

The 68000 assembly code of Darin Wayrynen from the file ST\_UNZIP.ZIP was used in UnZip 2.0 (modified to avoid trashing of register D3). The full text of the original files "CRC32.H" and "CRC32.C" from ST\_UNZIP.ZIP follow:

CRC32.H:

```
long crc32val;
```

```
/*#ifndef TEST
extern void UpdateCRC();
#endif*/
/* update running CRC calculation with contents of a buffer */
```

CRC32.C:

```
#define ATARI_ST 1
#ifndef ATARI_ST
#define HIGH_LOW 1
typedef unsigned char byte;
/* code assumes UNSIGNED bytes */
typedef long longint;
typedef unsigned word;
typedef char boolean;
#endif

/* ===== */
/* COPYRIGHT (C) 1986 Gary S. Brown. You may use this program, or */
/* code or tables extracted from it, as desired without restriction. */
/* */
/* First, the polynomial itself and its table of feedback terms. The */
/* polynomial is */
/*  $X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X^1+X^0$  */
/* */
/* Note that we take it "backwards" and put the highest-order term in */
/* the lowest-order bit. The  $X^{32}$  term is "implied"; the LSB is the */
/*  $X^{31}$  term, etc. The  $X^0$  term (usually shown as "+1") results in */
/* the MSB being 1. */
/* */
/* Note that the usual hardware shift register implementation, which */
/* is what we're using (we're merely optimizing it by doing eight-bit */
/* chunks at a time) shifts bits into the lowest-order term. In our */
/* implementation, that means shifting towards the right. Why do we */
/* do it this way? Because the calculated CRC must be transmitted in */
/* order from highest-order term to lowest-order term. UARTs transmit */
/* characters in order from LSB to MSB. By storing the CRC this way, */
/* we hand it to the UART in the order low-byte to high-byte; the UART */
/* sends each low-bit to high-bit; and the result is transmission bit */
/* by bit from highest- to lowest-order term without requiring any bit */
/* shuffling on our part. Reception works similarly. */
/* */
/* The feedback terms table consists of 256, 32-bit entries. Notes: */
/* */
/* The table can be generated at runtime if desired; code to do so */
/* is shown later. It might not be obvious, but the feedback */
/* terms simply represent the results of eight shift/xor opera- */
/* tions for all combinations of data and CRC register values. */
/* */
/* The values must be right-shifted by eight bits by the "updcrc" */
/* logic; the shift must be unsigned (bring in zeroes). On some */
/* hardware you could probably optimize the shift in assembler by */
/* using byte-swap instructions. */
/* */
```

```

/*      polynomial $edb88320
/*
/* ----- */
extern long crc32val;

long crc_32_tab[] =
0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL, 0x076dc419L,
0x706af48fL, 0xe963a535L, 0x9e6495a3L, 0x0edb8832L, 0x79dcb8a4L,
0xe0d5e91eL, 0x97d2d988L, 0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L,
0x90bf1d91L, 0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L, 0x136c9856L,
0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL, 0x14015c4fL, 0x63066cd9L,
0xfa0f3d63L, 0x8d080df5L, 0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L,
0xa2677172L, 0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
0x325b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L, 0x32d86ce3L,
0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L, 0x26d930acL, 0x51de003aL,
0xc8d75180L, 0xbfd06116L, 0x21b4f4b5L, 0x56b3c423L, 0xcfb9a959L,
0xb8bda50fL, 0x2802b89eL, 0xf5058808L, 0xc60cd9b2L, 0xb10be924L,
0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL, 0x76dc4190L,
0x01db7106L, 0x98d220bcL, 0xefd5102aL, 0x71b18589L, 0x06b6b51fL,
0x9fbfe4a5L, 0xe8b8d433L, 0x7807c9a2L, 0x0f00f934L, 0x9609a88eL,
0xe10e9818L, 0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
0xb6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL, 0x6c0695edL,
0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L, 0x65b0d9c6L, 0x12b7e950L,
0x8bbbeb8eaL, 0xfcb9887cL, 0x62dd1ddfl, 0x15da2d49L, 0x8cd37cf3L,
0xfbd44c65L, 0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL, 0x4369e96aL,
0x346ed9fcL, 0xad678846L, 0xda60b8d0L, 0x44042d73L, 0x33031de5L,
0xaa0a4c5fL, 0xdd0d7cc9L, 0x5005713cL, 0x270241aaL, 0xbe0b1010L,
0xc90c2086L, 0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L, 0x59b33d17L,
0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL, 0xedb88320L, 0x9abfb3b6L,
0x03b6e20cL, 0x74b1d29aL, 0xeada54739L, 0x9dd277afL, 0x04db2615L,
0x73dc1683L, 0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L, 0xf00f9344L,
0x8708a3d2L, 0x1e01f268L, 0x6906c2feL, 0xf762575dL, 0x806567cbL,
0x196c3671L, 0x6e6b06e7L, 0xfed41b76L, 0x89d32be0L, 0x10da7a5aL,
0x67dd4accL, 0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L, 0xd1bb67f1L,
0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL, 0xd80d2bdaL, 0xaf0a1b4cL,
0x36034af6L, 0x41047a60L, 0xdf60efc3L, 0xa867df55L, 0x316e8eefL,
0x4669be79L, 0xcb6b1b38L, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL, 0xc5ba3bbeL,
0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L, 0xc2d7ffa7L, 0xb5d0cf31L,
0x2cd99e8bL, 0x5bdeae1dL, 0x9b64c2b0L, 0xec63f226L, 0x756aa39cL,
0x026d930aL, 0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L, 0x92d28e9bL,
0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L, 0x86d3d2d4L, 0xf1d4e242L,
0x68ddb3f8L, 0x1fda836eL, 0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L,
0x18b74777L, 0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L, 0xa00ae278L,
0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L, 0xa7672661L, 0xd06016f7L,
0x4969474dL, 0x3e6e77dbL, 0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L,
0x37d83bf0L, 0xa9bcaee5L, 0xddebb9ecL, 0x47b2cf7fL, 0x30b5ffe9L,
0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L, 0xbad03605L,
0xcdd70693L, 0x54de7729L, 0x23d967bfL, 0xb3667a2eL, 0xc4614ab8L,
0x5d681b02L, 0x2a6f2b94L, 0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL,
0x2d02ef8dL
;
/*

```

```

* IBM PC Inline assembly version of 32 bit CRC calculation
* Copyright 1989 Samuel H. Smith
*
* You may use this program, or code extracted from it,
* as desired without restriction.
*
*/

```

```

#ifndef ATARI_ST
UpdateCRC(s,len)
register unsigned char *s;
register int len;
/* update running CRC calculation with contents of a buffer */

```

```

#ifdef HIGH_LOW

```

```

register long a = crc32val;

```

```

extern long crc_32_tab[];

```

```

while (len--)

```

```

    a=crc_32_tab[(byte)a^(byte)*s++] ^ ((a>>8) & 0x0FFFFFFL);

```

```

    crc32val = a;
#else

```

```

    asm    push cx
    asm    push si

```

```

    asm    mov cx,len

```

```

    asm    les si,s

```

```

    asm    mov dx,crc32val+2
    asm    mov ax,crc32val

```

```

crcNext:

```

```

    asm    mov bh,al    /* save crc32val[0] */

```

```

    asm    mov al,ah    /* (crc32 >> 8) & 0x00ffff */
    asm    mov ah,dl
    asm    mov dl,dh
    asm    xor dh,dh

```

```

    asm    mov bl,es:[si]
    asm    inc si

```

```

    asm    xor bl,bh    /* crcval[0] */

```

```

    asm    xor bh,bh
    asm    shl bx,1
    asm    shl bx,1
    asm    xor ax,crc_32_tab[bx]
    asm    xor dx,crc_32_tab[bx+2]

```

```

    asm    loop crcNext

```

```

        asm      mov crc32val+2,dx
        asm      mov crc32val,ax
        asm      pop si
        asm      pop cx
#endif

#endif

#ifndef ATARI_ST
#define UPDCRC32(res,oct) res=crc_32_tab[(byte)res^(byte)oct] ^ ((res>>8) & 0x0FFFFFFFL)
#endif

/*
 * macro UPDCRC32(res,oct)
 *  res=crc_32_tab[(byte)res ^ (byte)oct] ^ ((res >> 8) & 0x0FFFFFFFL)
 *
 */

/* ----- */

extern int UpdateCRC()

/*
 * ATARI ST Inline assembly version of 32 bit CRC calculation
 * Copyright 1989 Darin Wayrynen
 *
 */

asm

UpdateCRC:

move.l
4(A7), A1

;address of data to compute crc on

move.w
8(A7), D3

;length of data

move.l
crc32val, D2

;global crc value

```

```
lea.l  
crc_32_tab,A0
```

```
;address of crc table
```

```
subq.w  
#1,D3
```

```
;subtract 1 for dbf loop
```

```
@top_loop:
```

```
clr.w  
D0
```

```
move.b  
D2,D0
```

```
;lower byte of global crc
```

```
clr.w  
D1
```

```
move.b  
(A1)+,D1
```

```
;first byte of data
```

```
eor.w  
D1,D0
```

```
;exclusive or with crc
```

```
add.w
```

D0,D0

;2 \* result

add.w  
D0,D0

;4 \* result (for long index)

move.l  
0(A0,D0.w),D1

;crc value from table

clr.b  
D2

;clear lower byte

ror.l  
#8,D2

;32 bit -> 24 bit (upper byte clear)

eor.l  
D1,D2

;exclusive or with global crc

dbf  
D3,@top\_loop

```
;loop until end of data
```

```
move.l  
D2, crc32val
```

```
;update global crc value
```

```
rts
```