; Version 12.1.1.258 Build 20111";
; mark_description "-O3 -D_N_GP -FAcs";
.B6.3::
  0001a 8b 02           mov eax, DWORD PTR [rdx]
  0001c 89 c1           mov ecx, eax
  0001e 83 e1 03        and ecx, 3
  00021 75 1f           jne .B6.6
.B6.4::
  00023 48 8b 6a 01     mov rbp, QWORD PTR [1+rdx]
  00027 49 89 2a        mov QWORD PTR [r10], rbp
  0002a 4c 8b 5a 09     mov r11, QWORD PTR [9+rdx]
  0002e 4d 89 5a 08     mov QWORD PTR [8+r10], r11
.B6.5::
  00032 0f b6 c0        movzx eax, al
  00035 c1 e8 04        shr eax, 4
  00038 4c 03 d0        add r10, rax
  0003b ff c0           inc eax
  0003d 48 03 d0        add rdx, rax
  00040 eb 3a           jmp .B6.8
.B6.6::
  00042 c1 e1 03        shl ecx, 3
  00045 bd ff ff ff ff  mov ebp, -1
  0004a d3 ed           shr ebp, cl
  0004c 23 e8           and ebp, eax
  0004e c1 ed 04        shr ebp, 4
  00051 48 f7 dd        neg rbp
  00054 49 03 ea        add rbp, r10
  00057 4c 8b 5d 00     mov r11, QWORD PTR [rbp]
  0005b 4d 89 1a        mov QWORD PTR [r10], r11
  0005e 4c 8b 5d 08     mov r11, QWORD PTR [8+rbp]
  00062 4d 89 5a 08     mov QWORD PTR [8+r10], r11
.B6.7::
  00066 c1 e9 03        shr ecx, 3
  00069 83 e0 0c        and eax, 12
  0006c f7 d8           neg eax
  0006e f7 d9           neg ecx
  00070 83 c0 10        add eax, 16
  00073 83 c1 04        add ecx, 4
  00076 4c 03 d0        add r10, rax
  00079 48 03 d1        add rdx, rcx
.B6.8::
  0007c 49 3b d0        cmp rdx, r8
  0007f 72 99           jb .B6.3
```

Nakamichi '**Tengu**' vs **lzturbo** 1.2

Grmbl!

```
09/08/2014  07:42 AM           152,089 alice29.txt
09/30/2014  06:31 AM            73,787 alice29.txt.Tengu.Nakamichi
09/08/2014  11:33 AM            63,013 alice29.txt.v1.2_19.lzt
09/08/2014  11:55 AM            50,668 alice29.txt.v1.2_39.lzt
09/08/2014  07:42 AM         3,153,408 CalgaryCorpus.tar
09/30/2014  07:15 AM         1,296,773 CalgaryCorpus.tar.Tengu.Nakamichi
09/08/2014  11:33 AM         1,186,476 CalgaryCorpus.tar.v1.2_19.lzt
09/08/2014  11:55 AM           886,634 CalgaryCorpus.tar.v1.2_39.lzt
09/08/2014  07:42 AM        10,192,446 dickens
09/30/2014  07:49 AM         3,993,467 dickens.Tengu.Nakamichi
09/08/2014  11:33 AM         4,377,543 dickens.v1.2_19.lzt
09/08/2014  11:56 AM         2,861,165 dickens.v1.2_39.lzt
09/08/2014  07:42 AM       100,000,000 enwik8
09/30/2014  12:06 PM        40,860,927 enwik8.Tengu.Nakamichi
09/08/2014  11:35 AM        41,924,186 enwik8.v1.2_19.lzt
09/08/2014  11:59 AM        25,330,833 enwik8.v1.2_39.lzt
09/08/2014  08:19 AM        20,617,071 Large_traffic_log_file_of_a_popular_website_fp.log
09/30/2014  07:54 AM         3,096,667 Large_traffic_log_file_of_a_popular_website_fp.log.Tengu.Nakamichi
09/08/2014  11:36 AM         1,554,849 Large_traffic_log_file_of_a_popular_website_fp.log.v1.2_19.lzt
09/08/2014  11:59 AM           732,244 Large_traffic_log_file_of_a_popular_website_fp.log.v1.2_39.lzt
09/08/2014  07:57 AM         3,903,143 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd
09/30/2014  06:56 AM         1,308,271 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.Tengu.Nakamichi
09/08/2014  11:59 AM         1,516,168 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.v1.2_19.lzt
09/08/2014  11:59 AM           846,634 MASAKARI_General-Purpose_Grade_English_Wordlist.wrd.v1.2_39.lzt
09/08/2014  07:42 AM       206,908,949 OSHO.TXT
09/30/2014  06:31 AM        64,746,363 OSHO.TXT.Tengu.Nakamichi
09/08/2014  11:33 AM        70,050,926 OSHO.TXT.v1.2_19.lzt
09/08/2014  11:55 AM        40,006,461 osho.txt.v1.2_39.lzt
09/08/2014  07:42 AM         3,984,896 Quran.tar
09/30/2014  07:15 AM         1,173,663 Quran.tar.Tengu.Nakamichi
09/08/2014  11:33 AM         1,168,422 Quran.tar.v1.2_19.lzt
09/08/2014  11:55 AM           753,602 Quran.tar.v1.2_39.lzt
09/08/2014  07:42 AM         4,999,168 Sahih_Bukhari.tar
09/30/2014  07:15 AM         1,455,153 Sahih_Bukhari.tar.Tengu.Nakamichi
09/08/2014  11:33 AM         1,490,821 Sahih_Bukhari.tar.v1.2_19.lzt
09/08/2014  11:55 AM           935,699 Sahih_Bukhari.tar.v1.2_39.lzt
09/08/2014  07:42 AM         5,582,655 shaks12.txt
09/30/2014  07:15 AM         2,211,086 shaks12.txt.Tengu.Nakamichi
09/08/2014  11:33 AM         2,292,406 shaks12.txt.v1.2_19.lzt
09/08/2014  11:55 AM         1,584,881 shaks12.txt.v1.2_39.lzt
09/08/2014  07:42 AM       211,948,544 silesia.tar
09/30/2014  06:31 AM        82,918,238 silesia.tar.Nakamichi
09/08/2014  11:33 AM        77,052,909 silesia.tar.v1.2_19.lzt
09/08/2014  11:55 AM        51,704,548 silesia.tar.v1.2_39.lzt
09/08/2014  07:42 AM         4,612,608 Webster_Bible.tar
09/30/2014  07:15 AM         1,561,021 Webster_Bible.tar.Tengu.Nakamichi
09/08/2014  11:33 AM         1,656,134 Webster_Bible.tar.v1.2_19.lzt
09/08/2014  11:55 AM         1,122,782 Webster_Bible.tar.v1.2_39.lzt
09/08/2014  07:42 AM        28,762,624 _Agatha.Christie.Complete.Work.74.books.tar
09/30/2014  07:15 AM        11,234,791 _Agatha.Christie.Complete.Work.74.books.tar.Tengu.Nakamichi
09/08/2014  11:33 AM        11,709,136 _Agatha.Christie.Complete.Work.74.books.tar.v1.2_19.lzt
09/08/2014  11:55 AM         7,591,061 _Agatha.Christie.Complete.Work.74.books.tar.v1.2_39.lzt
```

Grmbl!

Grmbl!

Grmbl!

Grmbl!

Results on Core2 T7500 2200MHz:

```
D:\Nakamichi_Tengu>Nakamichi_Tengu_GP.exe OSHO.TXT.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source,
babealicious suggestion by m^2 enforced, muffinesque suggestion by Jim
Dempsey enforced.
Decompressing 64746363 bytes ...
RAM-to-RAM performance: 704 MB/s.
Memory pool starting address: 00000000042F0080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 280895 clocks or 1.866MB
per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 37%

D:\Nakamichi_Tengu>Nakamichi_Tengu_GP.exe enwik8.Nakamichi /report
Nakamichi 'Tengu', written by Kaze, based on Nobuo Ito's LZSS source,
babealicious suggestion by m^2 enforced, muffinesque suggestion by Jim
Dempsey enforced.
Decompressing 40860927 bytes ...
RAM-to-RAM performance: 512 MB/s.
Memory pool starting address: 0000000002B50080 ... 64 byte aligned, OK
Copying a 512MB block 1024 times i.e. 512GB READ + 512GB WRITTEN ...
memcpy(): (512MB block); 524288MB copied in 279507 clocks or 1.876MB
per clock
RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): 27%

D:\Nakamichi_Tengu>..\DDETT\LZ4.exe -9 -Sx -b -T1 enwik8
Nb of threads = 1 ; Compression Level = 9
enwik8  : 100000000 ->  42283793 ( 42.28%),   14.6 MB/s ,  745.5 MB/s

D:\Nakamichi_Tengu>..\DDETT\LZ4.exe -9 -Sx -b -T1 OSHO.TXT
Nb of threads = 1 ; Compression Level = 9
OSHO.TXT: 206908949 ->  71399094 ( 34.51%),   11.5 MB/s ,  747.3 MB/s
```