```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Copyright (c) 2012, Intel Corporation
;
; All rights reserved.
;
; Redistribution and use in source and binary forms, with or without
; modification, are permitted provided that the following conditions are
; met:
;
; * Redistributions of source code must retain the above copyright
;   notice, this list of conditions and the following disclaimer.
;
; * Redistributions in binary form must reproduce the above copyright
;   notice, this list of conditions and the following disclaimer in the
;   documentation and/or other materials provided with the
;   distribution.
;
; * Neither the name of the Intel Corporation nor the names of its
;   contributors may be used to endorse or promote products derived from
;   this software without specific prior written permission.
;
;
; THIS SOFTWARE IS PROVIDED BY INTEL CORPORATION "AS IS" AND ANY
; EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
; IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
; PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTEL CORPORATION OR
; CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
; EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
; PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
; PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
; LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
; NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
; SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Example YASM command lines:
; Windows:  yasm -Xvc -f x64 -rnasm -pnasm -o sha256_sse4.obj -g cv8 sha256_sse4.asm
; Linux:    yasm -f x64 -f elf64 -X gnu -g dwarf2 -D LINUX -o sha256_sse4.o sha256_sse4.asm
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This code is described in an Intel White-Paper:
; "Fast SHA-256 Implementations on Intel Architecture Processors"
;
; To find it, surf to http://www.intel.com/p/en_US/embedded
; and search for that title.
; The paper is expected to be released roughly at the end of April, 2012
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This code schedules 1 blocks at a time, with 4 lanes per block
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Modified by kerukuro for use in cppcrypto.

; Modified By Mounir IDRASSI for use in VeraCrypt

%define MOVDQ movdqu ;; assume buffers not aligned

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Define Macros

; addm [mem], reg
; Add reg to mem using reg-mem add and store
%macro addm 2
        add     %2, %1
```

```
        mov     %1, %2
%endm

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

; COPY_XMM_AND_BSWAP xmm, [mem], byte_flip_mask
; Load xmm with mem and byte swap each dword
%macro COPY_XMM_AND_BSWAP 3
        MOVDQ %1, %2
        pshufb %1, %3
%endmacro

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

%define X0 xmm4
%define X1 xmm5
%define X2 xmm6
%define X3 xmm7

%define XTMP0 xmm0
%define XTMP1 xmm1
%define XTMP2 xmm2
%define XTMP3 xmm3
%define XTMP4 xmm8
%define XFER  xmm9

%define SHUF_00BA       xmm10 ; shuffle xBxA -> 00BA
%define SHUF_DC00       xmm11 ; shuffle xDxC -> DC00
%define BYTE_FLIP_MASK  xmm12

%ifndef WINABI
%define NUM_BLKS rdx    ; 3rd arg
%define CTX     rsi     ; 2nd arg
%define INP     rdi     ; 1st arg

%define SRND    rdi     ; clobbers INP
%define c       ecx
%define d       r8d
%define e       edx
%else
%define NUM_BLKS r8     ; 3rd arg
%define CTX     rdx     ; 2nd arg
%define INP     rcx     ; 1st arg

%define SRND    rcx     ; clobbers INP
%define c       edi
%define d       esi
%define e       r8d

%endif
%define TBL     rbp
%define a eax
%define b ebx

%define f r9d
%define g r10d
%define h r11d

%define y0 r13d
%define y1 r14d
%define y2 r15d



_INP_END_SIZE   equ 8
```

```
_INP_SIZE       equ 8
_XFER_SIZE      equ 8
%ifndef WINABI
_XMM_SAVE_SIZE  equ 0
%else
_XMM_SAVE_SIZE  equ 7*16
%endif
; STACK_SIZE plus pushes must be an odd multiple of 8
_ALIGN_SIZE     equ 8

_INP_END        equ 0
_INP            equ _INP_END  + _INP_END_SIZE
_XFER           equ _INP      + _INP_SIZE
_XMM_SAVE       equ _XFER     + _XFER_SIZE + _ALIGN_SIZE
STACK_SIZE      equ _XMM_SAVE + _XMM_SAVE_SIZE

; rotate_Xs
; Rotate values of symbols X0...X3
%macro rotate_Xs 0
%xdefine X_ X0
%xdefine X0 X1
%xdefine X1 X2
%xdefine X2 X3
%xdefine X3 X_
%endm

; ROTATE_ARGS
; Rotate values of symbols a...h
%macro ROTATE_ARGS 0
%xdefine TMP_ h
%xdefine h g
%xdefine g f
%xdefine f e
%xdefine e d
%xdefine d c
%xdefine c b
%xdefine b a
%xdefine a TMP_
%endm

%macro FOUR_ROUNDS_AND_SCHED 0
                ;; compute s0 four at a time and s1 two at a time
                ;; compute W[-16] + W[-7] 4 at a time
                movdqa  XTMP0, X3
        mov     y0, e           ; y0 = e
        ror     y0, (25-11)     ; y0 = e >> (25-11)
        mov     y1, a           ; y1 = a
                palignr XTMP0, X2, 4    ; XTMP0 = W[-7]
        ror     y1, (22-13)     ; y1 = a >> (22-13)
        xor     y0, e           ; y0 = e ^ (e >> (25-11))
        mov     y2, f           ; y2 = f
        ror     y0, (11-6)      ; y0 = (e >> (11-6)) ^ (e >> (25-6))
                movdqa  XTMP1, X1
        xor     y1, a           ; y1 = a ^ (a >> (22-13)
        xor     y2, g           ; y2 = f^g
                paddd   XTMP0, X0       ; XTMP0 = W[-7] + W[-16]
        xor     y0, e           ; y0 = e ^ (e >> (11-6)) ^ (e >> (25-6))
        and     y2, e           ; y2 = (f^g)&e
        ror     y1, (13-2)      ; y1 = (a >> (13-2)) ^ (a >> (22-2))
                ;; compute s0
                palignr XTMP1, X0, 4    ; XTMP1 = W[-15]
        xor     y1, a           ; y1 = a ^ (a >> (13-2)) ^ (a >> (22-2))
        ror     y0, 6           ; y0 = S1 = (e>>6) & (e>>11) ^ (e>>25)
        xor     y2, g           ; y2 = CH = ((f^g)&e)^g
                movdqa  XTMP2, XTMP1    ; XTMP2 = W[-15]
```

```
        ror     y1, 2           ; y1 = S0 = (a>>2) ^ (a>>13) ^ (a>>22)
        add     y2, y0          ; y2 = S1 + CH
        add     y2, [rsp + _XFER + 0*4] ; y2 = k + w + S1 + CH
                movdqa  XTMP3, XTMP1    ; XTMP3 = W[-15]
        mov     y0, a           ; y0 = a
        add     h, y2           ; h = h + S1 + CH + k + w
        mov     y2, a           ; y2 = a
                pslld   XTMP1, (32-7)
        or      y0, c           ; y0 = a|c
        add     d, h            ; d = d + h + S1 + CH + k + w
        and     y2, c           ; y2 = a&c
                psrld   XTMP2, 7
        and     y0, b           ; y0 = (a|c)&b
        add     h, y1           ; h = h + S1 + CH + k + w + S0
                por     XTMP1, XTMP2    ; XTMP1 = W[-15] ror 7
        or      y0, y2          ; y0 = MAJ = (a|c)&b)|(a&c)
        add     h, y0           ; h = h + S1 + CH + k + w + S0 + MAJ

ROTATE_ARGS
                movdqa  XTMP2, XTMP3    ; XTMP2 = W[-15]
        mov     y0, e           ; y0 = e
        mov     y1, a           ; y1 = a
                movdqa  XTMP4, XTMP3    ; XTMP4 = W[-15]
        ror     y0, (25-11)     ; y0 = e >> (25-11)
        xor     y0, e           ; y0 = e ^ (e >> (25-11))
        mov     y2, f           ; y2 = f
        ror     y1, (22-13)     ; y1 = a >> (22-13)
                pslld   XTMP3, (32-18)
        xor     y1, a           ; y1 = a ^ (a >> (22-13)
        ror     y0, (11-6)      ; y0 = (e >> (11-6)) ^ (e >> (25-6))
        xor     y2, g           ; y2 = f^g
                psrld   XTMP2, 18
        ror     y1, (13-2)      ; y1 = (a >> (13-2)) ^ (a >> (22-2))
        xor     y0, e           ; y0 = e ^ (e >> (11-6)) ^ (e >> (25-6))
        and     y2, e           ; y2 = (f^g)&e
        ror     y0, 6           ; y0 = S1 = (e>>6) & (e>>11) ^ (e>>25)
                pxor    XTMP1, XTMP3
        xor     y1, a           ; y1 = a ^ (a >> (13-2)) ^ (a >> (22-2))
        xor     y2, g           ; y2 = CH = ((f^g)&e)^g
                psrld   XTMP4, 3        ; XTMP4 = W[-15] >> 3
        add     y2, y0          ; y2 = S1 + CH
        add     y2, [rsp + _XFER + 1*4] ; y2 = k + w + S1 + CH
        ror     y1, 2           ; y1 = S0 = (a>>2) ^ (a>>13) ^ (a>>22)
                pxor    XTMP1, XTMP2    ; XTMP1 = W[-15] ror 7 ^ W[-15] ror 18
        mov     y0, a           ; y0 = a
        add     h, y2           ; h = h + S1 + CH + k + w
        mov     y2, a           ; y2 = a
                pxor    XTMP1, XTMP4    ; XTMP1 = s0
        or      y0, c           ; y0 = a|c
        add     d, h            ; d = d + h + S1 + CH + k + w
        and     y2, c           ; y2 = a&c
                ;; compute low s1
                pshufd  XTMP2, X3, 11111010b    ; XTMP2 = W[-2] {BBAA}
        and     y0, b           ; y0 = (a|c)&b
        add     h, y1           ; h = h + S1 + CH + k + w + S0
                paddd   XTMP0, XTMP1    ; XTMP0 = W[-16] + W[-7] + s0
        or      y0, y2          ; y0 = MAJ = (a|c)&b)|(a&c)
        add     h, y0           ; h = h + S1 + CH + k + w + S0 + MAJ

ROTATE_ARGS
                movdqa  XTMP3, XTMP2    ; XTMP3 = W[-2] {BBAA}
        mov     y0, e           ; y0 = e
        mov     y1, a           ; y1 = a
        ror     y0, (25-11)     ; y0 = e >> (25-11)
                movdqa  XTMP4, XTMP2    ; XTMP4 = W[-2] {BBAA}
```

```
        xor     y0, e           ; y0 = e ^ (e >> (25-11))
        ror     y1, (22-13)     ; y1 = a >> (22-13)
        mov     y2, f           ; y2 = f
        xor     y1, a           ; y1 = a ^ (a >> (22-13)
        ror     y0, (11-6)      ; y0 = (e >> (11-6)) ^ (e >> (25-6))
                psrlq   XTMP2, 17       ; XTMP2 = W[-2] ror 17 {xBxA}
        xor     y2, g           ; y2 = f^g
                psrlq   XTMP3, 19       ; XTMP3 = W[-2] ror 19 {xBxA}
        xor     y0, e           ; y0 = e ^ (e >> (11-6)) ^ (e >> (25-6))
        and     y2, e           ; y2 = (f^g)&e
                psrld   XTMP4, 10       ; XTMP4 = W[-2] >> 10 {BBAA}
        ror     y1, (13-2)      ; y1 = (a >> (13-2)) ^ (a >> (22-2))
        xor     y1, a           ; y1 = a ^ (a >> (13-2)) ^ (a >> (22-2))
        xor     y2, g           ; y2 = CH = ((f^g)&e)^g
        ror     y0, 6           ; y0 = S1 = (e>>6) & (e>>11) ^ (e>>25)
                pxor    XTMP2, XTMP3
        add     y2, y0          ; y2 = S1 + CH
        ror     y1, 2           ; y1 = S0 = (a>>2) ^ (a>>13) ^ (a>>22)
        add     y2, [rsp + _XFER + 2*4] ; y2 = k + w + S1 + CH
                pxor    XTMP4, XTMP2    ; XTMP4 = s1 {xBxA}
        mov     y0, a           ; y0 = a
        add     h, y2           ; h = h + S1 + CH + k + w
        mov     y2, a           ; y2 = a
                pshufb  XTMP4, SHUF_00BA        ; XTMP4 = s1 {00BA}
        or      y0, c           ; y0 = a|c
        add     d, h            ; d = d + h + S1 + CH + k + w
        and     y2, c           ; y2 = a&c
                paddd   XTMP0, XTMP4    ; XTMP0 = {..., ..., W[1], W[0]}
        and     y0, b           ; y0 = (a|c)&b
        add     h, y1           ; h = h + S1 + CH + k + w + S0
                ;; compute high s1
                pshufd  XTMP2, XTMP0, 01010000b ; XTMP2 = W[-2] {DDCC}
        or      y0, y2          ; y0 = MAJ = (a|c)&b)|(a&c)
        add     h, y0           ; h = h + S1 + CH + k + w + S0 + MAJ

ROTATE_ARGS
                movdqa  XTMP3, XTMP2    ; XTMP3 = W[-2] {DDCC}
        mov     y0, e           ; y0 = e
        ror     y0, (25-11)     ; y0 = e >> (25-11)
        mov     y1, a           ; y1 = a
                movdqa  X0,    XTMP2    ; X0    = W[-2] {DDCC}
        ror     y1, (22-13)     ; y1 = a >> (22-13)
        xor     y0, e           ; y0 = e ^ (e >> (25-11))
        mov     y2, f           ; y2 = f
        ror     y0, (11-6)      ; y0 = (e >> (11-6)) ^ (e >> (25-6))
                psrlq   XTMP2, 17       ; XTMP2 = W[-2] ror 17 {xDxC}
        xor     y1, a           ; y1 = a ^ (a >> (22-13)
        xor     y2, g           ; y2 = f^g
                psrlq   XTMP3, 19       ; XTMP3 = W[-2] ror 19 {xDxC}
        xor     y0, e           ; y0 = e ^ (e >> (11-6)) ^ (e >> (25-6))
        and     y2, e           ; y2 = (f^g)&e
        ror     y1, (13-2)      ; y1 = (a >> (13-2)) ^ (a >> (22-2))
                psrld   X0,    10       ; X0 = W[-2] >> 10 {DDCC}
        xor     y1, a           ; y1 = a ^ (a >> (13-2)) ^ (a >> (22-2))
        ror     y0, 6           ; y0 = S1 = (e>>6) & (e>>11) ^ (e>>25)
        xor     y2, g           ; y2 = CH = ((f^g)&e)^g
                pxor    XTMP2, XTMP3
        ror     y1, 2           ; y1 = S0 = (a>>2) ^ (a>>13) ^ (a>>22)
        add     y2, y0          ; y2 = S1 + CH
        add     y2, [rsp + _XFER + 3*4] ; y2 = k + w + S1 + CH
                pxor    X0, XTMP2       ; X0 = s1 {xDxC}
        mov     y0, a           ; y0 = a
        add     h, y2           ; h = h + S1 + CH + k + w
        mov     y2, a           ; y2 = a
                pshufb  X0, SHUF_DC00   ; X0 = s1 {DC00}
```

```
        or      y0, c           ; y0 = a|c
        add     d, h            ; d = d + h + S1 + CH + k + w
        and     y2, c           ; y2 = a&c
                paddd   X0, XTMP0       ; X0 = {W[3], W[2], W[1], W[0]}
        and     y0, b           ; y0 = (a|c)&b
        add     h, y1           ; h = h + S1 + CH + k + w + S0
        or      y0, y2          ; y0 = MAJ = (a|c)&b)|(a&c)
        add     h, y0           ; h = h + S1 + CH + k + w + S0 + MAJ

ROTATE_ARGS
rotate_Xs
%endm

;; input is [rsp + _XFER + %1 * 4]
%macro DO_ROUND 1
        mov     y0, e           ; y0 = e
        ror     y0, (25-11)     ; y0 = e >> (25-11)
        mov     y1, a           ; y1 = a
        xor     y0, e           ; y0 = e ^ (e >> (25-11))
        ror     y1, (22-13)     ; y1 = a >> (22-13)
        mov     y2, f           ; y2 = f
        xor     y1, a           ; y1 = a ^ (a >> (22-13)
        ror     y0, (11-6)      ; y0 = (e >> (11-6)) ^ (e >> (25-6))
        xor     y2, g           ; y2 = f^g
        xor     y0, e           ; y0 = e ^ (e >> (11-6)) ^ (e >> (25-6))
        ror     y1, (13-2)      ; y1 = (a >> (13-2)) ^ (a >> (22-2))
        and     y2, e           ; y2 = (f^g)&e
        xor     y1, a           ; y1 = a ^ (a >> (13-2)) ^ (a >> (22-2))
        ror     y0, 6           ; y0 = S1 = (e>>6) & (e>>11) ^ (e>>25)
        xor     y2, g           ; y2 = CH = ((f^g)&e)^g
        add     y2, y0          ; y2 = S1 + CH
        ror     y1, 2           ; y1 = S0 = (a>>2) ^ (a>>13) ^ (a>>22)
        add     y2, [rsp + _XFER + %1 * 4]      ; y2 = k + w + S1 + CH
        mov     y0, a           ; y0 = a
        add     h, y2           ; h = h + S1 + CH + k + w
        mov     y2, a           ; y2 = a
        or      y0, c           ; y0 = a|c
        add     d, h            ; d = d + h + S1 + CH + k + w
        and     y2, c           ; y2 = a&c
        and     y0, b           ; y0 = (a|c)&b
        add     h, y1           ; h = h + S1 + CH + k + w + S0
        or      y0, y2          ; y0 = MAJ = (a|c)&b)|(a&c)
        add     h, y0           ; h = h + S1 + CH + k + w + S0 + MAJ
        ROTATE_ARGS
%endm

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; void sha256_sse4(void *input_data, UINT32 digest[8], UINT64 num_blks)
;; arg 1 : pointer to input data
;; arg 2 : pointer to digest
;; arg 3 : Num blocks
section .text
global sha256_sse4
global _sha256_sse4
align 32
sha256_sse4:
_sha256_sse4:
        push    rbx
%ifdef WINABI
        push    rsi
        push    rdi
%endif
        push    rbp
        push    r13
```

```
        push    r14
        push    r15

        sub     rsp,STACK_SIZE
%ifdef WINABI
        movdqa  [rsp + _XMM_SAVE + 0*16],xmm6
        movdqa  [rsp + _XMM_SAVE + 1*16],xmm7
        movdqa  [rsp + _XMM_SAVE + 2*16],xmm8
        movdqa  [rsp + _XMM_SAVE + 3*16],xmm9
        movdqa  [rsp + _XMM_SAVE + 4*16],xmm10
        movdqa  [rsp + _XMM_SAVE + 5*16],xmm11
        movdqa  [rsp + _XMM_SAVE + 6*16],xmm12
%endif

        shl     NUM_BLKS, 6     ; convert to bytes
        jz      done_hash
        add     NUM_BLKS, INP   ; pointer to end of data
        mov     [rsp + _INP_END], NUM_BLKS

        ;; load initial digest
        mov     a,[4*0 + CTX]
        mov     b,[4*1 + CTX]
        mov     c,[4*2 + CTX]
        mov     d,[4*3 + CTX]
        mov     e,[4*4 + CTX]
        mov     f,[4*5 + CTX]
        mov     g,[4*6 + CTX]
        mov     h,[4*7 + CTX]

        movdqa  BYTE_FLIP_MASK, [PSHUFFLE_BYTE_FLIP_MASK wrt rip]
        movdqa  SHUF_00BA, [_SHUF_00BA wrt rip]
        movdqa  SHUF_DC00, [_SHUF_DC00 wrt rip]

loop0:
        lea     TBL,[K256 wrt rip]

        ;; byte swap first 16 dwords
        COPY_XMM_AND_BSWAP      X0, [INP + 0*16], BYTE_FLIP_MASK
        COPY_XMM_AND_BSWAP      X1, [INP + 1*16], BYTE_FLIP_MASK
        COPY_XMM_AND_BSWAP      X2, [INP + 2*16], BYTE_FLIP_MASK
        COPY_XMM_AND_BSWAP      X3, [INP + 3*16], BYTE_FLIP_MASK

        mov     [rsp + _INP], INP

        ;; schedule 48 input dwords, by doing 3 rounds of 16 each
        mov     SRND, 3
align 16
loop1:
        movdqa  XFER, [TBL + 0*16]
        paddd   XFER, X0
        movdqa  [rsp + _XFER], XFER
        FOUR_ROUNDS_AND_SCHED

        movdqa  XFER, [TBL + 1*16]
        paddd   XFER, X0
        movdqa  [rsp + _XFER], XFER
        FOUR_ROUNDS_AND_SCHED

        movdqa  XFER, [TBL + 2*16]
        paddd   XFER, X0
        movdqa  [rsp + _XFER], XFER
        FOUR_ROUNDS_AND_SCHED

        movdqa  XFER, [TBL + 3*16]
        paddd   XFER, X0
```

```
        movdqa  [rsp + _XFER], XFER
        add     TBL, 4*16
        FOUR_ROUNDS_AND_SCHED

        sub     SRND, 1
        jne     loop1

        mov     SRND, 2
loop2:
        paddd   X0, [TBL + 0*16]
        movdqa  [rsp + _XFER], X0
        DO_ROUND        0
        DO_ROUND        1
        DO_ROUND        2
        DO_ROUND        3
        paddd   X1, [TBL + 1*16]
        movdqa  [rsp + _XFER], X1
        add     TBL, 2*16
        DO_ROUND        0
        DO_ROUND        1
        DO_ROUND        2
        DO_ROUND        3

        movdqa  X0, X2
        movdqa  X1, X3

        sub     SRND, 1
        jne     loop2

        addm    [4*0 + CTX],a
        addm    [4*1 + CTX],b
        addm    [4*2 + CTX],c
        addm    [4*3 + CTX],d
        addm    [4*4 + CTX],e
        addm    [4*5 + CTX],f
        addm    [4*6 + CTX],g
        addm    [4*7 + CTX],h

        mov     INP, [rsp + _INP]
        add     INP, 64
        cmp     INP, [rsp + _INP_END]
        jne     loop0

done_hash:
%ifdef WINABI
        movdqa  xmm6,[rsp + _XMM_SAVE + 0*16]
        movdqa  xmm7,[rsp + _XMM_SAVE + 1*16]
        movdqa  xmm8,[rsp + _XMM_SAVE + 2*16]
        movdqa  xmm9,[rsp + _XMM_SAVE + 3*16]
        movdqa  xmm10,[rsp + _XMM_SAVE + 4*16]
        movdqa  xmm11,[rsp + _XMM_SAVE + 5*16]
        movdqa  xmm12,[rsp + _XMM_SAVE + 6*16]
%endif

        add     rsp, STACK_SIZE

        pop     r15
        pop     r14
        pop     r13
        pop     rbp
%ifdef WINABI
        pop     rdi
        pop     rsi
%endif
        pop     rbx
```

```
        ret


section .data
align 64
K256:
        dd      0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5
        dd      0x3956c25b,0x59f111f1,0x923f82a4,0xab1c5ed5
        dd      0xd807aa98,0x12835b01,0x243185be,0x550c7dc3
        dd      0x72be5d74,0x80deb1fe,0x9bdc06a7,0xc19bf174
        dd      0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc
        dd      0x2de92c6f,0x4a7484aa,0x5cb0a9dc,0x76f988da
        dd      0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7
        dd      0xc6e00bf3,0xd5a79147,0x06ca6351,0x14292967
        dd      0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13
        dd      0x650a7354,0x766a0abb,0x81c2c92e,0x92722c85
        dd      0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3
        dd      0xd192e819,0xd6990624,0xf40e3585,0x106aa070
        dd      0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5
        dd      0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,0x682e6ff3
        dd      0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208
        dd      0x90befffa,0xa4506ceb,0xbef9a3f7,0xc67178f2

PSHUFFLE_BYTE_FLIP_MASK: ddq 0x0c0d0e0f08090a0b0405060700010203

; shuffle xBxA -> 00BA
_SHUF_00BA:              ddq 0xFFFFFFFFFFFFFFFF0b0a090803020100

; shuffle xDxC -> DC00
_SHUF_DC00:              ddq 0x0b0a090803020100FFFFFFFFFFFFFFFF

%ifidn __OUTPUT_FORMAT__,elf
section .note.GNU-stack noalloc noexec nowrite progbits
%endif
%ifidn __OUTPUT_FORMAT__,elf32
section .note.GNU-stack noalloc noexec nowrite progbits
%endif
%ifidn __OUTPUT_FORMAT__,elf64
section .note.GNU-stack noalloc noexec nowrite progbits
%endif
```