**XACQUIRE/XRELEASE — Hardware Lock Elision Prefix Hints**

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>64/32bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2 XACQUIRE</td>
<td>V/V</td>
<td>HLE1</td>
<td>A hint used with an &quot;XACQUIRE-enabled&quot; instruction to start lock elision on the instruction memory operand address.</td>
</tr>
<tr>
<td>F3 XRELEASE</td>
<td>V/V</td>
<td>HLE</td>
<td>A hint used with an &quot;XRELEASE-enabled&quot; instruction to end lock elision on the instruction memory operand address.</td>
</tr>
</tbody>
</table>

**NOTES:**

1. Software is not required to check the HLE feature flag to use XACQUIRE or XRELEASE, as they are treated as regular prefix if HLE feature flag reports 0.

**Description**

The XACQUIRE prefix is a hint to start lock elision on the memory address specified by the instruction and the XRELEASE prefix is a hint to end lock elision on the memory address specified by the instruction.

The XACQUIRE prefix hint can only be used with the following instructions (these instructions are also referred to as XACQUIRE-enabled when used with the XACQUIRE prefix):

- Instructions with an explicit LOCK prefix (F0H) prepended to forms of the instruction where the destination operand is a memory operand: ADD, ADC, AND, BTC, BTR, BTS, CMPXCHG, CMPXCHG8B, DEC, INC, NEG, NOT, OR, SBB, SUB, XOR, XADD, and XCHG.
- The XCHG instruction either with or without the presence of the LOCK prefix.

The XRELEASE prefix hint can only be used with the following instructions (also referred to as XRELEASE-enabled when used with the XRELEASE prefix):

- Instructions with an explicit LOCK prefix (F0H) prepended to forms of the instruction where the destination operand is a memory operand: ADD, ADC, AND, BTC, BTR, BTS, CMPXCHG, CMPXCHG8B, DEC, INC, NEG, NOT, OR, SBB, SUB, XOR, XADD, and XCHG.
- The XCHG instruction either with or without the presence of the LOCK prefix.
- The "MOV mem, reg" (Opcode 88H/89H) and "MOV mem, imm" (Opcode C6H/C7H) instructions. In these cases, the XRELEASE is recognized without the presence of the LOCK prefix.

The lock variables must satisfy the guidelines described in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*, Section 15.3.3, for elision to be successful, otherwise an HLE abort may be signaled.

If an encoded byte sequence that meets XACQUIRE/XRELEASE requirements includes both prefixes, then the HLE semantic is determined by the prefix byte that is placed closest to the instruction opcode. For example, an F3F2C6 will not be treated as a XRELEASE-enabled instruction since the F2H (XACQUIRE) is closest to the instruction opcode C6. Similarly, an F2F3F0 prefixed instruction will be treated as a XRELEASE-enabled instruction since F3H (XRELEASE) is closest to the instruction opcode.

**Intel 64 and IA-32 Compatibility**

The effect of the XACQUIRE/XRELEASE prefix hint is the same in non-64-bit modes and in 64-bit mode.

For instructions that do not support the XACQUIRE hint, the presence of the F2H prefix behaves the same way as prior hardware, according to:

- REPNE/REPNZ semantics for string instructions,
- Serve as SIMD prefix for legacy SIMD instructions operating on XMM register
- Cause #UD if prepending the VEX prefix.
- Undefined for non-string instructions or other situations.

For instructions that do not support the XRELEASE hint, the presence of the F3H prefix behaves the same way as in prior hardware, according to:

- REP/REPE/REPZ semantics for string instructions,
• Serve as SIMD prefix for legacy SIMD instructions operating on XMM register
• Cause #UD if prepending the VEX prefix.
• Undefined for non-string instructions or other situations.

Operation

**XACQUIRE**

IF XACQUIRE-enabled instruction
THEN
  IF (HLE_NEST_COUNT < MAX_HLE_NEST_COUNT) THEN
    HLE_NEST_COUNT++
    IF (HLE_NEST_COUNT = 1) THEN
      HLE_ACTIVE ← 1
      IF 64-bit mode
        THEN
          restartRIP ← instruction pointer of the XACQUIRE-enabled instruction
        ELSE
          restartEIP ← instruction pointer of the XACQUIRE-enabled instruction
        FI;
      Enter HLE Execution (* record register state, start tracking memory state *)
    FI; (* HLE_NEST_COUNT = 1*)
    IF ElisionBufferAvailable
      THEN
        Allocate elision buffer
        Record address and data for forwarding and commit checking
        Perform elision
      ELSE
        Perform lock acquire operation transactionally but without elision
      FI;
    ELSE (* HLE_NEST_COUNT = MAX_HLE_NEST_COUNT *)
      GOTO HLE_ABORT_PROCESSING
    FI;
  ELSE (* HLE_NEST_COUNT = MAX_HLE_NEST_COUNT *)
    GOTO HLE_ABORT_PROCESSING
  FI;
ELSE
  Treat instruction as non-XACQUIRE F2H prefixed legacy instruction
FI;

**XRELEASE**

IF XRELEASE-enabled instruction
THEN
  IF (HLE_NEST_COUNT > 0)
    THEN
      HLE_NEST_COUNT--
      IF lock address matches in elision buffer THEN
        IF lock satisfies address and value requirements THEN
          Deallocate elision buffer
        ELSE
          GOTO HLE_ABORT_PROCESSING
        FI;
      ELSE
        GOTO HLE_ABORT_PROCESSING
      FI;
    FI;
  IF (HLE_NEST_COUNT = 0)
    THEN
      IF NoAllocatedElisionBuffer
        THEN

4-564 Vol. 2B XACQUIRE/XRELEASE — Hardware Lock Elision Prefix Hints
Try to commit transactional execution
IF fail to commit transactional execution
THEN
   GOTO HLE_ABORT_PROCESSING;
ELSE (* commit success *)
   HLE_ACTIVE ← 0
   FI;
ELSE
   GOTO HLE_ABORT_PROCESSING
   FI;
FI; (* HLE_NEST_COUNT > 0 *)
ELSE
   Treat instruction as non-XRELEASE F3H prefixed legacy instruction
FI;
(* For any HLE abort condition encountered during HLE execution *)
HLE_ABORT_PROCESSING:
   HLE_ACTIVE ← 0
   HLE_NEST_COUNT ← 0
   Restore architectural register state
   Discard memory updates performed in transaction
   Free any allocated lock elision buffers
   IF 64-bit mode
      THEN
         RIP ← restartRIP
      ELSE
         EIP ← restartEIP
      FI;
   Execute and retire instruction at RIP (or EIP) and ignore any HLE hint
END

SIMD Floating-Point Exceptions
None

Other Exceptions
#GP(0) If the use of prefix causes instruction length to exceed 15 bytes.
**XABORT — Transactional Abort**

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32bit Mode</th>
<th>CPUID Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6 F8 ib XABORT imm8</td>
<td>A</td>
<td>V/V</td>
<td>RTM</td>
<td>Causes an RTM abort if in RTM execution</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>Operand 1</th>
<th>Operand2</th>
<th>Operand3</th>
<th>Operand4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>imm8</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Description**

XABORT forces an RTM abort. Following an RTM abort, the logical processor resumes execution at the fallback address computed through the outermost XBEGIN instruction. The EAX register is updated to reflect an XABORT instruction caused the abort, and the imm8 argument will be provided in bits 31:24 of EAX.

**Operation**

XABORT

IF RTM_ACTIVE = 0
  THEN
    Treat as NOP;
  ELSE
    GOTO RTM_ABORT_PROCESSING;
  FI;

(* For any RTM abort condition encountered during RTM execution *)

RTM_ABORT_PROCESSING:
  Restore architectural register state;
  Discard memory updates performed in transaction;
  Update EAX with status and XABORT argument;
  RTM_NEST_COUNT ← 0;
  RTM_ACTIVE ← 0;
  IF 64-bit Mode
    THEN
      RIP ← fallbackRIP;
    ELSE
      EIP ← fallbackEIP;
    FI;
  END

**Flags Affected**

None

**Intel C/C++ Compiler Intrinsic Equivalent**

XABORT: void __xabort( unsigned int);

**SIMD Floating-Point Exceptions**

None
Other Exceptions

#UD

CPUID.(EAX=7, ECX=0):EBX.RTM[bit 11] = 0.
If LOCK prefix is used.
### XBEGIN — Transactional Begin

**Description**

The XBEGIN instruction specifies the start of an RTM code region. If the logical processor was not already in transactional execution, then the XBEGIN instruction causes the logical processor to transition into transactional execution. The XBEGIN instruction that transitions the logical processor into transactional execution is referred to as the outermost XBEGIN instruction. The instruction also specifies a relative offset to compute the address of the fallback code path following a transactional abort.

On an RTM abort, the logical processor discards all architectural register and memory updates performed during the RTM execution and restores architectural state to that corresponding to the outermost XBEGIN instruction. The fallback address following an abort is computed from the outermost XBEGIN instruction.

**Operation**

```assembly
XBEGIN
IF RTM_NEST_COUNT < MAX_RTM_NEST_COUNT
    THEN
        RTM_NEST_COUNT++
        IF RTM_NEST_COUNT = 1 THEN
            IF 64-bit Mode
                THEN
                    fallbackRIP ← RIP + SgnExtend64(IMM)
                        (* RIP is instruction following XBEGIN instruction *)
            ELSE
                fallbackEIP ← EIP + SgnExtend32(IMM)
                        (* EIP is instruction following XBEGIN instruction *)
            FI;
        FI;
    IF (64-bit mode)
        THEN IF (fallbackRIP is not canonical)
            THEN #GP(0)
            FI;
        ELSE IF (fallbackEIP outside code segment limit)
            THEN #GP(0)
            FI;
        FI;
    RTM_ACTIVE ← 1
    Enter RTM Execution (* record register state, start tracking memory state*)
```

### Instruction Operand Encoding

<table>
<thead>
<tr>
<th>OPCODE/INSTRUCTION</th>
<th>OP/EN</th>
<th>OPERAND 1</th>
<th>OPERAND 2</th>
<th>OPERAND 3</th>
<th>OPERAND 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>C7 F8 XBEGIN rel16</td>
<td>A V/V</td>
<td>V/V</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C7 F8 XBEGIN rel32</td>
<td>A V/V</td>
<td>V/V</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Description**

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C7 F8 XBEGIN rel16</td>
<td>A V/V</td>
<td>V/V</td>
<td>RTM</td>
<td>Specifies the start of an RTM region. Provides a 16-bit relative offset to compute the address of the fallback instruction address at which execution resumes following an RTM abort.</td>
</tr>
<tr>
<td>C7 F8 XBEGIN rel32</td>
<td>A V/V</td>
<td>V/V</td>
<td>RTM</td>
<td>Specifies the start of an RTM region. Provides a 32-bit relative offset to compute the address of the fallback instruction address at which execution resumes following an RTM abort.</td>
</tr>
</tbody>
</table>
FI; (* RTM_NEST_COUNT = 1 *)
ELSE (* RTM_NEST_COUNT = MAX RTM_NEST_COUNT *)
    GOTO RTM_ABORT_PROCESSING
FI;

("For any RTM abort condition encountered during RTM execution")
RTM_ABORT_PROCESSING:
    Restore architectural register state
    Discard memory updates performed in transaction
    Update EAX with status
    RTM_NEST_COUNT ← 0
    RTM_ACTIVE ← 0
    IF 64-bit mode
        THEN
            RIP ← fallbackRIP
        ELSE
            EIP ← fallbackEIP
        FI;
    END

Flags Affected
None

Intel C/C++ Compiler Intrinsic Equivalent
XBEGIN: unsigned int _xbegin( void );

SIMD Floating-Point Exceptions
None

Protected Mode Exceptions
# UD CPUID (EAX=7, ECX=0): EBX.RTM[bit 11] = 0.
   If LOCK prefix is used.
# GP(0) If the fallback address is outside the CS segment.

Real-Address Mode Exceptions
# GP(0) If the fallback address is outside the address space 0000H and FFFFH.
# UD CPUID (EAX=7, ECX=0): EBX.RTM[bit 11] = 0.
   If LOCK prefix is used.

Virtual-8086 Mode Exceptions
# GP(0) If the fallback address is outside the address space 0000H and FFFFH.
# UD CPUID (EAX=7, ECX=0): EBX.RTM[bit 11] = 0.
   If LOCK prefix is used.

Compatibility Mode Exceptions
Same exceptions as in protected mode.

64-bit Mode Exceptions
# UD CPUID (EAX=7, ECX=0): EBX.RTM[bit 11] = 0.
   If LOCK prefix is used.
# GP(0)  If the fallback address is non-canonical.
XEND — Transactional End

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32bit Mode</th>
<th>OPUID Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XEND</td>
<td>A</td>
<td>V/V</td>
<td>RTM</td>
<td>Specifies the end of an RTM code region.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>Operand 1</th>
<th>Operand2</th>
<th>Operand3</th>
<th>Operand4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Description**

The instruction marks the end of an RTM code region. If this corresponds to the outermost scope (that is, including this XEND instruction, the number of XBEGIN instructions is the same as number of XEND instructions), the logical processor will attempt to commit the logical processor state atomically. If the commit fails, the logical processor will rollback all architectural register and memory updates performed during the RTM execution. The logical processor will resume execution at the fallback address computed from the outermost XBEGIN instruction. The EAX register is updated to reflect RTM abort information.

XEND executed outside a transactional region will cause a # GP (General Protection Fault).

**Operation**

XEND

IF (RTM_ACTIVE = 0) THEN
  SIGNAL #GP
ELSE
  RTM_NEST_COUNT--
  IF (RTM_NEST_COUNT = 0) THEN
    Try to commit transaction
    IF fail to commit transactional execution THEN
      GOTO RTM_ABORT_PROCESSING
    ELSE (* commit success *)
      RTM_ACTIVE ← 0
    FI;
  FI;
FI;

(* For any RTM abort condition encountered during RTM execution *)

RTM_ABORT_PROCESSING:
  Restore architectural register state
  Discard memory updates performed in transaction
  Update EAX with status
  RTM_NEST_COUNT ← 0
  RTM_ACTIVE ← 0
  IF 64-bit Mode THEN
    RIP ← fallbackRIP
  ELSE
    EIP ← fallbackEIP
  FI;
END
Flags Affected
None

Intel C/C++ Compiler Intrinsic Equivalent
XEND: void_xend(void);

SIMD Floating-Point Exceptions
None

Other Exceptions
# UD CPUID (EAX=7, ECX=0): EBX.RTM[bit 11] = 0.
If LOCK or 66H or F2H or F3H prefix is used.
# GP(0) If RTM_ACTIVE = 0.
**XTEST — Test If In Transactional Execution**

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32bit</th>
<th>CPUID Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF 01 D6 XTEST</td>
<td>A</td>
<td>V/V</td>
<td>HLE or RTM</td>
<td>Test if executing in a transactional region</td>
</tr>
</tbody>
</table>

### Instruction Operand Encoding

<table>
<thead>
<tr>
<th>Op/En</th>
<th>Operand 1</th>
<th>Operand2</th>
<th>Operand3</th>
<th>Operand4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Description**

The XTEST instruction queries the transactional execution status. If the instruction executes inside a transactionally executing RTM region or a transactionally executing HLE region, then the ZF flag is cleared; else it is set.

**Operation**

XTEST  
IF (RTM_ACTIVE = 1 OR HLE_ACTIVE = 1)  
THEN  
ZF ← 0  
ELSE  
ZF ← 1  
FI;

**Flags Affected**

The ZF flag is cleared if the instruction is executed transactionally; otherwise it is set to 1. The CF, OF, SF, PF, and AF flags are cleared.

**Intel C/C++ Compiler Intrinsic Equivalent**

XTEST: int _xtest( void );

**SIMD Floating-Point Exceptions**

None

**Other Exceptions**

#UD 
CPUID.(EAX=7, ECX=0):HLE[bit 4] = 0 and CPUID.(EAX=7, ECX=0):RTM[bit 11] = 0. If LOCK or 66H or F2H or F3H prefix is used.