White Paper

Vinodh Gopal

Jim Guilford

Erdinc Ozturk

Gil Wolrich

Wajdi Feghali

Martin Dixon

IA Architects

Intel Corporation

Deniz Karakoyunlu

PhD Worcester Polytechnic Institute

Fast CRC Computation for iSCSI Polynomial Using CRC32 Instruction

April 2011

323405

Executive Summary

Cyclic Redundancy Check (CRC) codes are widely used for integrity checking of data in fields such as storage and networking. There is an ever-increasing need for very high-speed CRC computations on processors for end-to-end integrity checks. We present fast and efficient methods of computing CRC on Intel processors for the fixed (degree-32) iSCSI polynomial, using the CRC32 instruction present in the Intel® Core™ i5 processor 650 .

Instead of computing CRC of the entire message with a traditional linear method, we use a faster method to split an arbitrary length buffer to a number of smaller fixed size segments, compute the CRC on these segments in parallel followed by a recombination step of computing the effective CRC using the partial CRCs of the segments.

Parallelized CRC computation is used to maximize the throughput of the CRC32 instruction. We show an efficient method for data buffers of arbitrary length.

The final recombination of CRCs adds an overhead and can be implemented with lookup tables on the Nehalem microarchitecture – we show how to do this with as few tables as possible while giving excellent overall performance on the range of sizes. The PCLMULQDQ instruction in the Westmere microarchitecture allows efficient recombination of CRCs without lookup tables. The various methods are thoroughly explained in this paper with real code examples.

These functions work across an arbitrary range of buffer sizes guaranteeing excellent performance across the range, achieving nearly 3X the performance of a linear implementation of CRC32. For instance, a single core of an Intel® Core™ i5 processor 650 can compute the CRC of a 1024-byte buffer at the rate of 0.145 cycles/byte with a single thread! [1]

The paper will enable customers to code and optimize their iSCSI CRC applications for maximum performance on Intel® processors. We use real optimized code examples in the paper.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. www.intel.com/embedded/edc.

§

[1] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section Performance of CRC32 for configurations and tests used. Testing conducted by Intel. For more information go to http://www.intel.com/performance.

Contents

Overview

A Cyclic Redundancy Check (CRC) is the remainder, or residue (typically 32 bits), of binary division of a potentially long message, by a CRC polynomial [7]. This technique is ubiquitously employed in communication and storage applications due to its effectiveness at detecting errors.

There is an increasing need to perform CRC operations on processor cores for end-to-end integrity at very high speeds [8][10][11]. In this paper we show how Intel processors can accelerate CRC computation with a fixed CRC polynomial [9] of degree 32 – the iSCSI CRC (0x11EDC6F41), using the CRC32 instruction present in the Intel® Core™ i5 processor 650 (based on the Westmere microarchitecture). We show methods that can compute the CRC of a single buffer of data at extremely high speeds, e.g., ~0.145 cycles/byte on a single core of an Intel® Core™ i5 650 processor. A related paper, [2] describes efficient algorithms on Intel® processors for generic polynomials, but for this paper, we focus on the fixed iSCSI polynomial.

CRC Preliminaries

The CRC32 instruction definition can be found in [1]. The instruction operates on two operands, the accumulated CRC value and the new data, and updates the accumulated CRC with computations based on the new data. The instruction can operate on a maximal data size of 64 bits (a Qword). In the Intel® Core™ i5 Processors, the instruction is implemented with a latency of 3 cycles and a throughput of 1 cycle. This implies that a traditional linear method to compute CRC of a buffer, will achieve about a third of the optimal performance.

Detecting the CRC32 Instruction

CRC32 belongs to the SSE4.2 family of instructions. To test for its presence, a programmer executes the CPUID instruction with EAX = 1, and then tests bit 20 of ECX on the output.

Microsoft's Visual Studio 2010 and GNU Compiler Collection (gcc) both provide easy mechanisms to query the CPUID instruction from C. In Visual Studio this is provided by an intrinsic from <intrin.h>. Sample code is available from Microsoft [3]. In the gcc, <cpuid.h> includes code to query the CPUID instruction. On gcc version 4.3.4, the following code sequence could be used to test for the presence of SSE 4.2:

```
#include <cpuid.h>
#include <stdio.h>

void
main () {
  unsigned int eax, ebx, ecx, edx;

  __get_cpuid(1, &eax, &ebx, &ecx, &edx);

  if (ecx & bit_SSE4_2)
    printf ("SSE4.2 is supported\n");

  return;
}
```

PCLMULQDQ Instruction

The PCLMULQDQ instruction performs carry-less multiplication of two 64-bit Qwords which are selected from the first and the second (128-bit) operands according to the immediate byte value [12].

Versions of the Algorithms

Four major versions of the algorithms/code are described in this paper. One pair only uses lookup tables for recombination, and can therefore run on the Nehalem microarchitecture. The second pair uses PCLMULQDQ and is designed to run on Westmere.

Within each pair, one operates on an arbitrary-sized buffer, and one operates on a fixed 1024-byte buffer. The latter illustrates that if the buffer size is fixed and known in advance, CRC code can be written for that length which avoids some of the overhead of the general version, resulting in smaller and faster code.

Table 1. CRC Code Versions

Versions	Lookup Table Based	PCLMULQDQ Based
Variable Buffer Size	CRC_216 CRC_240	CRC_PCL
Fixed 1K Buffer Size	CRC_1024	CRC_1024_PCL

Note: There are actually two minor versions for variable buffer size for the lookup table based code. This illustrates that although these two versions work on arbitrary sized buffers, if most of the buffers are of a particular size, then these functions can be "tuned" to operate more efficiently at this size (by varying the internal buffer sizes).

For example, if many of the data buffers are 900 bytes long, then CRC_216 performs faster than CRC_240, but if most of the data buffers are 1000 bytes long, then CRC_240 is faster. This will be described in detail in the performance results section.

Perform CRC of Multiple Parts of a Buffer

The basic concepts in this paper are derived from and explained in detail in the patents and pending applications [4][5][6]. If we divide the buffer into three non-overlapping parts, we can perform three CRC calculations in parallel. Since the calculation for each part is independent of the calculations for the other parts, each set of three CRC32 instructions is no longer dependent and can execute at the throughput rate.

At the end of this process, we have three separate CRC values. These need to be merged into a single value. This is the recombination "overhead" that we need to pay in order to process the buffer in parallel. Recombination can be done using lookup tables or carry-less multiplication operations.

Code / Data Sizes

The lookup table based versions tend to be larger than the corresponding PCLMULQDQ based versions, i.e., use of the PCLMULQDQ instruction mostly saves on data size. In the case of variable buffer sizes, the PCLMULQDQ instruction also allows us to take a different approach, which gives slightly better and more regular performance.

Table 2. CRC Code Version Sizes

Versions	Size in Bytes		
	Code Size	Data Size	Total Size
CRC_216	4,945	5,120	10,065
CRC_240	5,395	5,120	10,515
CRC_PCL	4,130	2,306	6,436
CRC_1024	1,376	2,048	3,424
CRC_1024_PCL	1,305	32	1,337

Performance of CRC32

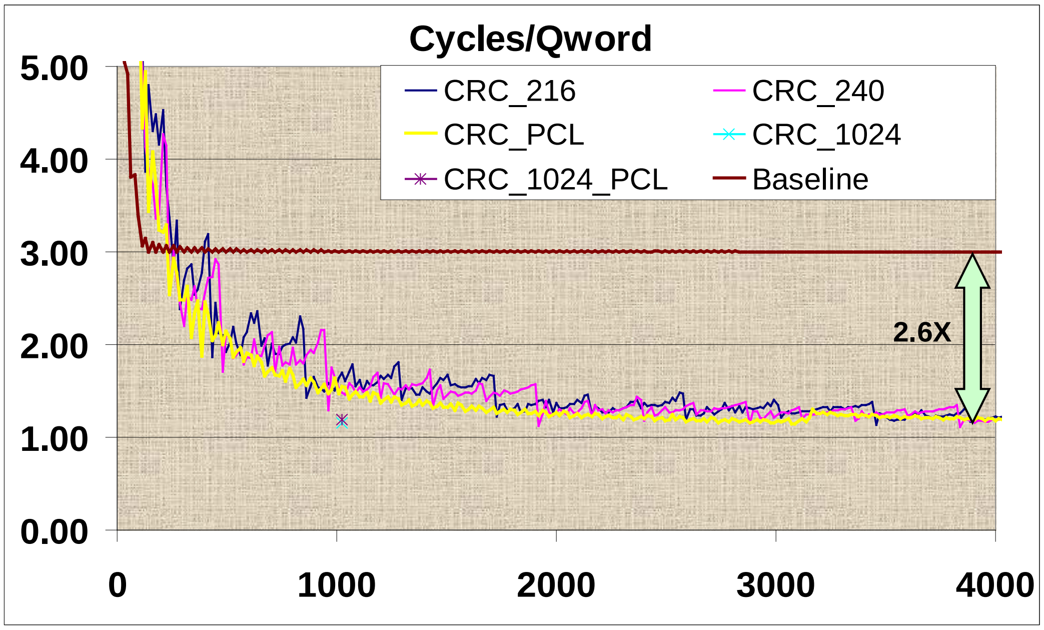
The performance results provided in this section were measured on an Intel® Core™ i5 processor 650 at a frequency of 3.20 GHz, supporting Intel PCLMULQDQ instruction. The tests were run with Intel® Turbo Boost Technology off, and represent the performance without Intel® Hyper-Threading Technology (Intel® HT Technology) on a **single core**.

For the fixed-size versions, for each run, we compute the CRC of the same data buffer 350,000 times. For the variable-size versions, the size of the buffer was varied from 16 up to 4096 by 16, and then back down to 16. This was repeated numerous times per run.

The CRC output of one calculation is used as the initial CRC for the next calculation to serialize them. The timing is measured using the **rdtsc**() function which returns the processor time stamp counter (TSC). The TSC is the number of clock cycles since the last reset. We sample the processors' TSC before and after the run, to get the number of timestamps per run. We then perform 256 runs, discarding the 64 fastest and 64 slowest times, and use the mean of the remaining 128 values.

The results are shown in Figure 1 as Cycles/Qword as a function of size in bytes:

Figure 1. Cycles/Qword Performance[2]



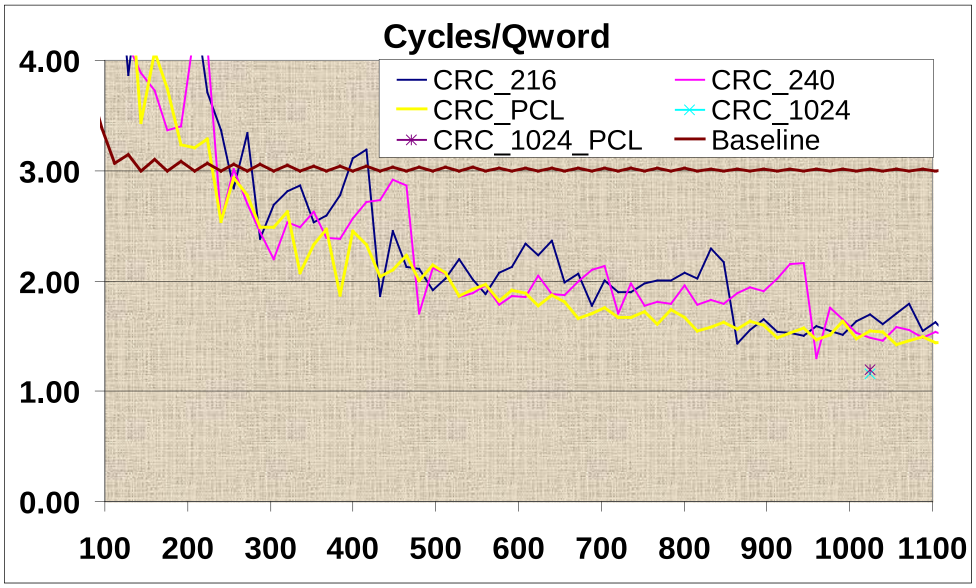
This shows that the performance for the variable-size version tends to about 1.2 cycles/Qword, as compared to 3.0 cycles/Qword when using the baseline/linear CRC32 implementation.

For the variable input size versions, the PCLMULQDQ version (CRC_PCL) gives slightly better performance and more regular performance. The curves for the lookup table versions (CRC_216 and CRC_240) show some amount of jaggedness, due to use of a small number of fixed-size segments.

[2] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section Performance of CRC32 for configurations and tests used. Testing conducted by Intel. For more information go to http://www.intel.com/performance.

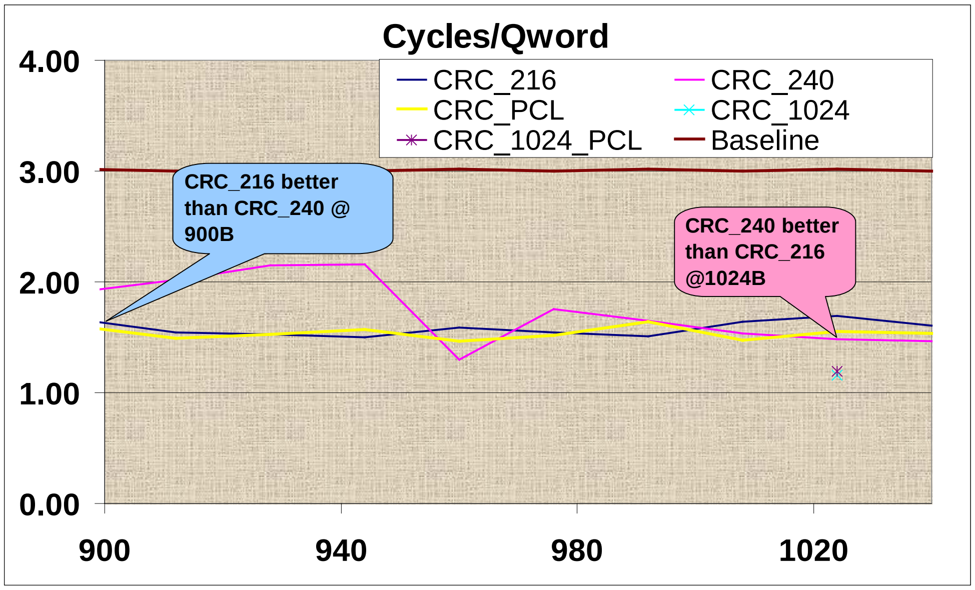
Figure 2 and Figure 3 show a close up:

Figure 2. Cycles/Qword Performance, Part A (Zoomed)[3]



[3] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section Performance of CRC32 for configurations and tests used. Testing conducted by Intel. For more information go to http://www.intel.com/performance.

Figure 3. Cycles/Qword Performance, Part B (Zoomed)[4]



For the versions without PCLMULQDQ for a variable-sized input, even though they handle all input sizes, the sizes of the internal buffers can be chosen to optimize performance at certain points. For example, if you needed to process any size buffer, but 900 byte buffers were particularly common, then CRC_216 would give better results than CRC_240.

On the other hand, if the size of the buffer is completely fixed, e.g., 1024 bytes, then a CRC implementation designed for that particular size will perform better (1.16 cycles/Qword) than the arbitrary size version.

Finally, the table lookup version for the fixed-sizes, e.g., CRC_1024, is marginally faster than the version using PCLMULQDQ! However, the PCLMULQDQ version uses significantly less memory, which may be of interest in certain applications.

[4] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section Performance of CRC32 for configurations and tests used. Testing conducted by Intel. For more information go to http://www.intel.com/performance.

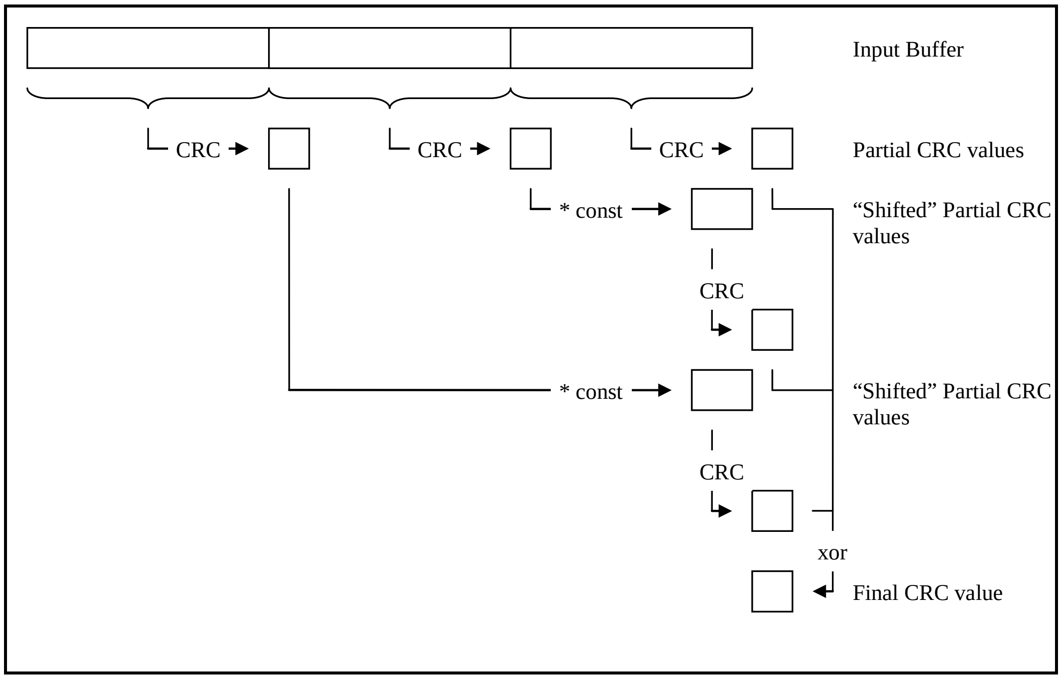
Implementation Details

This section discusses some of the internal details of these different versions and illustrates how the basic ideas can be applied. First it will describe the basic concepts in a generic sense, and then it will discuss the individual versions.

Basic Ideas in Detail

The basic idea is illustrated in Figure 4:

Figure 4. Performing CRC on Segments of a Buffer



The input data is divided into three equal-sized segments, and the CRC is computed in parallel for each segment. Call the three values (from left to right) crc0, crc1, and crc2. The value crc2 is in the proper bit location, but the other two values need to be shifted into the correct bit position. This can be done by carry less multiplication of crc0 and crc1 by appropriate constants. This multiplication "shifts" the value towards the right, but it also increases the size to 64 bits. For this reason, we cannot simply use the multiplication to "shift" crc0 and crc1 into the same bit position as crc2.

Instead, we shift crc0 and crc1 into a position 32 bits to the left of crc2. Then we take the CRC of each of these shifted values. This results in a 32-bit value that is shifted 32-bits to the right, to align with the correct bit position. We can then XOR these two new CRC values to crc2 to get the final CRC value.

CRC_1024_PCL

Perhaps the simplest to understand is CRC_1024_PCL, i.e., the version for a fixed 1024-byte buffer using explicit multiplies (PCLMULQDQ). The picture here would be very similar to Figure 4.

A 1024-byte (128 Qword) buffer can be broken into a most-significant segment of 43 Qwords, followed by two segments of 42 Qwords each, followed by a final and least significant Qword.

Each of the three segments can be processed in parallel, resulting in three partial CRCs. The value crc2 is generated 1 Qword short; we therefore take the final Qword and this crc2 value and compute an additional CRC with the crc32 instruction, to generate a residue in the correct final bit position. The other two partial CRCs are processed as described in the previous section, except that they are shifted an extra Qword, so that their new CRC values appear at the correct final bit offset.

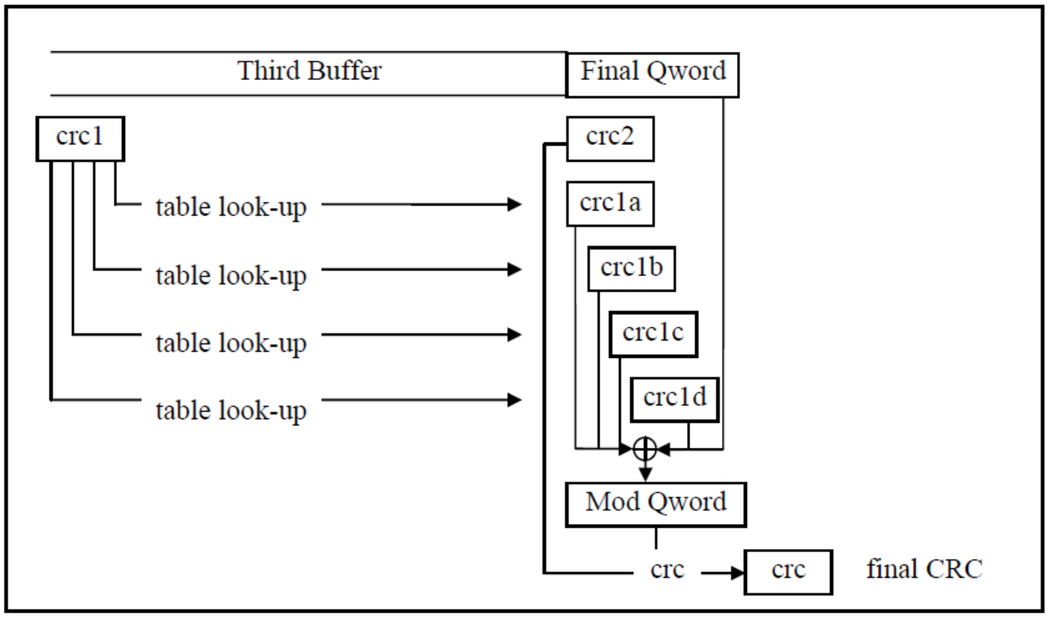
CRC_1024

The basic structure of this is the same as CRC_1024_PCL, except that it does not use PCLMULQDQ. Instead, a lookup table is used. The crc2 value is in the correct bit position with respect to the final Qword. The other two partial CRCs need to be “shifted” to the right, so that their new CRC values appear at the correct bit positions with respect to the final Qword. Since crc0 can be handled in the exact same way as crc1, we only describe crc1 in the rest of this section.

The “shifting” of a partial CRC value is done using a single lookup table. To keep the table size reasonable, the table only contains 256 entries, so that each byte of crc1 needs to be processed separately. Each table entry corresponding to a byte b, is the 32-bit result of applying CRC to a data buffer containing b shifted a fixed distance to the left. The table effectively moves the contribution of the byte to the right by a fixed number of bits. Since the four bytes are staggered by one byte, the results of the lookup operation are also staggered by one byte.

The details of the recombination of a partial CRC value are shown in Figure 5. This figure only considers crc1 and crc2. The value of crc0 is handled in the same manner as for crc1.

Figure 5. Recombining a Partial CRC



Doing the table lookup for one byte of crc1 gives crc1a, which is in the same bit position as crc2. However, the lookups for the other three bytes give results that are shifted (crc1b…crc1d). The four crc1x values are XOR'd to the final Qword, then a final CRC32 operation combines the modified final Qword with crc2 to give the final CRC value. The trade-off here is that we use just one table at the cost of doing some bit-shift operations to align the crc1x values correctly.

Another approach would be to use 4 different tables for the 4 bytes of crc1, each of which shifts the original byte down by 8 bits less than the previous table (e.g. crc1b would be shifted one byte less than crc1a). This results in four CRC values, each of which is in the same position. These can just be XOR'd together without requiring any aligning shift operations. However, to reduce the data size, we just use one table (for crc1) in our implementation.

CRC_216 / CRC_240

These versions handle arbitrary length input buffers without using PCLMULQDQ. Their structures are identical; the difference is in the sizes of the internal buffers and the values of the lookup tables. This section will describe CRC_240.

The basic approach is to divide up the input into a series of fixed-sized buffers, and then process each such buffer in a manner similar to the above. In the case of CRC_240, 1920 bytes of the input are processed until the size of the remaining bytes is less than 1920. It then processes 960 bytes, 480 bytes, and/or 240 bytes, depending on how many bytes remain after each step. Finally the remaining bytes, fewer than 240, are processed in a straight-forward linear manner.

The processing for each buffer is handled in a manner similar to that of CRC_1024 except that there is no "final Qword", since 1920 is divisible by 3. Instead, the processing of the third, and least significant, portion of the input, i.e., the processing that results in crc2, stops one Qword short. Then the recombination becomes identical to that of CRC_1024.

One further optimization is that the tables are defined by how many bits the value needs to be shifted down. This is 1/3 or 2/3 of the buffer size minus the 8 bytes to reflect that we are stopping one Qword early. So for the 240-byte buffer, we need tables that shift by 72 and 152 bytes. For the 480-byte buffer, we need tables that shift by 152 and 312 bytes, etc.

Note: In each case, the second table for a given buffer size is the same as the first table for the next larger buffer. That means that for four sizes of buffers, we only need five tables rather than eight.

The use of a small number of fixed buffer sizes is needed to keep the number of tables reasonable.

CRC_PCL

This version handles arbitrary length input buffers using PCLMULQDQ. It takes a totally different approach to CRC_240. Since each buffer size requires only two Qwords of data, it can handle many more sizes of buffer.

One threesome of CRC32 instructions handles 3 Qwords, or 24 bytes of data, so this becomes the fundamental granularity. It divides the input size by 24 to determine how many such threesomes need to be processed. This value is capped at 128 threesomes (3072 bytes). If the input buffer is greater than 3072 bytes, then it is processed 3072 bytes at a time until the size is less than or equal to 3072.

Based on the number of threesomes to be processed, control jumps to the appropriate point in a linear list of CRC32 instructions, which compute the partial CRCs for each third. This is also used to look up the appropriate multiplication constants in a table.

The three partial CRCs are then combined in a manner almost identical to that shown in Section "Basic Ideas in Detail", except that the two shifted values are added (XOR'd) to the final Qword in the buffer before a single final CRC is taken.

Finally the remaining bytes, fewer than 24, are handled in a linear manner.

As an optimization, if the buffer contains fewer than 96 bytes, it is handled in a simple linear fashion, to avoid paying the recombination penalty for very small buffers.

Conclusion

We developed optimized algorithms/code for computing the iSCSI CRC which unleashes the full potential of the CRC32 instruction. We have versions for both fixed-size and variable-sized buffers, as well as variations optimized for Intel® processors with and without the PCLMULQDQ instruction. On the Intel® Core™ i5 processor 650, these versions tend towards 1.16-1.2 cycles/Qword for large buffers, giving us about a 2.6X speed-up over the baseline linear method[5].

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. http://intel.com/embedded/edc.

[5] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section Performance of CRC32 for configurations and tests used. Testing conducted by Intel. For more information go to http://www.intel.com/performance.

References

[1] Intel(R) 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M http://www.intel.com/products/processor/manuals/

[2] Fast CRC Computation for Generic Polynomials Using PCLMULQDQ Instruction http://download.intel.com/design/intarch/papers/323102.pdf

[3] __cpuid, __cpuidex http://msdn.microsoft.com/en-us/library/hskdteyh.aspx

[4] Determining a Message Residue, Gopal et al. United States Patent 7,886,214

[5] Determining a Message Residue Gueron et al. United States Patent Application 20090019342

[6] Determining a Message Residue Gopal et al. United States Patent Application 20090158132

[7] "A Tutorial on CRC Computations", Ramabadran et al., Micro, IEEE, IEEE Computer Society, Aug. 1988, pp. 62-75.

[8] "High-Speed CRC Design for 10 Gbps applications", Lin et al., ISCAS 2006, IEEE, pp. 3177-3180.

[9] "Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks", Koopman et al., The International Conference on Dependable Systems and Networks, DSN- 2004, pp. 1-10.

[10] "Parallel CRC Realization", Campobello et al., IEEE Transactions on Computers, vol. 52, No. 10, Oct. 2003, Published by the IEEE Computer Society; pp. 1312-1319.

[11] "A Systematic Approach to Building High Performance Software- based CRC Generators", Kounavis et al., Proceedings of the 10th IEEE Symposium on Computers and Communications, ISCC 2005; pp. 855-862.

[12] Intel(R) 64 and IA-32 Architectures Software Developer's Manual http://www.intel.com/assets/PDF/manual/252046.pdf

Appendix A

We provide 64-bit code, written for the YASM assembler and the Microsoft* Windows* 64-bit ABI, for the various optimized functions described in the paper.

A.1 CRC_216

```
; Function to compute iscsi CRC32 with table-based recombination
; crc done "by 3" with block sizes 1728, 864, 432, 216

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; crcB3 MACRO to implement crc32 on 3 %%bSize-byte blocks
%macro  crcB3 3
%define %%bSize   %1    ; 1/3 of buffer size
%define %%td2     %2    ; table offset for crc0 (2/3 of buffer)
%define %%td1     %3    ; table offset for crc1 (1/3 of buffer)

%IF %%bSize=576
        sub rdx, %%bSize*3
        js %%crcB3_end      ;; jump to next level if 3*blockSize > len
%ELSE
        cmp rdx, %%bSize*3
        jnae %%crcB3_end    ;; jump to next level if 3*blockSize > len
%ENDIF
        ;;;;;; Calculate CRC of 3 blocks of the buffer ;;;;;;
%%crcB3_loop:
                                        ;; rax = crc0 = initial crc
        xor     rbx, rbx                ;; rbx = crc1 = 0;
        xor     r10, r10                ;; r10 = crc2 = 0;

 %assign i 0
 %rep %%bSize/8 - 1
        crc32 rax, [rdi+i + 0*%%bSize]  ;; update crc0
        crc32 rbx, [rdi+i + 1*%%bSize]  ;; update crc1
        crc32 r10, [rdi+i + 2*%%bSize]  ;; update crc2
        %assign i (i+8)
 %endrep
        crc32 rax, [rdi+i + 0*%%bSize]  ;; update crc0
        crc32 rbx, [rdi+i + 1*%%bSize]  ;; update crc1
; SKIP  ;crc32 r10, [rdi+i + 2*%%bSize] ;; update crc2

        ; merge in crc0
        movzx   ecx, al
        mov     r9d, [r8 + rcx*4 + %%td2]
        movzx   ecx, ah
        shr     eax, 16
        mov     r11d, [r8 + rcx*4 + %%td2]
        shl     r11, 8
        xor     r9, r11

        movzx   ecx, al
        mov     r11d, [r8 + rcx*4 + %%td2]
        movzx   ecx, ah
        shl     r11, 16
        xor     r9, r11
        mov     r11d, [r8 + rcx*4 + %%td2]
        shl     r11, 24
        xor     r9, r11

        ; merge in crc1

        movzx   ecx, bl
        mov     r11d, [r8 + rcx*4 + %%td1]
        movzx   ecx, bh
        shr     ebx, 16
        xor     r9, r11
        mov     r11d, [r8 + rcx*4 + %%td1]
        shl     r11, 8
        xor     r9, r11

        movzx   ecx, bl
        mov     r11d, [r8 + rcx*4 + %%td1]
        movzx   ecx, bh
        shl     r11, 16
        xor     r9, r11
        mov     r11d, [r8 + rcx*4 + %%td1]
        shl     r11, 24
        xor     r9, r11

        xor     r9, [rdi+i + 2*%%bSize]
        crc32   r10, r9
        mov     rax, r10

        add rdi, %%bSize*3              ;;      move to next block
        sub rdx, %%bSize*3
%IF %%bSize=576
        jns     %%crcB3_loop
%ENDIF
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
%%crcB3_end:
%IF %%bSize=576
        add rdx, %%bSize*3
%ENDIF
        je do_return                ;; return if remaining data is zero
%endmacro

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; ISCSI CRC 32 Implementation with crc32 Instruction

;;; unsigned int crc_216(unsigned char * buffer, int len, unsigned int
crc_init);
;;;
;;;        *buf = rcx
;;;         len = rdx
;;;    crc_init = r8
;;;

global  crc_216
crc_216:

        push    rdi
        push    rbx

        mov     rax, r8                 ;; rax = crc_init;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 1) ALIGN: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
        mov rdi, rcx                        ;; rdi = *buf
        neg rcx
        and rcx, 7                          ;; calculate the unalignment
                                            ;; amount of the address
        je proc_block                       ;; Skip if aligned

        ;;;; Calculate CRC of unaligned bytes of the buffer (if any) ;;;;
        mov rbx, [rdi]                      ;; load a Qword from buffer
        add rdi, rcx                        ;; align buffer pointer for
                                            ;; Qword processing
        sub rdx, rcx                        ;; update buffer length
align_loop:
        crc32 eax, bl                       ;;    compute crc32 of 1-byte
        shr rbx, 8                          ;;    get next byte
        dec rcx
        jne     align_loop
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 2) BLOCK LEVEL: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

proc_block:

        lea r8, [mul_table_64 wrt rip]  ;; load table base address

        crcB3 576, 0x1000, 0x0c00       ; 576*3 = 1920 (Tables 1152, 576)
        crcB3 288, 0x0c00, 0x0800       ; 288*3 =  960 (Tables  576, 288)
        crcB3 144, 0x0800, 0x0400       ; 144*3 =  480 (Tables  288, 144)
        crcB3  72, 0x0400, 0x0000       ;  72*3 =  240 (Tables  144,  72)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;4) LESS THAN 256-bytes REMAIN AT THIS POINT (8-bits of rdx are full)

bit8:
        shl dl, 1               ;; shift-out MSB (bit-7)
        jnc bit7                ;; jump to bit-6 if bit-7 == 0
 %assign i 0
 %rep 16
        crc32 rax, [rdi+i]      ;; compute crc32 of 8-byte data
        %assign i (i+8)
 %endrep
        je do_return            ;; return if remaining data is zero
        add rdi, 128                    ;; buf +=64; (next 64 bytes)

bit7:
        shl dl, 1               ;; shift-out MSB (bit-7)
        jnc bit6                ;; jump to bit-6 if bit-7 == 0
 %assign i 0
 %rep 8
        crc32 rax, [rdi+i]      ;; compute crc32 of 8-byte data
        %assign i (i+8)
 %endrep
        je do_return            ;; return if remaining data is zero
        add rdi, 64             ;; buf +=64; (next 64 bytes)
bit6:
        shl dl, 1               ;; shift-out MSB (bit-6)
        jnc bit5                ;; jump to bit-5 if bit-6 == 0
 %assign i 0
 %rep 4
        crc32 rax, [rdi+i]      ;;    compute crc32 of 8-byte data
        %assign i (i+8)
 %endrep
        je do_return            ;; return if remaining data is zero
        add rdi, 32             ;; buf +=32; (next 32 bytes)
bit5:
        shl dl, 1               ;; shift-out MSB (bit-5)
        jnc bit4                ;; jump to bit-4 if bit-5 == 0
 %assign i 0
 %rep 2
        crc32 rax, [rdi+i]      ;;    compute crc32 of 8-byte data
        %assign i (i+8)
 %endrep
        je do_return            ;; return if remaining data is zero
        add rdi, 16             ;; buf +=16; (next 16 bytes)
bit4:
        shl dl, 1               ;; shift-out MSB (bit-4)
        jnc bit3                ;; jump to bit-3 if bit-4 == 0
        crc32 rax, [rdi]        ;; compute crc32 of 8-byte data
        je do_return            ;; return if remaining data is zero
        add rdi, 8              ;; buf +=8; (next 8 bytes)
bit3:
        mov rbx, [rdi]          ;; load a 8-bytes from the buffer:
        shl dl, 1               ;; shift-out MSB (bit-3)
        jnc bit2                ;; jump to bit-2 if bit-3 == 0
        crc32 eax, ebx          ;; compute crc32 of 4-byte data
        je do_return            ;; return if remaining data is zero
        shr rbx, 32             ;; get next 3 bytes
bit2:
        shl dl, 1               ;; shift-out MSB (bit-2)
        jnc bit1                ;; jump to bit-1 if bit-2 == 0
        crc32 eax, bx           ;; compute crc32 of 2-byte data
        je do_return            ;; return if remaining data is zero
        shr rbx, 16             ;; next byte
bit1:
        crc32 eax, bl           ;; compute crc32 of 1-byte data
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

do_return:

        pop     rbx
        pop     rdi
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

section .data
align   8
mul_table_64:
DD 0x00000000,0x740eef02,0xe81dde04,0x9c133106
DD 0xd5d7caf9,0xa1d925fb,0x3dca14fd,0x49c4fbff
DD 0xae43e303,0xda4d0c01,0x465e3d07,0x3250d205
DD 0x7b9429fa,0x0f9ac6f8,0x9389f7fe,0xe78718fc
DD 0x596bb0f7,0x2d655ff5,0xb1766ef3,0xc57881f1
DD 0x8cbc7a0e,0xf8b2950c,0x64a1a40a,0x10af4b08
DD 0xf72853f4,0x8326bcf6,0x1f358df0,0x6b3b62f2
DD 0x22ff990d,0x56f1760f,0xcae24709,0xbeeca80b
DD 0xb2d761ee,0xc6d98eec,0x5acabfea,0x2ec450e8
DD 0x6700ab17,0x130e4415,0x8f1d7513,0xfb139a11
DD 0x1c9482ed,0x689a6def,0xf4895ce9,0x8087b3eb
DD 0xc9434814,0xbd4da716,0x215e9610,0x55507912
DD 0xebbcd119,0x9fb23e1b,0x03a10f1d,0x77afe01f
DD 0x3e6b1be0,0x4a65f4e2,0xd676c5e4,0xa2782ae6
DD 0x45ff321a,0x31f1dd18,0xade2ec1e,0xd9ec031c
DD 0x9028f8e3,0xe42617e1,0x783526e7,0x0c3bc9e5
DD 0x6042b52d,0x144c5a2f,0x885f6b29,0xfc51842b
DD 0xb5957fd4,0xc19b90d6,0x5d88a1d0,0x29864ed2
DD 0xce01562e,0xba0fb92c,0x261c882a,0x52126728
DD 0x1bd69cd7,0x6fd873d5,0xf3cb42d3,0x87c5add1
DD 0x392905da,0x4d27ead8,0xd134dbde,0xa53a34dc
DD 0xecfecf23,0x98f02021,0x04e31127,0x70edfe25
DD 0x976ae6d9,0xe36409db,0x7f7738dd,0x0b79d7df
DD 0x42bd2c20,0x36b3c322,0xaaa0f224,0xdeae1d26
DD 0xd295d4c3,0xa69b3bc1,0x3a880ac7,0x4e86e5c5
DD 0x07421e3a,0x734cf138,0xef5fc03e,0x9b512f3c
DD 0x7cd637c0,0x08d8d8c2,0x94cbe9c4,0xe0c506c6
DD 0xa901fd39,0xdd0f123b,0x411c233d,0x3512cc3f
DD 0x8bfe6434,0xfff08b36,0x63e3ba30,0x17ed5532
DD 0x5e29aecd,0x2a2741cf,0xb63470c9,0xc23a9fcb
DD 0x25bd8737,0x51b36835,0xcda05933,0xb9aeb631
DD 0xf06a4dce,0x8464a2cc,0x187793ca,0x6c797cc8
DD 0xc0856a5a,0xb48b8558,0x2898b45e,0x5c965b5c
DD 0x1552a0a3,0x615c4fa1,0xfd4f7ea7,0x894191a5
DD 0x6ec68959,0x1ac8665b,0x86db575d,0xf2d5b85f
DD 0xbb1143a0,0xcf1faca2,0x530c9da4,0x270272a6
DD 0x99eedaad,0xede035af,0x71f304a9,0x05fdebab
DD 0x4c391054,0x3837ff56,0xa424ce50,0xd02a2152
DD 0x37ad39ae,0x43a3d6ac,0xdfb0e7aa,0xabbe08a8
DD 0xe27af357,0x96741c55,0x0a672d53,0x7e69c251
DD 0x72520bb4,0x065ce4b6,0x9a4fd5b0,0xee413ab2
DD 0xa785c14d,0xd38b2e4f,0x4f981f49,0x3b96f04b
DD 0xdc11e8b7,0xa81f07b5,0x340c36b3,0x4002d9b1
DD 0x09c6224e,0x7dc8cd4c,0xe1dbfc4a,0x95d51348
DD 0x2b39bb43,0x5f375441,0xc3246547,0xb72a8a45
DD 0xfeee71ba,0x8ae09eb8,0x16f3afbe,0x62fd40bc
DD 0x857a5840,0xf174b742,0x6d678644,0x19696946
DD 0x50ad92b9,0x24a37dbb,0xb8b04cbd,0xccbea3bf
DD 0xa0c7df77,0xd4c93075,0x48da0173,0x3cd4ee71
DD 0x7510158e,0x011efa8c,0x9d0dcb8a,0xe9032488
DD 0x0e843c74,0x7a8ad376,0xe699e270,0x92970d72
DD 0xdb53f68d,0xaf5d198f,0x334e2889,0x4740c78b
DD 0xf9ac6f80,0x8da28082,0x11b1b184,0x65bf5e86
DD 0x2c7ba579,0x58754a7b,0xc4667b7d,0xb068947f
DD 0x57ef8c83,0x23e16381,0xbff25287,0xcbfcbd85
DD 0x8238467a,0xf636a978,0x6a25987e,0x1e2b777c
DD 0x1210be99,0x661e519b,0xfa0d609d,0x8e038f9f
DD 0xc7c77460,0xb3c99b62,0x2fdaaa64,0x5bd44566
DD 0xbc535d9a,0xc85db298,0x544e839e,0x20406c9c
DD 0x69849763,0x1d8a7861,0x81994967,0xf597a665
DD 0x4b7b0e6e,0x3f75e16c,0xa366d06a,0xd7683f68
DD 0x9eacc497,0xeaa22b95,0x76b11a93,0x02bff591
DD 0xe538ed6d,0x9136026f,0x0d253369,0x792bdc6b
DD 0x30ef2794,0x44e1c896,0xd8f2f990,0xacfc1692

mul_table_136:
DD 0x00000000,0xc96cfdc0,0x97358d71,0x5e5970b1
DD 0x2b876c13,0xe2eb91d3,0xbcb2e162,0x75de1ca2
DD 0x570ed826,0x9e6225e6,0xc03b5557,0x0957a897
DD 0x7c89b435,0xb5e549f5,0xebbc3944,0x22d0c484
DD 0xae1db04c,0x67714d8c,0x39283d3d,0xf044c0fd
DD 0x859adc5f,0x4cf6219f,0x12af512e,0xdbc3acee
DD 0xf913686a,0x307f95aa,0x6e26e51b,0xa74a18db
DD 0xd2940479,0x1bf8f9b9,0x45a18908,0x8ccd74c8
DD 0x59d71669,0x90bbeba9,0xcee29b18,0x078e66d8
DD 0x72507a7a,0xbb3c87ba,0xe565f70b,0x2c090acb
DD 0x0ed9ce4f,0xc7b5338f,0x99ec433e,0x5080befe
DD 0x255ea25c,0xec325f9c,0xb26b2f2d,0x7b07d2ed
DD 0xf7caa625,0x3ea65be5,0x60ff2b54,0xa993d694
DD 0xdc4dca36,0x152137f6,0x4b784747,0x8214ba87
DD 0xa0c47e03,0x69a883c3,0x37f1f372,0xfe9d0eb2
DD 0x8b431210,0x422fefd0,0x1c769f61,0xd51a62a1
DD 0xb3ae2cd2,0x7ac2d112,0x249ba1a3,0xedf75c63
DD 0x982940c1,0x5145bd01,0x0f1ccdb0,0xc6703070
DD 0xe4a0f4f4,0x2dcc0934,0x73957985,0xbaf98445
DD 0xcf2798e7,0x064b6527,0x58121596,0x917ee856
DD 0x1db39c9e,0xd4df615e,0x8a8611ef,0x43eaec2f
DD 0x3634f08d,0xff580d4d,0xa1017dfc,0x686d803c
DD 0x4abd44b8,0x83d1b978,0xdd88c9c9,0x14e43409
DD 0x613a28ab,0xa856d56b,0xf60fa5da,0x3f63581a
DD 0xea793abb,0x2315c77b,0x7d4cb7ca,0xb4204a0a
DD 0xc1fe56a8,0x0892ab68,0x56cbdbd9,0x9fa72619
DD 0xbd77e29d,0x741b1f5d,0x2a426fec,0xe32e922c
DD 0x96f08e8e,0x5f9c734e,0x01c503ff,0xc8a9fe3f
DD 0x44648af7,0x8d087737,0xd3510786,0x1a3dfa46
DD 0x6fe3e6e4,0xa68f1b24,0xf8d66b95,0x31ba9655
DD 0x136a52d1,0xda06af11,0x845fdfa0,0x4d332260
DD 0x38ed3ec2,0xf181c302,0xafd8b3b3,0x66b44e73
DD 0x62b02f55,0xabdcd295,0xf585a224,0x3ce95fe4
DD 0x49374346,0x805bbe86,0xde02ce37,0x176e33f7
DD 0x35bef773,0xfcd20ab3,0xa28b7a02,0x6be787c2
DD 0x1e399b60,0xd75566a0,0x890c1611,0x4060ebd1
DD 0xccad9f19,0x05c162d9,0x5b981268,0x92f4efa8
DD 0xe72af30a,0x2e460eca,0x701f7e7b,0xb97383bb
DD 0x9ba3473f,0x52cfbaff,0x0c96ca4e,0xc5fa378e
DD 0xb0242b2c,0x7948d6ec,0x2711a65d,0xee7d5b9d
DD 0x3b67393c,0xf20bc4fc,0xac52b44d,0x653e498d
DD 0x10e0552f,0xd98ca8ef,0x87d5d85e,0x4eb9259e
DD 0x6c69e11a,0xa5051cda,0xfb5c6c6b,0x323091ab
DD 0x47ee8d09,0x8e8270c9,0xd0db0078,0x19b7fdb8
DD 0x957a8970,0x5c1674b0,0x024f0401,0xcb23f9c1
DD 0xbefde563,0x779118a3,0x29c86812,0xe0a495d2
DD 0xc2745156,0x0b18ac96,0x5541dc27,0x9c2d21e7
DD 0xe9f33d45,0x209fc085,0x7ec6b034,0xb7aa4df4
DD 0xd11e0387,0x1872fe47,0x462b8ef6,0x8f477336
DD 0xfa996f94,0x33f59254,0x6dace2e5,0xa4c01f25
DD 0x8610dba1,0x4f7c2661,0x112556d0,0xd849ab10
DD 0xad97b7b2,0x64fb4a72,0x3aa23ac3,0xf3cec703
DD 0x7f03b3cb,0xb66f4e0b,0xe8363eba,0x215ac37a
DD 0x5484dfd8,0x9de82218,0xc3b152a9,0x0addaf69
DD 0x280d6bed,0xe161962d,0xbf38e69c,0x76541b5c
DD 0x038a07fe,0xcae6fa3e,0x94bf8a8f,0x5dd3774f
DD 0x88c915ee,0x41a5e82e,0x1ffc989f,0xd690655f
DD 0xa34e79fd,0x6a22843d,0x347bf48c,0xfd17094c
DD 0xdfc7cdc8,0x16ab3008,0x48f240b9,0x819ebd79
DD 0xf440a1db,0x3d2c5c1b,0x63752caa,0xaa19d16a
DD 0x26d4a5a2,0xefb85862,0xb1e128d3,0x788dd513
DD 0x0d53c9b1,0xc43f3471,0x9a6644c0,0x530ab900
DD 0x71da7d84,0xb8b68044,0xe6eff0f5,0x2f830d35
DD 0x5a5d1197,0x9331ec57,0xcd689ce6,0x04046126

mul_table_280:
DD 0x00000000,0xb6dd949b,0x68575fc7,0xde8acb5c
DD 0xd0aebf8e,0x66732b15,0xb8f9e049,0x0e2474d2
DD 0xa4b109ed,0x126c9d76,0xcce6562a,0x7a3bc2b1
DD 0x741fb663,0xc2c222f8,0x1c48e9a4,0xaa957d3f
DD 0x4c8e652b,0xfa53f1b0,0x24d93aec,0x9204ae77
DD 0x9c20daa5,0x2afd4e3e,0xf4778562,0x42aa11f9
DD 0xe83f6cc6,0x5ee2f85d,0x80683301,0x36b5a79a
DD 0x3891d348,0x8e4c47d3,0x50c68c8f,0xe61b1814
DD 0x991cca56,0x2fc15ecd,0xf14b9591,0x4796010a
DD 0x49b275d8,0xff6fe143,0x21e52a1f,0x9738be84
DD 0x3dadc3bb,0x8b705720,0x55fa9c7c,0xe32708e7
DD 0xed037c35,0x5bdee8ae,0x855423f2,0x3389b769
DD 0xd592af7d,0x634f3be6,0xbdc5f0ba,0x0b186421
DD 0x053c10f3,0xb3e18468,0x6d6b4f34,0xdbb6dbaf
DD 0x7123a690,0xc7fe320b,0x1974f957,0xafa96dcc
DD 0xa18d191e,0x17508d85,0xc9da46d9,0x7f07d242
DD 0x37d5e25d,0x810876c6,0x5f82bd9a,0xe95f2901
DD 0xe77b5dd3,0x51a6c948,0x8f2c0214,0x39f1968f
DD 0x9364ebb0,0x25b97f2b,0xfb33b477,0x4dee20ec
DD 0x43ca543e,0xf517c0a5,0x2b9d0bf9,0x9d409f62
DD 0x7b5b8776,0xcd8613ed,0x130cd8b1,0xa5d14c2a
DD 0xabf538f8,0x1d28ac63,0xc3a2673f,0x757ff3a4
DD 0xdfea8e9b,0x69371a00,0xb7bdd15c,0x016045c7
DD 0x0f443115,0xb999a58e,0x67136ed2,0xd1cefa49
DD 0xaec9280b,0x1814bc90,0xc69e77cc,0x7043e357
DD 0x7e679785,0xc8ba031e,0x1630c842,0xa0ed5cd9
DD 0x0a7821e6,0xbca5b57d,0x622f7e21,0xd4f2eaba
DD 0xdad69e68,0x6c0b0af3,0xb281c1af,0x045c5534
DD 0xe2474d20,0x549ad9bb,0x8a1012e7,0x3ccd867c
DD 0x32e9f2ae,0x84346635,0x5abead69,0xec6339f2
DD 0x46f644cd,0xf02bd056,0x2ea11b0a,0x987c8f91
DD 0x9658fb43,0x20856fd8,0xfe0fa484,0x48d2301f
DD 0x6fabc4ba,0xd9765021,0x07fc9b7d,0xb1210fe6
DD 0xbf057b34,0x09d8efaf,0xd75224f3,0x618fb068
DD 0xcb1acd57,0x7dc759cc,0xa34d9290,0x1590060b
DD 0x1bb472d9,0xad69e642,0x73e32d1e,0xc53eb985
DD 0x2325a191,0x95f8350a,0x4b72fe56,0xfdaf6acd
DD 0xf38b1e1f,0x45568a84,0x9bdc41d8,0x2d01d543
DD 0x8794a87c,0x31493ce7,0xefc3f7bb,0x591e6320
DD 0x573a17f2,0xe1e78369,0x3f6d4835,0x89b0dcae
DD 0xf6b70eec,0x406a9a77,0x9ee0512b,0x283dc5b0
DD 0x2619b162,0x90c425f9,0x4e4eeea5,0xf8937a3e
DD 0x52060701,0xe4db939a,0x3a5158c6,0x8c8ccc5d
DD 0x82a8b88f,0x34752c14,0xeaffe748,0x5c2273d3
DD 0xba396bc7,0x0ce4ff5c,0xd26e3400,0x64b3a09b
DD 0x6a97d449,0xdc4a40d2,0x02c08b8e,0xb41d1f15
DD 0x1e88622a,0xa855f6b1,0x76df3ded,0xc002a976
DD 0xce26dda4,0x78fb493f,0xa6718263,0x10ac16f8
DD 0x587e26e7,0xeea3b27c,0x30297920,0x86f4edbb
DD 0x88d09969,0x3e0d0df2,0xe087c6ae,0x565a5235
DD 0xfccf2f0a,0x4a12bb91,0x949870cd,0x2245e456
DD 0x2c619084,0x9abc041f,0x4436cf43,0xf2eb5bd8
DD 0x14f043cc,0xa22dd757,0x7ca71c0b,0xca7a8890
DD 0xc45efc42,0x728368d9,0xac09a385,0x1ad4371e
DD 0xb0414a21,0x069cdeba,0xd81615e6,0x6ecb817d
DD 0x60eff5af,0xd6326134,0x08b8aa68,0xbe653ef3
DD 0xc162ecb1,0x77bf782a,0xa935b376,0x1fe827ed
DD 0x11cc533f,0xa711c7a4,0x799b0cf8,0xcf469863
DD 0x65d3e55c,0xd30e71c7,0x0d84ba9b,0xbb592e00
DD 0xb57d5ad2,0x03a0ce49,0xdd2a0515,0x6bf7918e
DD 0x8dec899a,0x3b311d01,0xe5bbd65d,0x536642c6
DD 0x5d423614,0xeb9fa28f,0x351569d3,0x83c8fd48
DD 0x295d8077,0x9f8014ec,0x410adfb0,0xf7d74b2b
DD 0xf9f33ff9,0x4f2eab62,0x91a4603e,0x2779f4a5

mul_table_568:
DD 0x00000000,0x271d9844,0x4e3b3088,0x6926a8cc
DD 0x9c766110,0xbb6bf954,0xd24d5198,0xf550c9dc
DD 0x3d00b4d1,0x1a1d2c95,0x733b8459,0x54261c1d
DD 0xa176d5c1,0x866b4d85,0xef4de549,0xc8507d0d
DD 0x7a0169a2,0x5d1cf1e6,0x343a592a,0x1327c16e
DD 0xe67708b2,0xc16a90f6,0xa84c383a,0x8f51a07e
DD 0x4701dd73,0x601c4537,0x093aedfb,0x2e2775bf
DD 0xdb77bc63,0xfc6a2427,0x954c8ceb,0xb25114af
DD 0xf402d344,0xd31f4b00,0xba39e3cc,0x9d247b88
DD 0x6874b254,0x4f692a10,0x264f82dc,0x01521a98
DD 0xc9026795,0xee1fffd1,0x8739571d,0xa024cf59
DD 0x55740685,0x72699ec1,0x1b4f360d,0x3c52ae49
DD 0x8e03bae6,0xa91e22a2,0xc0388a6e,0xe725122a
DD 0x1275dbf6,0x356843b2,0x5c4eeb7e,0x7b53733a
DD 0xb3030e37,0x941e9673,0xfd383ebf,0xda25a6fb
DD 0x2f756f27,0x0868f763,0x614e5faf,0x4653c7eb
DD 0xede9d079,0xcaf4483d,0xa3d2e0f1,0x84cf78b5
DD 0x719fb169,0x5682292d,0x3fa481e1,0x18b919a5
DD 0xd0e964a8,0xf7f4fcec,0x9ed25420,0xb9cfcc64
DD 0x4c9f05b8,0x6b829dfc,0x02a43530,0x25b9ad74
DD 0x97e8b9db,0xb0f5219f,0xd9d38953,0xfece1117
DD 0x0b9ed8cb,0x2c83408f,0x45a5e843,0x62b87007
DD 0xaae80d0a,0x8df5954e,0xe4d33d82,0xc3cea5c6
DD 0x369e6c1a,0x1183f45e,0x78a55c92,0x5fb8c4d6
DD 0x19eb033d,0x3ef69b79,0x57d033b5,0x70cdabf1
DD 0x859d622d,0xa280fa69,0xcba652a5,0xecbbcae1
DD 0x24ebb7ec,0x03f62fa8,0x6ad08764,0x4dcd1f20
DD 0xb89dd6fc,0x9f804eb8,0xf6a6e674,0xd1bb7e30
DD 0x63ea6a9f,0x44f7f2db,0x2dd15a17,0x0accc253
DD 0xff9c0b8f,0xd88193cb,0xb1a73b07,0x96baa343
DD 0x5eeade4e,0x79f7460a,0x10d1eec6,0x37cc7682
DD 0xc29cbf5e,0xe581271a,0x8ca78fd6,0xabba1792
DD 0xde3fd603,0xf9224e47,0x9004e68b,0xb7197ecf
DD 0x4249b713,0x65542f57,0x0c72879b,0x2b6f1fdf
DD 0xe33f62d2,0xc422fa96,0xad04525a,0x8a19ca1e
DD 0x7f4903c2,0x58549b86,0x3172334a,0x166fab0e
DD 0xa43ebfa1,0x832327e5,0xea058f29,0xcd18176d
DD 0x3848deb1,0x1f5546f5,0x7673ee39,0x516e767d
DD 0x993e0b70,0xbe239334,0xd7053bf8,0xf018a3bc
DD 0x05486a60,0x2255f224,0x4b735ae8,0x6c6ec2ac
DD 0x2a3d0547,0x0d209d03,0x640635cf,0x431bad8b
DD 0xb64b6457,0x9156fc13,0xf87054df,0xdf6dcc9b
DD 0x173db196,0x302029d2,0x5906811e,0x7e1b195a
DD 0x8b4bd086,0xac5648c2,0xc570e00e,0xe26d784a
DD 0x503c6ce5,0x7721f4a1,0x1e075c6d,0x391ac429
DD 0xcc4a0df5,0xeb5795b1,0x82713d7d,0xa56ca539
DD 0x6d3cd834,0x4a214070,0x2307e8bc,0x041a70f8
DD 0xf14ab924,0xd6572160,0xbf7189ac,0x986c11e8
DD 0x33d6067a,0x14cb9e3e,0x7ded36f2,0x5af0aeb6
DD 0xafa0676a,0x88bdff2e,0xe19b57e2,0xc686cfa6
DD 0x0ed6b2ab,0x29cb2aef,0x40ed8223,0x67f01a67
DD 0x92a0d3bb,0xb5bd4bff,0xdc9be333,0xfb867b77
DD 0x49d76fd8,0x6ecaf79c,0x07ec5f50,0x20f1c714
DD 0xd5a10ec8,0xf2bc968c,0x9b9a3e40,0xbc87a604
DD 0x74d7db09,0x53ca434d,0x3aeceb81,0x1df173c5
DD 0xe8a1ba19,0xcfbc225d,0xa69a8a91,0x818712d5
DD 0xc7d4d53e,0xe0c94d7a,0x89efe5b6,0xaef27df2
DD 0x5ba2b42e,0x7cbf2c6a,0x159984a6,0x32841ce2
DD 0xfad461ef,0xddc9f9ab,0xb4ef5167,0x93f2c923
DD 0x66a200ff,0x41bf98bb,0x28993077,0x0f84a833
DD 0xbdd5bc9c,0x9ac824d8,0xf3ee8c14,0xd4f31450
DD 0x21a3dd8c,0x06be45c8,0x6f98ed04,0x48857540
DD 0x80d5084d,0xa7c89009,0xceee38c5,0xe9f3a081
DD 0x1ca3695d,0x3bbef119,0x529859d5,0x7585c191

mul_table_1144:
DD 0x00000000,0x86d8e4d2,0x085dbf55,0x8e855b87
DD 0x10bb7eaa,0x96639a78,0x18e6c1ff,0x9e3e252d
DD 0x2176fd54,0xa7ae1986,0x292b4201,0xaff3a6d3
DD 0x31cd83fe,0xb715672c,0x39903cab,0xbf48d879
DD 0x42edfaa8,0xc4351e7a,0x4ab045fd,0xcc68a12f
DD 0x52568402,0xd48e60d0,0x5a0b3b57,0xdcd3df85
DD 0x639b07fc,0xe543e32e,0x6bc6b8a9,0xed1e5c7b
DD 0x73207956,0xf5f89d84,0x7b7dc603,0xfda522d1
DD 0x85dbf550,0x03031182,0x8d864a05,0x0b5eaed7
DD 0x95608bfa,0x13b86f28,0x9d3d34af,0x1be5d07d
DD 0xa4ad0804,0x2275ecd6,0xacf0b751,0x2a285383
DD 0xb41676ae,0x32ce927c,0xbc4bc9fb,0x3a932d29
DD 0xc7360ff8,0x41eeeb2a,0xcf6bb0ad,0x49b3547f
DD 0xd78d7152,0x51559580,0xdfd0ce07,0x59082ad5
DD 0xe640f2ac,0x6098167e,0xee1d4df9,0x68c5a92b
DD 0xf6fb8c06,0x702368d4,0xfea63353,0x787ed781
DD 0x0e5b9c51,0x88837883,0x06062304,0x80dec7d6
DD 0x1ee0e2fb,0x98380629,0x16bd5dae,0x9065b97c
DD 0x2f2d6105,0xa9f585d7,0x2770de50,0xa1a83a82
DD 0x3f961faf,0xb94efb7d,0x37cba0fa,0xb1134428
DD 0x4cb666f9,0xca6e822b,0x44ebd9ac,0xc2333d7e
DD 0x5c0d1853,0xdad5fc81,0x5450a706,0xd28843d4
DD 0x6dc09bad,0xeb187f7f,0x659d24f8,0xe345c02a
DD 0x7d7be507,0xfba301d5,0x75265a52,0xf3febe80
DD 0x8b806901,0x0d588dd3,0x83ddd654,0x05053286
DD 0x9b3b17ab,0x1de3f379,0x9366a8fe,0x15be4c2c
DD 0xaaf69455,0x2c2e7087,0xa2ab2b00,0x2473cfd2
DD 0xba4deaff,0x3c950e2d,0xb21055aa,0x34c8b178
DD 0xc96d93a9,0x4fb5777b,0xc1302cfc,0x47e8c82e
DD 0xd9d6ed03,0x5f0e09d1,0xd18b5256,0x5753b684
DD 0xe81b6efd,0x6ec38a2f,0xe046d1a8,0x669e357a
DD 0xf8a01057,0x7e78f485,0xf0fdaf02,0x76254bd0
DD 0x1cb738a2,0x9a6fdc70,0x14ea87f7,0x92326325
DD 0x0c0c4608,0x8ad4a2da,0x0451f95d,0x82891d8f
DD 0x3dc1c5f6,0xbb192124,0x359c7aa3,0xb3449e71
DD 0x2d7abb5c,0xaba25f8e,0x25270409,0xa3ffe0db
DD 0x5e5ac20a,0xd88226d8,0x56077d5f,0xd0df998d
DD 0x4ee1bca0,0xc8395872,0x46bc03f5,0xc064e727
DD 0x7f2c3f5e,0xf9f4db8c,0x7771800b,0xf1a964d9
DD 0x6f9741f4,0xe94fa526,0x67cafea1,0xe1121a73
DD 0x996ccdf2,0x1fb42920,0x913172a7,0x17e99675
DD 0x89d7b358,0x0f0f578a,0x818a0c0d,0x0752e8df
DD 0xb81a30a6,0x3ec2d474,0xb0478ff3,0x369f6b21
DD 0xa8a14e0c,0x2e79aade,0xa0fcf159,0x2624158b
DD 0xdb81375a,0x5d59d388,0xd3dc880f,0x55046cdd
DD 0xcb3a49f0,0x4de2ad22,0xc367f6a5,0x45bf1277
DD 0xfaf7ca0e,0x7c2f2edc,0xf2aa755b,0x74729189
DD 0xea4cb4a4,0x6c945076,0xe2110bf1,0x64c9ef23
DD 0x12eca4f3,0x94344021,0x1ab11ba6,0x9c69ff74
DD 0x0257da59,0x848f3e8b,0x0a0a650c,0x8cd281de
DD 0x339a59a7,0xb542bd75,0x3bc7e6f2,0xbd1f0220
DD 0x2321270d,0xa5f9c3df,0x2b7c9858,0xada47c8a
DD 0x50015e5b,0xd6d9ba89,0x585ce10e,0xde8405dc
DD 0x40ba20f1,0xc662c423,0x48e79fa4,0xce3f7b76
DD 0x7177a30f,0xf7af47dd,0x792a1c5a,0xfff2f888
DD 0x61ccdda5,0xe7143977,0x699162f0,0xef498622
DD 0x973751a3,0x11efb571,0x9f6aeef6,0x19b20a24
DD 0x878c2f09,0x0154cbdb,0x8fd1905c,0x0909748e
DD 0xb641acf7,0x30994825,0xbe1c13a2,0x38c4f770
DD 0xa6fad25d,0x2022368f,0xaea76d08,0x287f89da
DD 0xd5daab0b,0x53024fd9,0xdd87145e,0x5b5ff08c
DD 0xc561d5a1,0x43b93173,0xcd3c6af4,0x4be48e26
DD 0xf4ac565f,0x7274b28d,0xfcf1e90a,0x7a290dd8
DD 0xe41728f5,0x62cfcc27,0xec4a97a0,0x6a927372
```

A.2 CRC_240

```
; Function to compute iscsi CRC32 with table-based recombination
; crc done "by 3" with block sizes 1920, 960, 480, 240

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; crcB3 MACRO to implement crc32 on 3 %%bSize-byte blocks
%macro  crcB3 3
%define %%bSize    %1    ; 1/3 of buffer size
%define %%td2      %2    ; table offset for crc0 (2/3 of buffer)
%define %%td1      %3    ; table offset for crc1 (1/3 of buffer)

%IF %%bSize=640
        sub rdx, %%bSize*3
        js %%crcB3_end         ;; jump to next level if 3*blockSize > len
%ELSE
        cmp rdx, %%bSize*3
        jnae %%crcB3_end       ;; jump to next level if 3*blockSize > len
%ENDIF
        ;;;;;; Calculate CRC of 3 blocks of the buffer ;;;;;;
%%crcB3_loop:
                                         ;; rax = crc0 = initial crc
        xor     rbx, rbx                ;; rbx = crc1 = 0;
        xor     r10, r10                ;; r10 = crc2 = 0;

 %assign i 0
 %rep %%bSize/8 - 1
        crc32 rax, [rdi+i + 0*%%bSize]  ;; update crc0
        crc32 rbx, [rdi+i + 1*%%bSize]  ;; update crc1
        crc32 r10, [rdi+i + 2*%%bSize]  ;; update crc2
        %assign i (i+8)
 %endrep
        crc32 rax, [rdi+i + 0*%%bSize]  ;; update crc0
        crc32 rbx, [rdi+i + 1*%%bSize]  ;; update crc1
; SKIP  ;crc32 r10, [rdi+i + 2*%%bSize] ;; update crc2

        ; merge in crc0
        movzx   ecx, al
        mov     r9d, [r8 + rcx*4 + %%td2]
        movzx   ecx, ah
        shr     eax, 16
        mov     r11d, [r8 + rcx*4 + %%td2]
        shl     r11, 8
        xor     r9, r11

        movzx   ecx, al
        mov     r11d, [r8 + rcx*4 + %%td2]
        movzx   ecx, ah
        shl     r11, 16
        xor     r9, r11
        mov     r11d, [r8 + rcx*4 + %%td2]
        shl     r11, 24
        xor     r9, r11

        ; merge in crc1

        movzx   ecx, bl
        mov     r11d, [r8 + rcx*4 + %%td1]
        movzx   ecx, bh
        shr     ebx, 16
        xor     r9, r11
        mov     r11d, [r8 + rcx*4 + %%td1]
        shl     r11, 8
        xor     r9, r11

        movzx   ecx, bl
        mov     r11d, [r8 + rcx*4 + %%td1]
        movzx   ecx, bh
        shl     r11, 16
        xor     r9, r11
        mov     r11d, [r8 + rcx*4 + %%td1]
        shl     r11, 24
        xor     r9, r11

        xor     r9, [rdi+i + 2*%%bSize]
        crc32   r10, r9
        mov     rax, r10

        add rdi, %%bSize*3              ;;      move to next block
        sub rdx, %%bSize*3
%IF %%bSize=640
        jns     %%crcB3_loop
%ENDIF
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
%%crcB3_end:
%IF %%bSize=640
        add rdx, %%bSize*3
%ENDIF
        je do_return                ;; return if remaining data is zero
```

```
%endmacro

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; ISCSI CRC 32 Implementation with crc32 Instruction

;;; unsigned int crc_240(unsigned char * buffer, int len, unsigned int
crc_init);
;;;
;;;        *buf = rcx
;;;         len = rdx
;;;    crc_init = r8
;;;

global  crc_240
crc_240:

        push    rdi
        push    rbx

        mov     rax, r8                 ;; rax = crc_init;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 1) ALIGN: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        mov rdi, rcx                    ;; rdi = *buf
        neg rcx
        and rcx, 7                      ;; calculate the unalignment
                                        ;; amount of the address
        je proc_block                   ;; Skip if aligned

        ;;;; Calculate CRC of unaligned bytes of the buffer (if any) ;;;;
        mov rbx, [rdi]                  ;; load a Qword from the buffer
        add rdi, rcx                    ;; align buffer pointer for
                                        ;; Qword processing
        sub rdx, rcx                    ;; update buffer length
align_loop:
        crc32 eax, bl                   ;;    compute crc32 of 1-byte
        shr rbx, 8                      ;;    get next byte
        dec rcx
        jne     align_loop
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 2) BLOCK LEVEL: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

proc_block:

        lea r8, [mul_table_72 wrt rip]  ;; load table base address

        crcB3 640, 0x1000, 0x0c00       ; 640*3 = 1920 (Tables 1280, 640)
        crcB3 320, 0x0c00, 0x0800       ; 320*3 =  960 (Tables  640, 320)
        crcB3 160, 0x0800, 0x0400       ; 160*3 =  480 (Tables  320, 160)
        crcB3  80, 0x0400, 0x0000       ;  80*3 =  240 (Tables  160,  80)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;4) LESS THAN 256-bytes REMAIN AT THIS POINT (8-bits of rdx are full)

bit8:
        shl dl, 1               ;; shift-out MSB (bit-7)
        jnc bit7                ;; jump to bit-6 if bit-7 == 0
 %assign i 0
 %rep 16
```

```
        crc32 rax, [rdi+i]      ;; compute crc32 of 8-byte data
        %assign i (i+8)
 %endrep
        je do_return            ;; return if remaining data is zero
        add rdi, 128                    ;; buf +=64; (next 64 bytes)

bit7:
        shl dl, 1               ;; shift-out MSB (bit-7)
        jnc bit6                ;; jump to bit-6 if bit-7 == 0
 %assign i 0
 %rep 8
        crc32 rax, [rdi+i]      ;; compute crc32 of 8-byte data
        %assign i (i+8)
 %endrep
        je do_return            ;; return if remaining data is zero
        add rdi, 64             ;; buf +=64; (next 64 bytes)
bit6:
        shl dl, 1               ;; shift-out MSB (bit-6)
        jnc bit5                ;; jump to bit-5 if bit-6 == 0
 %assign i 0
 %rep 4
        crc32 rax, [rdi+i]      ;;   compute crc32 of 8-byte data
        %assign i (i+8)
 %endrep
        je do_return            ;; return if remaining data is zero
        add rdi, 32             ;; buf +=32; (next 32 bytes)
bit5:
        shl dl, 1               ;; shift-out MSB (bit-5)
        jnc bit4                ;; jump to bit-4 if bit-5 == 0
 %assign i 0
 %rep 2
        crc32 rax, [rdi+i]      ;;   compute crc32 of 8-byte data
        %assign i (i+8)
 %endrep
        je do_return            ;; return if remaining data is zero
        add rdi, 16             ;; buf +=16; (next 16 bytes)
bit4:
        shl dl, 1               ;; shift-out MSB (bit-4)
        jnc bit3                ;; jump to bit-3 if bit-4 == 0
        crc32 rax, [rdi]        ;; compute crc32 of 8-byte data
        je do_return            ;; return if remaining data is zero
        add rdi, 8              ;; buf +=8; (next 8 bytes)
bit3:
        mov rbx, [rdi]          ;; load a 8-bytes from the buffer:
        shl dl, 1               ;; shift-out MSB (bit-3)
        jnc bit2                ;; jump to bit-2 if bit-3 == 0
        crc32 eax, ebx          ;; compute crc32 of 4-byte data
        je do_return            ;; return if remaining data is zero
        shr rbx, 32             ;; get next 3 bytes
bit2:
        shl dl, 1               ;; shift-out MSB (bit-2)
        jnc bit1                ;; jump to bit-1 if bit-2 == 0
        crc32 eax, bx           ;; compute crc32 of 2-byte data
        je do_return            ;; return if remaining data is zero
        shr rbx, 16             ;; next byte
bit1:
        crc32 eax, bl           ;; compute crc32 of 1-byte data
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

do_return:

        pop     rbx
        pop     rdi
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

section .data
align   8
mul_table_72:
DD 0x00000000,0x39d3b296,0x73a7652c,0x4a74d7ba
DD 0xe74eca58,0xde9d78ce,0x94e9af74,0xad3a1de2
DD 0xcb71e241,0xf2a250d7,0xb8d6876d,0x810535fb
DD 0x2c3f2819,0x15ec9a8f,0x5f984d35,0x664bffa3
DD 0x930fb273,0xaadc00e5,0xe0a8d75f,0xd97b65c9
DD 0x7441782b,0x4d92cabd,0x07e61d07,0x3e35af91
DD 0x587e5032,0x61ade2a4,0x2bd9351e,0x120a8788
DD 0xbf309a6a,0x86e328fc,0xcc97ff46,0xf5444dd0
DD 0x23f31217,0x1a20a081,0x5054773b,0x6987c5ad
DD 0xc4bdd84f,0xfd6e6ad9,0xb71abd63,0x8ec90ff5
DD 0xe882f056,0xd15142c0,0x9b25957a,0xa2f627ec
DD 0x0fcc3a0e,0x361f8898,0x7c6b5f22,0x45b8edb4
DD 0xb0fca064,0x892f12f2,0xc35bc548,0xfa8877de
DD 0x57b26a3c,0x6e61d8aa,0x24150f10,0x1dc6bd86
DD 0x7b8d4225,0x425ef0b3,0x082a2709,0x31f9959f
DD 0x9cc3887d,0xa5103aeb,0xef64ed51,0xd6b75fc7
DD 0x47e6242e,0x7e3596b8,0x34414102,0x0d92f394
DD 0xa0a8ee76,0x997b5ce0,0xd30f8b5a,0xeadc39cc
DD 0x8c97c66f,0xb54474f9,0xff30a343,0xc6e311d5
DD 0x6bd90c37,0x520abea1,0x187e691b,0x21addb8d
DD 0xd4e9965d,0xed3a24cb,0xa74ef371,0x9e9d41e7
DD 0x33a75c05,0x0a74ee93,0x40003929,0x79d38bbf
DD 0x1f98741c,0x264bc68a,0x6c3f1130,0x55eca3a6
DD 0xf8d6be44,0xc1050cd2,0x8b71db68,0xb2a269fe
DD 0x64153639,0x5dc684af,0x17b25315,0x2e61e183
DD 0x835bfc61,0xba884ef7,0xf0fc994d,0xc92f2bdb
DD 0xaf64d478,0x96b766ee,0xdcc3b154,0xe51003c2
DD 0x482a1e20,0x71f9acb6,0x3b8d7b0c,0x025ec99a
DD 0xf71a844a,0xcec936dc,0x84bde166,0xbd6e53f0
DD 0x10544e12,0x2987fc84,0x63f32b3e,0x5a2099a8
DD 0x3c6b660b,0x05b8d49d,0x4fcc0327,0x761fb1b1
DD 0xdb25ac53,0xe2f61ec5,0xa882c97f,0x91517be9
DD 0x8fcc485c,0xb61ffaca,0xfc6b2d70,0xc5b89fe6
DD 0x68828204,0x51513092,0x1b25e728,0x22f655be
DD 0x44bdaa1d,0x7d6e188b,0x371acf31,0x0ec97da7
DD 0xa3f36045,0x9a20d2d3,0xd0540569,0xe987b7ff
DD 0x1cc3fa2f,0x251048b9,0x6f649f03,0x56b72d95
DD 0xfb8d3077,0xc25e82e1,0x882a555b,0xb1f9e7cd
DD 0xd7b2186e,0xee61aaf8,0xa4157d42,0x9dc6cfd4
DD 0x30fcd236,0x092f60a0,0x435bb71a,0x7a88058c
DD 0xac3f5a4b,0x95ece8dd,0xdf983f67,0xe64b8df1
DD 0x4b719013,0x72a22285,0x38d6f53f,0x010547a9
DD 0x674eb80a,0x5e9d0a9c,0x14e9dd26,0x2d3a6fb0
DD 0x80007252,0xb9d3c0c4,0xf3a7177e,0xca74a5e8
DD 0x3f30e838,0x06e35aae,0x4c978d14,0x75443f82
DD 0xd87e2260,0xe1ad90f6,0xabd9474c,0x920af5da
DD 0xf4410a79,0xcd92b8ef,0x87e66f55,0xbe35ddc3
DD 0x130fc021,0x2adc72b7,0x60a8a50d,0x597b179b
DD 0xc82a6c72,0xf1f9dee4,0xbb8d095e,0x825ebbc8
DD 0x2f64a62a,0x16b714bc,0x5cc3c306,0x65107190
DD 0x035b8e33,0x3a883ca5,0x70fceb1f,0x492f5989
DD 0xe415446b,0xddc6f6fd,0x97b22147,0xae6193d1
DD 0x5b25de01,0x62f66c97,0x2882bb2d,0x115109bb
DD 0xbc6b1459,0x85b8a6cf,0xcfcc7175,0xf61fc3e3
DD 0x90543c40,0xa9878ed6,0xe3f3596c,0xda20ebfa
DD 0x771af618,0x4ec9448e,0x04bd9334,0x3d6e21a2
DD 0xebd97e65,0xd20accf3,0x987e1b49,0xa1ada9df
DD 0x0c97b43d,0x354406ab,0x7f30d111,0x46e36387
DD 0x20a89c24,0x197b2eb2,0x530ff908,0x6adc4b9e
DD 0xc7e6567c,0xfe35e4ea,0xb4413350,0x8d9281c6
DD 0x78d6cc16,0x41057e80,0x0b71a93a,0x32a21bac
DD 0x9f98064e,0xa64bb4d8,0xec3f6362,0xd5ecd1f4
DD 0xb3a72e57,0x8a749cc1,0xc0004b7b,0xf9d3f9ed
DD 0x54e9e40f,0x6d3a5699,0x274e8123,0x1e9d33b5

mul_table_152:
DD 0x00000000,0x878a92a7,0x0af953bf,0x8d73c118
DD 0x15f2a77e,0x927835d9,0x1f0bf4c1,0x98816666
DD 0x2be54efc,0xac6fdc5b,0x211c1d43,0xa6968fe4
DD 0x3e17e982,0xb99d7b25,0x34eeba3d,0xb364289a
DD 0x57ca9df8,0xd0400f5f,0x5d33ce47,0xdab95ce0
DD 0x42383a86,0xc5b2a821,0x48c16939,0xcf4bfb9e
DD 0x7c2fd304,0xfba541a3,0x76d680bb,0xf15c121c
DD 0x69dd747a,0xee57e6dd,0x632427c5,0xe4aeb562
DD 0xaf953bf0,0x281fa957,0xa56c684f,0x22e6fae8
DD 0xba679c8e,0x3ded0e29,0xb09ecf31,0x37145d96
DD 0x8470750c,0x03fae7ab,0x8e8926b3,0x0903b414
DD 0x9182d272,0x160840d5,0x9b7b81cd,0x1cf1136a
DD 0xf85fa608,0x7fd534af,0xf2a6f5b7,0x752c6710
DD 0xedad0176,0x6a2793d1,0xe75452c9,0x60dec06e
DD 0xd3bae8f4,0x54307a53,0xd943bb4b,0x5ec929ec
DD 0xc6484f8a,0x41c2dd2d,0xccb11c35,0x4b3b8e92
DD 0x5ac60111,0xdd4c93b6,0x503f52ae,0xd7b5c009
DD 0x4f34a66f,0xc8be34c8,0x45cdf5d0,0xc2476777
DD 0x71234fed,0xf6a9dd4a,0x7bda1c52,0xfc508ef5
DD 0x64d1e893,0xe35b7a34,0x6e28bb2c,0xe9a2298b
DD 0x0d0c9ce9,0x8a860e4e,0x07f5cf56,0x807f5df1
DD 0x18fe3b97,0x9f74a930,0x12076828,0x958dfa8f
DD 0x26e9d215,0xa16340b2,0x2c1081aa,0xab9a130d
DD 0x331b756b,0xb491e7cc,0x39e226d4,0xbe68b473
DD 0xf5533ae1,0x72d9a846,0xffaa695e,0x7820fbf9
DD 0xe0a19d9f,0x672b0f38,0xea58ce20,0x6dd25c87
DD 0xdeb6741d,0x593ce6ba,0xd44f27a2,0x53c5b505
DD 0xcb44d363,0x4cce41c4,0xc1bd80dc,0x4637127b
DD 0xa299a719,0x251335be,0xa860f4a6,0x2fea6601
DD 0xb76b0067,0x30e192c0,0xbd9253d8,0x3a18c17f
DD 0x897ce9e5,0x0ef67b42,0x8385ba5a,0x040f28fd
DD 0x9c8e4e9b,0x1b04dc3c,0x96771d24,0x11fd8f83
DD 0xb58c0222,0x32069085,0xbf75519d,0x38ffc33a
DD 0xa07ea55c,0x27f437fb,0xaa87f6e3,0x2d0d6444
DD 0x9e694cde,0x19e3de79,0x94901f61,0x131a8dc6
DD 0x8b9beba0,0x0c117907,0x8162b81f,0x06e82ab8
DD 0xe2469fda,0x65cc0d7d,0xe8bfcc65,0x6f355ec2
DD 0xf7b438a4,0x703eaa03,0xfd4d6b1b,0x7ac7f9bc
DD 0xc9a3d126,0x4e294381,0xc35a8299,0x44d0103e
DD 0xdc517658,0x5bdbe4ff,0xd6a825e7,0x5122b740
DD 0x1a1939d2,0x9d93ab75,0x10e06a6d,0x976af8ca
DD 0x0feb9eac,0x88610c0b,0x0512cd13,0x82985fb4
DD 0x31fc772e,0xb676e589,0x3b052491,0xbc8fb636
DD 0x240ed050,0xa38442f7,0x2ef783ef,0xa97d1148
DD 0x4dd3a42a,0xca59368d,0x472af795,0xc0a06532
DD 0x58210354,0xdfab91f3,0x52d850eb,0xd552c24c
DD 0x6636ead6,0xe1bc7871,0x6ccfb969,0xeb452bce
DD 0x73c44da8,0xf44edf0f,0x793d1e17,0xfeb78cb0
DD 0xef4a0333,0x68c09194,0xe5b3508c,0x6239c22b
DD 0xfab8a44d,0x7d3236ea,0xf041f7f2,0x77cb6555
DD 0xc4af4dcf,0x4325df68,0xce561e70,0x49dc8cd7
DD 0xd15deab1,0x56d77816,0xdba4b90e,0x5c2e2ba9
DD 0xb8809ecb,0x3f0a0c6c,0xb279cd74,0x35f35fd3
DD 0xad7239b5,0x2af8ab12,0xa78b6a0a,0x2001f8ad
DD 0x9365d037,0x14ef4290,0x999c8388,0x1e16112f
DD 0x86977749,0x011de5ee,0x8c6e24f6,0x0be4b651
DD 0x40df38c3,0xc755aa64,0x4a266b7c,0xcdacf9db
DD 0x552d9fbd,0xd2a70d1a,0x5fd4cc02,0xd85e5ea5
DD 0x6b3a763f,0xecb0e498,0x61c32580,0xe649b727
DD 0x7ec8d141,0xf94243e6,0x743182fe,0xf3bb1059
DD 0x1715a53b,0x909f379c,0x1decf684,0x9a666423
DD 0x02e70245,0x856d90e2,0x081e51fa,0x8f94c35d
DD 0x3cf0ebc7,0xbb7a7960,0x3609b878,0xb1832adf
DD 0x29024cb9,0xae88de1e,0x23fb1f06,0xa4718da1

mul_table_312:
DD 0x00000000,0xbac2fd7b,0x70698c07,0xcaab717c
DD 0xe0d3180e,0x5a11e575,0x90ba9409,0x2a786972
DD 0xc44a46ed,0x7e88bb96,0xb423caea,0x0ee13791
DD 0x24995ee3,0x9e5ba398,0x54f0d2e4,0xee322f9f
DD 0x8d78fb2b,0x37ba0650,0xfd11772c,0x47d38a57
DD 0x6dabe325,0xd7691e5e,0x1dc26f22,0xa7009259
DD 0x4932bdc6,0xf3f040bd,0x395b31c1,0x8399ccba
DD 0xa9e1a5c8,0x132358b3,0xd98829cf,0x634ad4b4
DD 0x1f1d80a7,0xa5df7ddc,0x6f740ca0,0xd5b6f1db
DD 0xffce98a9,0x450c65d2,0x8fa714ae,0x3565e9d5
DD 0xdb57c64a,0x61953b31,0xab3e4a4d,0x11fcb736
DD 0x3b84de44,0x8146233f,0x4bed5243,0xf12faf38
DD 0x92657b8c,0x28a786f7,0xe20cf78b,0x58ce0af0
DD 0x72b66382,0xc8749ef9,0x02dfef85,0xb81d12fe
DD 0x562f3d61,0xecedc01a,0x2646b166,0x9c844c1d
DD 0xb6fc256f,0x0c3ed814,0xc695a968,0x7c575413
DD 0x3e3b014e,0x84f9fc35,0x4e528d49,0xf4907032
DD 0xdee81940,0x642ae43b,0xae819547,0x1443683c
DD 0xfa7147a3,0x40b3bad8,0x8a18cba4,0x30da36df
DD 0x1aa25fad,0xa060a2d6,0x6acbd3aa,0xd0092ed1
DD 0xb343fa65,0x0981071e,0xc32a7662,0x79e88b19
DD 0x5390e26b,0xe9521f10,0x23f96e6c,0x993b9317
DD 0x7709bc88,0xcdcb41f3,0x0760308f,0xbda2cdf4
DD 0x97daa486,0x2d1859fd,0xe7b32881,0x5d71d5fa
DD 0x212681e9,0x9be47c92,0x514f0dee,0xeb8df095
DD 0xc1f599e7,0x7b37649c,0xb19c15e0,0x0b5ee89b
DD 0xe56cc704,0x5fae3a7f,0x95054b03,0x2fc7b678
DD 0x05bfdf0a,0xbf7d2271,0x75d6530d,0xcf14ae76
DD 0xac5e7ac2,0x169c87b9,0xdc37f6c5,0x66f50bbe
DD 0x4c8d62cc,0xf64f9fb7,0x3ce4eecb,0x862613b0
DD 0x68143c2f,0xd2d6c154,0x187db028,0xa2bf4d53
DD 0x88c72421,0x3205d95a,0xf8aea826,0x426c555d
DD 0x7c76029c,0xc6b4ffe7,0x0c1f8e9b,0xb6dd73e0
DD 0x9ca51a92,0x2667e7e9,0xeccc9695,0x560e6bee
DD 0xb83c4471,0x02feb90a,0xc855c876,0x7297350d
DD 0x58ef5c7f,0xe22da104,0x2886d078,0x92442d03
DD 0xf10ef9b7,0x4bcc04cc,0x816775b0,0x3ba588cb
DD 0x11dde1b9,0xab1f1cc2,0x61b46dbe,0xdb7690c5
DD 0x3544bf5a,0x8f864221,0x452d335d,0xffefce26
DD 0xd597a754,0x6f555a2f,0xa5fe2b53,0x1f3cd628
DD 0x636b823b,0xd9a97f40,0x13020e3c,0xa9c0f347
DD 0x83b89a35,0x397a674e,0xf3d11632,0x4913eb49
DD 0xa721c4d6,0x1de339ad,0xd74848d1,0x6d8ab5aa
DD 0x47f2dcd8,0xfd3021a3,0x379b50df,0x8d59ada4
DD 0xee137910,0x54d1846b,0x9e7af517,0x24b8086c
DD 0x0ec0611e,0xb4029c65,0x7ea9ed19,0xc46b1062
DD 0x2a593ffd,0x909bc286,0x5a30b3fa,0xe0f24e81
DD 0xca8a27f3,0x7048da88,0xbae3abf4,0x0021568f
DD 0x424d03d2,0xf88ffea9,0x32248fd5,0x88e672ae
DD 0xa29e1bdc,0x185ce6a7,0xd2f797db,0x68356aa0
DD 0x8607453f,0x3cc5b844,0xf66ec938,0x4cac3443
DD 0x66d45d31,0xdc16a04a,0x16bdd136,0xac7f2c4d
DD 0xcf35f8f9,0x75f70582,0xbf5c74fe,0x059e8985
DD 0x2fe6e0f7,0x95241d8c,0x5f8f6cf0,0xe54d918b
DD 0x0b7fbe14,0xb1bd436f,0x7b163213,0xc1d4cf68
DD 0xebaca61a,0x516e5b61,0x9bc52a1d,0x2107d766
DD 0x5d508375,0xe7927e0e,0x2d390f72,0x97fbf209
DD 0xbd839b7b,0x07416600,0xcdea177c,0x7728ea07
DD 0x991ac598,0x23d838e3,0xe973499f,0x53b1b4e4
DD 0x79c9dd96,0xc30b20ed,0x09a05191,0xb362acea
DD 0xd028785e,0x6aea8525,0xa041f459,0x1a830922
DD 0x30fb6050,0x8a399d2b,0x4092ec57,0xfa50112c
DD 0x14623eb3,0xaea0c3c8,0x640bb2b4,0xdec94fcf
DD 0xf4b126bd,0x4e73dbc6,0x84d8aaba,0x3e1a57c1

mul_table_632:
DD 0x00000000,0x6b749fb2,0xd6e93f64,0xbd9da0d6
DD 0xa83e0839,0xc34a978b,0x7ed7375d,0x15a3a8ef
DD 0x55906683,0x3ee4f931,0x837959e7,0xe80dc655
DD 0xfdae6eba,0x96daf108,0x2b4751de,0x4033ce6c
DD 0xab20cd06,0xc05452b4,0x7dc9f262,0x16bd6dd0
DD 0x031ec53f,0x686a5a8d,0xd5f7fa5b,0xbe8365e9
DD 0xfeb0ab85,0x95c43437,0x285994e1,0x432d0b53
DD 0x568ea3bc,0x3dfa3c0e,0x80679cd8,0xeb13036a
DD 0x53adecfd,0x38d9734f,0x8544d399,0xee304c2b
DD 0xfb93e4c4,0x90e77b76,0x2d7adba0,0x460e4412
DD 0x063d8a7e,0x6d4915cc,0xd0d4b51a,0xbba02aa8
DD 0xae038247,0xc5771df5,0x78eabd23,0x139e2291
DD 0xf88d21fb,0x93f9be49,0x2e641e9f,0x4510812d
DD 0x50b329c2,0x3bc7b670,0x865a16a6,0xed2e8914
DD 0xad1d4778,0xc669d8ca,0x7bf4781c,0x1080e7ae
DD 0x05234f41,0x6e57d0f3,0xd3ca7025,0xb8beef97
DD 0xa75bd9fa,0xcc2f4648,0x71b2e69e,0x1ac6792c
DD 0x0f65d1c3,0x64114e71,0xd98ceea7,0xb2f87115
DD 0xf2cbbf79,0x99bf20cb,0x2422801d,0x4f561faf
DD 0x5af5b740,0x318128f2,0x8c1c8824,0xe7681796
DD 0x0c7b14fc,0x670f8b4e,0xda922b98,0xb1e6b42a
DD 0xa4451cc5,0xcf318377,0x72ac23a1,0x19d8bc13
DD 0x59eb727f,0x329fedcd,0x8f024d1b,0xe476d2a9
DD 0xf1d57a46,0x9aa1e5f4,0x273c4522,0x4c48da90
DD 0xf4f63507,0x9f82aab5,0x221f0a63,0x496b95d1
DD 0x5cc83d3e,0x37bca28c,0x8a21025a,0xe1559de8
DD 0xa1665384,0xca12cc36,0x778f6ce0,0x1cfbf352
DD 0x09585bbd,0x622cc40f,0xdfb164d9,0xb4c5fb6b
DD 0x5fd6f801,0x34a267b3,0x893fc765,0xe24b58d7
DD 0xf7e8f038,0x9c9c6f8a,0x2101cf5c,0x4a7550ee
DD 0x0a469e82,0x61320130,0xdcafa1e6,0xb7db3e54
DD 0xa27896bb,0xc90c0909,0x7491a9df,0x1fe5366d
DD 0x4b5bc505,0x202f5ab7,0x9db2fa61,0xf6c665d3
DD 0xe365cd3c,0x8811528e,0x358cf258,0x5ef86dea
DD 0x1ecba386,0x75bf3c34,0xc8229ce2,0xa3560350
DD 0xb6f5abbf,0xdd81340d,0x601c94db,0x0b680b69
DD 0xe07b0803,0x8b0f97b1,0x36923767,0x5de6a8d5
DD 0x4845003a,0x23319f88,0x9eac3f5e,0xf5d8a0ec
DD 0xb5eb6e80,0xde9ff132,0x630251e4,0x0876ce56
DD 0x1dd566b9,0x76a1f90b,0xcb3c59dd,0xa048c66f
DD 0x18f629f8,0x7382b64a,0xce1f169c,0xa56b892e
DD 0xb0c821c1,0xdbbcbe73,0x66211ea5,0x0d558117
DD 0x4d664f7b,0x2612d0c9,0x9b8f701f,0xf0fbefad
DD 0xe5584742,0x8e2cd8f0,0x33b17826,0x58c5e794
DD 0xb3d6e4fe,0xd8a27b4c,0x653fdb9a,0x0e4b4428
DD 0x1be8ecc7,0x709c7375,0xcd01d3a3,0xa6754c11
DD 0xe646827d,0x8d321dcf,0x30afbd19,0x5bdb22ab
DD 0x4e788a44,0x250c15f6,0x9891b520,0xf3e52a92
DD 0xec001cff,0x8774834d,0x3ae9239b,0x519dbc29
DD 0x443e14c6,0x2f4a8b74,0x92d72ba2,0xf9a3b410
DD 0xb9907a7c,0xd2e4e5ce,0x6f794518,0x040ddaaa
DD 0x11ae7245,0x7adaedf7,0xc7474d21,0xac33d293
DD 0x4720d1f9,0x2c544e4b,0x91c9ee9d,0xfabd712f
DD 0xef1ed9c0,0x846a4672,0x39f7e6a4,0x52837916
DD 0x12b0b77a,0x79c428c8,0xc459881e,0xaf2d17ac
DD 0xba8ebf43,0xd1fa20f1,0x6c678027,0x07131f95
DD 0xbfadf002,0xd4d96fb0,0x6944cf66,0x023050d4
DD 0x1793f83b,0x7ce76789,0xc17ac75f,0xaa0e58ed
DD 0xea3d9681,0x81490933,0x3cd4a9e5,0x57a03657
DD 0x42039eb8,0x2977010a,0x94eaa1dc,0xff9e3e6e
DD 0x148d3d04,0x7ff9a2b6,0xc2640260,0xa9109dd2
DD 0xbcb3353d,0xd7c7aa8f,0x6a5a0a59,0x012e95eb
DD 0x411d5b87,0x2a69c435,0x97f464e3,0xfc80fb51
DD 0xe92353be,0x8257cc0c,0x3fca6cda,0x54bef368

mul_table_1272:
DD 0x00000000,0xdd66cbbb,0xbf21e187,0x62472a3c
DD 0x7bafb5ff,0xa6c97e44,0xc48e5478,0x19e89fc3
DD 0xf75f6bfe,0x2a39a045,0x487e8a79,0x951841c2
DD 0x8cf0de01,0x519615ba,0x33d13f86,0xeeb7f43d
DD 0xeb52a10d,0x36346ab6,0x5473408a,0x89158b31
DD 0x90fd14f2,0x4d9bdf49,0x2fdcf575,0xf2ba3ece
DD 0x1c0dcaf3,0xc16b0148,0xa32c2b74,0x7e4ae0cf
DD 0x67a27f0c,0xbac4b4b7,0xd8839e8b,0x05e55530
DD 0xd34934eb,0x0e2fff50,0x6c68d56c,0xb10e1ed7
DD 0xa8e68114,0x75804aaf,0x17c76093,0xcaa1ab28
DD 0x24165f15,0xf97094ae,0x9b37be92,0x46517529
DD 0x5fb9eaea,0x82df2151,0xe0980b6d,0x3dfec0d6
DD 0x381b95e6,0xe57d5e5d,0x873a7461,0x5a5cbfda
DD 0x43b42019,0x9ed2eba2,0xfc95c19e,0x21f30a25
DD 0xcf44fe18,0x122235a3,0x70651f9f,0xad03d424
DD 0xb4eb4be7,0x698d805c,0x0bcaaa60,0xd6ac61db
DD 0xa37e1f27,0x7e18d49c,0x1c5ffea0,0xc139351b
DD 0xd8d1aad8,0x05b76163,0x67f04b5f,0xba9680e4
DD 0x542174d9,0x8947bf62,0xeb00955e,0x36665ee5
DD 0x2f8ec126,0xf2e80a9d,0x90af20a1,0x4dc9eb1a
DD 0x482cbe2a,0x954a7591,0xf70d5fad,0x2a6b9416
DD 0x33830bd5,0xeee5c06e,0x8ca2ea52,0x51c421e9
DD 0xbf73d5d4,0x62151e6f,0x00523453,0xdd34ffe8
DD 0xc4dc602b,0x19baab90,0x7bfd81ac,0xa69b4a17
DD 0x70372bcc,0xad51e077,0xcf16ca4b,0x127001f0
DD 0x0b989e33,0xd6fe5588,0xb4b97fb4,0x69dfb40f
DD 0x87684032,0x5a0e8b89,0x3849a1b5,0xe52f6a0e
DD 0xfcc7f5cd,0x21a13e76,0x43e6144a,0x9e80dff1
DD 0x9b658ac1,0x4603417a,0x24446b46,0xf922a0fd
DD 0xe0ca3f3e,0x3dacf485,0x5febdeb9,0x828d1502
DD 0x6c3ae13f,0xb15c2a84,0xd31b00b8,0x0e7dcb03
DD 0x179554c0,0xcaf39f7b,0xa8b4b547,0x75d27efc
DD 0x431048bf,0x9e768304,0xfc31a938,0x21576283
DD 0x38bffd40,0xe5d936fb,0x879e1cc7,0x5af8d77c
DD 0xb44f2341,0x6929e8fa,0x0b6ec2c6,0xd608097d
DD 0xcfe096be,0x12865d05,0x70c17739,0xada7bc82
DD 0xa842e9b2,0x75242209,0x17630835,0xca05c38e
DD 0xd3ed5c4d,0x0e8b97f6,0x6cccbdca,0xb1aa7671
DD 0x5f1d824c,0x827b49f7,0xe03c63cb,0x3d5aa870
DD 0x24b237b3,0xf9d4fc08,0x9b93d634,0x46f51d8f
DD 0x90597c54,0x4d3fb7ef,0x2f789dd3,0xf21e5668
DD 0xebf6c9ab,0x36900210,0x54d7282c,0x89b1e397
DD 0x670617aa,0xba60dc11,0xd827f62d,0x05413d96
DD 0x1ca9a255,0xc1cf69ee,0xa38843d2,0x7eee8869
DD 0x7b0bdd59,0xa66d16e2,0xc42a3cde,0x194cf765
DD 0x00a468a6,0xddc2a31d,0xbf858921,0x62e3429a
DD 0x8c54b6a7,0x51327d1c,0x33755720,0xee139c9b
DD 0xf7fb0358,0x2a9dc8e3,0x48dae2df,0x95bc2964
DD 0xe06e5798,0x3d089c23,0x5f4fb61f,0x82297da4
DD 0x9bc1e267,0x46a729dc,0x24e003e0,0xf986c85b
DD 0x17313c66,0xca57f7dd,0xa810dde1,0x7576165a
DD 0x6c9e8999,0xb1f84222,0xd3bf681e,0x0ed9a3a5
DD 0x0b3cf695,0xd65a3d2e,0xb41d1712,0x697bdca9
DD 0x7093436a,0xadf588d1,0xcfb2a2ed,0x12d46956
DD 0xfc639d6b,0x210556d0,0x43427cec,0x9e24b757
DD 0x87cc2894,0x5aaae32f,0x38edc913,0xe58b02a8
DD 0x33276373,0xee41a8c8,0x8c0682f4,0x5160494f
DD 0x4888d68c,0x95ee1d37,0xf7a9370b,0x2acffcb0
DD 0xc478088d,0x191ec336,0x7b59e90a,0xa63f22b1
DD 0xbfd7bd72,0x62b176c9,0x00f65cf5,0xdd90974e
DD 0xd875c27e,0x051309c5,0x675423f9,0xba32e842
DD 0xa3da7781,0x7ebcbc3a,0x1cfb9606,0xc19d5dbd
DD 0x2f2aa980,0xf24c623b,0x900b4807,0x4d6d83bc
DD 0x54851c7f,0x89e3d7c4,0xeba4fdf8,0x36c23643
```

A.3 CRC_1024

```
; Function to compute iSCSI CRC32 with table-based recombination
; crc done "by 3" for fixed input size of 1024 bytes

;;; ISCSI CRC 32 Implementation with crc32 Instruction

;; 1024 = 336 * 3 * 8 + 16
%define BLOCKSIZE 336

mul_table1_336_offset equ mul_table1_336 - mul_table1_336
mul_table1_672_offset equ mul_table1_672 - mul_table1_336

;;; unsigned int crc_1024(unsigned char * buffer, unsigned int crc_init);
;;;
;;;         *buf = rcx
;;;     crc_init = rdx
;;;

global crc_1024
crc_1024:

        push    rbx
        mov     r9, rcx

                                ;; rdx = crc0 = crc_init
        xor     rbx, rbx        ;; rbx = crc1
        xor     rax, rax        ;; rax = crc2

        ;; Do first 8 bytes here for better pipelining
        crc32   rdx, [r9]

        ;; process block inline
        ;; process rdx last to avoid dependency with crc32 above
%assign i 8
%rep BLOCKSIZE/8
        crc32   rbx, [r9 + i + 1*BLOCKSIZE]     ; crc1
        crc32   rax, [r9 + i + 2*BLOCKSIZE]     ; crc2
        crc32   rdx, [r9 + i + 0*BLOCKSIZE]     ; crc0
%assign i (i+8)
%endrep

        lea     r8, [mul_table1_336 wrt rip]

        ;; merge in crc1
        movzx   ecx, bl
        mov     r10d, [r8 + rcx*4 + mul_table1_336_offset]
        movzx   ecx, bh
        shr     ebx, 16
        mov     r11d, [r8 + rcx*4 + mul_table1_336_offset]
        shl     r11, 8
        xor     r10, r11

        movzx   ecx, bl
        mov     r11d, [r8 + rcx*4 + mul_table1_336_offset]
        movzx   ecx, bh
        shl     r11, 16
        xor     r10, r11
        mov     r11d, [r8 + rcx*4 + mul_table1_336_offset]
        shl     r11, 24
        xor     r10, r11

        ;; merge in crc0

        movzx   ecx, dl
        mov     r11d, [r8 + rcx*4 + mul_table1_672_offset]
        movzx   ecx, dh
        shr     edx, 16
        xor     r10, r11
        mov     r11d, [r8 + rcx*4 + mul_table1_672_offset]
        shl     r11, 8
        xor     r10, r11

        movzx   ecx, dl
        mov     r11d, [r8 + rcx*4 + mul_table1_672_offset]
        movzx   ecx, dh
        shl     r11, 16
        xor     r10, r11
        mov     r11d, [r8 + rcx*4 + mul_table1_672_offset]
        shl     r11, 24
        xor     r10, r11


        xor     r10, [r9 + 1016]
        crc32   rax, r10                        ; crc2

        pop     rbx
        ret

section .data
align 16
mul_table1_336:
DD 0x00000000,0x8f158014,0x1bc776d9,0x94d2f6cd
DD 0x378eedb2,0xb89b6da6,0x2c499b6b,0xa35c1b7f
DD 0x6f1ddb64,0xe0085b70,0x74daadbd,0xfbcf2da9
DD 0x589336d6,0xd786b6c2,0x4354400f,0xcc41c01b
DD 0xde3bb6c8,0x512e36dc,0xc5fcc011,0x4ae94005
DD 0xe9b55b7a,0x66a0db6e,0xf2722da3,0x7d67adb7
DD 0xb1266dac,0x3e33edb8,0xaae11b75,0x25f49b61
DD 0x86a8801e,0x09bd000a,0x9d6ff6c7,0x127a76d3
DD 0xb99b1b61,0x368e9b75,0xa25c6db8,0x2d49edac
DD 0x8e15f6d3,0x010076c7,0x95d2800a,0x1ac7001e
DD 0xd686c005,0x59934011,0xcd41b6dc,0x425436c8
DD 0xe1082db7,0x6e1dada3,0xfacf5b6e,0x75dadb7a
DD 0x67a0ada9,0xe8b52dbd,0x7c67db70,0xf3725b64
DD 0x502e401b,0xdf3bc00f,0x4be936c2,0xc4fcb6d6
DD 0x08bd76cd,0x87a8f6d9,0x137a0014,0x9c6f8000
DD 0x3f339b7f,0xb0261b6b,0x24f4eda6,0xabe16db2
DD 0x76da4033,0xf9cfc027,0x6d1d36ea,0xe208b6fe
DD 0x4154ad81,0xce412d95,0x5a93db58,0xd5865b4c
DD 0x19c79b57,0x96d21b43,0x0200ed8e,0x8d156d9a
DD 0x2e4976e5,0xa15cf6f1,0x358e003c,0xba9b8028
DD 0xa8e1f6fb,0x27f476ef,0xb3268022,0x3c330036
DD 0x9f6f1b49,0x107a9b5d,0x84a86d90,0x0bbded84
DD 0xc7fc2d9f,0x48e9ad8b,0xdc3b5b46,0x532edb52
DD 0xf072c02d,0x7f674039,0xebb5b6f4,0x64a036e0
DD 0xcf415b52,0x4054db46,0xd4862d8b,0x5b93ad9f
DD 0xf8cfb6e0,0x77da36f4,0xe308c039,0x6c1d402d
DD 0xa05c8036,0x2f490022,0xbb9bf6ef,0x348e76fb
DD 0x97d26d84,0x18c7ed90,0x8c151b5d,0x03009b49
DD 0x117aed9a,0x9e6f6d8e,0x0abd9b43,0x85a81b57
DD 0x26f40028,0xa9e1803c,0x3d3376f1,0xb226f6e5
DD 0x7e6736fe,0xf172b6ea,0x65a04027,0xeab5c033
DD 0x49e9db4c,0xc6fc5b58,0x522ead95,0xdd3b2d81
DD 0xedb48066,0x62a10072,0xf673f6bf,0x796676ab
DD 0xda3a6dd4,0x552fedc0,0xc1fd1b0d,0x4ee89b19
DD 0x82a95b02,0x0dbcdb16,0x996e2ddb,0x167badcf
DD 0xb527b6b0,0x3a3236a4,0xaee0c069,0x21f5407d
DD 0x338f36ae,0xbc9ab6ba,0x28484077,0xa75dc063
DD 0x0401db1c,0x8b145b08,0x1fc6adc5,0x90d32dd1
DD 0x5c92edca,0xd3876dde,0x47559b13,0xc8401b07
DD 0x6b1c0078,0xe409806c,0x70db76a1,0xffcef6b5
DD 0x542f9b07,0xdb3a1b13,0x4fe8edde,0xc0fd6dca
DD 0x63a176b5,0xecb4f6a1,0x7866006c,0xf7738078
DD 0x3b324063,0xb427c077,0x20f536ba,0xafe0b6ae
DD 0x0cbcadd1,0x83a92dc5,0x177bdb08,0x986e5b1c
DD 0x8a142dcf,0x0501addb,0x91d35b16,0x1ec6db02
DD 0xbd9ac07d,0x328f4069,0xa65db6a4,0x294836b0
DD 0xe509f6ab,0x6a1c76bf,0xfece8072,0x71db0066
DD 0xd2871b19,0x5d929b0d,0xc9406dc0,0x4655edd4
DD 0x9b6ec055,0x147b4041,0x80a9b68c,0x0fbc3698
DD 0xace02de7,0x23f5adf3,0xb7275b3e,0x3832db2a
DD 0xf4731b31,0x7b669b25,0xefb46de8,0x60a1edfc
DD 0xc3fdf683,0x4ce87697,0xd83a805a,0x572f004e
DD 0x4555769d,0xca40f689,0x5e920044,0xd1878050
DD 0x72db9b2f,0xfdce1b3b,0x691cedf6,0xe6096de2
DD 0x2a48adf9,0xa55d2ded,0x318fdb20,0xbe9a5b34
DD 0x1dc6404b,0x92d3c05f,0x06013692,0x8914b686
DD 0x22f5db34,0xade05b20,0x3932aded,0xb6272df9
DD 0x157b3686,0x9a6eb692,0x0ebc405f,0x81a9c04b
DD 0x4de80050,0xc2fd8044,0x562f7689,0xd93af69d
DD 0x7a66ede2,0xf5736df6,0x61a19b3b,0xeeb41b2f
DD 0xfcce6dfc,0x73dbede8,0xe7091b25,0x681c9b31
DD 0xcb40804e,0x4455005a,0xd087f697,0x5f927683
DD 0x93d3b698,0x1cc6368c,0x8814c041,0x07014055
DD 0xa45d5b2a,0x2b48db3e,0xbf9a2df3,0x308fade7

mul_table1_672:
DD 0x00000000,0xe417f38a,0xcdc391e5,0x29d4626f
DD 0x9e6b553b,0x7a7ca6b1,0x53a8c4de,0xb7bf3754
DD 0x393adc87,0xdd2d2f0d,0xf4f94d62,0x10eebee8
DD 0xa75189bc,0x43467a36,0x6a921859,0x8e85ebd3
DD 0x7275b90e,0x96624a84,0xbfb628eb,0x5ba1db61
DD 0xec1eec35,0x08091fbf,0x21dd7dd0,0xc5ca8e5a
DD 0x4b4f6589,0xaf589603,0x868cf46c,0x629b07e6
DD 0xd52430b2,0x3133c338,0x18e7a157,0xfcf052dd
DD 0xe4eb721c,0x00fc8196,0x2928e3f9,0xcd3f1073
DD 0x7a802727,0x9e97d4ad,0xb743b6c2,0x53544548
DD 0xddd1ae9b,0x39c65d11,0x10123f7e,0xf405ccf4
DD 0x43bafba0,0xa7ad082a,0x8e796a45,0x6a6e99cf
DD 0x969ecb12,0x72893898,0x5b5d5af7,0xbf4aa97d
DD 0x08f59e29,0xece26da3,0xc5360fcc,0x2121fc46
DD 0xafa41795,0x4bb3e41f,0x62678670,0x867075fa
DD 0x31cf42ae,0xd5d8b124,0xfc0cd34b,0x181b20c1
DD 0xcc3a92c9,0x282d6143,0x01f9032c,0xe5eef0a6
DD 0x5251c7f2,0xb6463478,0x9f925617,0x7b85a59d
DD 0xf5004e4e,0x1117bdc4,0x38c3dfab,0xdcd42c21
DD 0x6b6b1b75,0x8f7ce8ff,0xa6a88a90,0x42bf791a
DD 0xbe4f2bc7,0x5a58d84d,0x738cba22,0x979b49a8
DD 0x20247efc,0xc4338d76,0xede7ef19,0x09f01c93
DD 0x8775f740,0x636204ca,0x4ab666a5,0xaea1952f
DD 0x191ea27b,0xfd0951f1,0xd4dd339e,0x30cac014
DD 0x28d1e0d5,0xccc6135f,0xe5127130,0x010582ba
DD 0xb6bab5ee,0x52ad4664,0x7b79240b,0x9f6ed781
DD 0x11eb3c52,0xf5fccfd8,0xdc28adb7,0x383f5e3d
DD 0x8f806969,0x6b979ae3,0x4243f88c,0xa6540b06
DD 0x5aa459db,0xbeb3aa51,0x9767c83e,0x73703bb4
DD 0xc4cf0ce0,0x20d8ff6a,0x090c9d05,0xed1b6e8f
DD 0x639e855c,0x878976d6,0xae5d14b9,0x4a4ae733
DD 0xfdf5d067,0x19e223ed,0x30364182,0xd421b208
DD 0x9d995363,0x798ea0e9,0x505ac286,0xb44d310c
DD 0x03f20658,0xe7e5f5d2,0xce3197bd,0x2a266437
DD 0xa4a38fe4,0x40b47c6e,0x69601e01,0x8d77ed8b
DD 0x3ac8dadf,0xdedf2955,0xf70b4b3a,0x131cb8b0
DD 0xefecea6d,0x0bfb19e7,0x222f7b88,0xc6388802
DD 0x7187bf56,0x95904cdc,0xbc442eb3,0x5853dd39
DD 0xd6d636ea,0x32c1c560,0x1b15a70f,0xff025485
DD 0x48bd63d1,0xacaa905b,0x857ef234,0x616901be
DD 0x7972217f,0x9d65d2f5,0xb4b1b09a,0x50a64310
DD 0xe7197444,0x030e87ce,0x2adae5a1,0xcecd162b
DD 0x4048fdf8,0xa45f0e72,0x8d8b6c1d,0x699c9f97
DD 0xde23a8c3,0x3a345b49,0x13e03926,0xf7f7caac
DD 0x0b079871,0xef106bfb,0xc6c40994,0x22d3fa1e
DD 0x956ccd4a,0x717b3ec0,0x58af5caf,0xbcb8af25
DD 0x323d44f6,0xd62ab77c,0xfffed513,0x1be92699
DD 0xac5611cd,0x4841e247,0x61958028,0x858273a2
DD 0x51a3c1aa,0xb5b43220,0x9c60504f,0x7877a3c5
DD 0xcfc89491,0x2bdf671b,0x020b0574,0xe61cf6fe
DD 0x68991d2d,0x8c8eeea7,0xa55a8cc8,0x414d7f42
DD 0xf6f24816,0x12e5bb9c,0x3b31d9f3,0xdf262a79
DD 0x23d678a4,0xc7c18b2e,0xee15e941,0x0a021acb
DD 0xbdbd2d9f,0x59aade15,0x707ebc7a,0x94694ff0
DD 0x1aeca423,0xfefb57a9,0xd72f35c6,0x3338c64c
DD 0x8487f118,0x60900292,0x494460fd,0xad539377
DD 0xb548b3b6,0x515f403c,0x788b2253,0x9c9cd1d9
DD 0x2b23e68d,0xcf341507,0xe6e07768,0x02f784e2
DD 0x8c726f31,0x68659cbb,0x41b1fed4,0xa5a60d5e
DD 0x12193a0a,0xf60ec980,0xdfdaabef,0x3bcd5865
DD 0xc73d0ab8,0x232af932,0x0afe9b5d,0xeee968d7
DD 0x59565f83,0xbd41ac09,0x9495ce66,0x70823dec
DD 0xfe07d63f,0x1a1025b5,0x33c447da,0xd7d3b450
DD 0x606c8304,0x847b708e,0xadaf12e1,0x49b8e16b
```

A.4 CRC_PCL

```
;;; ISCSI CRC 32 Implementation with crc32 and pclmulqdq Instruction

%define CONCAT(a,b,c)    a %+ b %+ c

;;; unsigned int crc_pcl(unsigned char * buffer, int len, unsigned int
crc_init);
;;;
;;;        *buf = rcx
;;;         len = rdx
;;;    crc_init = r8

global  crc_pcl
crc_pcl:

        push    rbx
        push    rdi
        push    rsi

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; 1) ALIGN: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        mov     rdi, rcx        ;; rdi = *buf
        neg     rcx
        and     rcx, 7          ;; calculate the unalignment amount of
                                ;; the address
        je      proc_block      ;; Skip if aligned

        ;;;; Calculate CRC of unaligned bytes of the buffer (if any) ;;;
        mov     rbx, [rdi]      ;; load a Qword from the buffer
        add     rdi, rcx        ;; align buffer pointer for Qword
                                ;; processing
        sub     rdx, rcx        ;; update buffer length
align_loop:
        crc32   r8d, bl         ;; compute crc32 of 1-byte
        shr     rbx, 8          ;; get next byte
        dec     rcx
        jne     align_loop
```

```
proc_block:

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; 2) PROCESS  BLOCKS: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        ;; compute num of bytes to be processed
        mov     rbx, rdx        ;; save num bytes in rbx

        cmp     rdx, 128*24
        jae     full_block

continue_block:
        ;; rdx < 128*24
        mov     rax, 2731       ;; 2731 = ceil(2^16 / 24)
        mul     edx
        shr     rax, 16
        ;; eax contains floor(bytes / 24) = num 24-byte chunks to do

        ;; process rax 24-byte chunks (128 >= rax >= 0)

        ;; compute end address of each block
        ;; rdi -> block 0 (base addr + RAX * 8)
        ;; rsi -> block 1 (base addr + RAX * 16)
        ;; r11 -> block 2 (base addr + RAX * 24)
        lea     rdi, [rdi + rax * 8]
        lea     rsi, [rdi + rax * 8]
        lea     r11, [rsi + rax * 8]

        xor     r9,r9
        xor     r10,r10

        ;; branch into array
        lea     rcx, [jump_table wrt rip]
        movzx   rdx, word [rcx + rax * 2] ;; rdx is offset from crc_array
        lea     rcx, [rcx + rdx + crc_array - jump_table]
        jmp     rcx

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; 2a) PROCESS FULL BLOCKS: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
full_block:
        mov     rax, 128
        lea     rsi, [rdi + 128*8*2]
        lea     r11, [rdi + 128*8*3]
        add     rdi, 128*8*1

        xor     r9,r9
        xor     r10,r10

        ;; branch into array
        jmp     CONCAT(crc_,128,)

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; 3) CRC Array: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

crc_array:
%assign i 128
%rep 128-3
CONCAT(crc_,i,:)
        crc32   r8,  [rdi - i*8]
        crc32   r9,  [rsi - i*8]
        crc32   r10, [r11 - i*8]
%assign i (i-1)
%endrep

%rep 2
        crc32   r8,  [rdi - i*8]
        crc32   r9,  [rsi - i*8]
        crc32   r10, [r11 - i*8]
%assign i (i-1)
%endrep

        crc32   r8,  [rdi - i*8]
        crc32   r9,  [rsi - i*8]
; SKIP  ;crc32  r10, [r11 - i*8] ; Don't do this one yet

        mov     rdi, r11

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; 4) Combine three results: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        lea     rcx, [K_table - 16 wrt rip]    ; first entry is for idx 1
        shl     rax, 3                         ; rax *= 8
        sub     rbx, rax                       ; rbx -= rax*8
        shl     rax, 1
        sub     rbx, rax        ; rbx -= rax*16 (total rbx -= rax*24)
        add     rcx, rax

        movdqa  xmm0, [rcx]                    ; 2 consts: K1:K2

        movq    xmm1, r8                       ; CRC for block 1
        pclmulqdq       xmm1, xmm0, 0x00       ; Multiply by K2

        movq    xmm2, r9                       ; CRC for block 2
        pclmulqdq       xmm2, xmm0, 0x10       ; Multiply by K1

        pxor    xmm1, xmm2
        movq    rax, xmm1
        xor     rax, [r11 - i*8]
        mov     r8, r10
        crc32   r8, rax

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; 5) Check for end: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CONCAT(crc_,0,:)
        mov     rdx, rbx
        cmp     rbx, 128*24
        jae     full_block
        cmp     rbx, 24
        jae     continue_block

fewer_than_24:
        ;; now fewer than 24 bytes remain
        cmp     rbx, 16
        jae     do_16
        cmp     rbx, 8
        jae     do_8

        ;; 0 <= rbx <= 7
        shl     ebx, 29         ; size now in bits 31:29
        jz      do_return
check_4:
        mov     rcx, [rdi]
        shl     ebx, 1          ; shift out into carry MSB (orig size & 4)
        jnc     check_2
        crc32   r8d, ecx
        jz      do_return
        shr     rcx, 32         ; shift data down by 4 bytes
check_2:
        shl     ebx, 1          ; shift out into carry MSB (orig size & 2)
        jnc     check_1
        crc32   r8d, cx
        jz      do_return
        shr     rcx, 16         ; shift data down by 2 bytes
check_1:
        crc32   r8d, cl

do_return:
        mov     rax, r8
        pop     rsi
        pop     rdi
        pop     rbx
        ret

do_8:
        crc32   r8, [rdi]
        add     rdi, 8
        shl     ebx, 29         ; size (0...7) in bits 31:29
        jnz     check_4
        mov     rax, r8
        pop     rsi
        pop     rdi
        pop     rbx
        ret

do_16:
        crc32   r8, [rdi]
        crc32   r8, [rdi+8]
        add     rdi, 16
        shl     ebx, 29         ; size (0...7) in bits 31:29
        jnz     check_4
        mov     rax, r8
        pop     rsi
        pop     rdi
        pop     rbx
        ret

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; 6) For small blocks, do it by 1: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; This handles cases: 1, 2, or 3

%assign i 3
%rep 3
CONCAT(crc_,i,:)
        crc32   r8, [r11 - i*24]
        crc32   r8, [r11 - i*24 + 8]
        crc32   r8, [r11 - i*24 + 16]
%assign i (i-1)
%endrep
        mov     rdi, r11

        shl     rax, 3           ;; rax *= 8
        sub     rbx, rax         ;; rbx -= rax*8
        shl     rax, 1
        sub     rbx, rax         ;; rbx -= rax*16 (total rbx -= rax*24)

        jmp     fewer_than_24
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; jump table        ;; Table is 129 entries x 2 bytes each
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
align 4
jump_table:
%assign i 0
%rep 129
        dw      CONCAT(crc_,i,) - crc_array
%assign i (i+1)
%endrep



        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;; PCLMULQDQ tables
        ;; Table is 128 entries x 2 quad words each
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
section .data
align 64
K_table:
        dq 0x14cd00bd6, 0x105ec76f0
        dq 0x0ba4fc28e, 0x14cd00bd6
        dq 0x1d82c63da, 0x0f20c0dfe
        dq 0x09e4addf8, 0x0ba4fc28e
        dq 0x039d3b296, 0x1384aa63a
        dq 0x102f9b8a2, 0x1d82c63da
        dq 0x14237f5e6, 0x01c291d04
        dq 0x00d3b6092, 0x09e4addf8
        dq 0x0c96cfdc0, 0x0740eef02
        dq 0x18266e456, 0x039d3b296
        dq 0x0daece73e, 0x0083a6eec
        dq 0x0ab7aff2a, 0x102f9b8a2
        dq 0x1248ea574, 0x1c1733996
        dq 0x083348832, 0x14237f5e6
        dq 0x12c743124, 0x02ad91c30
        dq 0x0b9e02b86, 0x00d3b6092
        dq 0x018b33a4e, 0x06992cea2
        dq 0x1b331e26a, 0x0c96cfdc0
        dq 0x17d35ba46, 0x07e908048
        dq 0x1bf2e8b8a, 0x18266e456
        dq 0x1a3e0968a, 0x11ed1f9d8
        dq 0x0ce7f39f4, 0x0daece73e
        dq 0x061d82e56, 0x0f1d0f55e
        dq 0x0d270f1a2, 0x0ab7aff2a
        dq 0x1c3f5f66c, 0x0a87ab8a8
        dq 0x12ed0daac, 0x1248ea574
        dq 0x065863b64, 0x08462d800
        dq 0x11eef4f8e, 0x083348832
        dq 0x1ee54f54c, 0x071d111a8
        dq 0x0b3e32c28, 0x12c743124
        dq 0x0064f7f26, 0x0ffd852c6
        dq 0x0dd7e3b0c, 0x0b9e02b86
        dq 0x0f285651c, 0x0dcb17aa4
        dq 0x010746f3c, 0x018b33a4e
        dq 0x1c24afea4, 0x0f37c5aee
        dq 0x0271d9844, 0x1b331e26a
        dq 0x08e766a0c, 0x06051d5a2
        dq 0x093a5f730, 0x17d35ba46
        dq 0x06cb08e5c, 0x11d5ca20e
        dq 0x06b749fb2, 0x1bf2e8b8a
        dq 0x1167f94f2, 0x021f3d99c
        dq 0x0cec3662e, 0x1a3e0968a
        dq 0x19329634a, 0x08f158014
        dq 0x0e6fc4e6a, 0x0ce7f39f4
        dq 0x08227bb8a, 0x1a5e82106
        dq 0x0b0cd4768, 0x061d82e56
        dq 0x13c2b89c4, 0x188815ab2
        dq 0x0d7a4825c, 0x0d270f1a2
        dq 0x10f5ff2ba, 0x105405f3e
        dq 0x00167d312, 0x1c3f5f66c
        dq 0x0f6076544, 0x0e9adf796
        dq 0x026f6a60a, 0x12ed0daac
        dq 0x1a2adb74e, 0x096638b34
        dq 0x19d34af3a, 0x065863b64
        dq 0x049c3cc9c, 0x1e50585a0
        dq 0x068bce87a, 0x11eef4f8e
        dq 0x1524fa6c6, 0x19f1c69dc
        dq 0x16cba8aca, 0x1ee54f54c
        dq 0x042d98888, 0x12913343e
        dq 0x1329d9f7e, 0x0b3e32c28
        dq 0x1b1c69528, 0x088f25a3a
        dq 0x02178513a, 0x0064f7f26
        dq 0x0e0ac139e, 0x04e36f0b0
        dq 0x0170076fa, 0x0dd7e3b0c
        dq 0x141a1a2e2, 0x0bd6f81f8
        dq 0x16ad828b4, 0x0f285651c
        dq 0x041d17b64, 0x19425cbba
        dq 0x1fae1cc66, 0x010746f3c
        dq 0x1a75b4b00, 0x18db37e8a
        dq 0x0f872e54c, 0x1c24afea4
        dq 0x01e41e9fc, 0x04c144932
        dq 0x086d8e4d2, 0x0271d9844
        dq 0x160f7af7a, 0x052148f02
        dq 0x05bb8f1bc, 0x08e766a0c
        dq 0x0a90fd27a, 0x0a3c6f37a
        dq 0x0b3af077a, 0x093a5f730
        dq 0x04984d782, 0x1d22c238e
        dq 0x0ca6ef3ac, 0x06cb08e5c
        dq 0x0234e0b26, 0x063ded06a
        dq 0x1d88abd4a, 0x06b749fb2
        dq 0x04597456a, 0x04d56973c
        dq 0x0e9e28eb4, 0x1167f94f2
        dq 0x07b3ff57a, 0x19385bf2e
        dq 0x0c9c8b782, 0x0cec3662e
        dq 0x13a9cba9e, 0x0e417f38a
        dq 0x093e106a4, 0x19329634a
        dq 0x167001a9c, 0x14e727980
        dq 0x1ddffc5d4, 0x0e6fc4e6a
        dq 0x00df04680, 0x0d104b8fc
        dq 0x02342001e, 0x08227bb8a
        dq 0x00a2a8d7e, 0x05b397730
        dq 0x168763fa6, 0x0b0cd4768
        dq 0x1ed5a407a, 0x0e78eb416
        dq 0x0d2c3ed1a, 0x13c2b89c4
        dq 0x0995a5724, 0x1641378f0
        dq 0x19b1afbc4, 0x0d7a4825c
        dq 0x109ffedc0, 0x08d96551c
        dq 0x0f2271e60, 0x10f5ff2ba
        dq 0x00b0bf8ca, 0x00bf80dd2
        dq 0x123888b7a, 0x00167d312
        dq 0x1e888f7dc, 0x18dcddd1c
        dq 0x002ee03b2, 0x0f6076544
        dq 0x183e8d8fe, 0x06a45d2b2
        dq 0x133d7a042, 0x026f6a60a
        dq 0x116b0f50c, 0x1dd3e10e8
        dq 0x05fabe670, 0x1a2adb74e
        dq 0x130004488, 0x0de87806c
        dq 0x000bcf5f6, 0x19d34af3a
        dq 0x18f0c7078, 0x014338754
        dq 0x017f27698, 0x049c3cc9c
        dq 0x058ca5f00, 0x15e3e77ee
        dq 0x1af900c24, 0x068bce87a
        dq 0x0b5cfca28, 0x0dd07448e
        dq 0x0ded288f8, 0x1524fa6c6
        dq 0x059f229bc, 0x1d8048348
        dq 0x06d390dec, 0x16cba8aca
        dq 0x037170390, 0x0a3e3e02c
        dq 0x06353c1cc, 0x042d98888
        dq 0x0c4584f5c, 0x0d73c7bea
        dq 0x1f16a3418, 0x1329d9f7e
        dq 0x0531377e2, 0x185137662
        dq 0x1d8d9ca7c, 0x1b1c69528
        dq 0x0b25b29f2, 0x18a08b5bc
        dq 0x19fb2a8b0, 0x02178513a
        dq 0x1a08fe6ac, 0x1da758ae0
        dq 0x045cddf4e, 0x0e0ac139e
        dq 0x1a91647f2, 0x169cf9eb0
        dq 0x1a0f717c4, 0x0170076fa
```

A.5 CRC_1024_PCL

```
; Function to compute iSCSI CRC32 with PCLMULQDQ Instruction
; crc done "by 3" for fixed input size of 1024 bytes

;;; ISCSI CRC 32 Implementation with crc32 Instruction

;; 1024 = 336 * 3 * 8 + 16
%define BLOCKSIZE 336

;;; unsigned int crc_1024_pcl(unsigned char * buffer, unsigned
int crc_init);
;;;
;;;         *buf = rcx
;;;     crc_init = rdx
;;;

global crc_1024_pcl
crc_1024_pcl:

        mov     r9, rcx

                                ;; rdx = crc0 = crc_init
        xor     r8, r8          ;; r8 = crc1
        xor     rax, rax        ;; rax = crc2

        ;; Do first 8 bytes here for better pipelining
        crc32   rdx, [r9]

        ;; process block inline
        ;; process rdx last to avoid dependency with crc32 above
%assign i 8
%rep BLOCKSIZE/8
        crc32   r8,  [r9 + i + 1*BLOCKSIZE]     ; crc1
        crc32   rax, [r9 + i + 2*BLOCKSIZE]     ; crc2
        crc32   rdx, [r9 + i + 0*BLOCKSIZE]     ; crc0
%assign i (i+8)
%endrep

        movdqa  xmm0, [K344_1 wrt rip]          ; 2 consts: K1:K2
        ; Merge crc0 and crc1 into crc2

        movq    xmm1, r8                            ; crc1
        pclmulqdq       xmm1, xmm0, 0x10            ; Multiply by K1

        movq    xmm2, rdx                           ; crc0
        pclmulqdq       xmm2, xmm0, 0x00            ; Multiply by K2

        crc32   rax, [r9 + 1016]

        xor     r9, r9
        movq    r8, xmm1
        crc32   r9, r8

        xor     rax, r9

        xor     r9, r9
        movq    r8, xmm2
        crc32   r9, r8

        xor     rax, r9

        ret

section .data
align   16
K344_1: dq 0x0e417f38a
K344_2: dq 0x08f158014
```

Authors

Vinodh Gopal, **Jim Guilford**, **Erdinc Ozturk**, **Gil Wolrich**, **Wajdi Feghali** and **Martin Dixon** are IA Architects with the IAG Group at Intel Corporation.

Deniz Karakoyunlu is a PhD alumnus of the Worcester Polytechnic Institute, currently working for Marvell Technology Group.

323405