

new/usr/src/common/crypto/sha1/amd64/THIRDPARTYLICENSE

1

1758 Mon Mar 3 13:14:56 2008

new/usr/src/common/crypto/sha1/amd64/THIRDPARTYLICENSE

6662791 Need a SHA1 implementation optimized for 64-bit x86

1 Copyright (c) 2006, CRYPTOGAMS by <appro@openssl.org>

2 All rights reserved.

4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions
6 are met:

8 * Redistributions of source code must retain copyright notices,
9 this list of conditions and the following disclaimer.

11 * Redistributions in binary form must reproduce the above
12 copyright notice, this list of conditions and the following
13 disclaimer in the documentation and/or other materials
14 provided with the distribution.

16 * Neither the name of the CRYPTOGAMS nor the names of its
17 copyright holder and contributors may be used to endorse or
18 promote products derived from this software without specific
19 prior written permission.

21 ALTERNATIVELY, provided that this notice is retained in full, this
22 product may be distributed under the terms of the GNU General Public
23 License (GPL), in which case the provisions of the GPL apply INSTEAD OF
24 those given above.

26 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS
27 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
28 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
29 A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
30 OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
31 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
32 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
33 DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
34 THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
35 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
36 OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

new/usr/src/common/crypto/sha1/amd64/THIRDPARTYLICENSE.descript1

31 Mon Mar 3 13:14:57 2008
new/usr/src/common/crypto/sha1/amd64/THIRDPARTYLICENSE.descript
6662791 Need a SHA1 implementation optimized for 64-bit x86

1 PORTIONS OF SHA1 FUNCTIONALITY

new/usr/src/common/crypto/sha1/amd64/sha1-x86_64.pl

1

```
*****
9216 Mon Mar 3 13:14:58 2008
new/usr/src/common/crypto/sha1/amd64/sha1-x86_64.pl
6662791 Need a SHA1 implementation optimized for 64-bit x86
*****

1 #!/usr/bin/env perl
2 #
3 # =====
4 # Written by Andy Polyakov <appro@fy.chalmers.se> for the OpenSSL
5 # project. The module is, however, dual licensed under OpenSSL and
6 # CRYPTOGAMS licenses depending on where you obtain it. For further
7 # details see http://www.openssl.org/~appro/cryptogams/.
8 # =====
9 #
10 # sha1_block procedure for x86_64.
11 #
12 # It was brought to my attention that on EM64T compiler-generated code
13 # was far behind 32-bit assembler implementation. This is unlike on
14 # Opteron where compiler-generated code was only 15% behind 32-bit
15 # assembler, which originally made it hard to motivate the effort.
16 # There was suggestion to mechanically translate 32-bit code, but I
17 # dismissed it, reasoning that x86_64 offers enough register bank
18 # capacity to fully utilize SHA-1 parallelism. Therefore this fresh
19 # implementation:-) However! While 64-bit code does performs better
20 # on Opteron, I failed to beat 32-bit assembler on EM64T core. Well,
21 # x86_64 does offer larger *addressable* bank, but out-of-order core
22 # reaches for even more registers through dynamic aliasing, and EM64T
23 # core must have managed to run-time optimize even 32-bit code just as
24 # good as 64-bit one. Performance improvement is summarized in the
25 # following table:
26 #
27 # gcc 3.4      32-bit asm      cycles/byte
28 # Opteron      +45%           +20%           6.8
29 # Xeon P4       +65%           +0%           9.9
30 # Core2        +60%           +10%          7.0
31 #
32 #
33 # OpenSolaris OS modifications
34 #
35 # Sun elects to use this software under the BSD license.
36 #
37 # This source originates from OpenSSL file sha1-x86_64.pl at
38 # ftp://ftp.openssl.org/snapshot/openssl-0.9.8-stable-SNAP-20080131.tar.gz
39 # (presumably for future OpenSSL release 0.9.8h), with these changes:
40 #
41 # 1. Added perl "use strict" and declared variables.
42 #
43 # 2. Added OpenSolaris ENTRY_NP/SET_SIZE macros from
44 # /usr/include/sys/asm_linkage.h, .ident keywords, and lint(1B) guards.
45 #
46 # 3. Added perl function &lea_offset_eax_register_register() to handle
47 # Solaris as(1) bug.
48 #
49 # 4. Removed x86_64-xlate.pl script (not needed for as(1) or gas(1) assemblers).
50 #
51
52 use strict;
53 my ($code, $ctx, $inp, $num, $xi, $t0, $t1, $i, @V, $A, $B, $C, $D, $E, $T);
54 my $soutp = shift;
55 open STDOUT, ">$soutp";
56
57 sub lea_offset_eax_register_register
58 # Workaround for a Solaris "gas" assembler bug where compiling the source
59 # errors out and does not generate a valid "lea" instruction. Specifically,
60 # &lea OFFSET(%eax, SOURCE_REGISTER), DESTINATION_REGISTER
```

new/usr/src/common/crypto/sha1/amd64/sha1-x86_64.pl

2

```
62 #
63 # For Solaris as, "as -a32" must be used to compile this.
64 # For Solaris gas 2.15, this errors out with this message:
65 # Error: '0x5a827999(%eax,%r11d)' is not a valid 64 bit base/index expression
66 #
67 # This should be fixed in Solaris gas 2.16.
68 # It assembles with the Linux "as --64" gas 2.17 assembler and runs OK.
69 #
70 # For the ONBLD NV tools, the aw wrapper script fails when -a32 is used:
71 # /ws/onnv-tools/onbld/bin/i386/aw -xarch=amd64 -P -a32 -o lea.o lea.s
72 # aw: as->gas mapping failed at or near arg '-a32'
73 #
74 # For more information, see CRs 6644870 and 6628627.
75 {
76     use Switch;
77     my ($offset, $reg_src, $reg_dest) = @_;
78
79     # Failed "lea" instruction.
80     # This instruction errors out from the Solaris as assembler.
81     # It assembles with the Linux "as --64" assembler and runs OK.
82     $code .= "    /lea    $offset(%eax,$reg_src),$reg_dest\n";
83
84     # Workaround
85     # This workaround hand-generates hex machine code for lea.
86     $code .= "    / Solaris as assembly bug CR 6628627 errors out for\n";
87     $code .= "    / the above, so we specify the machine code in hex:\n";
88     $code .= "    .byte 0x67    / lea\n";
89
90     switch ($reg_src) {
91     case "%ebp"
92     {
93         switch ($reg_dest) {
94         case "%r11d" { $code .=
95             "    .byte 0x44,0x8d,0x9c,0x28    "
96             . "/ (%eax,$reg_src),$reg_dest\n"; }
97         else { $code .= "Unknown register $reg_dest\n"; }
98         }
99     case "%edi"
100     {
101         switch ($reg_dest) {
102         case "%ebp" { $code .=
103             "    .byte 0x8d,0xac,0x38    "
104             . "/ (%eax,$reg_src),$reg_dest\n"; }
105         else { $code .= "Unknown register $reg_dest\n"; }
106         }
107     case "%edx"
108     {
109         switch ($reg_dest) {
110         case "%esi" { $code .=
111             "    .byte 0x8d,0xb4,0x10    "
112             . "/ (%eax,$reg_src),$reg_dest\n"; }
113         else { $code .= "Unknown register $reg_dest\n"; }
114         }
115     case "%esi"
116     {
117         switch ($reg_dest) {
118         case "%edi" { $code .=
119             "    .byte 0x8d,0xbc,0x30    "
120             . "/ (%eax,$reg_src),$reg_dest\n"; }
121         else { $code .= "Unknown register $reg_dest\n"; }
122         }
123     case "%r11d"
124     {
125         switch ($reg_dest) {
126         case "%r12d" { $code .=
127             "    .byte 0x46,0x8d,0xa4,0x18    "
128             . "/ (%eax,$reg_src),$reg_dest\n"; }
129         else { $code .= "Unknown register $reg_dest\n"; }
130         }
131     }
132     }
```

```

128         else { $code .= "Unknown register $reg_dest\n"; }
129     }
130 }
131 case "%r12d" {
132     switch ($reg_dest) {
133         case "%edx" { $code .=
134             ".byte    0x42,0x8d,0x94,0x20    "
135             . "/ (%eax,$reg_src),$reg_dest\n"; }
136         else { $code .= "Unknown register $reg_dest\n"; }
137     }
138 }
139 else { $code .= "Unknown register $reg_src\n"; }
140 }

142 $code .= "        .long    $offset / offset\n";
143 }

```

```

146 #
147 # void sha1_block_data_order(SHA1_CTX *ctx, const void *inpp, size_t blocks);
148 #

```

```

150 # Arguments:
151 $ctx="%rdi";    # 1st arg
152 $inp="%rsi";    # 2nd arg
153 $num="%rdx";    # 3rd arg

```

```

155 # reassign arguments in order to produce more compact code
156 $ctx="%r8";
157 $inp="%r9";
158 $num="%r10";

```

```

160 # Temporaries:
161 $xi="%eax";
162 $t0="%ebx";
163 $t1="%ecx";
164 # State information from SHA-1 context:
165 $A="%edx";
166 $B="%esi";
167 $C="%edi";
168 $D="%ebp";
169 $E="%r11d";
170 # Temporary:
171 $T="%r12d";

```

```

173 @V=($A,$B,$C,$D,$E,$T);

```

```

175 sub PROLOGUE {
176     my $func=shift;
177     $code.=<<<__ ;
178     ENTRY_NP($func)
179     /* EXPORT DELETE START */
180     push    %rbx
181     push    %rbp
182     push    %r12
183     mov     %rsp,%rax
184     mov     %rdi,$ctx    # reassigned argument
185     sub     \$(8+16*4),%rsp
186     mov     %rsi,$inp    # reassigned argument
187     and     \$(64,%rsp
188     mov     %rdx,$num    # reassigned argument
189     mov     %rax,'16*4'(%rsp)

191     mov     0($ctx),$A
192     mov     4($ctx),$B
193     mov     8($ctx),$C

```

```

194     mov     12($ctx),$D
195     mov     16($ctx),$E
196
197 }

```

```

199 sub EPILOGUE {
200     my $func=shift;
201     $code.=<<<__ ;
202     mov     '16*4'(%rsp),%rsp
203     pop     %r12
204     pop     %rbp
205     pop     %rbx
206     /* EXPORT DELETE END */
207     ret
208     SET_SIZE($func)
209
210 }

```

```

212 sub BODY_00_19 {
213     my ($i,$a,$b,$c,$d,$e,$f,$host)=@_ ;
214     my $j=$i+1;
215     $code.=<<<__ if ($i==0);
216     mov     '4*$i'($inp),$xi
217     "bswap $xi" if(!defined($host))'
218     mov     $xi,'4*$i'(%rsp)
219
220     &lea_offset_eax_register_register("0x5a827999", $e, $f) if ($i < 15);
221     $code.=<<<__ if ($i<15);
222     /lea     0x5a827999($xi,$e),$f
223     mov     $c,$t0
224     mov     '4*$j'($inp),$xi
225     mov     $a,$e
226     xor     $d,$t0
227     "bswap $xi" if(!defined($host))'
228     rol     \$(5,$e
229     and     $b,$t0
230     mov     $xi,'4*$j'(%rsp)
231     add     $e,$f
232     xor     $d,$t0
233     rol     \$(30,$b
234     add     $t0,$f
235
236     &lea_offset_eax_register_register("0x5a827999", $e, $f) if ($i >= 15);
237     $code.=<<<__ if ($i>=15);
238     /lea     0x5a827999($xi,$e),$f
239     mov     '4*($j%16)'(%rsp),$xi
240     mov     $c,$t0
241     mov     $a,$e
242     xor     '4*(($j+2)%16)'(%rsp),$xi
243     xor     $d,$t0
244     rol     \$(5,$e
245     xor     '4*(($j+8)%16)'(%rsp),$xi
246     and     $b,$t0
247     add     $e,$f
248     xor     '4*(($j+13)%16)'(%rsp),$xi
249     xor     $d,$t0
250     rol     \$(30,$b
251     add     $t0,$f
252     rol     \$(1,$xi
253     mov     $xi,'4*($j%16)'(%rsp)
254
255 }

```

```

257 sub BODY_20_39 {
258     my ($i,$a,$b,$c,$d,$e,$f)=@_ ;
259     my $j=$i+1;

```

```

260 my $K=(($i<40)?0x6ed9eba1:0xca62c1d6;
261      &lea_offset_eax_register_register($K, $e, $f) if ($i < 79);
262 $code.=<<__ if ($i<79);
263      /lea      $K($xi,$e),$f
264      mov      '4*($j%16)'(%rsp),$xi
265      mov      $c,$t0
266      mov      $a,$e
267      xor      '4*((($j+2)%16)'(%rsp),$xi
268      xor      $b,$t0
269      rol      \5,$e
270      xor      '4*((($j+8)%16)'(%rsp),$xi
271      xor      $d,$t0
272      add      $e,$f
273      xor      '4*((($j+13)%16)'(%rsp),$xi
274      rol      \30,$b
275      add      $t0,$f
276      rol      \1,$xi
277
278 $code.=<<__ if ($i<76);
279      mov      $xi,'4*($j%16)'(%rsp)
280
281      &lea_offset_eax_register_register($K, $e, $f) if ($i == 79);
282 $code.=<<__ if ($i==79);
283      /lea      $K($xi,$e),$f
284      mov      $c,$t0
285      mov      $a,$e
286      xor      $b,$t0
287      rol      \5,$e
288      xor      $d,$t0
289      add      $e,$f
290      rol      \30,$b
291      add      $t0,$f
292
293 }

295 sub BODY_40_59 {
296 my ($i,$a,$b,$c,$d,$e,$f)=@_;
297 my $j=$i+1;
298      &lea_offset_eax_register_register("0x8f1bbcdc", $e, $f);
299 $code.=<<__;
300      /lea      0x8f1bbcdc($xi,$e),$f
301      mov      '4*($j%16)'(%rsp),$xi
302      mov      $b,$t0
303      mov      $b,$t1
304      xor      '4*((($j+2)%16)'(%rsp),$xi
305      mov      $a,$e
306      and      $c,$t0
307      xor      '4*((($j+8)%16)'(%rsp),$xi
308      or       $c,$t1
309      rol      \5,$e
310      xor      '4*((($j+13)%16)'(%rsp),$xi
311      and      $d,$t1
312      add      $e,$f
313      rol      \1,$xi
314      or       $t1,$t0
315      rol      \30,$b
316      mov      $xi,'4*($j%16)'(%rsp)
317      add      $t0,$f
318
319 }

321 $code=<<__;
322 #if !defined(lint) && !defined(__lint)
323      .ident      "@(#)sha1-x86_64.pl      1.1      08/03/02 SMI"
324 #include <sys/asm_linkage.h>
325

```

```

328 &PROLOGUE("sha1_block_data_order");
329 $code.="align 4\n.Lloop:\n";
330 for($i=0;$i<20;$i++) { &BODY_00_19($i,@v); unshift(@v,pop(@v)); }
331 for(; $i<40;$i++) { &BODY_20_39($i,@v); unshift(@v,pop(@v)); }
332 for(; $i<60;$i++) { &BODY_40_59($i,@v); unshift(@v,pop(@v)); }
333 for(; $i<80;$i++) { &BODY_20_39($i,@v); unshift(@v,pop(@v)); }
334 $code.=<<__;
335      / Update and save state information in SHA-1 context
336      add      0($ctx),$E
337      add      4($ctx),$T
338      add      8($ctx),$A
339      add      12($ctx),$B
340      add      16($ctx),$C
341      mov      $E,0($ctx)
342      mov      $T,4($ctx)
343      mov      $A,8($ctx)
344      mov      $B,12($ctx)
345      mov      $C,16($ctx)

347      xchg     $E,$A # mov $E,$A
348      xchg     $T,$B # mov $T,$B
349      xchg     $E,$C # mov $A,$C
350      xchg     $T,$D # mov $B,$D
351                      # mov $C,$E
352      lea      '16*4'($inp),$inp
353      sub      \1,$num
354      jnz      .Lloop
355
356 &EPILOGUE("sha1_block_data_order");
357 $code.=<<__;
358 .asciz      "SHA1 block transform for x86_64, CRYPTOGAMS by <appro@openssl.org>"

360 #else
361      /* LINTED */
362      /* Nothing to be linted in this file--it's pure assembly source. */
363 #endif /* !lint && !__lint */
364
366 #####

368 $code =~ s/\'([^\']*)\'/eval $1/gem;
369 print $code;
370 close STDOUT;

```

new/usr/src/common/crypto/sha1/sha1.c

1

```
*****
31673 Mon Mar  3 13:14:59 2008
new/usr/src/common/crypto/sha1/sha1.c
6662791 Need a SHA1 implementation optimized for 64-bit x86
*****
1 /*
2  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
3  * Copyright 2007 Sun Microsystems, Inc. All rights reserved.
4  * Use is subject to license terms.
5  */
6 #pragma ident    "@(#)sha1.c      1.27      08/03/02 SMI"
6 #pragma ident    "@(#)sha1.c      1.26      07/04/10 SMI"
7
8 /*
9  * The basic framework for this code came from the reference
10 * implementation for MD5. That implementation is Copyright (C)
11 * 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
12 *
13 * License to copy and use this software is granted provided that it
14 * is identified as the "RSA Data Security, Inc. MD5 Message-Digest
15 * Algorithm" in all material mentioning or referencing this software
16 * or this function.
17 *
18 * License is also granted to make and use derivative works provided
19 * that such works are identified as "derived from the RSA Data
20 * Security, Inc. MD5 Message-Digest Algorithm" in all material
21 * mentioning or referencing the derived work.
22 *
23 * RSA Data Security, Inc. makes no representations concerning either
24 * the merchantability of this software or the suitability of this
25 * software for any particular purpose. It is provided "as is"
26 * without express or implied warranty of any kind.
27 *
28 * These notices must be retained in any copies of any part of this
29 * documentation and/or software.
30 *
31 * NOTE: Cleaned-up and optimized, version of SHA1, based on the FIPS 180-1
32 * standard, available at http://www.itl.nist.gov/div897/pubs/fip180-1.htm
33 * Not as fast as one would like -- further optimizations are encouraged
34 * and appreciated.
35 */
36
37 #include <sys/types.h>
38 #include <sys/param.h>
39 #include <sys/system.h>
40 #include <sys/sysmacros.h>
41 #include <sys/sha1.h>
42 #include <sys/sha1-consts.h>
43
44 #ifndef _KERNEL
45 #include <strings.h>
46 #include <stdlib.h>
47 #include <errno.h>
48 #include <sys/systeminfo.h>
49 #endif /* !_KERNEL */
50
51 static void Encode(uint8_t *, const uint32_t *, size_t);
52
53 #if defined(__sparc)
54
55 #define SHA1_TRANSFORM(ctx, in) \
56     SHA1Transform((ctx)->state[0], (ctx)->state[1], (ctx)->state[2], \
57         (ctx)->state[3], (ctx)->state[4], (ctx), (in))
58
59 static void SHA1Transform(uint32_t, uint32_t, uint32_t, uint32_t, uint32_t,
```

new/usr/src/common/crypto/sha1/sha1.c

2

```
60     SHA1_CTX *, const uint8_t *);
61
62 #elif defined(__amd64)
63
64 #define SHA1_TRANSFORM(ctx, in) sha1_block_data_order((ctx), (in), 1)
65 #define SHA1_TRANSFORM_BLOCKS(ctx, in, num) sha1_block_data_order((ctx), \
66     (in), (num))
67
68 void sha1_block_data_order(SHA1_CTX *ctx, const void *inpp, size_t num_blocks);
69
70 #else
71
72 #define SHA1_TRANSFORM(ctx, in) SHA1Transform((ctx), (in))
73
74 static void SHA1Transform(SHA1_CTX *, const uint8_t *);
75
76 #endif
77
78 static uint8_t PADDING[64] = { 0x80, /* all zeros */ };
79
80 /*
81  * F, G, and H are the basic SHA1 functions.
82  */
83 #define F(b, c, d) (((b) & (c)) | ((~b) & (d)))
84 #define G(b, c, d) ((b) ^ (c) ^ (d))
85 #define H(b, c, d) (((b) & (c)) | (((b)|(c)) & (d)))
86
87 /*
88  * ROTATE_LEFT rotates x left n bits.
89  */
90
91 #if defined(__GNUC__) && defined(__LP64__)
92 static __inline__ uint64_t
93 ROTATE_LEFT(uint64_t value, uint32_t n)
94 {
95     uint32_t t32;
96
97     t32 = (uint32_t)value;
98     return ((t32 << n) | (t32 >> (32 - n)));
99 }
100 }
101
102 #ifndef _unchanged_portion_omitted_
103
104 #ifdef VIS_SHA1
105 #ifndef _KERNEL
106
107 #include <sys/regset.h>
108 #include <sys/vis.h>
109 #include <sys/fpu/fpusystem.h>
110
111 /* the alignment for block stores to save fp registers */
112 #define VIS_ALIGN (64)
113
114 extern int sha1_savefp(kfpu_t *, int);
115 extern void sha1_restorefp(kfpu_t *);
116
117 uint32_t vis_sha1_svfp_threshold = 128;
118
119 #endif /* !_KERNEL */
120
121 /*
122  * VIS SHA-1 consts.
123  */
124 static uint64_t VIS[] = {
125     0x8000000000000000ULL,
126     0x0002000200020002ULL,
127
128     0x0000000000000000ULL,
129     0x0000000000000000ULL,
130     0x0000000000000000ULL,
131     0x0000000000000000ULL,
132     0x0000000000000000ULL,
133     0x0000000000000000ULL,
134     0x0000000000000000ULL,
135     0x0000000000000000ULL,
136     0x0000000000000000ULL,
137     0x0000000000000000ULL,
138     0x0000000000000000ULL,
139     0x0000000000000000ULL,
140     0x0000000000000000ULL,
141     0x0000000000000000ULL,
142     0x0000000000000000ULL,
143     0x0000000000000000ULL,
144     0x0000000000000000ULL,
145     0x0000000000000000ULL,
146     0x0000000000000000ULL,
147     0x0000000000000000ULL,
148     0x0000000000000000ULL,
149     0x0000000000000000ULL,
150     0x0000000000000000ULL,
151     0x0000000000000000ULL,
152     0x0000000000000000ULL,
153     0x0000000000000000ULL,
154     0x0000000000000000ULL,
155     0x0000000000000000ULL,
156     0x0000000000000000ULL,
157     0x0000000000000000ULL,
158     0x0000000000000000ULL,
159     0x0000000000000000ULL,
160     0x0000000000000000ULL,
161     0x0000000000000000ULL,
162     0x0000000000000000ULL,
163     0x0000000000000000ULL,
164     0x0000000000000000ULL,
165     0x0000000000000000ULL,
166     0x0000000000000000ULL,
167     0x0000000000000000ULL,
168     0x0000000000000000ULL,
169     0x0000000000000000ULL,
170     0x0000000000000000ULL,
171     0x0000000000000000ULL,
172     0x0000000000000000ULL,
173     0x0000000000000000ULL,
174     0x0000000000000000ULL,
175     0x0000000000000000ULL,
176     0x0000000000000000ULL,
177     0x0000000000000000ULL,
178     0x0000000000000000ULL,
179     0x0000000000000000ULL,
180     0x0000000000000000ULL,
181     0x0000000000000000ULL,
182     0x0000000000000000ULL,
183     0x0000000000000000ULL,
184     0x0000000000000000ULL,
185     0x0000000000000000ULL,
186     0x0000000000000000ULL,
187     0x0000000000000000ULL,
188     0x0000000000000000ULL,
189     0x0000000000000000ULL,
190     0x0000000000000000ULL,
191     0x0000000000000000ULL,
192     0x0000000000000000ULL,
193     0x0000000000000000ULL,
194     0x0000000000000000ULL,
195     0x0000000000000000ULL,
196     0x0000000000000000ULL,
197     0x0000000000000000ULL,
198     0x0000000000000000ULL,
199     0x0000000000000000ULL,
200     0x0000000000000000ULL,
201     0x0000000000000000ULL,
202     0x0000000000000000ULL,
203     0x0000000000000000ULL,
204     0x0000000000000000ULL,
205     0x0000000000000000ULL,
206     0x0000000000000000ULL,
207     0x0000000000000000ULL,
208     0x0000000000000000ULL,
209     0x0000000000000000ULL,
210     0x0000000000000000ULL,
211     0x0000000000000000ULL,
212     0x0000000000000000ULL,
213     0x0000000000000000ULL,
214     0x0000000000000000ULL,
215     0x0000000000000000ULL,
216     0x0000000000000000ULL,
217     0x0000000000000000ULL,
218     0x0000000000000000ULL,
219     0x0000000000000000ULL,
220     0x0000000000000000ULL,
221     0x0000000000000000ULL,
222     0x0000000000000000ULL,
223     0x0000000000000000ULL,
224     0x0000000000000000ULL,
225     0x0000000000000000ULL,
226     0x0000000000000000ULL,
227     0x0000000000000000ULL,
228     0x0000000000000000ULL,
229     0x0000000000000000ULL,
230     0x0000000000000000ULL,
231     0x0000000000000000ULL,
232     0x0000000000000000ULL,
233     0x0000000000000000ULL,
234     0x0000000000000000ULL,
235     0x0000000000000000ULL,
236     0x0000000000000000ULL,
237     0x0000000000000000ULL,
238     0x0000000000000000ULL,
239     0x0000000000000000ULL,
240     0x0000000000000000ULL,
241     0x0000000000000000ULL,
242     0x0000000000000000ULL,
243     0x0000000000000000ULL,
244     0x0000000000000000ULL,
245     0x0000000000000000ULL,
246     0x0000000000000000ULL,
247     0x0000000000000000ULL,
248     0x0000000000000000ULL,
249     0x0000000000000000ULL,
250     0x0000000000000000ULL,
251     0x0000000000000000ULL,
252     0x0000000000000000ULL,
253     0x0000000000000000ULL,
254     0x0000000000000000ULL,
255     0x0000000000000000ULL,
256     0x0000000000000000ULL,
257     0x0000000000000000ULL,
258     0x0000000000000000ULL,
259     0x0000000000000000ULL,
260     0x0000000000000000ULL,
261     0x0000000000000000ULL,
262     0x0000000000000000ULL,
263     0x0000000000000000ULL,
264     0x0000000000000000ULL,
265     0x0000000000000000ULL,
266     0x0000000000000000ULL,
267     0x0000000000000000ULL,
268     0x0000000000000000ULL,
269     0x0000000000000000ULL,
270     0x0000000000000000ULL,
271     0x0000000000000000ULL,
272     0x0000000000000000ULL,
273     0x0000000000000000ULL,
274     0x0000000000000000ULL,
275     0x0000000000000000ULL,
276     0x0000000000000000ULL,
277     0x0000000000000000ULL,
278     0x0000000000000000ULL,
279     0x0000000000000000ULL,
280     0x0000000000000000ULL,
281     0x0000000000000000ULL,
282     0x0000000000000000ULL,
283     0x0000000000000000ULL,
284     0x0000000000000000ULL,
285     0x0000000000000000ULL,
286     0x0000000000000000ULL,
287     0x0000000000000000ULL,
288     0x0000000000000000ULL,
289     0x0000000000000000ULL,
290     0x0000000000000000ULL,
291     0x0000000000000000ULL,
292     0x0000000000000000ULL,
293     0x0000000000000000ULL,
294     0x0000000000000000ULL,
295     0x0000000000000000ULL,
296     0x0000000000000000ULL,
297     0x0000000000000000ULL,
298     0x0000000000000000ULL,
299     0x0000000000000000ULL,
300     0x0000000000000000ULL,
301     0x0000000000000000ULL,
302     0x0000000000000000ULL,
303     0x0000000000000000ULL,
304     0x0000000000000000ULL,
305     0x0000000000000000ULL,
306     0x0000000000000000ULL,
307     0x0000000000000000ULL,
308     0x0000000000000000ULL,
309     0x0000000000000000ULL,
310     0x0000000000000000ULL,
311     0x0000000000000000ULL,
312     0x0000000000000000ULL,
313     0x0000000000000000ULL,
314     0x0000000000000000ULL,
315     0x0000000000000000ULL,
316     0x0000000000000000ULL,
317     0x0000000000000000ULL,
318     0x0000000000000000ULL,
319     0x0000000000000000ULL,
320     0x0000000000000000ULL,
321     0x0000000000000000ULL,
322     0x0000000000000000ULL,
323     0x0000000000000000ULL,
324     0x0000000000000000ULL,
325     0x0000000000000000ULL,
326     0x0000000000000000ULL,
327     0x0000000000000000ULL,
328     0x0000000000000000ULL,
329     0x0000000000000000ULL,
330     0x0000000000000000ULL,
331     0x0000000000000000ULL,
332     0x0000000000000000ULL,
333     0x0000000000000000ULL,
334     0x0000000000000000ULL,
335     0x0000000000000000ULL,
336     0x0000000000000000ULL,
337     0x0000000000000000ULL,
338     0x0000000000000000ULL,
339     0x0000000000000000ULL,
340     0x0000000000000000ULL,
341     0x0000000000000000ULL,
342     0x0000000000000000ULL,
343     0x0000000000000000ULL,
344     0x0000000000000000ULL,
345     0x0000000000000000ULL,
346     0x0000000000000000ULL,
347     0x0000000000000000ULL,
348     0x0000000000000000ULL,
349     0x0000000000000000ULL,
350     0x0000000000000000ULL,
351     0x0000000000000000ULL,
352     0x0000000000000000ULL,
353     0x0000000000000000ULL,
354     0x0000000000000000ULL,
355     0x0000000000000000ULL,
356     0x0000000000000000ULL,
357     0x0000000000000000ULL,
358     0x0000000000000000ULL,
359     0x0000000000000000ULL,
360     0x0000000000000000ULL,
361     0x0000000000000000ULL,
362     0x0000000000000000ULL,
363     0x0000000000000000ULL,
364     0x0000000000000000ULL,
365     0x0000000000000000ULL,
366     0x0000000000000000ULL,
367     0x0000000000000000ULL,
368     0x0000000000000000ULL,
369     0x0000000000000000ULL,
370     0x0000000000000000ULL,
371     0x0000000000000000ULL,
372     0x0000000000000000ULL,
373     0x0000000000000000ULL,
374     0x0000000000000000ULL,
375     0x0000000000000000ULL,
376     0x0000000000000000ULL,
377     0x0000000000000000ULL,
378     0x0000000000000000ULL,
379     0x0000000000000000ULL,
380     0x0000000000000000ULL,
381     0x0000000000000000ULL,
382     0x0000000000000000ULL,
383     0x0000000000000000ULL,
384     0x0000000000000000ULL,
385     0x0000000000000000ULL,
386     0x0000000000000000ULL,
387     0x0000000000000000ULL,
388     0x0000000000000000ULL,
389     0x0000000000000000ULL,
390     0x0000000000000000ULL,
391     0x0000000000000000ULL,
392     0x0000000000000000ULL,
393     0x0000000000000000ULL,
394     0x0000000000000000ULL,
395     0x0000000000000000ULL,
396     0x0000000000000000ULL,
397     0x0000000000000000ULL,
398     0x0000000000000000ULL,
399     0x0000000000000000ULL,
400     0x0000000000000000ULL,
401     0x0000000000000000ULL,
402     0x0000000000000000ULL,
403     0x0000000000000000ULL,
404     0x0000000000000000ULL,
405     0x0000000000000000ULL,
406     0x0000000000000000ULL,
407     0x0000000000000000ULL,
408     0x0000000000000000ULL,
409     0x0000000000000000ULL,
410     0x0000000000000000ULL,
411     0x0000000000000000ULL,
412     0x0000000000000000ULL,
413     0x0000000000000000ULL,
414     0x0000000000000000ULL,
415     0x0000000000000000ULL,
416     0x0000000000000000ULL,
417     0x0000000000000000ULL,
418     0x0000000000000000ULL,
419     0x0000000000000000ULL,
420     0x0000000000000000ULL,
421     0x0000000000000000ULL,
422     0x0000000000000000ULL,
423     0x0000000000000000ULL,
424     0x0000000000000000ULL,
425     0x0000000000000000ULL,
426     0x0000000000000000ULL,
427     0x0000000000000000ULL,
428     0x0000000000000000ULL,
429     0x0000000000000000ULL,
430     0x0000000000000000ULL,
431     0x0000000000000000ULL,
432     0x0000000000000000ULL,
433     0x0000000000000000ULL,
434     0x0000000000000000ULL,
435     0x0000000000000000ULL,
436     0x0000000000000000ULL,
437     0x0000000000000000ULL,
438     0x0000000000000000ULL,
439     0x0000000000000000ULL,
440     0x0000000000000000ULL,
441     0x0000000000000000ULL,
442     0x0000000000000000ULL,
443     0x0000000000000000ULL,
444     0x0000000000000000ULL,
445     0x0000000000000000ULL,
446     0x0000000000000000ULL,
447     0x0000000000000000ULL,
448     0x0000000000000000ULL,
449     0x0000000000000000ULL,
450     0x0000000000000000ULL,
451     0x0000000000000000ULL,
452     0x0000000000000000ULL,
453     0x0000000000000000ULL,
454     0x0000000000000000ULL,
455     0x0000000000000000ULL,
456     0x0000000000000000ULL,
457     0x0000000000000000ULL,
458     0x0000000000000000ULL,
459     0x0000000000000000ULL,
460     0x0000000000000000ULL,
461     0x0000000000000000ULL,
462     0x0000000000000000ULL,
463     0x0000000000000000ULL,
464     0x0000000000000000ULL,
465     0x0000000000000000ULL,
466     0x0000000000000000ULL,
467     0x0000000000000000ULL,
468     0x0000000000000000ULL,
469     0x0000000000000000ULL,
470     0x0000000000000000ULL,
471     0x0000000000000000ULL,
472     0x0000000000000000ULL,
473     0x0000000000000000ULL,
474     0x0000000000000000ULL,
475     0x0000000000000000ULL,
476     0x0000000000000000ULL,
477     0x0000000000000000ULL,
478     0x0000000000000000ULL,
479     0x0000000000000000ULL,
480     0x0000000000000000ULL,
481     0x0000000000000000ULL,
482     0x0000000000000000ULL,
483     0x0000000000000000ULL,
484     0x0000000000000000ULL,
485     0x0000000000000000ULL,
486     0x0000000000000000ULL,
487     0x0000000000000000ULL,
488     0x0000000000000000ULL,
489     0x0000000000000000ULL,
490     0x0000000000000000ULL,
491     0x0000000000000000ULL,
492     0x0000000000000000ULL,
493     0x0000000000000000ULL,
494     0x0000000000000000ULL,
495     0x0000000000000000ULL,
496     0x0000000000000000ULL,
497     0x0000000000000000ULL,
498     0x0000000000000000ULL,
499     0x0000000000000000ULL,
500     0x0000000000000000ULL,
501     0x0000000000000000ULL,
502     0x0000000000000000ULL,
503     0x0000000000000000ULL,
504     0x0000000000000000ULL,
505     0x0000000000000000ULL,
506     0x0000000000000000ULL,
507     0x0000000000000000ULL,
508     0x0000000000000000ULL,
509     0x0000000000000000ULL,
510     0x0000000000000000ULL,
511     0x0000000000000000ULL,
512     0x0000000000000000ULL,
513     0x0000000000000000ULL,
514     0x0000000000000000ULL,
515     0x0000000000000000ULL,
516     0x0000000000000000ULL,
517     0x0000000000000000ULL,
518     0x0000000000000000ULL,
519     0x0000000000000000ULL,
520     0x0000000000000000ULL,
521     0x0000000000000000ULL,
522     0x0000000000000000ULL,
523     0x0000000000000000ULL,
524     0x0000000000000000ULL,
525     0x0000000000000000ULL,
526     0x0000000000000000ULL,
527     0x0000000000000000ULL,
528     0x0000000000000000ULL,
529     0x0000000000000000ULL,
530     0x0000000000000000ULL,
531     0x0000000
```

```

169     0x5a8279996ed9eba1ULL,
170     0x8f1bbcdcca62c1d6ULL,
171     0x012389ab456789abULL};

173 extern void SHA1TransformVIS(uint64_t *, uint32_t *, uint32_t *, uint64_t *);

176 /*
177  * SHA1Update()
178  *
179  * purpose: continues an sha1 digest operation, using the message block
180  *           to update the context.
181  * input:  SHA1_CTX * : the context to update
182  *         void *      : the message block
183  *         size_t      : the length of the message block in bytes
184  * output: void
185  */

187 void
188 SHA1Update(SHA1_CTX *ctx, const void *inptr, size_t input_len)
189 {
190     uint32_t i, buf_index, buf_len;
191     uint64_t X0[40], input64[8];
192     const uint8_t *input = inptr;
193 #ifdef _KERNEL
194     int usevis = 0;
195 #else
196     int usevis = 1;
197 #endif /* _KERNEL */

199     /* check for noop */
200     if (input_len == 0)
201         return;

203     /* compute number of bytes mod 64 */
204     buf_index = (ctx->count[1] >> 3) & 0x3F;

206     /* update number of bits */
207     if ((ctx->count[1] += (input_len << 3)) < (input_len << 3))
208         ctx->count[0]++;

210     ctx->count[0] += (input_len >> 29);

212     buf_len = 64 - buf_index;

214     /* transform as many times as possible */
215     i = 0;
216     if (input_len >= buf_len) {
217 #ifdef _KERNEL
218         kfpu_t *fpu;
219         if (fpu_exists) {
220             uint8_t fpu_a[sizeof(kfpu_t) + GSR_SIZE + VIS_ALIGN];
221             uint32_t len = (input_len + buf_index) & ~0x3f;
222             int svfp_ok;

224             fpu = (kfpu_t *)P2ROUNDUP((uintptr_t)fpu_a, 64);
225             svfp_ok = ((len >= vis_sha1_svfp_threshold) ? 1 : 0);
226             usevis = fpu_exists && sha1_savefp(fpu, svfp_ok);
227         } else {
228             usevis = 0;
229         }
230 #endif /* _KERNEL */

232         /*
233          * general optimization:
234          */

```

```

235     * only do initial bcopy() and SHA1Transform() if
236     * buf_index != 0. if buf_index == 0, we're just
237     * wasting our time doing the bcopy() since there
238     * wasn't any data left over from a previous call to
239     * SHA1Update().
240     */

242     if (buf_index) {
243         bcopy(input, &ctx->buf_un.buf8[buf_index], buf_len);
244         if (usevis) {
245             SHA1TransformVIS(X0,
246                             ctx->buf_un.buf32,
247                             &ctx->state[0], VIS);
248         } else {
249             SHA1_TRANSFORM(ctx, ctx->buf_un.buf8);
250         }
251         i = buf_len;
252     }

254     /*
255     * VIS SHA-1: uses the VIS 1.0 instructions to accelerate
256     * SHA-1 processing. This is achieved by "offloading" the
257     * computation of the message schedule (MS) to the VIS units.
258     * This allows the VIS computation of the message schedule
259     * to be performed in parallel with the standard integer
260     * processing of the remainder of the SHA-1 computation.
261     * performance by up to around 1.37X, compared to an optimized
262     * integer-only implementation.
263     *
264     * The VIS implementation of SHA1Transform has a different API
265     * to the standard integer version:
266     *
267     * void SHA1TransformVIS(
268     *     uint64_t *, // Pointer to MS for ith block
269     *     uint32_t *, // Pointer to ith block of message data
270     *     uint32_t *, // Pointer to SHA state i.e ctx->state
271     *     uint64_t *, // Pointer to various VIS constants
272     * )
273     *
274     * Note: the message data must be 4-byte aligned.
275     *
276     * Function requires VIS 1.0 support.
277     *
278     * Handling is provided to deal with arbitrary byte alignment
279     * of the input data but the performance gains are reduced
280     * for alignments other than 4-bytes.
281     */
282     if (usevis) {
283         if (!IS_P2ALIGNED(&input[i], sizeof(uint32_t))) {
284             /*
285              * Main processing loop - input misaligned
286              */
287             for (; i + 63 < input_len; i += 64) {
288                 bcopy(&input[i], input64, 64);
289                 SHA1TransformVIS(X0,
290                                 (uint32_t *)input64,
291                                 SHA1TransformVIS(X0, (uint32_t *)input64,
292                                                     &ctx->state[0], VIS);
293             }
294         } else {
295             /*
296              * Main processing loop - input 8-byte aligned
297              */
298             for (; i + 63 < input_len; i += 64) {
299                 SHA1TransformVIS(X0,

```

```

300             (uint32_t *)&input[i],
301             &ctx->state[0], VIS);
302         }
303     }
304 }
305 #ifdef _KERNEL
306     sha1_restorefp(fpu);
307 #endif /* _KERNEL */
308 } else {
309     for (; i + 63 < input_len; i += 64) {
310         SHA1_TRANSFORM(ctx, &input[i]);
311     }
312 }
313
314 /*
315  * general optimization:
316  * if i and input_len are the same, return now instead
317  * of calling bcopy(), since the bcopy() in this case
318  * will be an expensive nop.
319  */
320
321 if (input_len == i)
322     return;
323
324 buf_index = 0;
325 }
326
327 /* buffer remaining input */
328 bcopy(&input[i], &ctx->buf_un.buf8[buf_index], input_len - i);
329 }
330 }
331
332 #else /* VIS_SHA1 */
333
334 void
335 SHA1Update(SHA1_CTX *ctx, const void *inptr, size_t input_len)
336 {
337     uint32_t i, buf_index, buf_len;
338     const uint8_t *input = inptr;
339 #if defined(__amd64)
340     uint32_t block_count;
341 #endif /* __amd64 */
342
343     /* check for noop */
344     if (input_len == 0)
345         return;
346
347     /* compute number of bytes mod 64 */
348     buf_index = (ctx->count[1] >> 3) & 0x3F;
349
350     /* update number of bits */
351     if ((ctx->count[1] += (input_len << 3)) < (input_len << 3))
352         ctx->count[0]++;
353
354     ctx->count[0] += (input_len >> 29);
355
356     buf_len = 64 - buf_index;
357
358     /* transform as many times as possible */
359     i = 0;
360     if (input_len >= buf_len) {
361
362         /*
363          * general optimization:
364          * only do initial bcopy() and SHA1Transform() if

```

```

366         * buf_index != 0. if buf_index == 0, we're just
367         * wasting our time doing the bcopy() since there
368         * wasn't any data left over from a previous call to
369         * SHA1Update().
370         */
371
372         if (buf_index) {
373             bcopy(input, &ctx->buf_un.buf8[buf_index], buf_len);
374             SHA1_TRANSFORM(ctx, &ctx->buf_un.buf8);
375             i = buf_len;
376         }
377
378 #if !defined(__amd64)
379         for (; i + 63 < input_len; i += 64)
380             SHA1_TRANSFORM(ctx, &input[i]);
381 #else
382         block_count = (input_len - i) >> 6;
383         if (block_count > 0) {
384             SHA1_TRANSFORM_BLOCKS(ctx, &input[i], block_count);
385             i += block_count << 6;
386         }
387 #endif /* !__amd64 */
388
389     /*
390      * general optimization:
391      * if i and input_len are the same, return now instead
392      * of calling bcopy(), since the bcopy() in this case
393      * will be an expensive nop.
394      */
395
396     if (input_len == i)
397         return;
398
399     buf_index = 0;
400 }
401
402 /* buffer remaining input */
403 bcopy(&input[i], &ctx->buf_un.buf8[buf_index], input_len - i);
404 }
405 }
406
407 #endif /* VIS_SHA1 */
408
409 /*
410  * SHA1Final()
411  * purpose: ends an sha1 digest operation, finalizing the message digest and
412  *          zeroing the context.
413  * input: uchar_t * : A buffer to store the digest.
414  * input: uchar_t * : a buffer to store the digest in
415  *                  : The function actually uses void* because many
416  *                  : callers pass things other than uchar_t here.
417  * SHA1_CTX * : the context to finalize, save, and zero
418  * output: void
419  */
420
421 void
422 SHA1Final(void *digest, SHA1_CTX *ctx)
423 {
424     uint8_t bitcount_be[sizeof (ctx->count)];
425     uint32_t index = (ctx->count[1] >> 3) & 0x3f;
426
427     /* store bit count, big endian */
428     Encode(bitcount_be, ctx->count, sizeof (bitcount_be));
429
430     /* pad out to 56 mod 64 */

```

```

431     SHA1Update(ctx, PADDING, ((index < 56) ? 56 : 120) - index);

433     /* append length (before padding) */
434     SHA1Update(ctx, bitcount_be, sizeof (bitcount_be));

436     /* store state in digest */
437     Encode(digest, ctx->state, sizeof (ctx->state));

439     /* zeroize sensitive information */
440     bzero(ctx, sizeof (*ctx));
441 }

444 #if !defined(__amd64)

446 typedef uint32_t sha1word;

448 /*
449  * sparc optimization:
450  *
451  * on the sparc, we can load big endian 32-bit data easily. note that
452  * special care must be taken to ensure the address is 32-bit aligned.
453  * in the interest of speed, we don't check to make sure, since
454  * careful programming can guarantee this for us.
455  */

457 #if defined(_BIG_ENDIAN)

459 #define LOAD_BIG_32(addr)      (*(uint32_t *) (addr))

461 #else /* !defined(_BIG_ENDIAN) */

463 #if defined(HAVE_BSWAP)

465 #define LOAD_BIG_32(addr) bswap(*(uint32_t *) (addr))

467 #else /* !defined(HAVE_BSWAP) */

469 /* little endian -- will work on big endian, but slowly */
470 #define LOAD_BIG_32(addr)      \
471     (((addr)[0] << 24) | ((addr)[1] << 16) | ((addr)[2] << 8) | (addr)[3])

473 #endif /* !defined(HAVE_BSWAP) */

475 #endif /* !defined(_BIG_ENDIAN) */

477 /*
478  * SHA1Transform()
479  */
480 #if defined(W_ARRAY)
481 #define W(n) w[n]
482 #else /* !defined(W_ARRAY) */
483 #define W(n) w_ ## n
484 #endif /* !defined(W_ARRAY) */

487 #if defined(__sparc)

489 /*
490  * sparc register window optimization:
491  *
492  * 'a', 'b', 'c', 'd', and 'e' are passed into SHA1Transform
493  * explicitly since it increases the number of registers available to
494  * the compiler. under this scheme, these variables can be held in
495  * %i0 - %i4, which leaves more local and out registers available.
496  */

```

```

497  * purpose: sha1 transformation -- updates the digest based on 'block'
498  * input:  uint32_t : bytes 1 - 4 of the digest
499  *          uint32_t : bytes 5 - 8 of the digest
500  *          uint32_t : bytes 9 - 12 of the digest
501  *          uint32_t : bytes 12 - 16 of the digest
502  *          uint32_t : bytes 16 - 20 of the digest
503  *          SHA1_CTX * : the context to update
504  *          uint8_t [64]: the block to use to update the digest
505  * output: void
506  */

508 void
509 SHA1Transform(uint32_t a, uint32_t b, uint32_t c, uint32_t d, uint32_t e,
510              SHA1_CTX *ctx, const uint8_t blk[64])
511 {
512     /*
513      * sparc optimization:
514      *
515      * while it is somewhat counter-intuitive, on sparc, it is
516      * more efficient to place all the constants used in this
517      * function in an array and load the values out of the array
518      * than to manually load the constants. this is because
519      * setting a register to a 32-bit value takes two ops in most
520      * cases: a 'sethi' and an 'or', but loading a 32-bit value
521      * from memory only takes one 'ld' (or 'lduw' on v9). while
522      * this increases memory usage, the compiler can find enough
523      * other things to do while waiting to keep the pipeline does
524      * not stall. additionally, it is likely that many of these
525      * constants are cached so that later accesses do not even go
526      * out to the bus.
527      *
528      * this array is declared 'static' to keep the compiler from
529      * having to bcopy() this array onto the stack frame of
530      * SHA1Transform() each time it is called -- which is
531      * unacceptably expensive.
532      *
533      * the 'const' is to ensure that callers are good citizens and
534      * do not try to munge the array. since these routines are
535      * going to be called from inside multithreaded kernelland,
536      * this is a good safety check. -- 'sha1_consts' will end up in
537      * .rodata.
538      *
539      * unfortunately, loading from an array in this manner hurts
540      * performance under intel. so, there is a macro,
541      * SHA1_CONST(), used in SHA1Transform(), that either expands to
542      * a reference to this array, or to the actual constant,
543      * depending on what platform this code is compiled for.
544      */

546     static const uint32_t sha1_consts[] = {
547         SHA1_CONST_0,  SHA1_CONST_1,  SHA1_CONST_2,  SHA1_CONST_3,
548     };

550     /*
551      * general optimization:
552      *
553      * use individual integers instead of using an array. this is a
554      * win, although the amount it wins by seems to vary quite a bit.
555      */

557     uint32_t      w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7;
558     uint32_t      w_8, w_9, w_10, w_11, w_12, w_13, w_14, w_15;

560     /*
561      * sparc optimization:
562      */

```

```

563  * if 'block' is already aligned on a 4-byte boundary, use
564  * LOAD_BIG_32() directly.  otherwise, bcopy() into a
565  * buffer that *is* aligned on a 4-byte boundary and then do
566  * the LOAD_BIG_32() on that buffer.  benchmarks have shown
567  * that using the bcopy() is better than loading the bytes
568  * individually and doing the endian-swap by hand.
569  *
570  * even though it's quite tempting to assign to do:
571  *
572  * blk = bcopy(ctx->buf_un.buf32, blk, sizeof (ctx->buf_un.buf32));
573  *
574  * and only have one set of LOAD_BIG_32()'s, the compiler
575  * *does not* like that, so please resist the urge.
576  */
577
578  if ((uintptr_t)blk & 0x3) {          /* not 4-byte aligned? */
579      bcopy(blk, ctx->buf_un.buf32, sizeof (ctx->buf_un.buf32));
580      w_15 = LOAD_BIG_32(ctx->buf_un.buf32 + 15);
581      w_14 = LOAD_BIG_32(ctx->buf_un.buf32 + 14);
582      w_13 = LOAD_BIG_32(ctx->buf_un.buf32 + 13);
583      w_12 = LOAD_BIG_32(ctx->buf_un.buf32 + 12);
584      w_11 = LOAD_BIG_32(ctx->buf_un.buf32 + 11);
585      w_10 = LOAD_BIG_32(ctx->buf_un.buf32 + 10);
586      w_9 = LOAD_BIG_32(ctx->buf_un.buf32 + 9);
587      w_8 = LOAD_BIG_32(ctx->buf_un.buf32 + 8);
588      w_7 = LOAD_BIG_32(ctx->buf_un.buf32 + 7);
589      w_6 = LOAD_BIG_32(ctx->buf_un.buf32 + 6);
590      w_5 = LOAD_BIG_32(ctx->buf_un.buf32 + 5);
591      w_4 = LOAD_BIG_32(ctx->buf_un.buf32 + 4);
592      w_3 = LOAD_BIG_32(ctx->buf_un.buf32 + 3);
593      w_2 = LOAD_BIG_32(ctx->buf_un.buf32 + 2);
594      w_1 = LOAD_BIG_32(ctx->buf_un.buf32 + 1);
595      w_0 = LOAD_BIG_32(ctx->buf_un.buf32 + 0);
596  } else {
597      /*LINTED*/
598      w_15 = LOAD_BIG_32(blk + 60);
599      /*LINTED*/
600      w_14 = LOAD_BIG_32(blk + 56);
601      /*LINTED*/
602      w_13 = LOAD_BIG_32(blk + 52);
603      /*LINTED*/
604      w_12 = LOAD_BIG_32(blk + 48);
605      /*LINTED*/
606      w_11 = LOAD_BIG_32(blk + 44);
607      /*LINTED*/
608      w_10 = LOAD_BIG_32(blk + 40);
609      /*LINTED*/
610      w_9 = LOAD_BIG_32(blk + 36);
611      /*LINTED*/
612      w_8 = LOAD_BIG_32(blk + 32);
613      /*LINTED*/
614      w_7 = LOAD_BIG_32(blk + 28);
615      /*LINTED*/
616      w_6 = LOAD_BIG_32(blk + 24);
617      /*LINTED*/
618      w_5 = LOAD_BIG_32(blk + 20);
619      /*LINTED*/
620      w_4 = LOAD_BIG_32(blk + 16);
621      /*LINTED*/
622      w_3 = LOAD_BIG_32(blk + 12);
623      /*LINTED*/
624      w_2 = LOAD_BIG_32(blk + 8);
625      /*LINTED*/
626      w_1 = LOAD_BIG_32(blk + 4);
627      /*LINTED*/
628      w_0 = LOAD_BIG_32(blk + 0);

```

```

629  }
630  #else /* !defined(__sparc) */
631
632  void
633  SHA1Transform(SHA1_CTX *ctx, const uint8_t blk[64])
634  {
635      sha1word a = ctx->state[0];
636      sha1word b = ctx->state[1];
637      sha1word c = ctx->state[2];
638      sha1word d = ctx->state[3];
639      sha1word e = ctx->state[4];
640
641      #if defined(W_ARRAY)
642          sha1word w[16];
643      #else /* !defined(W_ARRAY) */
644          sha1word w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7;
645          sha1word w_8, w_9, w_10, w_11, w_12, w_13, w_14, w_15;
646      #endif /* !defined(W_ARRAY) */
647
648      W(0) = LOAD_BIG_32(blk + 0);
649      W(1) = LOAD_BIG_32(blk + 4);
650      W(2) = LOAD_BIG_32(blk + 8);
651      W(3) = LOAD_BIG_32(blk + 12);
652      W(4) = LOAD_BIG_32(blk + 16);
653      W(5) = LOAD_BIG_32(blk + 20);
654      W(6) = LOAD_BIG_32(blk + 24);
655      W(7) = LOAD_BIG_32(blk + 28);
656      W(8) = LOAD_BIG_32(blk + 32);
657      W(9) = LOAD_BIG_32(blk + 36);
658      W(10) = LOAD_BIG_32(blk + 40);
659      W(11) = LOAD_BIG_32(blk + 44);
660      W(12) = LOAD_BIG_32(blk + 48);
661      W(13) = LOAD_BIG_32(blk + 52);
662      W(14) = LOAD_BIG_32(blk + 56);
663      W(15) = LOAD_BIG_32(blk + 60);
664
665      #endif /* !defined(__sparc) */
666
667      /*
668       * general optimization:
669       *
670       * even though this approach is described in the standard as
671       * being slower algorithmically, it is 30-40% faster than the
672       * "faster" version under SPARC, because this version has more
673       * of the constraints specified at compile-time and uses fewer
674       * variables (and therefore has better register utilization)
675       * than its "speedier" brother.  (i've tried both, trust me)
676       *
677       * for either method given in the spec, there is an "assignment"
678       * phase where the following takes place:
679       *
680       *     tmp = (main_computation);
681       *     e = d; d = c; c = rotate_left(b, 30); b = a; a = tmp;
682       *
683       * we can make the algorithm go faster by not doing this work,
684       * but just pretending that 'd' is now 'e', etc. this works
685       * really well and obviates the need for a temporary variable.
686       * however, we still explicitly perform the rotate action,
687       * however, we still explicitly perform the rotate action,
688       * since it is cheaper on SPARC to do it once than to have to
689       * do it over and over again.
690       */
691
692      /* round 1 */
693      e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(0) + SHA1_CONST(0); /* 0 */
694      b = ROTATE_LEFT(b, 30);

```

```

695     d = ROTATE_LEFT(e, 5) + F(a, b, c) + d + W(1) + SHA1_CONST(0); /* 1 */
696     a = ROTATE_LEFT(a, 30);

698     c = ROTATE_LEFT(d, 5) + F(e, a, b) + c + W(2) + SHA1_CONST(0); /* 2 */
699     e = ROTATE_LEFT(e, 30);

701     b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(3) + SHA1_CONST(0); /* 3 */
702     d = ROTATE_LEFT(d, 30);

704     a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(4) + SHA1_CONST(0); /* 4 */
705     c = ROTATE_LEFT(c, 30);

707     e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(5) + SHA1_CONST(0); /* 5 */
708     b = ROTATE_LEFT(b, 30);

710     d = ROTATE_LEFT(e, 5) + F(a, b, c) + d + W(6) + SHA1_CONST(0); /* 6 */
711     a = ROTATE_LEFT(a, 30);

713     c = ROTATE_LEFT(d, 5) + F(e, a, b) + c + W(7) + SHA1_CONST(0); /* 7 */
714     e = ROTATE_LEFT(e, 30);

716     b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(8) + SHA1_CONST(0); /* 8 */
717     d = ROTATE_LEFT(d, 30);

719     a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(9) + SHA1_CONST(0); /* 9 */
720     c = ROTATE_LEFT(c, 30);

722     e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(10) + SHA1_CONST(0); /* 10 */
723     b = ROTATE_LEFT(b, 30);

725     d = ROTATE_LEFT(e, 5) + F(a, b, c) + d + W(11) + SHA1_CONST(0); /* 11 */
726     a = ROTATE_LEFT(a, 30);

728     c = ROTATE_LEFT(d, 5) + F(e, a, b) + c + W(12) + SHA1_CONST(0); /* 12 */
729     e = ROTATE_LEFT(e, 30);

731     b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(13) + SHA1_CONST(0); /* 13 */
732     d = ROTATE_LEFT(d, 30);

734     a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(14) + SHA1_CONST(0); /* 14 */
735     c = ROTATE_LEFT(c, 30);

737     e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(15) + SHA1_CONST(0); /* 15 */
738     b = ROTATE_LEFT(b, 30);

740     W(0) = ROTATE_LEFT((W(13) ^ W(8) ^ W(2) ^ W(0)), 1); /* 16 */
741     d = ROTATE_LEFT(e, 5) + F(a, b, c) + d + W(0) + SHA1_CONST(0);
742     a = ROTATE_LEFT(a, 30);

744     W(1) = ROTATE_LEFT((W(14) ^ W(9) ^ W(3) ^ W(1)), 1); /* 17 */
745     c = ROTATE_LEFT(d, 5) + F(e, a, b) + c + W(1) + SHA1_CONST(0);
746     e = ROTATE_LEFT(e, 30);

748     W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1); /* 18 */
749     b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(2) + SHA1_CONST(0);
750     d = ROTATE_LEFT(d, 30);

752     W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1); /* 19 */
753     a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(3) + SHA1_CONST(0);
754     c = ROTATE_LEFT(c, 30);

756     /* round 2 */
757     W(4) = ROTATE_LEFT((W(1) ^ W(12) ^ W(6) ^ W(4)), 1); /* 20 */
758     e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(4) + SHA1_CONST(1);
759     b = ROTATE_LEFT(b, 30);

```

```

761     W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1); /* 21 */
762     d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(5) + SHA1_CONST(1);
763     a = ROTATE_LEFT(a, 30);

765     W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1); /* 22 */
766     c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(6) + SHA1_CONST(1);
767     e = ROTATE_LEFT(e, 30);

769     W(7) = ROTATE_LEFT((W(4) ^ W(15) ^ W(9) ^ W(7)), 1); /* 23 */
770     b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(7) + SHA1_CONST(1);
771     d = ROTATE_LEFT(d, 30);

773     W(8) = ROTATE_LEFT((W(5) ^ W(0) ^ W(10) ^ W(8)), 1); /* 24 */
774     a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(8) + SHA1_CONST(1);
775     c = ROTATE_LEFT(c, 30);

777     W(9) = ROTATE_LEFT((W(6) ^ W(1) ^ W(11) ^ W(9)), 1); /* 25 */
778     e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(9) + SHA1_CONST(1);
779     b = ROTATE_LEFT(b, 30);

781     W(10) = ROTATE_LEFT((W(7) ^ W(2) ^ W(12) ^ W(10)), 1); /* 26 */
782     d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(10) + SHA1_CONST(1);
783     a = ROTATE_LEFT(a, 30);

785     W(11) = ROTATE_LEFT((W(8) ^ W(3) ^ W(13) ^ W(11)), 1); /* 27 */
786     c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(11) + SHA1_CONST(1);
787     e = ROTATE_LEFT(e, 30);

789     W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1); /* 28 */
790     b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(12) + SHA1_CONST(1);
791     d = ROTATE_LEFT(d, 30);

793     W(13) = ROTATE_LEFT((W(10) ^ W(5) ^ W(15) ^ W(13)), 1); /* 29 */
794     a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(13) + SHA1_CONST(1);
795     c = ROTATE_LEFT(c, 30);

797     W(14) = ROTATE_LEFT((W(11) ^ W(6) ^ W(0) ^ W(14)), 1); /* 30 */
798     e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(14) + SHA1_CONST(1);
799     b = ROTATE_LEFT(b, 30);

801     W(15) = ROTATE_LEFT((W(12) ^ W(7) ^ W(1) ^ W(15)), 1); /* 31 */
802     d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(15) + SHA1_CONST(1);
803     a = ROTATE_LEFT(a, 30);

805     W(0) = ROTATE_LEFT((W(13) ^ W(8) ^ W(2) ^ W(0)), 1); /* 32 */
806     c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(0) + SHA1_CONST(1);
807     e = ROTATE_LEFT(e, 30);

809     W(1) = ROTATE_LEFT((W(14) ^ W(9) ^ W(3) ^ W(1)), 1); /* 33 */
810     b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(1) + SHA1_CONST(1);
811     d = ROTATE_LEFT(d, 30);

813     W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1); /* 34 */
814     a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(2) + SHA1_CONST(1);
815     c = ROTATE_LEFT(c, 30);

817     W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1); /* 35 */
818     e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(3) + SHA1_CONST(1);
819     b = ROTATE_LEFT(b, 30);

821     W(4) = ROTATE_LEFT((W(1) ^ W(12) ^ W(6) ^ W(4)), 1); /* 36 */
822     d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(4) + SHA1_CONST(1);
823     a = ROTATE_LEFT(a, 30);

825     W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1); /* 37 */

```

```

826     c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + w(5) + SHA1_CONST(1);
827     e = ROTATE_LEFT(e, 30);

829     w(6) = ROTATE_LEFT((w(3) ^ w(14) ^ w(8) ^ w(6)), 1); /* 38 */
830     b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + w(6) + SHA1_CONST(1);
831     d = ROTATE_LEFT(d, 30);

833     w(7) = ROTATE_LEFT((w(4) ^ w(15) ^ w(9) ^ w(7)), 1); /* 39 */
834     a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + w(7) + SHA1_CONST(1);
835     c = ROTATE_LEFT(c, 30);

837     /* round 3 */
838     w(8) = ROTATE_LEFT((w(5) ^ w(0) ^ w(10) ^ w(8)), 1); /* 40 */
839     e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + w(8) + SHA1_CONST(2);
840     b = ROTATE_LEFT(b, 30);

842     w(9) = ROTATE_LEFT((w(6) ^ w(1) ^ w(11) ^ w(9)), 1); /* 41 */
843     d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + w(9) + SHA1_CONST(2);
844     a = ROTATE_LEFT(a, 30);

846     w(10) = ROTATE_LEFT((w(7) ^ w(2) ^ w(12) ^ w(10)), 1); /* 42 */
847     c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + w(10) + SHA1_CONST(2);
848     e = ROTATE_LEFT(e, 30);

850     w(11) = ROTATE_LEFT((w(8) ^ w(3) ^ w(13) ^ w(11)), 1); /* 43 */
851     b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + w(11) + SHA1_CONST(2);
852     d = ROTATE_LEFT(d, 30);

854     w(12) = ROTATE_LEFT((w(9) ^ w(4) ^ w(14) ^ w(12)), 1); /* 44 */
855     a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + w(12) + SHA1_CONST(2);
856     c = ROTATE_LEFT(c, 30);

858     w(13) = ROTATE_LEFT((w(10) ^ w(5) ^ w(15) ^ w(13)), 1); /* 45 */
859     e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + w(13) + SHA1_CONST(2);
860     b = ROTATE_LEFT(b, 30);

862     w(14) = ROTATE_LEFT((w(11) ^ w(6) ^ w(0) ^ w(14)), 1); /* 46 */
863     d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + w(14) + SHA1_CONST(2);
864     a = ROTATE_LEFT(a, 30);

866     w(15) = ROTATE_LEFT((w(12) ^ w(7) ^ w(1) ^ w(15)), 1); /* 47 */
867     c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + w(15) + SHA1_CONST(2);
868     e = ROTATE_LEFT(e, 30);

870     w(0) = ROTATE_LEFT((w(13) ^ w(8) ^ w(2) ^ w(0)), 1); /* 48 */
871     b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + w(0) + SHA1_CONST(2);
872     d = ROTATE_LEFT(d, 30);

874     w(1) = ROTATE_LEFT((w(14) ^ w(9) ^ w(3) ^ w(1)), 1); /* 49 */
875     a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + w(1) + SHA1_CONST(2);
876     c = ROTATE_LEFT(c, 30);

878     w(2) = ROTATE_LEFT((w(15) ^ w(10) ^ w(4) ^ w(2)), 1); /* 50 */
879     e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + w(2) + SHA1_CONST(2);
880     b = ROTATE_LEFT(b, 30);

882     w(3) = ROTATE_LEFT((w(0) ^ w(11) ^ w(5) ^ w(3)), 1); /* 51 */
883     d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + w(3) + SHA1_CONST(2);
884     a = ROTATE_LEFT(a, 30);

886     w(4) = ROTATE_LEFT((w(1) ^ w(12) ^ w(6) ^ w(4)), 1); /* 52 */
887     c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + w(4) + SHA1_CONST(2);
888     e = ROTATE_LEFT(e, 30);

890     w(5) = ROTATE_LEFT((w(2) ^ w(13) ^ w(7) ^ w(5)), 1); /* 53 */
891     b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + w(5) + SHA1_CONST(2);

```

```

892     d = ROTATE_LEFT(d, 30);

894     w(6) = ROTATE_LEFT((w(3) ^ w(14) ^ w(8) ^ w(6)), 1); /* 54 */
895     a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + w(6) + SHA1_CONST(2);
896     c = ROTATE_LEFT(c, 30);

898     w(7) = ROTATE_LEFT((w(4) ^ w(15) ^ w(9) ^ w(7)), 1); /* 55 */
899     e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + w(7) + SHA1_CONST(2);
900     b = ROTATE_LEFT(b, 30);

902     w(8) = ROTATE_LEFT((w(5) ^ w(0) ^ w(10) ^ w(8)), 1); /* 56 */
903     d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + w(8) + SHA1_CONST(2);
904     a = ROTATE_LEFT(a, 30);

906     w(9) = ROTATE_LEFT((w(6) ^ w(1) ^ w(11) ^ w(9)), 1); /* 57 */
907     c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + w(9) + SHA1_CONST(2);
908     e = ROTATE_LEFT(e, 30);

910     w(10) = ROTATE_LEFT((w(7) ^ w(2) ^ w(12) ^ w(10)), 1); /* 58 */
911     b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + w(10) + SHA1_CONST(2);
912     d = ROTATE_LEFT(d, 30);

914     w(11) = ROTATE_LEFT((w(8) ^ w(3) ^ w(13) ^ w(11)), 1); /* 59 */
915     a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + w(11) + SHA1_CONST(2);
916     c = ROTATE_LEFT(c, 30);

918     /* round 4 */
919     w(12) = ROTATE_LEFT((w(9) ^ w(4) ^ w(14) ^ w(12)), 1); /* 60 */
920     e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + w(12) + SHA1_CONST(3);
921     b = ROTATE_LEFT(b, 30);

923     w(13) = ROTATE_LEFT((w(10) ^ w(5) ^ w(15) ^ w(13)), 1); /* 61 */
924     d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + w(13) + SHA1_CONST(3);
925     a = ROTATE_LEFT(a, 30);

927     w(14) = ROTATE_LEFT((w(11) ^ w(6) ^ w(0) ^ w(14)), 1); /* 62 */
928     c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + w(14) + SHA1_CONST(3);
929     e = ROTATE_LEFT(e, 30);

931     w(15) = ROTATE_LEFT((w(12) ^ w(7) ^ w(1) ^ w(15)), 1); /* 63 */
932     b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + w(15) + SHA1_CONST(3);
933     d = ROTATE_LEFT(d, 30);

935     w(0) = ROTATE_LEFT((w(13) ^ w(8) ^ w(2) ^ w(0)), 1); /* 64 */
936     a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + w(0) + SHA1_CONST(3);
937     c = ROTATE_LEFT(c, 30);

939     w(1) = ROTATE_LEFT((w(14) ^ w(9) ^ w(3) ^ w(1)), 1); /* 65 */
940     e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + w(1) + SHA1_CONST(3);
941     b = ROTATE_LEFT(b, 30);

943     w(2) = ROTATE_LEFT((w(15) ^ w(10) ^ w(4) ^ w(2)), 1); /* 66 */
944     d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + w(2) + SHA1_CONST(3);
945     a = ROTATE_LEFT(a, 30);

947     w(3) = ROTATE_LEFT((w(0) ^ w(11) ^ w(5) ^ w(3)), 1); /* 67 */
948     c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + w(3) + SHA1_CONST(3);
949     e = ROTATE_LEFT(e, 30);

951     w(4) = ROTATE_LEFT((w(1) ^ w(12) ^ w(6) ^ w(4)), 1); /* 68 */
952     b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + w(4) + SHA1_CONST(3);
953     d = ROTATE_LEFT(d, 30);

955     w(5) = ROTATE_LEFT((w(2) ^ w(13) ^ w(7) ^ w(5)), 1); /* 69 */
956     a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + w(5) + SHA1_CONST(3);
957     c = ROTATE_LEFT(c, 30);

```

```

959     W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1); /* 70 */
960     e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(6) + SHA1_CONST(3);
961     b = ROTATE_LEFT(b, 30);

963     W(7) = ROTATE_LEFT((W(4) ^ W(15) ^ W(9) ^ W(7)), 1); /* 71 */
964     d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(7) + SHA1_CONST(3);
965     a = ROTATE_LEFT(a, 30);

967     W(8) = ROTATE_LEFT((W(5) ^ W(0) ^ W(10) ^ W(8)), 1); /* 72 */
968     c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(8) + SHA1_CONST(3);
969     e = ROTATE_LEFT(e, 30);

971     W(9) = ROTATE_LEFT((W(6) ^ W(1) ^ W(11) ^ W(9)), 1); /* 73 */
972     b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(9) + SHA1_CONST(3);
973     d = ROTATE_LEFT(d, 30);

975     W(10) = ROTATE_LEFT((W(7) ^ W(2) ^ W(12) ^ W(10)), 1); /* 74 */
976     a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(10) + SHA1_CONST(3);
977     c = ROTATE_LEFT(c, 30);

979     W(11) = ROTATE_LEFT((W(8) ^ W(3) ^ W(13) ^ W(11)), 1); /* 75 */
980     e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(11) + SHA1_CONST(3);
981     b = ROTATE_LEFT(b, 30);

983     W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1); /* 76 */
984     d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(12) + SHA1_CONST(3);
985     a = ROTATE_LEFT(a, 30);

987     W(13) = ROTATE_LEFT((W(10) ^ W(5) ^ W(15) ^ W(13)), 1); /* 77 */
988     c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(13) + SHA1_CONST(3);
989     e = ROTATE_LEFT(e, 30);

991     W(14) = ROTATE_LEFT((W(11) ^ W(6) ^ W(0) ^ W(14)), 1); /* 78 */
992     b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(14) + SHA1_CONST(3);
993     d = ROTATE_LEFT(d, 30);

995     W(15) = ROTATE_LEFT((W(12) ^ W(7) ^ W(1) ^ W(15)), 1); /* 79 */

997     ctx->state[0] += ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(15) +
998     SHA1_CONST(3);
999     ctx->state[1] += b;
1000    ctx->state[2] += ROTATE_LEFT(c, 30);
1001    ctx->state[3] += d;
1002    ctx->state[4] += e;

1004    /* zeroize sensitive information */
1005    W(0) = W(1) = W(2) = W(3) = W(4) = W(5) = W(6) = W(7) = W(8) = 0;
1006    W(9) = W(10) = W(11) = W(12) = W(13) = W(14) = W(15) = 0;
1007 }
1008 #endif /* !__amd64 */

1011 /*
1012  * Encode()
1013  *
1014  * purpose: to convert a list of numbers from little endian to big endian
1015  * input: uint8_t * : place to store the converted big endian numbers
1016  *        uint32_t * : place to get numbers to convert from
1017  *        size_t : the length of the input in bytes
1018  * output: void
1019  */

1021 static void
1022 Encode(uint8_t *_RESTRICT_KYWD output, const uint32_t *_RESTRICT_KYWD input,
1023        size_t len)

```

```

1024 {
1025     size_t i, j;

1027     #if defined(__sparc)
1028     if (IS_P2ALIGNED(output, sizeof (uint32_t))) {
1029         for (i = 0, j = 0; j < len; i++, j += 4) {
1030             /* LINTED: pointer alignment */
1031             *((uint32_t *) (output + j)) = input[i];
1032         }
1033     } else {
1034         #endif /* little endian -- will work on big endian, but slowly */
1035         for (i = 0, j = 0; j < len; i++, j += 4) {
1036             output[j] = (input[i] >> 24) & 0xff;
1037             output[j + 1] = (input[i] >> 16) & 0xff;
1038             output[j + 2] = (input[i] >> 8) & 0xff;
1039             output[j + 3] = input[i] & 0xff;
1040         }
1041     } #if defined(__sparc)
1042     #endif
1043 #endif
1044 }
_____unchanged_portion_omitted_____

```

new/usr/src/lib/libmd/Makefile.com

1

```
*****
3380 Mon Mar  3 13:15:02 2008
new/usr/src/lib/libmd/Makefile.com
6662791 Need a SHA1 implementation optimized for 64-bit x86
*****

1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
23 # Use is subject to license terms.
24 #
25 # ident "@(#)Makefile.com      1.4      08/03/02 SMI"
26 # ident "@(#)Makefile.com      1.3      08/01/02 SMI"

28 # $LIBRARY is set in lower makefiles so we can have platform and
29 # processor optimised versions of this library via libmd_psr and libmd_hwcapN

31 #LIBRARY= libmd.a
32 VERS= .1

34 OBJECTS= md4.o md5.o $(MD5_PSR_OBJECTS) sha1.o $(SHA1_PSR_OBJECTS) sha2.o
34 OBJECTS= md4.o md5.o $(MD5_PSR_OBJECTS) sha1.o sha2.o

36 # Use $(SRC) to include makefiles rather than ../../ because the
37 # platform subdirs are one level deeper so it would be ../../ for them
38 include $(SRC)/lib/Makefile.lib
39 include $(SRC)/lib/Makefile.rootfs

41 LIBS =          $(DYNLIB) $(LINTLIB)
42 SRCS = \
43     $(COMDIR)/md4/md4.c \
44     $(COMDIR)/md5/md5.c \
45     $(COMDIR)/sha1/sha1.c \
46     $(COMDIR)/sha2/sha2.c

48 COMDIR= $(SRC)/common/crypto

50 $(LINTLIB) :=      SRCS = $(SRCDIR)/$(LINTSRC)
51 LDLIBS +=         -lc

53 SRCDIR =          ../common
54 COMDIR =          $(SRC)/common/crypto

56 CFLAGS += $(CCVERBOSE) $(C_BIGPICFLAGS)
57 CFLAGS64 += $(C_BIGPICFLAGS)
58 CPPFLAGS += -I$(SRCDIR)
```

new/usr/src/lib/libmd/Makefile.com

2

```
60 # The md5 and sha1 code is very careful about data alignment
61 # but lint doesn't know that, so just shut lint up.
62 LINTFLAGS += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
63 LINTFLAGS64 += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED

66 ROOTLINT= $(LINTSRC:%=$(ROOTLIBDIR)/%)

68 .KEEP_STATE:

70 all: $(LIBS) fnamecheck

72 lint: lintcheck

74 pics/%.o: $(COMDIR)/md4/%.c
75     $(COMPILE.c) -I$(COMDIR)/md4 -o $$@ $<
76     $(POST_PROCESS_O)

78 pics/%.o: $(COMDIR)/md5/%.c
79     $(COMPILE.c) -I$(COMDIR)/md5 $(INLINES) -o $$@ $<
80     $(POST_PROCESS_O)

82 pics/%.o: $(COMDIR)/sha1/%.c
83     $(COMPILE.c) -I$(COMDIR)/sha1 -o $$@ $<
84     $(POST_PROCESS_O)

86 pics/%.o: $(COMDIR)/sha1/sparc/$(PLATFORM)/sha1_asm.s
87     $(COMPILE.s) -P -DPIC -D_ASM -o $$@ $<
88     $(POST_PROCESS_O)

90 pics/%.o: $(COMDIR)/sha2/%.c
91     $(COMPILE.c) -I$(COMDIR)/sha2 -o $$@ $<
92     $(POST_PROCESS_O)

94 #
95 # Used when building links in /platform/$(PLATFORM)/lib for libmd_psr.so.1
96 #

98 LIBMD_PSR_DIRS = $(LINKED_PLATFORMS:%=$(ROOT_PLAT_DIR)/%/lib)
99 LIBMD_PSR_LINKS = $(LINKED_PLATFORMS:%=$(ROOT_PLAT_DIR)/%/lib/$(MODULE))

101 LIBMD_PSR64_DIRS = $(LINKED_PLATFORMS:%=$(ROOT_PLAT_DIR)/%/lib/$(MACH64))
102 LIBMD_PSR64_LINKS = $(LINKED_PLATFORMS:%=$(ROOT_PLAT_DIR)/%/lib/$(MACH64)/$(MODU

104 INS.slink6 = $(RM) -r $$@; $(SYMLINK) ../../$(PLATFORM)/lib/$(MODULE) $$@ $(CHOWNL

106 INS.slink64 = $(RM) -r $$@; $(SYMLINK) ../../$(PLATFORM)/lib/$(MACH64)/$(MODUL

108 $(LIBMD_PSR_DIRS):
109     -$(INS.dir.root.bin)

111 $(LIBMD_PSR_LINKS): $(LIBMD_PSR_DIRS)
112     -$(INS.slink6)

114 $(LIBMD_PSR64_DIRS):
115     -$(INS.dir.root.bin)

117 $(LIBMD_PSR64_LINKS): $(LIBMD_PSR64_DIRS)
118     -$(INS.slink64)

120 include $(SRC)/lib/Makefile.targ
```

new/usr/src/lib/libmd/amd64/Makefile

1

```
*****
1563 Mon Mar  3 13:15:04 2008
new/usr/src/lib/libmd/amd64/Makefile
6662791 Need a SHA1 implementation optimized for 64-bit x86
*****
```

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # ident "@(#)Makefile 1.3 08/03/02 SMI"
26 # ident "@(#)Makefile 1.2 08/01/02 SMI"
27 #
```

28 LIBRARY= libmd.a

30 MD5_PSR_OBJECTS = md5_amd64.o

31 SHA1_PSR_OBJECTS = sha1-x86_64.o

33 include ../Makefile.com

34 include \$(SRC)/lib/Makefile.lib.64

36 CLEANFILES += md5_amd64.s sha1-x86_64.s

35 CLEANFILES += md5_amd64.s

38 # This prevents <sys/asm_linkage.h> from including C source:

39 AS_CPPFLAGS += -D_ASM

41 install: all \$(ROOTLIBS64) \$(ROOTLINKS64) \$(ROOTLINT64)

43 pics/md5_amd64.o: md5_amd64.s

44 \$(COMPILE.s) -o \$@ \${@F:.o=.s}

45 \$(POST_PROCESS_O)

47 pics/sha1-x86_64.o: sha1-x86_64.s

48 \$(COMPILE.s) -o \$@ \${@F:.o=.s}

49 \$(POST_PROCESS_O)

51 md5_amd64.s: \$(COMDIR)/md5/amd64/md5_amd64.pl

52 \$(PERL) \$? \$@

54 sha1-x86_64.s: \$(COMDIR)/sha1/amd64/sha1-x86_64.pl

55 \$(PERL) \$? \$@

45 pics/md5_amd64.o: md5_amd64.s

46 \$(COMPILE.s) -o \$@ md5_amd64.s

47 \$(POST_PROCESS_O)

new/usr/src/uts/intel/sha1/Makefile

1

```
*****
2655 Mon Mar  3 13:15:06 2008
new/usr/src/uts/intel/sha1/Makefile
6662791 Need a SHA1 implementation optimized for 64-bit x86
*****

1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # uts/intel/sha1/Makefile
23 #
24 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24 # Copyright 2006 Sun Microsystems, Inc. All rights reserved.
25 # Use is subject to license terms.
26 #
27 # ident "@(#)Makefile 1.7 08/03/02 SMI"
27 #ident "@(#)Makefile 1.6 06/11/02 SMI"
28 #
29 # This makefile drives the production of the sha1 crypto kernel module.
30 #
31 # intel architecture dependent
32 #
33 #
34 #
35 # Path to the base of the uts directory tree (usually /usr/src/uts).
36 #
37 UTSBASE = ../../
38 COMDIR = $(COMMONBASE)/crypto/sha1

40 #
41 # Define the module and object file sets.
42 #
43 MODULE = sha1
44 LINTS = $(SHA1_OBJS:.o=$(LINTS_DIR)/%.ln)
45 SHA1_OBJS_32 =
46 SHA1_OBJS_64 = sha1-x86_64.o
47 SHA1_OBJS += $(SHA1_OBJS_$(CLASS))
48 OBJECTS = $(SHA1_OBJS:%=$(OBJDIR)/%)
44 LINTS = $(SHA1_OBJS:.o=$(LINTS_DIR)/%.ln)
49 ROOTMODULE = $(ROOT_CRYPTO_DIR)/$(MODULE)
50 ROOTLINK = $(ROOT_MISC_DIR)/$(MODULE)

52 #
53 # Include common rules.
54 #
55 include $(UTSBASE)/intel/Makefile.intel

57 #
58 # Override defaults
```

new/usr/src/uts/intel/sha1/Makefile

2

```
59 #
60 CLEANFILES += sha1-x86_64.s

62 #
63 # For now, disable these lint checks; maintainers should endeavor
64 # to investigate and remove these for maximum lint coverage.
65 # Please do not carry these forward to new Makefiles.
66 #
67 LINTTAGS += -erroff=E_BAD_PTR_CAST_ALIGN
68 LINTTAGS += -erroff=E_PTRDIFF_OVERFLOW

71 #
72 # Define targets
73 #
74 ALL_TARGET = $(BINARY)
75 LINT_TARGET = $(MODULE).lint
76 INSTALL_TARGET = $(BINARY) $(ROOTMODULE) $(ROOTLINK)

78 #
79 # Default build targets.
80 #
81 .KEEP_STATE:

83 def: $(DEF_DEPS)

85 all: $(ALL_DEPS)

87 clean: $(CLEAN_DEPS)

89 clobber: $(CLOBBER_DEPS)

91 lint: $(LINT_DEPS)

93 modlintlib: $(MODLINTLIB_DEPS)

95 clean.lint: $(CLEAN_LINT_DEPS)

97 install: $(INSTALL_DEPS)

99 $(ROOTLINK): $(ROOT_MISC_DIR) $(ROOTMODULE)
100 -$(RM) $@; ln $(ROOTMODULE) $@

102 #
103 # Include common targets.
104 #
105 include $(UTSBASE)/intel/Makefile.targ

107 $(OBJDIR)/sha1-x86_64.o: sha1-x86_64.s
108 $(COMPILE.s) -o $@ ${@F:.o=.s}
109 $(POST_PROCESS_O)
110
111 $(OBJDIR)/sha1-x86_64.ln: sha1-x86_64.s
112 @$(LHEAD) $(LINT.c) ${@F:.ln=.s} $(LTAIL))

114 sha1-x86_64.s: $(COMDIR)/amd64/sha1-x86_64.pl
115 $(PERL) $? $@
```