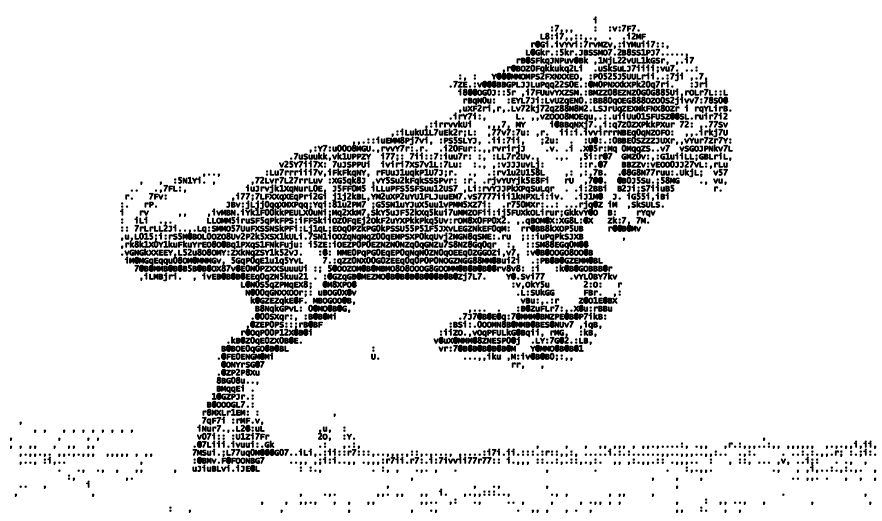
# 中道悪馬 - *NAKAMICHI 'Akuuma' a.k.a. 'Mustang'*

Fast Outlawish (freefull) LZSS decompressor: *http://www.sanmayce.com/Hayabusa/index.html*



```
uint64_t Decompress (char* ret, char* src, uint64_t srcSize) {
                char* retLOCAL = ret;  char* srcLOCAL = src;
                char* srcEndLOCAL = src+srcSize;
                unsigned int DWORDtrio; uint64_t Flag; //FIX
                uint64_t FlagMASK; //=       0xFFFFFFFFFFFFFFFF;
                uint64_t FlagMASKnegated; //=0x0000000000000000;
                while (srcLOCAL < srcEndLOCAL) {
                                DWORDtrio = *(unsigned int*)srcLOCAL;
// |1stLSB    |2ndLSB  |3rdLSB   |
// ---------------------------------
// |OO|LL|xxxx|xxxxxxxx|xxxxxx|xx|
// ---------------------------------
// [1bit         16bit]    24bit]
// OOLL = 0000 means Literal - to ease the initial check (branchless also)
// LL = 00b means 00 MatchLength, (2+OO)+LL or 2+0 + 0|4|8|12 = 2|6|10|14
// LL = 01b means 04 MatchLength, (2+OO)+LL or 2+1 + 0|4|8|12 = 3|7|11|15
// LL = 10b means 08 MatchLength, (2+OO)+LL or 2+2 + 0|4|8|12 = 4|8|12|16
// LL = 11b means 12 MatchLength, (2+OO)+LL or 2+3 + 0|4|8|12 = 5|9|13|17
// OO = 00b MatchOffset, 0xFFFFFFFF>>(3-OO), 1 bytes long i.e. Sliding Window is 1*8-LL-OO=(1+OO)*8-4=04 or   16B
// OO = 01b MatchOffset, 0xFFFFFFFF>>(3-OO), 2 bytes long i.e. Sliding Window is 2*8-LL-OO=(1+OO)*8-4=12 or   4KB
// OO = 10b MatchOffset, 0xFFFFFFFF>>(3-OO), 3 bytes long i.e. Sliding Window is 3*8-LL-OO=(1+OO)*8-4=20 or   1MB
// OO = 11b MatchOffset, 0xFFFFFFFF>>(3-OO), 4 bytes long i.e. Sliding Window is 4*8-LL-OO=(1+OO)*8-4=28 or 256MB
// (2+0 +  0):1 =  2:1 priority #12              #01 14:1 = 14
// (2+0 +  4):1 =  6:1 priority #04              #02 10:1 = 10
// (2+0 +  8):1 = 10:1 priority #02              #03 15:2 =  7.5
// (2+0 + 12):1 = 14:1 priority #01              #04  6:1 =  6
// (2+1 +  0):2 =  3:2 priority #13              #05 11:2 =  5.5
// (2+1 +  4):2 =  7:2 priority #08              #06 16:3 =  5.3
// (2+1 +  8):2 = 11:2 priority #05              #07 12:3 =  4
// (2+1 + 12):2 = 15:2 priority #03              #08  7:2 =  3.5
// (2+2 +  0):3 =  4:3 priority #14              #09 13:4 =  3.2
// (2+2 +  4):3 =  8:3 priority #10              #10  8:3 =  2.6
// (2+2 +  8):3 = 12:3 priority #07              #11  9:4 =  2.2
// (2+2 + 12):3 = 16:3 priority #06              #12  2:1 =  2 (NOT
// (2+3 +  0):4 =  5:4 priority #15              #13  3:2 =  1.5
// (2+3 +  4):4 =  9:4 priority #11              #14  4:3 =  1.3
// (2+3 +  8):4 = 13:4 priority #09              #15  5:4 =  1.2
// (2+3 + 12):4 = 17:4 not used in 'Mustang'
                                                              #ifdef _N_Branchfull
// Branchfull [
                                if ( (DWORDtrio & 0x0F) == 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>>4)&0x0F);
                                srcLOCAL+= ((DWORDtrio>>4)&0x0F)+1;
                                } else {
                                DWORDtrio = DWORDtrio&( 0xFFFFFFFF >> ((3-(DWORDtrio & 0x03))<<3) );
                                                              #ifdef _N_GP
                                                memcpy(retLOCAL, (const char *)( (uint64_t)(retLOCAL-(DWORDtrio>>4)) ), 16);
                                                              #endif
                                                              #ifdef _N_XMM
                                                SlowCopy128bit( (const char *)( (uint64_t)(retLOCAL-(DWORDtrio>>4)) ), retLOCAL );
                                                              #endif
                                srcLOCAL+= 1+(DWORDtrio&0x03); // 4|3|2|1
                                //retLOCAL+= 2+(DWORDtrio&0x03) + (DWORDtrio&0x0C); // 2/3/4/5/6/7/8/9/10/11/12/13/14/15/16 // Hoshimikou
                                retLOCAL+= 2+(DWORDtrio&0x0F); // 2/3/4/5/6/7/8/9/10/11/12/13/14/15/16 // Hoshimikou
                                }
// Branchfull ]
                                                              #endif
                                                              #ifdef _N_Branchless
// Branchless [
                                Flag=!(DWORDtrio & 0x0F);
                                // In here Flag=0|1
                                FlagMASKnegated= Flag - 1; // -1|0
                                FlagMASK= ~FlagMASKnegated; // if Flag is 1 then FlagMASK is -1/True
                                DWORDtrio = DWORDtrio&( 0xFFFFFFFF >> ((3-(DWORDtrio & 0x03))<<3) );
                                                              #ifdef _N_XMM
//                              SlowCopy128bit( (const char *)( ((uint64_t)(srcLOCAL+1)&FlagMASK) + ((uint64_t)(retLOCAL-(DWORDtrio>>4))&FlagMASKnegated) ), retLOCAL);
// Another (incompatible with Branchfull variant, though) way to avoid 'LEA' is to put the '+1' outside the FlagMASK but then the encoder has to count literals from zero in order to compensate '-((DWORDtrio>>4)-1) = -(DWORDtrio>>4)+1' within FlagMASKnegated:
                                SlowCopy128bit( (const char *)( 1+ ((uint64_t)(srcLOCAL)&FlagMASK) + ((uint64_t)(retLOCAL-(DWORDtrio>>4)-1)&FlagMASKnegated) ), retLOCAL);
                                                              #endif
                                                              #ifdef _N_GP
//                              memcpy(retLOCAL, (const char *)( ((uint64_t)(srcLOCAL+1)&FlagMASK) + ((uint64_t)(retLOCAL-(DWORDtrio>>4))&FlagMASKnegated) ), 16);
// Another (incompatible with Branchfull variant, though) way to avoid 'LEA' is to put the '+1' outside the FlagMASK but then the encoder has to count literals from zero in order to compensate '-((DWORDtrio>>4)-1) = -(DWORDtrio>>4)+1' within FlagMASKnegated:
                                memcpy(retLOCAL, (const char *)( 1+ ((uint64_t)(srcLOCAL)&FlagMASK) + ((uint64_t)(retLOCAL-(DWORDtrio>>4)-1)&FlagMASKnegated) ), 16);
                                                              #endif
                                retLOCAL+= ((uint64_t)((DWORDtrio>>4))&FlagMASK) +   ((uint64_t)(2+(DWORDtrio&0x0F))&FlagMASKne
                                srcLOCAL+= ((uint64_t)((DWORDtrio>>4)+1)&FlagMASK) + ((uint64_t)(1+(DWORDtrio&0x03))&FlagMASKne
// Branchless ]
                                                              #endif
                }
                return (uint64_t)(retLOCAL - ret);
}
```

**outlaw**:

- contrary to or forbidden by law; "an illegitimate seizure of power".
- disobedient to or defiant of law.
- a person who rebels against established rules or practices; nonconformist.
- *Western U.S.*:
  a. a horse that cannot be broken.
  b. any rogue animal.
- A wild or vicious horse or other animal.
- A rebel; a nonconformist: *a social outlaw.*

Source: *http://www.thefreedictionary.com/outlaw*

On Core 2 Q9550s @2.83GHz:

```
D:\_KAZE\Nakamichi_Mustang>Nakamichi_Mustang_branchfull.exe enwik8.Nakamichi /report
Decompressing 35453517 bytes ...
RAM-to-RAM performance: 256 MB/s.

D:\_KAZE\Nakamichi_Mustang>Nakamichi_Mustang_branchfull.exe Agatha_Christie_89-ebooks_TXT.tar.Nakamichi /report
Decompressing 11006648 bytes ...
RAM-to-RAM performance: 320 MB/s.

D:\_KAZE\Nakamichi_Mustang>zstd.exe -b enwik8
enwik8           : 100000000 ->  39573323 (39.57%),  107.7 MB/s ,  324.6 MB/s

D:\_KAZE\Nakamichi_Mustang>zstd.exe -b Agatha_Christie_89-ebooks_TXT.tar
Agatha_Christie_ :  33258496 ->  12404504 (37.30%),  105.1 MB/s ,  304.5 MB/s
```

If **1111b** is used as literal tag (instead of **0000b**) then compression ratio is slightly bettered, but speed is hurt a bit.

```
; 'Nakamichi_Mustang_branchfull' decompression loop:
; Size in bytes: 7c-14+2=106
; Size in instructions: 34
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for
applications running on Intel(R) 64, Version 15.0.0.108 Build 20140";
; mark_description "-O3 -QxSSE2 -D_N_XMM -D_N_prefetch_4096 -
D_N_Branchfull -FAcs";
.B7.3::
  00014 41 8b 11         mov edx, DWORD PTR [r9]
  00017 f6 c2 0f         test dl, 15
  0001a 75 20            jne .B7.5
.B7.4::
  0001c c1 ea 04         shr edx, 4
  0001f 89 d1            mov ecx, edx
  00021 83 e2 0f         and edx, 15
  00024 f3 41 0f 6f 41
        01               movdqu xmm0, XMMWORD PTR [1+r9]
  0002a ff c2            inc edx
  0002c 48 83 e1 0f      and rcx, 15
  00030 f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
  00034 48 03 c1         add rax, rcx
  00037 4c 03 ca         add r9, rdx
  0003a eb 3d            jmp .B7.6
.B7.5::
  0003c 89 d1            mov ecx, edx
  0003e 41 bb ff ff ff
        ff               mov r11d, -1
  00044 83 f1 03         xor ecx, 3
  00047 c1 e1 03         shl ecx, 3
  0004a 41 d3 eb         shr r11d, cl
  0004d 41 23 d3         and edx, r11d
  00050 89 d1            mov ecx, edx
  00052 41 89 d3         mov r11d, edx
  00055 c1 e9 04         shr ecx, 4
  00058 41 83 e3 03      and r11d, 3
  0005c 83 e2 0f         and edx, 15
  0005f 48 f7 d9         neg rcx
  00062 41 ff c3         inc r11d
  00065 48 03 c8         add rcx, rax
  00068 83 c2 02         add edx, 2
  0006b 4d 03 cb         add r9, r11
  0006e f3 0f 6f 01      movdqu xmm0, XMMWORD PTR [rcx]
  00072 f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
  00076 48 03 c2         add rax, rdx
.B7.6::
  00079 4d 3b ca         cmp r9, r10
  0007c 72 96            jb .B7.3

; 'Nakamichi_Mustang_branchless' decompression loop:
; Size in bytes: bc-3a+6=136
; Size in instructions: 40
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for
applications running on Intel(R) 64, Version 15.0.0.108 Build 20140";
; mark_description "-O3 -QxSSE2 -D_N_XMM -D_N_prefetch_4096 -
D_N_Branchless -FAcs";
.B7.3::
                                30         mov r14d, DWORD PTR [r8]
                                f1         mov ecx, r14d
                                03         xor ecx, 3
                                ff ff ff
                                           mov r15d, -1
                                03         shl ecx, 3
                                ed         xor r13d, r13d
                                c6 0f 00
                                           test r14d, 15
                                44 ea      cmove r13d, edx
                                ef         shr r15d, cl
                                c1         mov rcx, rax
  00060 45 23 f7                           and r14d, r15d
  00063 4d 89 c7                           mov r15, r8
  00066 44 89 f5                           mov ebp, r14d
  00069 c1 ed 04                           shr ebp, 4
  0006c 49 ff cd                           dec r13
  0006f 41 89 eb                           mov r11d, ebp
  00072 4d 89 ec                           mov r12, r13
  00075 49 2b cb                           sub rcx, r11
  00078 49 f7 d4                           not r12
  0007b 48 ff c9                           dec rcx
  0007e 4d 23 fc                           and r15, r12
  00081 49 23 cd                           and rcx, r13
  00084 ff c5                              inc ebp
  00086 4d 23 dc                           and r11, r12
  00089 49 23 ec                           and rbp, r12
  0008c f3 42 0f 6f 44
        39 01                              movdqu xmm0, XMMWORD PTR [1+rcx+r15]
  00093 44 89 f1                           mov ecx, r14d
  00096 41 83 e6 03                        and r14d, 3
  0009a 83 e1 0f                           and ecx, 15
  0009d 41 ff c6                           inc r14d
  000a0 83 c1 02                           add ecx, 2
  000a3 4d 23 f5                           and r14, r13
  000a6 49 23 cd                           and rcx, r13
  000a9 49 03 ee                           add rbp, r14
  000ac 4c 03 d9                           add r11, rcx
  000af 4c 03 c5                           add r8, rbp
  000b2 f3 0f 7f 00                        movdqu XMMWORD PTR [rax], xmm0
  000b6 49 03 c3                           add rax, r11
  000b9 4d 3b c2                           cmp r8, r10
  000bc 0f 82 78 ff ff
        ff                                 jb .B7.3
```

```
07/26/2015  11:55 PM        10,192,446 dickens
08/05/2015  07:48 AM         3,776,411 dickens.Nakamichi
08/05/2015  04:32 AM         3,681,828 dickens.zip          ! 7za a -tgzip -mx9 dickens.zip dickens !

On Core 2 Q9550s @2.83GHz laptop:

D:\_KAZE\Nakamichi_Mustang>Nakamichi_Mustang_branchfull.exe dickens.Nakamichi /report
Decompressing 3776411 bytes ...
RAM-to-RAM performance: 448 MB/s.

D:\_KAZE\Nakamichi_Mustang>\lz4 -9 -b -Sx -T1 dickens
Nb of threads = 1 ; Compression Level = 9
dickens          : 10192446 ->   4442965 ( 43.59%),   11.8 MB/s ,  896.2 MB/s
```