Inside Intel® Next Generation Nehalem Microarchitecture

Ronak Singhal
Nehalem Architect

NGMS001
Agenda

• Nehalem Design Philosophy
• Enhanced Processor Core
  • Performance Features
  • Simultaneous Multi-Threading
• Feeding the Engine
  • New Cache Hierarchy
  • New Platform Architecture
• Performance Acceleration
  • Virtualization
  • New Instructions
Tick Tock Development Model

2005-06

TICK

Intel® Pentium® D, Xeon®, Core™ processors

65nm

TOCK

2007-08

TICK

Intel® Core™2, Xeon® processors

45nm

TOCK

PENRYN

NEHALEM

2009-10

TICK

WESTMERE

SANDY BRIDGE

32nm

Nehalem - The Intel® 45 nm Tock Processor
Nehalem Design Goals

World class performance combined with superior energy efficiency - Optimized for:

- Single Thread
- Multi-thread

A single, scalable, foundation optimized across each segment and power envelope

Dynamically scaled performance when needed to maximize energy efficiency

A Dynamic and Design Scalable Microarchitecture

Optimized for: Optimized for:

- Workstation / Server
- Desktop / Mobile
- All Usages

Existing Apps

Emerging Apps
Scalable Nehalem Core

Same core for all segments

Common feature set

Common software optimization

Nehalem: Next Generation Intel® Microarchitecture

Optimized Nehalem core to meet all market segments

Servers/Workstations
- Energy Efficiency
- Performance
- Virtualization
- Reliability
- Capacity
- Scalability

Desktop
- Performance
- Graphics
- Energy Efficiency
- Idle Power
- Security

Mobile
- Battery Life
- Performance
- Energy Efficiency
- Graphics
- Security
Intel® Core™ Microarchitecture Recap

- Wide Dynamic Execution
  - 4-wide decode/ rename/ retire
- Advanced Digital Media Boost
  - 128-bit wide SSE execution units
- Intel HD Boost
  - New SSE4.1 Instructions
- Smart Memory Access
  - Memory Disambiguation
  - Hardware Prefetching
- Advanced Smart Cache
  - Low latency, high BW shared L2 cache

Nehalem builds on the great Intel® Core™ microarchitecture
Agenda

• Nehalem Design Philosophy

• Enhanced Processor Core
  • Performance Features
  • Simultaneous Multi-Threading

• Feeding the Engine
  • New Cache Hierarchy
  • New Platform Architecture

• Performance Acceleration
  • Virtualization
  • New Instructions
Designed for Performance

- New SSE4.2 Instructions
- Improved Lock Support
- Additional Caching Hierarchy
- Deeper Buffers
- Improved Loop Streaming
- Simultaneous Multi-Threading
- Faster Virtualization
- Better Branch Prediction

Execution Units

- L1 Data Cache
- Memory Ordering & Execution

Out-of-Order Scheduling & Retirement

- Instruction Decode & Microcode

L2 Cache & Interrupt Servicing

Paging

Branch Prediction

Instruction Fetch & L1 Cache

Additional Caching Hierarchy
Enhanced Processor Core

Instruction Fetch and Pre Decode

Instruction Queue

Decode

Rename/Allocate

Retirement Unit (ReOrder Buffer)

Reservation Station

Execution Units

ITLB

32kB Instruction Cache

2nd Level TLB

256kB 2nd Level Cache

32kB Data Cache

2nd Level Cache

L3 and beyond

Front End Execution Engine Memory

Intel Developer FORUM
Front-end

• Responsible for feeding the compute engine
  - Decode instructions
  - Branch Prediction

• Key Intel® Core™2 processor features
  - 4-wide decode
  - Macrofusion
  - Loop Stream Detector
Macrofusion Recap

• Introduced in the Intel® Core™ 2 Processor Family
• TEST/CMP instruction followed by a conditional branch treated as a single instruction
  - Decode as one instruction
  - Execute as one instruction
  - Retire as one instruction
• Higher performance
  - Improves throughput
  - Reduces execution latency
• Improved power efficiency
  - Less processing required to accomplish the same work
Nehalem Macrofusion

- Goal: Identify more macrofusion opportunities for increased *performance* and *power efficiency*
- Support all the cases in Intel® Core™ 2 processor **PLUS**
  - CMP+Jcc macrofusion added for the following branch conditions
    - JL/JNGE
    - JGE/JNL
    - JLE/JNG
    - JG/JNLE
- Intel® Core™ 2 processor only supports macrofusion in 32-bit mode
  - Nehalem supports macrofusion in both 32-bit and 64-bit modes

*Increased macrofusion benefit on Nehalem*
Loop Stream Detector Reminder

- Loops are very common in most software
- Take advantage of knowledge of loops in HW
  - Decoding the same instructions over and over
  - Making the same branch predictions over and over
- Loop Stream Detector identifies software loops
  - Stream from Loop Stream Detector instead of normal path
  - Disable unneeded blocks of logic for power savings
  - Higher performance by removing instruction fetch limitations

Intel® Core™2 processor Loop Stream Detector

- Nehalem: Next Generation Intel® Microarchitecture
Nehalem Loop Stream Detector

- Same concept as in prior implementations
- **Higher performance:** Expand the size of the loops detected
- **Improved power efficiency:** Disable even more logic
Branch Prediction Reminder

• Goal: Keep powerful compute engine fed
• Options:
  - Stall pipeline while determining branch direction/target
  - Predict branch direction/target and correct if wrong
• High branch prediction accuracy leads to:
  - **Performance**: Minimize amount of time wasted correcting from incorrect branch predictions
    ▪ Through higher branch prediction accuracy
    ▪ Through faster correction when prediction is wrong
  - **Power efficiency**: Minimize number of speculative/incorrect micro-ops that are executed

*Continued focus on branch prediction improvements*
L2 Branch Predictor

• Problem: Software with a large code footprint not able to fit well in existing branch predictors
  - Example: Database applications

• Solution: Use multi-level branch prediction scheme

• Benefits:
  - Higher *performance* through improved branch prediction accuracy
  - Greater *power efficiency* through less mis-speculation
Renamed Return Stack Buffer (RSB)

• Instruction Reminder
  - CALL: Entry into functions
  - RET: Return from functions

• Classical Solution
  - Return Stack Buffer (RSB) used to predict RET
  - Problems:
    ▫ RSB can be corrupted by speculative path
    ▫ RSB can overflow, overwriting valid entries

• The **Renamed RSB**
  - No RET mispredicts in the common case
  - No stack overflow issue
Execution Engine

• Start with powerful Intel® Core™ 2 processor execution engine
  - Dynamic 4-wide Execution
  - Advanced Digital Media Boost
    ▪ 128-bit wide SSE
  - HD Boost (Penryn)
    ▪ SSE4.1 instructions
  - Super Shuffler (Penryn)
  - Add Nehalem enhancements
  - Additional parallelism for higher performance
Execute 6 operations/cycle
- 3 Memory Operations
  - 1 Load
  - 1 Store Address
  - 1 Store Data
- 3 “Computational” Operations

Unified Reservation Station
- Schedules operations to Execution units
- Single Scheduler for all Execution Units
- Can be used by all integer, all FP, etc.
Increased Parallelism

- Goal: Keep powerful execution engine fed
- Nehalem increases size of out of order window by 33%
- Must also increase other corresponding structures

<table>
<thead>
<tr>
<th>Structure</th>
<th>Merom</th>
<th>Nehalem</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservation Station</td>
<td>32</td>
<td>36</td>
<td>Dispatches operations to execution units</td>
</tr>
<tr>
<td>Load Buffers</td>
<td>32</td>
<td>48</td>
<td>Tracks all load operations allocated</td>
</tr>
<tr>
<td>Store Buffers</td>
<td>20</td>
<td>32</td>
<td>Tracks all store operations allocated</td>
</tr>
</tbody>
</table>

Increased Resources for Higher Performance

Source: Microarchitecture specifications

Nehalem: Next Generation Intel® Microarchitecture
Enhanced Memory Subsystem

• Start with great Intel® Core™ 2 Processor Features
  - Memory Disambiguation
  - Hardware Prefetchers
  - Advanced Smart Cache

• New Nehalem Features
  - New TLB Hierarchy
  - Fast 16-Byte unaligned accesses
  - Faster Synchronization Primitives
New TLB Hierarchy

• Problem: Applications continue to grow in data size
• Need to increase TLB size to keep the pace for performance
• Nehalem adds new low-latency unified 2\textsuperscript{nd} level TLB

<table>
<thead>
<tr>
<th></th>
<th># of Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1\text{st} Level Instruction TLBs</strong></td>
<td></td>
</tr>
<tr>
<td>Small Page (4k)</td>
<td>128</td>
</tr>
<tr>
<td>Large Page (2M/4M)</td>
<td>7 per thread</td>
</tr>
<tr>
<td><strong>1\text{st} Level Data TLBs</strong></td>
<td></td>
</tr>
<tr>
<td>Small Page (4k)</td>
<td>64</td>
</tr>
<tr>
<td>Large Page (2M/4M)</td>
<td>32</td>
</tr>
<tr>
<td><strong>New 2\text{nd} Level Unified TLB</strong></td>
<td></td>
</tr>
<tr>
<td>Small Page Only</td>
<td>512</td>
</tr>
</tbody>
</table>
Fast Unaligned Cache Accesses

- Two flavors of 16-byte SSE loads/stores exist
  - Aligned (MOVAPS/D, MOVDQA) -- Must be aligned on a 16-byte boundary
  - Unaligned (MOVUPS/D, MOVDQU) -- No alignment requirement
- Prior to Nehalem
  - Optimized for Aligned instructions
  - Unaligned instructions slower, lower throughput -- Even for aligned accesses!
    - Required multiple uops (not energy efficient)
  - Compilers would largely avoid unaligned load
    - 2-instruction sequence (MOVSD+MOVHPD) was faster
- Nehalem optimizes Unaligned instructions
  - Same speed/throughput as Aligned instructions on aligned accesses
  - Optimizations for making accesses that cross 64-byte boundaries fast
    - Lower latency/higher throughput than Intel® Core™ 2 processor
  - Aligned instructions remain fast
- No reason to use aligned instructions on Nehalem!
- Benefits:
  - Compiler can now use unaligned instructions without fear
  - Higher performance on key media algorithms
  - More energy efficient than prior implementations
Faster Synchronization Primitives

- Multi-threaded software becoming more prevalent
- **Scalability** of multi-thread applications can be limited by synchronization
- Synchronization primitives: LOCK prefix, XCHG
- Reduce synchronization latency for legacy software

Greater thread **scalability** with Nehalem
Simultaneous Multi-Threaded (SMT)

- **SMT**
  - Run 2 threads at the same time per core
- Take advantage of 4-wide execution engine
  - Keep it fed with multiple threads
  - Hide latency of a single thread
- Most *power efficient* performance feature
  - Very low die area cost
  - Can provide significant performance benefit depending on application
  - Much more efficient than adding an entire core
- Nehalem advantages
  - Larger caches
  - Massive memory BW

Simultaneous multi-threading enhances performance and energy efficiency
SMT Implementation Details

• Multiple policies possible for implementation of SMT
  • Replicated – Duplicate state for SMT
    - Register state
    - Renamed RSB
    - Large page ITLB
  • Partitioned – Statically allocated between threads
    - Key buffers: Load, store, Reorder
    - Small page ITLB
  • Competitively shared – Depends on thread’s dynamic behavior
    - Reservation station
    - Caches
    - Data TLBs, 2nd level TLB
• Unaware
  - Execution units
Agenda

• Nehalem Design Philosophy
• Enhanced Processor Core
  • Performance Features
  • Simultaneous Multi-Threading

• Feeding the Engine
  • New Cache Hierarchy
  • New Platform Architecture

• Performance Acceleration
  • Virtualization
  • New Instructions
Feeding the Execution Engine

- Powerful 4-wide dynamic execution engine
- Need to keep providing fuel to the execution engine
- Nehalem Goals
  - **Low latency** to retrieve data
    ▪ Keep execution engine fed w/o stalling
  - High data **bandwidth**
    ▪ Handle requests from multiple cores/threads seamlessly
  - **Scalability**
    ▪ Design for increasing core counts
- Combination of great **cache hierarchy** and **new platform**

Nehalem designed to feed the execution engine
Designed For Modularity

Differentiation in the “Uncore”:

- # cores
- # mem channels
- # QPI Links
- Size of cache
- Type of Memory
- Power Management
- Integrated graphics

2008 - 2009 Servers & Desktops

Optimal price / performance / energy efficiency for server, desktop and mobile products
Intel Smart Cache – Core Caches

- New 3-level Cache Hierarchy
- 1st level caches
  - 32kB Instruction cache
  - 32kB Data Cache
    - Support more L1 misses in parallel than Core 2
- 2nd level Cache
  - New cache introduced in Nehalem
  - Unified (holds code and data)
  - 256 kB per core
  - **Performance:** Very low latency
  - **Scalability:** As core count increases, reduce pressure on shared cache
Intel Smart Cache -- 3rd Level Cache

- New 3rd level cache
- Shared across all cores
- Size depends on # of cores
  - Quad-core: Up to 8MB
  - **Scalability:**
    - Built to vary size with varied core counts
    - Built to easily increase L3 size in future parts

- Inclusive cache policy for best **performance**
  - Data residing in L1/L2 **must** be present in 3rd level cache
Inclusive vs. Exclusive Caches – Cache Miss

**Exclusive**

Data request from Core 0 misses Core 0’s L1 and L2
Request sent to the L3 cache

**Inclusive**
Inclusive vs. Exclusive Caches – Cache Miss

**Exclusive**

Core 0 looks up the L3 Cache
Data not in the L3 Cache

**Inclusive**

Core 0 looks up the L3 Cache
Data not in the L3 Cache
Inclusive vs. Exclusive Caches – Cache Miss

**Exclusive**

```
L3 Cache
Core 0  Core 1  Core 2  Core 3
MISS!
```

Must check other cores

**Inclusive**

```
L3 Cache
Core 0  Core 1  Core 2  Core 3
MISS!
```

Guaranteed data is not on-die

Greater *scalability* from inclusive approach
Inclusive vs. Exclusive Caches – Cache Hit

**Exclusive**

- L3 Cache
  - Core 0
  - Core 1
  - Core 2
  - Core 3

  *HIT!*

  No need to check other cores

**Inclusive**

- L3 Cache
  - Core 0
  - Core 1
  - Core 2
  - Core 3

  *HIT!*

  Data could be in another core

**BUT** Nehalem is smart...
Inclusive vs. Exclusive Caches – Cache Hit

- Maintain a set of "core valid" bits per cache line in the L3 cache
- Each bit represents a core
- If the L1/L2 of a core may contain the cache line, then core valid bit is set to "1"
- No snoops of cores are needed if no bits are set
- If more than 1 bit is set, line cannot be in Modified state in any core

Core valid bits limit unnecessary snoops

Inclusive Caches:

- L3 Cache
  - Hit!
  - Core valid bits: 0000
Inclusive vs. Exclusive Caches – Read from other core

**Exclusive**

- L3 Cache
- Core 0
- Core 1
- Core 2
- Core 3

Must check all other cores

**Inclusive**

- L3 Cache
- Core 0
- Core 1
- Core 2
- Core 3

Only need to check the core whose core valid bit is set
Today’s Platform Architecture
Nehalem-EP Platform Architecture

- Integrated Memory Controller
  - 3 DDR3 channels per socket
  - Massive memory **bandwidth**
  - Memory Bandwidth scales with # of processors
  - Very **low memory latency**
- Intel® QuickPath Interconnect (QPI)
  - New point-to-point interconnect
  - Socket to socket connections
  - Socket to chipset connections
  - Build **scalable** solutions

**Significant performance leap from new platform**
Intel® QuickPath Interconnect

- Nehalem introduces new Intel® QuickPath Interconnect (QPI)
- **High bandwidth, low latency** point to point interconnect
- Up to 6.4 GT/sec initially
  - 6.4 GT/sec -> 12.8 GB/sec
  - Bi-directional link -> 25.6 GB/sec per link
  - Future implementations at even higher speeds
- Highly **scalable** for systems with varying # of sockets
Integrated Memory Controller (IMC)

• Memory controller optimized per market segment
• Initial Nehalem products
  - Native DDR3 IMC
  - Up to 3 channels per socket
  - Speeds up to DDR3-1333
    ▪ Massive memory bandwidth
  - Designed for low latency
  - Support RDIMM and UDIMM
  - RAS Features
• Future products
  - Scalability
    ▪ Vary # of memory channels
    ▪ Increase memory speeds
    ▪ Buffered and Non-Buffered solutions
  - Market specific needs
    ▪ Higher memory capacity
    ▪ Integrated graphics

Significant performance through new IMC
IMC Memory Bandwidth (BW)

- 3 memory channels per socket
- Up to DDR3-1333 at launch
  - Massive memory BW
  - HEDT: 32 GB/sec peak
  - 2S server: 64 GB/sec peak

**Scalability**
- Design IMC and core to take advantage of BW
- Allow performance to scale with cores
  - Core enhancements
    - Support more cache misses per core
    - Aggressive hardware prefetching w/ throttling enhancements
  - Example IMC Features
    - Independent memory channels
    - Aggressive Request Reordering

Massive memory BW provides performance and scalability
Non-Uniform Memory Access (NUMA)

- FSB architecture
  - All memory in one location
- Starting with Nehalem
  - Memory located in multiple places
- Latency to memory dependent on location
- Local memory
  - Highest BW
  - Lowest latency
- Remote Memory
  - Higher latency

Ensure software is NUMA-optimized for best performance
Local Memory Access

- CPU0 requests cache line X, not present in any CPU0 cache
  - CPU0 requests data from its DRAM
  - CPU0 snoops CPU1 to check if data is present
- Step 2:
  - DRAM returns data
  - CPU1 returns snoop response
- Local memory latency is the maximum latency of the two responses
- Nehalem optimized to keep key latencies close to each other
Remote Memory Access

• CPU0 requests cache line X, not present in any CPU0 cache
  - CPU0 requests data from CPU1
  - Request sent over QPI to CPU1
  - CPU1’s IMC makes request to its DRAM
  - CPU1 snoops internal caches
  - Data returned to CPU0 over QPI

• Remote memory latency a function of having a low latency interconnect
Memory Latency Comparison

- **Low memory latency** critical to high performance
- Design integrated memory controller for low latency
- Need to optimize both local and remote memory latency
- Nehalem delivers
  - Huge reduction in local memory latency
  - Even remote memory latency is fast
- Effective memory latency depends per application/OS
  - Percentage of local vs. remote accesses
  - NHM has lower latency regardless of mix

![Relative Memory Latency Comparison](image)
Agenda

- Nehalem Design Philosophy
- Enhanced Processor Core
  - Performance Features
  - Simultaneous Multi-Threading
- Feeding the Engine
  - New Cache Hierarchy
  - New Platform Architecture
- Performance Acceleration
  - Virtualization
  - New Instructions
Virtualization

• To get best virtualized performance
  - Have best native performance
  - Reduce:
    ▪ # of transitions into/out of virtual machine
    ▪ Latency of transitions

• Nehalem virtualization features
  - Reduced latency for transitions
  - Virtual Processor ID (VPID) to reduce effective cost of transitions
  - Extended Page Table (EPT) to reduce # of transitions

Great virtualization performance w/ Nehalem
Virtualization Enhancements

Round Trip Virtualization Latency

![Bar chart showing relative latency for Merom, Penryn, and Nehalem]

- **>40% Faster**

Source: pre-silicon/post-silicon measurements

Virtualization Latency:
Time to enter and exit virtual machine

**Architecture:**

- Extended Page Tables (EPT)
  - Hardware to translate guest to host physical address
  - Previously done in VMM software
  - Removes #1 cause of exits from virtual machine
  - Allows guests full control over page tables

- Virtual Processor ID (VPIID)
  - Reduce frequency of invalidation of TLB entries

Greatly improved Virtualization performance

- Nehalem: Next Generation Intel® Microarchitecture
Extended Page Tables (EPT) Motivation

- A VMM needs to protect physical memory
  - Multiple Guest OSs share the same physical memory
  - Protections are implemented through page-table virtualization
- Page table virtualization accounts for a significant portion of virtualization overheads
  - VM Exits / Entries
- The goal of EPT is to reduce these overheads

Guest page table changes cause exits into the VMM

Guest OS

VMM

VMM maintains the active page table, which is used by the CPU
EPT Solution

• **Intel® 64 Page Tables**
  - Map Guest Linear Address to Guest Physical Address
  - Can be read and written by the guest OS

• **New EPT Page Tables under VMM Control**
  - Map Guest Physical Address to Host Physical Address
  - Referenced by new EPT base pointer

• **No VM Exits due to Page Faults, INVLPG or CR3 accesses**
Extending Performance and Energy Efficiency

- **SSE4.2 Instruction Set Architecture (ISA) Leadership in 2008**

**Accelerated String and Text Processing**
- Faster XML parsing
- Faster search and pattern matching
- Novel parallel data matching and comparison operations

**Accelerated Searching & Pattern Recognition of Large Data Sets**
- Improved performance for Genome Mining, Handwriting recognition
- Fast Hamming distance / Population count

**New Communications Capabilities**
- Hardware based CRC instruction
- Accelerated Network attached storage
- Improved power efficiency for Software I-SCSI, RDMA, and SCTP

**SSE4 (45nm CPUs)**

**SSE4.1 (Penryn Core)**
- STTNI

**SSE4.2 (Nehalem Core)**
- ATA

**ATA (Application Targeted Accelerators)**

**STTNI**
- e.g. XML acceleration

**POPCNT**
- e.g. Genome Mining

**CRC32**
- e.g. iSCSI Application

**Hardware based CRC instruction**
- Accelerated Network attached storage
- Improved power efficiency for Software I-SCSI, RDMA, and SCTP

**What should the applications, OS and VMM vendors do?:**

*Understand the benefits & take advantage of new instructions in 2008.*

*Provide us feedback on instructions ISV would like to see for next generation of applications*
STTNI - STring & Text New Instructions

Operates on strings of bytes or words (16b)

Equal Each Instruction
True for each character in Src2 if same position in Src1 is equal
Src1: Test\tday
Src2: tad tseT
Mask: 01101111

Equal Any Instruction
True for each character in Src2 if any character in Src1 matches
Src1: Example\n
Src2: atad tsT
Mask: 10100000

Ranges Instruction
True if a character in Src2 is in at least one of up to 8 ranges in Src1
Src1: AZ’0’9zzz
Src2: taD tseT
Mask: 00100001

Equal Ordered Instruction
Finds the start of a substring (Src1) within another string (Src2)
Src1: ABCA0XYZ
Src2: SOBACBAB
Mask: 0000010

Projected 3.8x kernel speedup on XML parsing & 2.7x savings on instruction cycles
STTNI Model

**EQUAL ANY**

Example:

<table>
<thead>
<tr>
<th>Source1 (XMM)</th>
<th>Source2 (XMM / M128)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F F F F F F F F</td>
<td>F F F F F F F F</td>
</tr>
<tr>
<td>T T F F F F F F</td>
<td>T T F F F F F F</td>
</tr>
<tr>
<td>F F F F F F F F</td>
<td>F F F F F F F F</td>
</tr>
<tr>
<td>T T T T F F F F</td>
<td>T T T T F F F F</td>
</tr>
<tr>
<td>F F F F F F F F</td>
<td>F F F F F F F F</td>
</tr>
<tr>
<td>T T T T T T T T</td>
<td>T T T T T T T T</td>
</tr>
</tbody>
</table>

IntRes1: 1 0 1 0 0 0 0

Check each bit in the diagonal

**EQUAL EACH**

<table>
<thead>
<tr>
<th>Source2 (XMM / M128)</th>
<th>Source1 (XMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T T T T T T T T</td>
<td>T T T T T T T T</td>
</tr>
<tr>
<td>T T T T T T T T</td>
<td>T T T T T T T T</td>
</tr>
<tr>
<td>T T T T T T T T</td>
<td>T T T T T T T T</td>
</tr>
<tr>
<td>T T T T T T T T</td>
<td>T T T T T T T T</td>
</tr>
</tbody>
</table>

IntRes1: 0 1 1 0 1 1 1

Check each bit in the diagonal

**RANGES**

<table>
<thead>
<tr>
<th>Source2 (XMM / M128)</th>
<th>Source1 (XMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F F F F T T T T</td>
<td>F F F F T T T T</td>
</tr>
<tr>
<td>F F F F T T T T</td>
<td>F F F F T T T T</td>
</tr>
<tr>
<td>F F F F T T T T</td>
<td>F F F F T T T T</td>
</tr>
<tr>
<td>F F F F T T T T</td>
<td>F F F F T T T T</td>
</tr>
</tbody>
</table>

IntRes1: 1 0 0 1 0 0 1

First Compare does GE, next does LE

**EQUAL ORDERED**

<table>
<thead>
<tr>
<th>Source2 (XMM / M128)</th>
<th>Source1 (XMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S O B A C B A B</td>
<td>S O B A C B A B</td>
</tr>
<tr>
<td>B T T F F T</td>
<td>B T T F F T</td>
</tr>
<tr>
<td>A B T T T F</td>
<td>A B T T T F</td>
</tr>
<tr>
<td>A T T T T T</td>
<td>A T T T T T</td>
</tr>
</tbody>
</table>

IntRes1: 0 0 0 0 0 0 1 0

AND the results along each diagonal
Example Code For strlen()

```c
int sttni_strlen(const char * src)
{
    char eom_vals[32] = {1, 255, 0};
    __asm{
        mov eax, src
        movdqu xmm2, eom_vals
        xor ecx, ecx
        topofloop:
        add eax, ecx
        movdqu xmm1, OWORD PTR [eax]
        pcmpistri xmm2, xmm1, imm8
        jnz topofloop
        endofstring:
        add eax, ecx
        sub eax, src
        ret
    }
    string equ [esp + 4]
    mov ecx,string ; ecx -> string
    test ecx, 3 ; test if string is aligned on 32 bits
    je short main_loop
    str_misaligned:
    ; simple byte loop until string is aligned
    mov al, byte ptr [ecx]
    add ecx, 1
    test al, al
    je short byte_3
    test ecx, 3
    jne short str_misaligned
    add eax, dword ptr 0 ; 5 byte nop to align label below
    align 16 ; should be redundant
    main_loop:
    mov eax, dword ptr [ecx] ; read 4 bytes
    mov edx, 7efefeffh
    add edx, eax
    xor eax, -1
    xor eax, edx
    test eax, 81010100h
    je short main_loop
    ; found zero byte in the loop
    mov eax, [ecx - 4]
    test al, al
    je short byte_2
    test eax, 0ff000000h
    je short byte_3
    jmp short main_loop
    byte_2:
    lea eax, [ecx - 2]
    mov ecx,string
    sub eax, ecx
    ret
    byte_1:
    lea eax, [ecx - 3]
    mov ecx,string
    sub eax, ecx
    ret
    byte_0:
    lea eax, [ecx - 4]
    mov ecx,string
    sub eax, ecx
    ret
    strlen endp
}
```

Current Code: Minimum of 11 instructions; Inner loop processes 4 bytes with 8 instructions

STTNI Code: Minimum of 10 instructions; A single inner loop processes 16 bytes with only 4 instructions
ATA - Application Targeted Accelerators

**CRC32**

Accumulates a CRC32 value using the iSCSI polynomial

One register maintains the running CRC value as a software loop iterates over data.
Fixed CRC polynomial = 11EDC6F41h

Replaces complex instruction sequences for CRC in Upper layer data protocols:
- iSCSI, RDMA, SCTP

---

**POPCNT**

POPCNT determines the number of nonzero bits in the source.

POPCNT is useful for speeding up fast matching in data mining workloads including:
- DNA/Genome Matching
- Voice Recognition

ZFlag set if result is zero. All other flags (C,S,O,A,P) reset

---

Enables enterprise class data assurance with high data rates in networked storage in any user environment.
CRC32 Preliminary Performance

CRC32 optimized Code

crc32c_sse42_optimized_version(uint32 crc, unsigned char const *p, size_t len)
{
    // Assuming len is a multiple of 0x10
    asm("pusha");
    asm("mov %0, %eax" :: "m" (crc));
    asm("mov %0, %ebx" :: "m" (p));
    asm("mov %0, %ecx" :: "m" (len));
    asm("1:");
    // Processing four byte at a time: Unrolled four times:
    asm("crc32 %eax, 0x0(%ebx)");
    asm("crc32 %eax, 0x4(%ebx)");
    asm("crc32 %eax, 0x8(%ebx)");
    asm("crc32 %eax, 0xc(%ebx)");
    asm("add $0x10, %ebx");
    asm("sub $0x10, %ecx");
    asm("jecxz 2f");
    asm("jmp 1b");
    asm("2:");
    asm("mov %eax, %0" : "=m" (crc));
    asm("popa");
    return crc;
}

Preliminary tests involved Kernel code implementing CRC algorithms commonly used by iSCSI drivers.

- 32-bit and 64-bit versions of the Kernel under test
- 32-bit version processes 4 bytes of data using 1 CRC32 instruction
- 64-bit version processes 8 bytes of data using 1 CRC32 instruction
- Input strings of sizes 48 bytes and 4KB used for the test

<table>
<thead>
<tr>
<th></th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Data Size = 48 bytes</td>
<td>6.53 X</td>
<td>9.85 X</td>
</tr>
<tr>
<td>Input Data Size = 4 KB</td>
<td>9.3 X</td>
<td>18.63 X</td>
</tr>
</tbody>
</table>

Preliminary Results show CRC32 instruction outperforming the fastest CRC32C software algorithm by a big margin.
Tools Support of New Instructions

- Intel® C++ Compiler 10.x supports the new instructions
  - SSE4.2 supported via intrinsics
  - Inline assembly supported on both IA-32 and Intel64 targets
  - Necessary to include required header files in order to access intrinsics
    - `<tmmintrin.h>` for Supplemental SSE3
    - `<smmintrin.h>` for SSE4.1
    - `<nmmintrin.h>` for SSE4.2

- Intel Library Support
  - XML Parser Library using string instructions will beta Spring ’08 and release product in Fall ’08
  - IPP is investigating possible usages of new instructions

- Microsoft* Visual Studio 2008 VC++
  - SSE4.2 supported via intrinsics
  - Inline assembly supported on IA-32 only
  - Necessary to include required header files in order to access intrinsics
    - `<tmmintrin.h>` for Supplemental SSE3
    - `<smmintrin.h>` for SSE4.1
    - `<nmmintrin.h>` for SSE4.2
  - VC++ 2008 tools masm, msdis, and debuggers recognize the new instructions
Software Optimization Guidelines

• Most optimizations for Intel® Core™ microarchitecture still hold
• Examples of new optimization guidelines:
  - 16-byte unaligned loads/stores
  - Enhanced macrofusion rules
  - NUMA optimizations
• Nehalem SW Optimization Guide will be published
• Intel® C++ Compiler will support settings for Nehalem optimizations
Summary

• Nehalem - The Intel® 45 nm Tock Processor
• Designed for
  - *Power Efficiency*
  - *Scalability*
  - *Performance*
• Enhanced Processor Core
• Brand New Platform Architecture
• Extending ISA Leadership
Additional sources of information on this topic:

- DPTS003-High End Desktop Platform Overview for the Next Generation Intel® Microarchitecture (Nehalem) Processor
  - April 3 11:10-12:00; Room 3C+3D, 3rd floor

- NGMS002-Upcoming Intel® 64 Instruction Set Architecture Extensions
  - April 3 16:00 – 17:50; Auditorium, 3rd floor

- NGMC001-Next Generation Intel® Microarchitecture (Nehalem) and New Instruction Extensions - Chalk Talk
  - April 3 17:50 – 18:30; Auditorium, 3rd floor

- Demos in the Advance Technology Zone on the 3rd floor –

Session Presentations - PDFs

The PDF of this Session presentation is available from our IDF Content Catalog:

https://intel.wingatweb.com/SHchina/catalog/controller/catalog

These can also be found from links on www.intel.com/idf
Please Fill out the Session Evaluation Form

Put in your lucky draw coupon to win the prize at the end of the track!

You must be present to win!

Thank You for your input, we use it to improve future Intel Developer Forum events
Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.

- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Nehalem, Harpertown, Tylersburg, Merom, Dothan, Penryn, Bloomfield and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel’s internal code names is at the sole risk of the user
- Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.
- Intel, Intel Inside, Xeon, Pentium, Core and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
- Copyright © 2008 Intel Corporation.
Risk Factors

This presentation contains forward-looking statements that involve a number of risks and uncertainties. These statements do not reflect the potential impact of any mergers, acquisitions, divestitures, investments or other similar transactions that may be completed in the future, with the exception of the Numonyx transaction. Our forward-looking statements for 2008 reflect the expectation that the Numonyx transaction will close during the first quarter. The information presented is accurate only as of today’s date and will not be updated. In addition to any factors discussed in the presentation, the important factors that could cause actual results to differ materially include the following: Factors that could cause demand to be different from Intel's expectations include changes in business and economic conditions, including conditions in the credit market that could affect consumer confidence; customer acceptance of Intel’s and competitors’ products; changes in customer order patterns, including order cancellations; and changes in the level of inventory at customers. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Additionally, Intel is in the process of transitioning to its next generation of products on 45 nm process technology, and there could be execution issues associated with these changes, including product defects and errata along with lower than anticipated manufacturing yields. Revenue and the gross margin percentage are affected by the timing of new Intel product introductions and the demand for and market acceptance of Intel’s products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; Intel’s ability to respond quickly to technological developments and to incorporate new features into its products; and the availability of sufficient components from suppliers to meet demand. The gross margin percentage could vary significantly from expectations based on changes in revenue levels; product mix and pricing; capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; excess or obsolete inventory; manufacturing yields; changes in unit costs; impairments of long-lived assets, including manufacturing, assembly/test and intangible assets; and the timing and execution of the manufacturing ramp and associated costs, including startup costs. Expenses, particularly certain marketing and compensation expenses, vary depending on the level of demand for Intel's products, the level of revenue and profits, and impairments of long-lived assets. Intel is in the midst of a structure and efficiency program that is resulting in several actions that could have an impact on expected expense levels and gross margin. We expect to complete the divestiture of our NOR flash memory assets to Numonyx. A delay or failure of the transaction to close, or a change in the financial performance of the contributed businesses could have a negative impact on our financial statements. Intel’s equity proportion of the new company’s results will be reflected on its financial statements below operating income and with a one quarter lag. Intel’s results could be affected by the amount, type, and valuation of share-based awards granted as well as the amount of awards cancelled due to employee turnover and the timing of award exercises by employees. Intel's results could be impacted by adverse economic, social, political and physical/infrastructure conditions in the countries in which Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the report on Form 10-K for the fiscal year ended December 29, 2007.