“This is the book Intel should have written, but now they don’t have to.”
—Bob Colwell

The Unabridged Pentium® 4

IA32 Processor Genealogy

MindShare, Inc.
Tom Shanley

PC System Architecture Series
MindShare recognizes and addresses your company’s technical training issues with:

- Scalable cost training
- Customizable training options
- Just-in-time training
- Overview and advanced topic courses
- Reducing time away from work
- Training delivered effectively globally
- Concurrently delivered multiple-site training
- Training in a classroom, at your cubicle or home office

MindShare training courses expand your technical skillset:

- PCI Express 2.0
- Intel Core 2 Processor Architecture
- AMD Opteron Processor Architecture
- Intel 64 and IA-32 Software Architecture
- Intel PC and Chipset Architecture
- PC Virtualization
- USB 2.0
- Wireless USB
- Serial ATA (SATA)
- Serial Attached SCSI (SAS)
- DDR2/DDR3 DRAM Technology
- PC BIOS Firmware
- High-Speed Design
- Windows Internals and Drivers
- Linux Fundamentals
- ... and many more.

All courses can be customized to meet your group’s needs. Detailed course outlines can be found at www.mindshare.com
Engage MindShare

Have knowledge that you want to bring to life? MindShare will work with you to “Bring Your Knowledge to Life.” Engage us to transform your knowledge and design courses that can be delivered in classroom or virtual classroom settings, create online eLearning modules, or publish a book that you author.

We are proud to be the preferred training provider at an extensive list of clients that include:

- ADAPTEC
- AMD
- AGILENT TECHNOLOGIES
- APPLE
- BROADCOM
- CADENCE
- CRAY
- CISCO
- DELL
- FREESCALE
- GENERAL DYNAMICS
- HP
- IBM
- KODAK
- LSI LOGIC
- MOTOROLA
- MICROSOFT
- NASA
- NATIONAL SEMICONDUCTOR
- NETAPP
- NOKIA
- NVIDIA
- PLX TECHNOLOGY
- QLOGIC
- SIEMENS
- SUN MICROSYSTEMS
- SYNOPSYS
- TI
- UNISYS
## At-a-Glance

### Table of Contents

**Part 1, Introduction**, introduces the processor’s role in the system. It consist of the following chapter:

- “Overview of the Processor Role” on page 9.

**Part 2, Single-/Multi-Task OS Background**, introduces the goals of single-task and multi-task OSs and consists of the following chapters:

- “Single-Task OS and Application” on page 23.
- “Definition of Multitasking” on page 27.
- “Multitasking Problems” on page 31.

**Part 3, The 386**, provides a detailed description of the 386 processor, the baseline ancestor of the IA32 processor family. It consists of the following chapters:

- “386 Real Mode Operation” on page 39.
- “Protected Mode Introduction” on page 103.
- “Intro to Segmentation in Protected Mode” on page 109.
- “Code Segments” on page 133.
- “Data and Stack Segments” on page 157.
- “Creating a Task” on page 171.
- “Mechanics of a Task Switch” on page 191.
- “386 Demand Mode Paging” on page 209.
- “The Flat Model” on page 247.
- “Interrupts and Exceptions” on page 251.
- “Virtual 8086 Mode” on page 329.
- “The Debug Registers” on page 375.

**Part 4, 486**, provides an introduction to the 486 processor’s hardware design. The 486 was the first IA32 processor to incorporate a cache and all subsequent IA32 processors include on-die caches. For this reason, a cache primer is provided. Finally, a detailed description of the 486 software enhancements is provided. This part consists of the following chapters:
Part 5, Pentium®, provides an overview of the Pentium® processor’s hardware design and a detailed description of its software enhancements. It consists of the following chapters:

- “Pentium® Hardware Overview” on page 463.
- “Pentium® Software Enhancements” on page 489.

Part 6, Intro to the P6 Core and FSB, provides a brief introduction to the P6 roadmap, the P6 processor core, and the P6 FSB.

It consists of the following chapters:

- “P6 Road Map” on page 539.
- “P6 Hardware Overview” on page 543.

Part 7, Pentium® Pro Software Enhancements, provides a detailed description of the software enhancements introduced in the Pentium® Pro processor. It also provides a detailed description of the Microcode Update feature which was introduced on the Pentium® Pro processor. This part consists of the following chapters:

- “Pentium® Pro Software Enhancements” on page 553.
- “MicroCode Update Feature” on page 631.

Part 8, Pentium® II, provides an overview of the Pentium® II’s hardware design, a detailed description of its power management features, software enhancements, and a description of the Pentium® II Xeon processor (the very first Xeon). This part consists of the following chapters:

- “Pentium® II Hardware Overview” on page 657.
- “Pentium® II Power Management Features” on page 683.
- “Pentium® II Software Enhancements” on page 695.
- “Pentium® II Xeon Features” on page 719.

Part 9, Pentium® III, provides an overview of the Pentium® III’s hardware design, a detailed description of its software enhancements, and a description of the Pentium® III Xeon processor. This part consists of the following chapters:

- “Pentium® III Hardware Overview” on page 741.
- “Pentium® III Software Enhancements” on page 757.
- “Pentium® III Xeon Features” on page 795.
Part 10, Pentium® 4, provides a detailed description of the hardware design and software enhancements encompassed in the Pentium® 4 processor family. It consists of the following chapters:

- “Pentium® 4 Road Map” on page 813.
- “Pentium® 4 System Overview” on page 823.
- “Pentium® 4 Processor Overview” on page 835.
- “Pentium® 4 PowerOn Configuration” on page 855.
- “Pentium® 4 Processor Startup” on page 875.
- “Pentium® 4 Core Description” on page 897.
- “Hyper-Threading” on page 965.
- “The Pentium® 4 Caches” on page 1009.
- “Pentium® 4 Handling of Loads and Stores” on page 1061.
- “The Pentium® 4 Prescott” on page 1091.
- “Pentium® 4 FSB Electrical Characteristics” on page 1115.
- “Intro to the Pentium® 4 FSB” on page 1137.
- “Pentium® 4 CPU Arbitration” on page 1149.
- “Pentium® 4 Priority Agent Arbitration” on page 1165.
- “Pentium® 4 Locked Transaction Series” on page 1177.
- “Pentium® 4 FSB Blocking” on page 1189.
- “Pentium® 4 FSB Request Phase” on page 1201.
- “Pentium® 4 FSB Snoop Phase” on page 1225.
- “Pentium® 4 FSB Response and Data Phases” on page 1241.
- “Pentium® 4 FSB Transaction Deferral” on page 1277.
- “Pentium® 4 FSB IO Transactions” on page 1295.
- “Pentium® 4 FSB Central Agent Transactions” on page 1301.
- “Pentium® 4 FSB Miscellaneous Signals” on page 1313.
- “Pentium® 4 Software Enhancements” on page 1321.
- “Pentium® 4 Xeon Features” on page 1421.

Part 11, Pentium® M, describes the hardware and software characteristics of the Pentium® M processor and consists of the following chapter:

- “Pentium® M Processor” on page 1425.

Part 12, Additional Topics, provides a detailed description of processor identification, System Management Mode, and the IO and Local APICs. It consists of the following chapters:

- “CPU Identification” on page 1443.
- “System Management Mode (SMM)” on page 1463.
- “The Local and IO APICs” on page 1497.
About This Book
The IA32 Architecture Specification ................................................................. 1
The Pentium® 4 Is the Sum of Its Ancestors .................................................... 1
The CD ............................................................................................................. 2
The MindShare Architecture Series ............................................................... 2
Cautionary Note .............................................................................................. 3
The Specification Is the Final Word ............................................................... 4
Documentation Conventions ........................................................................... 4
  Hexadecimal Notation ................................................................................... 4
  Binary Notation ............................................................................................ 4
  Decimal Notation .......................................................................................... 4
  Bits Versus Bytes Notation ......................................................................... 5
  Bit Fields (Logical Groups of Bits or Signals) ............................................ 5
  Signal Names ............................................................................................... 5
Visit Our Web Site .......................................................................................... 6
We Want Your Feedback .................................................................................. 6

Part 1: Introduction

Chapter 1: Overview of the Processor Role
The IA32 Specification ...................................................................................... 9
IA32 Processors ............................................................................................... 10
IA32 Instructions vs. µops ............................................................................... 10
Processor = Instruction Fetch/Decode/Execute Engine ............................... 10
Some Instructions Result in FSB Transactions ........................................... 11
  Many Instructions Do Not Require FSB Transactions ............................ 11
  Instructions That Do Require FSB Transactions .................................... 12
    IO Read and Write .................................................................................... 12
    IO Read Instruction .................................................................................. 12
    IO Write Instruction ............................................................................... 12
  Memory Data Read ..................................................................................... 12
  Memory Data Write .................................................................................... 13
  Memory Instruction Read .......................................................................... 13
The Processor’s Role in Today’s Systems ...................................................... 14
  Processor Activities at Startup ................................................................. 14
  Processor Activities During Run-Time .................................................... 15
    Load and Run Application Programs .................................................... 15
    Application Program Calls the OS ......................................................... 15
    Handling External Hardware Interrupts ............................................... 15
    Calling a Device Driver .......................................................................... 15
    Handling Software Exceptions ............................................................. 15
Contents

System Overview ..................................................................................................................... 16
  Pentium® 4 Processor........................................................................................................ 16
  Memory Control Hub (MCH) ......................................................................................... 16
  IO Control Hub (ICH) ...................................................................................................... 16
  Super IO (SIO) Chip........................................................................................................... 17
  DDR RAM........................................................................................................................... 17
  IDE RAID Controller ........................................................................................................ 18
  USB 2.0 Controller........................................................................................................... 18
  Five PCI Card Slots......................................................................................................... 18

Part 2: Single-/MultiTask OS Background

Chapter 2: Single-Task OS and Application
  Operating System Overview.................................................................................................. 23
    Command Line Interpreter (CLI) .................................................................................... 24
    Program Loader ............................................................................................................... 24
    OS Services..................................................................................................................... 25
  Direct IO Access .................................................................................................................... 25
  Application Program Memory Usage................................................................................... 26
  Task Initiation, Execution and Termination....................................................................... 26

Chapter 3: Definition of Multitasking
  Concept.................................................................................................................................. 27
  An Example—Timeslicing...................................................................................................... 28
  Another Example—Awaiting an Event................................................................................ 28
    Task Issues Call to OS for Disk Read .............................................................................. 28
    OSSuspends Task.............................................................................................................. 29
    OS Initiates Disk Read..................................................................................................... 29
    OS Makes Entry in Event Queue ..................................................................................... 29
    OS Starts or Resumes Another Task............................................................................... 29
    Disk-Generated Interrupt Causes Jump to OS .............................................................. 30
    Task Queue Checked........................................................................................................ 30
    OS Resumes Task ............................................................................................................ 30

Chapter 4: Multitasking Problems
  OS Protects Territorial Integrity............................................................................................ 31
  Stay in Your Own Memory Area......................................................................................... 32
  IO Port Anarchy.................................................................................................................... 32
  Unauthorized Use of OS’s Tools............................................................................................ 33
  No Interrupts, Please! ......................................................................................................... 34
  BIOS Calls .......................................................................................................................... 35
Chapter 5: 386 Real Mode Operation

Special Note ................................................................................................................... 39
An Overview of the 386 Internal Architecture ................................................................. 39
An Overview of the 386DX FSB ....................................................................................... 41
  Address Bus Selects Dword.............................................................................................. 42
  Byte Enables Select Location(s) in Dword.................................................................... 42
  Misaligned Transfers Affect Performance.................................................................. 43
  Alignment Is Important!............................................................................................... 43
The 386 Register Set......................................................................................................... 45
  Control Registers.......................................................................................................... 45
    CR0............................................................................................................................ 45
    CR1............................................................................................................................ 47
    CR2............................................................................................................................ 47
    CR3............................................................................................................................ 47
  EFlags Register ........................................................................................................... 48
  General Purpose Registers (GPRs) .............................................................................. 52
    Introduction................................................................................................................. 52
    EAX, EBX, ECX and EDX Registers ................................................................... 52
    EBP Register............................................................................................................. 53
    Index Registers ........................................................................................................... 53
  Segment Registers ....................................................................................................... 55
    Real Mode Usage ....................................................................................................... 55
    Protected Mode Usage ............................................................................................ 55
    DS, ES, FS and GS .................................................................................................... 55
    CS .............................................................................................................................. 55
    SS .............................................................................................................................. 55
  Extended Instruction Pointer (EIP) Register ................................................................ 57
  Task Register ............................................................................................................... 57
    What Is a TSS?............................................................................................................ 57
    The Purpose of the Task Register ........................................................................... 58
  GDTR and LDTR............................................................................................................ 60
  IDTR............................................................................................................................ 62
    Hardware Interrupts.................................................................................................... 62
    Software Exceptions and Interrupts.......................................................................... 62
    IDTR Points To the Interrupt Table.......................................................................... 63
  The Debug Registers................................................................................................... 64
  Test Registers.............................................................................................................. 66
386 Power-Up State ......................................................................................................... 66
### Contents

Initial Memory Reads........................................................................................................... 69  
IO Port Addressing ................................................................................................................ 70  
Memory Addressing................................................................................................................ 71  
  General........................................................................................................................ ..... 71  
  Accessing the Code Segment............................................................................................ 73  
  Accessing the Stack Segment............................................................................................ 76  
    Introduction..................................................................................................................... 76  
    Pushing Data Onto the Stack ...................................................................................... 77  
    Popping Data From the Stack ................................................................................... 78  
    Processor Stack Usage............................................................................................... 80  
  Accessing the DS Data Segment...................................................................................... 81  
  Accessing the ES/FS/GS Data Segments ....................................................................... 84  
    An Example.................................................................................................................. 84  
  Accessing Extended Memory in Real Mode ................................................................. 87  
    Big Real Mode......................................................................................................... 90  
Real Mode Instructions and Registers................................................................................. 91  
  Registers Accessible in Real Mode ............................................................................. 91  
  Registers Inaccessible in Real Mode............................................................................ 91  
  Instructions Usable in Real Mode............................................................................... 91  
  Instructions Unusable in Real Mode........................................................................... 92  
Real Mode Interrupt/Exception Handling .......................................................................... 92  
Protection in Real Mode ..................................................................................................... 101  

### Chapter 6: Protected Mode Introduction

General ..................................................................................................................................... 103  
Memory Protection ................................................................................................................ 104  
  Segmentation ................................................................................................................ 104  
  Virtual Memory Paging ................................................................................................. 105  
IO Protection ......................................................................................................................... 105  
Privilege Levels ...................................................................................................................... 106  
Virtual 8086 Mode.............................................................................................................. .... 106  
Task Switching ....................................................................................................................... 106  
Interrupt Handling .............................................................................................................. 107  
  Real Mode Interrupt Handling .................................................................................... 107  
  Protected Mode Interrupt Handling ........................................................................... 107  

### Chapter 7: Intro to Segmentation in Protected Mode

Special Note ......................................................................................................................... 109  
Real Mode Limitations.......................................................................................................... 110  
Segment Descriptor Describes a Memory Area in Detail................................................. 110  
Segment Register—Selects Descriptor Table and Entry .............................................. 112
Contents

Introduction to the Descriptor Tables ................................................................. 114
  Segment Descriptors Reside in Memory .................................................. 114
  Global Descriptor Table (GDT) ............................................................ 115
    GDT Description .............................................................................. 115
    Setting the GDT Base Address and Size .......................................... 115
  Local Descriptor Tables (LDTs) ......................................................... 116

General Segment Descriptor Format .............................................................. 121
  Granularity Bit ................................................................................. 122
  Segment Base Address Field .............................................................. 123
  Segment Size Field .......................................................................... 123
  Default/Big Bit .................................................................................. 123
    In a Code Segment, It’s the Descriptor’s “Default” Bit ..................... 123
    In a Stack Segment, It’s the Descriptor’s “Big” Bit ......................... 124
  Segment Type Field ........................................................................... 125
    Introduction to the Type Field ....................................................... 125
    Non-System Segment Types .......................................................... 125
  Segment Present Bit ........................................................................... 129
  Descriptor Privilege Level (DPL) Field ............................................... 130
  System Bit ....................................................................................... 130
  Available Bit .................................................................................... 131

Chapter 8: Code Segments

Selecting the Code Segment to Execute ......................................................... 133
Code Segment Descriptor Format .............................................................. 134
Accessing the Code Segment ....................................................................... 137
Privilege Checking ...................................................................................... 139
  General .............................................................................................. 139
  Some Definitions ............................................................................... 139
    Definition of a Task ....................................................................... 139
    Definition of a Procedure ............................................................. 140
    CPL Definition .............................................................................. 140
    DPL Definition ............................................................................. 140
    Conforming and Non-Conforming Code Segments ....................... 141
    RPL Definition ............................................................................. 141
Calling a Procedure in the Current Task ........................................................ 142
Call Gate ............................................................................................... 143
  The Problem .................................................................................... 143
  The Solution—Different Gateways .................................................... 143
  Call Gate Example ......................................................................... 145
  Execution Begins ............................................................................. 145
  Call Gate Descriptor Read ................................................................ 146
  Call Gate Contains Code Segment Selector ...................................... 148
Contents

Code Segment Descriptor Read ................................................................. 148
The Big Picture ......................................................................................... 150
The Call Gate Privilege Check ................................................................. 151
Privilege Check for a Call through a Call Gate ........................................ 151
Privilege Check for a Jump through a Call Gate ...................................... 152
Automatic Stack Switch ........................................................................... 153

Chapter 9: Data and Stack Segments
A Note Regarding Stack Segments ............................................................ 157
The Data Segments ................................................................................... 158
  Selecting and Accessing a Data Segment ................................................. 158
  Data Segment Privilege Check ............................................................... 159
Selecting and Accessing a Stack Segment .................................................. 161
  Introduction ............................................................................................ 161
  Expand-Up Stack ..................................................................................... 162
  Expand-Down Stack ............................................................................... 164
    The Problem .......................................................................................... 164
    Expand-Down Stack Description .......................................................... 165
    An Example ........................................................................................... 165
    Another Example .................................................................................. 166
Stack Segment Privilege Check ................................................................. 170

Chapter 10: Creating a Task
What Is a Task? ....................................................................................... 171
Basics of Task Creation and Startup ....................................................... 171
  Load All or Part of the Task into Memory ................................................ 172
  Create a TSS and a TSS Descriptor for the Task ..................................... 172
  Trigger the Timeslice Timer .................................................................... 172
  Scheduler Causes a Task Switch ............................................................... 172
  Interrupt on Timer Expiration ................................................................. 173
TSS Structure .......................................................................................... 173
  General ..................................................................................................... 173
  IO Port Access Protection ........................................................................ 175
    IO Protection in Real Mode ................................................................. 175
    Definition of IO Privilege Level (IOPL) ................................................ 176
    IO Permission Check in Protected Mode ............................................. 177
    IO Permission Check in VM86 Mode ................................................... 178
  IO Permission Bit Map Offset Field ....................................................... 178
  Interrupt Redirection Bit Map ................................................................. 180
  OS-Specific Data Structures ................................................................. 180
  Debug Trap Bit (T) ................................................................................. 181
## Contents

LDT Selector Field ................................................................. 181  
Segment Register Fields .......................................................... 181  
General Register Fields .......................................................... 181  
Extended Stack Pointer (ESP) Register Field .......................... 182  
Extended Flags (EFlags) Register Field ................................. 182  
Extended Instruction Pointer (EIP) Register Field ................... 182  
Control Register 3 (CR3) Field ................................................. 182  
Privilege Level 0 - 2 Stack Definition Fields ............................. 183  
Link Field (to Old TSS Selector) ................................................. 184  

**TSS Descriptor** ........................................................................................................................................ 185  

**How the OS Starts a Task** .................................................................................................................. 186  

**What Happens When a Task Starts** ..................................................................................................... 187  

**Use of the LTR and STR Instructions** ................................................................................................ 187  

**General** ............................................................................................................................................... 187  

**The STR Instruction** ............................................................................................................................ 188  

**The LTR Instruction** ............................................................................................................................ 188  

---

### Chapter 11: Mechanics of a Task Switch

**Events that Initiate a Task Switch** ........................................................................................................ 191  

**Switch Via a TSS Descriptor** .............................................................................................................. 194  

**Task Gate Descriptor** ......................................................................................................................... 194  

  - Task Gate Selected by a Far Call/Jump ............................................................................................ 194  
  - Task Gate Selected by a Hardware Interrupt or a Software Exception ........................................ 195  
  - Task Gate Selected by an INT Instruction .................................................................................. 195  

**Task Switch Details** ............................................................................................................................. 196  

  - Switch Due To an Interrupt or Exception .................................................................................... 196  
  - Switch as a Result of a Far Call .................................................................................................... 197  
  - Switch as the Result of a Far Jump .............................................................................................. 197  
  - Switch Due to a BOUND/INT/INTO/INT3 Instruction .............................................................. 198  
  - Switch Due to Execution of an IRET .......................................................................................... 198  

**Linked Tasks** ......................................................................................................................................... 201  

**Linkage Modification** .......................................................................................................................... 203  

**The Busy Bit** ....................................................................................................................................... 204  

**Address Mapping** ............................................................................................................................... 205  

  - The Linear vs. the Physical Memory Address .............................................................................. 205  
  - The GDT Purpose and Location .................................................................................................... 206  
  - The LDT Purpose and Location .................................................................................................... 206  
  - Paging-Related Issues ................................................................................................................... 207  
    - Background .............................................................................................................................. 207  
    - Each Task Can Have Different Linear-to-Physical Mapping ................................................ 207  
    - TSS Mapping Must Remain the Same for All Tasks ............................................................. 207  
    - Placement of a TSS Within a Page(s) .................................................................................... 208  

---

xvii
Chapter 12: 386 Demand Mode Paging

Problem—Loading Entire Task into Memory is Wasteful............................................. 209
Solution—Load Part and Keep Remainder on Disk ....................................................... 210
  Load on Demand.............................................................................................................. 210
  Track Usage....................................................................................................................... 211
  Capabilities Required ...................................................................................................... 211

Problem—Running Two (or more) DOS Programs ....................................................... 211
Solution—Redirect Memory Accesses to Separate Memory Areas.............................. 212
  Global Solution—Map Linear Address to Disk Address or to a
  Different Physical Memory Address.............................................................................. 213

The Paging Unit Is the Translator....................................................................................... 214
  Linear Memory Space Is Divided into $2^{20}$ 4KB Pages ................................................ 214
  Physical Memory Space Is Divided into $2^{20}$ 4KB Pages ............................................. 215
  Mass Storage Space Is Divided into 4KB Pages........................................................... 215
  The Paging Unit Uses Directories to Remap the Address........................................ 215

Three Possible Page Lookup Methods............................................................................... 215
  First Method: Sequential Scan through a Large Table................................................ 216
  Second Method: Index into a Large Table.................................................................... 216
  Third Method: Index into a Selected Small Table ....................................................... 217

IA32 Page Lookup Method................................................................................................... 219

Enabling Paging..................................................................................................................... 219

Page Directory and Page Tables.......................................................................................... 220

Finding the Location of a Physical Page............................................................................... 222
  Find the Page Table First................................................................................................. 222
  When the Target Page Table Is in Memory................................................................. 223
  When the Target Page Table Isn’t in Memory.............................................................. 225
  Find the Page Using an Entry in a Page Table............................................................ 229
  When the Target Page Is in Memory ..................................................................... 229
  When the Target Page Isn’t in Memory................................................................. 230

Eliminating the Directory Lookup...................................................................................... 234
  The 386 TLB ...................................................................................................................... 234
  TLB Maintenance ............................................................................................................. 235
  The TLBs Are Cleared on a Task Switch or a Page Directory Change ....................... 236
  Updating a Single Page Table Entry ...................................................................... 236

Checking Page Access Permission...................................................................................... 237
  The Privilege Check....................................................................................................... 237
  Segment Privilege Check Takes Precedence Over Page Check .................................... 237
  U/S Bit in Page Directory and Page Table Entry Is Checked ...................................... 238
  Accesses with Special Privilege ............................................................................... 238
  The Read/Write Check ................................................................................................. 238
Contents

Page Faults ........................................................................................................................................ 239
  Page Fault Causes .......................................................................................................................... 239
  Second Page Fault while in the Page Fault Handler ................................................................. 240
  A Page Fault During a Task Switch ............................................................................................. 240
  A Page Fault while Changing to a Different Stack ................................................................. 241
  Page Fault Error Code ................................................................................................................ 241
Usage of the Dirty and Accessed Bits ......................................................................................... 243
Demand Mode Paging Evolution ................................................................................................. 244

Chapter 13: The Flat Model
Segments Complicate Things ...................................................................................................... 247
Paging Can Do It All .................................................................................................................... 247
Eliminating Segmentation ........................................................................................................... 248
The Privilege Check .................................................................................................................... 249
The Read/Write Check .................................................................................................................. 249
Each Task (including the OS) Has Its Own TSS ....................................................................... 249
  Switch to an Application Task ................................................................................................. 249
  Switch to an OS Kernel Task ................................................................................................. 250

Chapter 14: Interrupts and Exceptions
Special Note .................................................................................................................................... 251
General .......................................................................................................................................... 252
Hardware Interrupts .................................................................................................................... 252
  Maskable Interrupt Requests ................................................................................................. 253
  Maskable Interrupt Servicing ............................................................................................... 254
  Automatic Actions .................................................................................................................. 254
  Actions Performed by the Software Handler .......................................................................... 255
  PC-Compatible Vector Assignment ....................................................................................... 255
Non-Maskable Interrupt Requests ............................................................................................ 259
Software-Generated Exceptions .................................................................................................. 260
  General ..................................................................................................................................... 260
  Faults, Traps, and Aborts ....................................................................................................... 260
  Instruction Restart .................................................................................................................. 265
  Software Interrupt Instructions .............................................................................................. 266
Interrupt/Exception Priority ......................................................................................................... 266
Real Mode Interrupt/Exception Handling .................................................................................... 270
  Real Mode Interrupt Descriptor Table (IDT) Structure .......................................................... 270
Real Mode Interrupt/Exception Handling .................................................................................. 271
Protected Mode Interrupt/Exception Handling ......................................................................... 272
  General ................................................................................................................................... 272
  Protected Mode Interrupt Descriptor Table (IDT) Structure .................................................. 272
## Contents

Handlers Can Only Be Entered From Program of Equal or Lesser Privilege ........ 275  
Lower-Privilege Programs Could Call More Privileged Programs ...................... 275  
Gates Prevent Anarchy ........................................................................................ 275  
Interrupts/Exceptions Bypass the Gate Privilege Check ..................................... 276  
Interrupt Gates ..................................................................................................... 276  
  General .................................................................................................................. 276  
  Actions Taken when an Interrupt Selects an Interrupt Gate .............................. 278  
Trap Gates ............................................................................................................. 281  
Using a Procedure as an Interrupt/Exception Handler ......................................... 282  
  State Save ............................................................................................................. 282  
  Jump to the Handler ............................................................................................. 285  
  Return to the Interrupted Program ..................................................................... 286  
    Returning to the Same Privilege Level ............................................................. 286  
    Returning to a Different Privilege Level ......................................................... 286  
Using a Task as an Interrupt/Exception Handler .................................................. 286  
Interrupt/Exception Handling in VM86 Mode ...................................................... 287  
Exception Error Codes ............................................................................................ 288  
The Resume Flag Prevents Multiple Debug Exceptions ......................................... 291  
Special Case—Interrupts Disabled While Updating SS:ESP ................................. 292  
  The Problem ....................................................................................................... 292  
  The Solution ....................................................................................................... 292  
Detailed Description of the Software Exceptions ................................................. 292  
Divide-by-Zero Exception (0) .................................................................................. 292  
  Processor Introduced In ..................................................................................... 292  
  Exception Class ................................................................................................. 293  
  Description ........................................................................................................ 293  
  Error Code ....................................................................................................... 293  
  Saved Instruction Pointer .................................................................................. 293  
  Processor State .................................................................................................. 293  
Debug Exception (1) ............................................................................................. 293  
  Processor Introduced In ..................................................................................... 293  
  Exception Class ................................................................................................. 293  
  Description ........................................................................................................ 293  
  Error Code ....................................................................................................... 293  
  Saved Instruction Pointer .................................................................................. 294  
  Processor State .................................................................................................. 294  
NMI (2) ..................................................................................................................... 295  
  Processor Introduced In ..................................................................................... 295  
  Exception Class ................................................................................................. 295  
  Error Code ....................................................................................................... 295  
  Saved Instruction Pointer .................................................................................. 295  
  Processor State .................................................................................................. 295
<table>
<thead>
<tr>
<th>Exception Type</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakpoint Exception (3)</td>
<td>295</td>
</tr>
<tr>
<td>Processor Introduced In</td>
<td>295</td>
</tr>
<tr>
<td>Exception Class</td>
<td>295</td>
</tr>
<tr>
<td>Description</td>
<td>296</td>
</tr>
<tr>
<td>Error Code</td>
<td>296</td>
</tr>
<tr>
<td>Saved Instruction Pointer</td>
<td>296</td>
</tr>
<tr>
<td>Processor State</td>
<td>296</td>
</tr>
<tr>
<td>Overflow Exception (4)</td>
<td>296</td>
</tr>
<tr>
<td>Processor Introduced In</td>
<td>296</td>
</tr>
<tr>
<td>Exception Class</td>
<td>297</td>
</tr>
<tr>
<td>Description</td>
<td>297</td>
</tr>
<tr>
<td>Error Code</td>
<td>297</td>
</tr>
<tr>
<td>Saved Instruction Pointer</td>
<td>297</td>
</tr>
<tr>
<td>Processor State</td>
<td>297</td>
</tr>
<tr>
<td>Array Bounds Check Exception (5)</td>
<td>297</td>
</tr>
<tr>
<td>Processor Introduced In</td>
<td>297</td>
</tr>
<tr>
<td>Exception Class</td>
<td>297</td>
</tr>
<tr>
<td>Description</td>
<td>297</td>
</tr>
<tr>
<td>Error Code</td>
<td>298</td>
</tr>
<tr>
<td>Saved Instruction Pointer</td>
<td>298</td>
</tr>
<tr>
<td>Processor State</td>
<td>298</td>
</tr>
<tr>
<td>Invalid OpCode Exception (6)</td>
<td>298</td>
</tr>
<tr>
<td>Exception Class</td>
<td>298</td>
</tr>
<tr>
<td>Description</td>
<td>298</td>
</tr>
<tr>
<td>Error Code</td>
<td>299</td>
</tr>
<tr>
<td>Saved Instruction Pointer</td>
<td>299</td>
</tr>
<tr>
<td>Processor State</td>
<td>299</td>
</tr>
<tr>
<td>Device Not Available Exception (7)</td>
<td>299</td>
</tr>
<tr>
<td>Processor Introduced In</td>
<td>299</td>
</tr>
<tr>
<td>Exception Class</td>
<td>300</td>
</tr>
<tr>
<td>Description</td>
<td>300</td>
</tr>
<tr>
<td>Error Code</td>
<td>301</td>
</tr>
<tr>
<td>Saved Instruction Pointer</td>
<td>301</td>
</tr>
<tr>
<td>Processor State</td>
<td>301</td>
</tr>
<tr>
<td>Double Fault Exception (8)</td>
<td>301</td>
</tr>
<tr>
<td>Processor Introduced In</td>
<td>301</td>
</tr>
<tr>
<td>Exception Class</td>
<td>301</td>
</tr>
<tr>
<td>Description</td>
<td>301</td>
</tr>
<tr>
<td>Shutdown Mode</td>
<td>303</td>
</tr>
<tr>
<td>Error Code</td>
<td>304</td>
</tr>
<tr>
<td>Saved Instruction Pointer</td>
<td>304</td>
</tr>
<tr>
<td>Processor State</td>
<td>304</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Exception Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coprocessor Segment Overrun Exception (9)</td>
<td>304</td>
</tr>
<tr>
<td>Invalid TSS Exception (10)</td>
<td>305</td>
</tr>
<tr>
<td>Segment Not Present Exception (11)</td>
<td>308</td>
</tr>
<tr>
<td>Stack Exception (12)</td>
<td>309</td>
</tr>
<tr>
<td>General Protection (GP) Exception (13)</td>
<td>311</td>
</tr>
<tr>
<td>Page Fault Exception (14)</td>
<td>314</td>
</tr>
</tbody>
</table>
Processor State ........................................................................................................... 316
The More Common Case .................................................................................. 316
Page Fault During a Task Switch .................................................................... 317
Page Fault During a Stack Switch .......................................................................... 317
Vector 15 ...................................................................................................................... 318
FPU Exception (16) ........................................................................................................ 318
Processor Introduced In .................................................................................. 318
Exception Class ......................................................................................................... 318
Description ................................................................................................................. 318
Handling of Masked Errors .................................................................................. 319
Handling of Unmasked Errors ............................................................................... 320
Error Code .................................................................................................................. 321
Saved Instruction Pointer ......................................................................................... 321
Processor State .......................................................................................................... 321
Alignment Check Exception (17) ................................................................................... 321
Processor Introduced In .................................................................................. 321
Exception Class ......................................................................................................... 321
Description ................................................................................................................. 322
Implicit Privilege Level 0 Accesses ........................................................................ 323
Storing GDTR, LDTR, IDTR or TR ......................................................................... 323
FP/MMX/SSE/SSE2 Save and Restore Accesses ..................................................... 324
MOVUPS and MOVUPD Accesses ........................................................................ 324
FSAVE and FRSTOR Accesses ............................................................................... 324
Error Code .................................................................................................................. 324
Saved Instruction Pointer ......................................................................................... 324
Processor State .......................................................................................................... 324
Machine Check Exception (18) ................................................................................... 324
Processor Introduced In .................................................................................. 324
Exception Class ......................................................................................................... 324
Description ................................................................................................................. 324
Error Code .................................................................................................................. 325
Saved Instruction Pointer ......................................................................................... 325
Processor State .......................................................................................................... 326
SIMD Floating-Point Exception (19) ............................................................................ 326
Processor Introduced In .................................................................................. 326
Exception Class ......................................................................................................... 326
Description ................................................................................................................. 326
Exception Error Code ............................................................................................... 328
Saved Instruction Pointer ......................................................................................... 328
Processor State .......................................................................................................... 328
Contents

Chapter 15: Virtual 8086 Mode
A Special Note ........................................................................................................................ 329
DOS Application—Portrait of an Anarchist ..................................................................... 330
Solution—Set a Watchdog on the DOS Application ...................................................... 330
The Virtual Machine Monitor (VMM)............................................................................... 331
Entering or Reentering VM86 Mode .................................................................................. 332
  Task Creation, Startup and Suspension.......................................................................... 332
  Create a TSS................................................................................................................... 332
  Each Task Gets a Timeslice.............................................................................................. 332
  Select DOS Task via a Far Call or a Far Jump............................................................ 333
An Interrupt or Exception Causes an Exit From VM86 Mode....................................... 333
  General........................................................................................................................ 333
  An Interrupt or Exception Clears EFlags[VM]......................................................... 334
  IRET Sets EFlags[VM] Again..................................................................................... 334
A Task Switch Causes an EFlags Update ...................................................................... 334
DOS Task's Memory Usage ................................................................................................. 335
  1st MB Is DOS Memory............................................................................................... 335
  Paging Provides Each DOS Task with Its Own Copy of the 1st MB......................... 336
  The VMM Should Not Reside in the HMA.................................................................. 337
Dealing with Segment Wraparound ................................................................................ 338
  8088/8086 Processor............................................................................................... 338
  Post-8086 Processors............................................................................................... 338
  Solutions..................................................................................................................... 338
Segment Register Interpretation in VM86 Mode......................................................... 339
Using the Address Size Override Prefix......................................................................... 339
The Privilege Level of a VM86 Task ............................................................................... 339
Restricting IO Accesses ................................................................................................. 340
  The Problem............................................................................................................... 340
  IO-Mapped IO.............................................................................................................. 341
    IO Permission in Protected Mode............................................................................... 341
    IO Permission in VM86 Mode................................................................................... 343
  Memory-Mapped IO..................................................................................................... 343
    Segregate Ports into Two Groups of Memory Pages.............................................. 344
    Set Up Task's Page Tables to Permit or Deny Access......................................... 344
    Handling Display Frame Buffer Updates............................................................. 344
IOPL-Sensitive Instructions............................................................................................. 345
  The Problem—Instructions with Side Effects............................................................ 345
    CLI (Clear Interrupt Enable) Instruction............................................................. 345
    STI (Set Interrupt Enable) Instruction................................................................ 346
    PUSHF (Push Flags) Instruction ......................................................................... 347
    POPF (Pop Flags) Instruction ............................................................................. 347
INT nn (Software Interrupt) Instruction ............................................................... 347
IRET (Interrupt Return) Instruction ....................................................................... 347
The Solution—IOPL Sensitive Instructions............................................................. 348

Interrupt/Exception Generation and Handling........................................................ 348
Introduction ...................................................................................................................... 348
Normally, There’s Only One IDT .................................................................................. 350
VM86 Mode—There Are Two IDTs .............................................................................. 350
Which IDT Is Used? ......................................................................................................... 351
Processor Actions when a Hardware Interrupt Occurs in VM86 Mode.................. 351
Obtain the Vector from the Interrupt Controller .................................................... 351
The Vector Selects a Protected Mode IDT Entry ....................................................... 351
Critical Values Are Stored on VM86 Task’s Level 0 Stack ....................................... 352
Jump to the Handler ..................................................................................................... 355
The Handler May Expect Values in the Data Segment Registers ......... 355
A Handler May Need to Know It Was Entered from a VM86 Task ......................... 355
A Handler May Need to Return Values in the Data Segment Registers .................. 355
Exit the Handler and Return to the Interrupted VM86 Task .................................. 357
Why the Data Segment Registers Were Cleared ..................................................... 357
Execution of the IRET Instruction ............................................................................ 357
Processor Actions When an INT nn Is Executed in VM86 Mode ............................. 358
When the IOPL < 3 ................................................................................................... 358
When the IOPL = 3 ................................................................................................... 358
Processor Actions when an Exception Occurs in VM86 Mode ................................. 359
Execute the Protected Mode Handler or Pass Control to the VMM .......................... 360
The VMM Chooses Its Actions Based on the Vector .................................................. 360
The VMM Passes the Ball to a Real Mode Handler ................................................. 361
Sometimes, the VMM Handles the Event ............................................................... 364
General ................................................................................................................ 364
Attempt to Access a Forbidden IO Port ................................................................... 364
Attempted Execution of a CLI Instruction ............................................................... 365
Attempted Execution of the STI Instruction ............................................................. 368
Attempted Execution of a PUSHF Instruction ......................................................... 369
Attempted Execution of a POPF Instruction ............................................................ 369
Attempted Execution of the INT nn Instruction ..................................................... 369
Attempted Execution of an IRET Instruction ......................................................... 370
Using a Separate Task as a Handler in VM86 Mode ............................................... 370
Registers Accessible in Real/VM86 Mode............................................................... 372
Instructions Usable in Real/VM86 Mode ................................................................. 373
VM86 Mode Evolution .............................................................................................. 374

Chapter 16: The Debug Registers
The Debug Registers .................................................................................................. 375
Chapter 17: Caching Overview

Definition of a Load and a Store ................................................................. 385

The Cache’s Purpose .................................................................................. 386
   Without a Cache, Core Stalls Were Common ....................................... 386
   An On-Die Cache Elminates Many Core Stalls ........................................ 387
      Introduction ......................................................................................... 387
      On a Cache Miss ............................................................................... 387
      The Cache Line .................................................................................. 387
      The Directory Entry .......................................................................... 388
      Repeat Accesses to the Same Areas Result in Cache Hits ................... 388

The Write-Through Cache ........................................................................ 388
   Introduction .......................................................................................... 388
   On a Load Miss ..................................................................................... 389
   On a Load Hit ....................................................................................... 389
   On a Store Miss .................................................................................... 390
   On a Store Hit .................................................................................... 390
   Additional Information on the WT Cache ............................................. 391

The Write Back Cache .............................................................................. 391
   A Line Can Be in One of Four Possible States ........................................ 391
   Before Storing to a Shared Line, Kill All Other Copies ......................... 392
   On a Store Miss, Perform an RWITM ................................................. 392
   Additional Information on the WB Cache ............................................. 393

Snooping ...................................................................................................... 393
   General .................................................................................................. 393
   Snooping and the WT Cache ................................................................. 396
      Introduction ......................................................................................... 396
      Snooping a Memory Read in a WT Cache .......................................... 397
      Snooping a Memory Write in a WT Cache .......................................... 397
   Snooping and the WB Cache ................................................................. 397
      Introduction ......................................................................................... 397
      Snooping a Memory Read in a WB Cache .......................................... 398
      Snooping a Memory Write in a WB Cache .......................................... 398

The Overall Cache Architecture ............................................................... 399
   Introduction .......................................................................................... 399
   The Fully-Associative Cache ................................................................. 399
   Two-Way Set Associative Cache ........................................................... 400
   Four-Way Set Associative Cache ........................................................... 402
   Eight-Way Set Associative Cache .......................................................... 404
Cache Real Estate Management ................................................................. 406
  The Lookup .......................................................................................... 406
  The Cache Initiates the Fetch of the New Line ....................................... 407
  And Immediately Decides Where to Store It ......................................... 407
  Example 1: Castout of a Modified Line .............................................. 407
  Example 2: Castout of an E or S Line ................................................. 408
A Unified Cache ...................................................................................... 408
Split Caches .......................................................................................... 409
Non-Blocking Caches ............................................................................ 410

Chapter 18: 486 Hardware Overview

486 Flavors .......................................................................................... 412
An Overview of the 486 Internal Architecture ....................................... 412
An Overview of the 486 FSB .................................................................. 415
  Address/Data Bus Structure............................................................... 415
  On a Cache Miss, an Entire Line Must Be Fetched ......................... 415
  486 Implemented a Burst Line Fill Transaction ............................... 415
    Background ....................................................................................... 415
    Toggle Mode Transfer Order ............................................................ 416
A20 Mask ............................................................................................ 419
  Accessing Extended Memory in Real Mode ....................................... 419
  Segment Wraparound ........................................................................ 421
  486 Integrated the A20M# Gate .......................................................... 422
On-Chip Cache Added ........................................................................... 425
  General ................................................................................................. 425
  Cache Operation ................................................................................ 425
  An Example ....................................................................................... 426
  On a Memory Read (i.e., a Load) Lookup ......................................... 427
  On a Memory Write (i.e., a Store) Lookup ........................................ 427

Chapter 19: 486 Software Enhancements

FPU Added On-Die ................................................................................ 432
  Introduction ....................................................................................... 432
  FPU-Related Register Set Changes .................................................. 434
  The CR0 FPU Control Bits ............................................................... 435
  The FP Data Registers ...................................................................... 436
  The FCW Register ........................................................................... 438
  The FSW Register ........................................................................... 440
  The FTW Register .......................................................................... 442
  The Instruction Pointer Register ...................................................... 443
  The Data Pointer Register ............................................................... 443
Contents

The FP Data Operand Format ................................................................. 443
FP Error Reporting .............................................................................. 444
Precise Error Reporting ................................................................. 444
Imprecise Error Reporting ............................................................. 444
Why Deferred Error Reporting Is Used ............................................. 445
The WAIT/FWAIT Instruction ............................................................. 445
The NE Bit ..................................................................................... 445
DOS-Compatible FP Error Reporting ................................................ 445
FP Error Reporting Via Exception 16 ................................................ 446
Ignoring FP Errors ......................................................................... 446
Alignment Checking Feature .............................................................. 448
Paging-Related Changes ................................................................. 449
The Write Protect Feature ............................................................... 450
Description .................................................................................. 450
Example Usage: Unix Copy-on-Write Strategy ................................. 451
Directory/Table and Page Caching ..................................................... 451
Page Directory Caching .................................................................. 451
Page Table Caching ...................................................................... 452
Page Caching ............................................................................... 453
Caching-Related Changes to the Programming Environment .......... 454
CR4 Was Added in the Later Models of the 486 .................................. 455
Test Registers Added ..................................................................... 456
Instruction Set Changes ................................................................. 456
Exchange and Add (XADD) ............................................................... 457
Compare and Exchange (CMPXCHG) ............................................. 457
Invalidate Cache (INVD) ................................................................. 457
Write Back and Invalidate (WBINVD) ........................................... 458
Invalidate TLB Entry (INVLPG) ........................................................ 458
Resume from System Management Mode (RSM) ............................... 458
Byte Swap (BSWAP) .................................................................... 459
New/Altered Exceptions ................................................................. 459
Exception 9 Is Now Reserved .......................................................... 459
Exception 17 (Alignment Check) Added .......................................... 460
System Management Mode (SMM) .................................................. 460

Part 5: Pentium® 461

Chapter 20: Pentium® Hardware Overview
Pentium® Flavors ............................................................................... 464
An Overview of the Pentium® Internal Architecture .......................... 464
The First Superscalar IA32 Processor ............................................... 464
Brief Core Description .................................................................. 467
Contents

An Overview of the Pentium® FSB .................................................................................... 469
  Address/Data Bus Structure.......................................................................................... 469
    Address Bus Selects Qword .................................................................................... 469
    Byte Enables Select Location(s) in Qword............................................................. 469
  On a Cache Miss, an Entire Line Must Be Fetched ..................................................... 472
  The Burst Transaction...................................................................................................... 472
    Background .................................................................................................................. 472
    Toggle Mode Transfer Order .................................................................................. 473
    Burst Write Transaction ........................................................................................... 474
The Caches..................................................................................................................... 476
  Split Cache Structure ....................................................................................................... 476
  Pentium® Code Cache..................................................................................................... 476
  Pentium® Data Cache...................................................................................................... 478
Local APIC Added in the P54C............................................................................................ 479
Test Access Port (TAP) ................................................................................................... 481
  General ......................................................................................................................... 481
  Operational Description .............................................................................................. 481
FRC Mode ....................................................................................................................... 483
Soft Reset (INIT#) ......................................................................................................... 485
  Hot Reset and 286 DOS Extender Programs ................................................................. 485
  Alternate (Fast) Hot Reset ............................................................................................ 486
  286 DOS Extenders on Post-286 Processors ................................................................. 487

Chapter 21: Pentium® Software Enhancements

VM86 Extensions ............................................................................................................ 490
  Introduction .................................................................................................................... 490
  Efficient CLI/STI Instruction Handling ..................................................................... 492
    Background .................................................................................................................. 492
    CLI Handling ............................................................................................................... 492
    STI Handling .............................................................................................................. 495
  Efficient Handling of the INT Instruction .................................................................. 495
Protected Mode Virtual Interrupts .................................................................................... 497
Debug Extension .............................................................................................................. 497
Time Stamp Counter ....................................................................................................... 498
  Reading the TSC .......................................................................................................... 498
  Writing to the TSC ...................................................................................................... 499
  Restricting Access to the TSC .................................................................................... 499
  Counter Wraparound .................................................................................................. 499
  RDTSC Is Not a Serializing Instruction ..................................................................... 499
4MB Pages ...................................................................................................................... 501
  The Problem ............................................................................................................... 501
  How To Set Up a 4MB Page ....................................................................................... 501
Part 6: Intro to the P6 Core and FSB

Chapter 22: P6 Road Map
The P6 Processor Family ................................................................. 539
The Klamath Core ........................................................................ 540
The Deschutes Core ...................................................................... 541
The Katmai Core.......................................................................... 542

Chapter 23: P6 Hardware Overview
For More Detail.............................................................................. 544
Introduction..................................................................................... 544
The P6 Processor Core ................................................................... 545
The FSB Interface Unit ................................................................. 546
  The Agent Types ......................................................................... 546
  The Request Agent Types .......................................................... 546
  The Transaction Phases .............................................................. 547
  The Transaction Types ............................................................... 547
The Backside Bus (BSB) Interface Unit ....................................... 547
The Unified L2 Cache ................................................................. 548
The L1 Data Cache ......................................................................... 548
The L1 Code Cache ........................................................................ 548
The Processor Core ....................................................................... 548
The Local APIC Unit ...................................................................... 548

Part 7: Pentium® Pro Software Enhancements

Chapter 24: Pentium® Pro Software Enhancements
Paging Enhancements .................................................................... 554
PAE-36 Mode ............................................................................... 554
  The Problem ............................................................................... 554
  The Solution: PAE-36 Mode ....................................................... 555
  Enabling PAE-36 Mode .............................................................. 556
  The Application Is Still Limited to a 4GB Virtual Address Space 556
  The OS Creates the Application’s Address Translation Tables .... 557
  CR3 Is Loaded with the Top Level Address Translation Table Pointer 557
  The Page Directory Pointer Table Lookup ................................. 558
  The Page Directory Lookup ....................................................... 559
  PDE Points to a Page Table ...................................................... 560
  PDE Points to a 2MB Physical Page .......................................... 561
  The Page Table Lookup ............................................................ 563
Windows OS PAE Support ......................................................... 566
## Contents

- The MCA Register Set .............................................................................................. 589
- The Global Registers ........................................................................................................ 591
  - Introduction ............................................................................................................... 591
  - The Global Count and Present Register ................................................................ 592
  - The Global Status Register ....................................................................................... 593
  - The Global Control Register .................................................................................... 593
- The Composition of a Register Bank ............................................................................. 594
  - Overview .................................................................................................................... 594
- The Bank Control Register ....................................................................................... 594
- The Bank Status Register .......................................................................................... 594
  - General ................................................................................................................ 595
  - Error Valid Bit .................................................................................................... 596
  - Overflow Bit ....................................................................................................... 596
  - Uncorrectable Error Bit ..................................................................................... 596
  - Error Enabled Bit ............................................................................................... 596
  - Miscellaneous Register Valid Bit ..................................................................... 596
  - Address Register Valid Bit ............................................................................... 597
  - Processor Context Corrupt Bit............................................................................. 597
  - MCA Error Code and Model Specific Error Code ............................................. 597
  - Other Information .............................................................................................. 597
- The Bank Address Register ..................................................................................... 598
- The Bank Miscellaneous Register ........................................................................... 598
- The Error Code ................................................................................................................. 598
  - The Error Code Fields .......................................................................................... 598
  - Simple MCA Error Codes .................................................................................... 598
  - Compound MCA Error Codes ............................................................................. 599
  - FSB Error Interpretation ..................................................................................... 602
- MC Exception May or May Not Be Recoverable ......................................................... 605
- Machine Check and BINIT# ........................................................................................... 605
- Additional Error Logging Notes .................................................................................... 605
  - Error Buffering Capability ................................................................................ 605
  - Additional Information for Each Log Entry ....................................................... 606
- Initialization of the MCA Register Set ........................................................................ 606
  - The Performance Counters ...................................................................................... 606
    - Purpose of the Performance Monitoring Facility ............................................ 607
    - Performance Monitoring Registers ..................................................................... 607
      - PerfEvtSel0 and PerfEvtSel1 MSRs ............................................................... 608
      - PerfCtr0 and PerfCtr1 .................................................................................... 609
    - Accessing the Performance Monitoring Registers .......................................... 610
      - Accessing the PerfEvtSel MSRs ................................................................. 610
      - Accessing the PerfCtr MSRs ....................................................................... 610
    - Accessing Using RDPMC Instruction .............................................................. 610
Contents

Accessing Using RDMSR/WRMSR Instructions.................................................. 610
Event Types........................................................................................................... 611
Starting and Stopping the Counters................................................................. 611
Starting the Counters ...................................................................................... 611
Stopping the Counters ..................................................................................... 611
Performance Monitoring Interrupt on Overflow .............................................. 611
MSRs Added .......................................................................................................... 612
Some Notes ......................................................................................................... 612
Test Control Register (TEST_CTL) ................................................................. 620
ROB_CR_BKUPTMPDR6 MSR............................................................................. 621
DebugCtl MSR................................................................................................. 621
General .............................................................................................................. 621
BPM and BP Pin Usage ..................................................................................... 622
Enable Branch Trace Messaging ..................................................................... 623
The Branch, Exception, Interrupt Recording Facility ...................................... 623
General .............................................................................................................. 623
LastBranchFromIP and LastBranchToIP Register Pair ................................. 623
LastExceptionFromIP and LastExceptionToIP Register Pair ......................... 623
Single-Step on Branch, Exception, or Interrupt .............................................. 625
Instruction Set Changes.................................................................................... 626
MMX Not Implemented ..................................................................................... 626
New Instructions .............................................................................................. 626
Conditional Move (CMOV) Eliminates Branches ........................................... 627
Problem It Addresses ...................................................................................... 627
Description ...................................................................................................... 627
Conditional FP Move (FCMOV) Eliminates Branches ..................................... 627
Problem Addressed ......................................................................................... 627
Description ...................................................................................................... 627
FCOMI, FCOMIP, FUCOMI, and FUCOMIP ..................................................... 628
RDPMC ............................................................................................................. 628
Problem Addressed ......................................................................................... 628
Description ...................................................................................................... 628
UD2 ..................................................................................................................... 629
The CPUID Instruction Enhanced ..................................................................... 629
New/Altered Exceptions .................................................................................... 629

Chapter 25: MicroCode Update Feature
The Problem ...................................................................................................... 632
The Solution ...................................................................................................... 632
The Microcode Update Image .......................................................................... 633
Introduction ...................................................................................................... 633
The Microcode Update Header ......................................................................... 634
Matching the Image to a Processor ................................................................. 636
CPUID Enhanced to Supply Update Signature ........................................... 636
Processor/Image Match Determination ....................................................... 636

The Microcode Update Loader ........................................................................ 637
The Bare Bones Loader .................................................................................. 637
The Trigger Initiates the Upload Process ..................................................... 638
After the Upload, the Signature Is Updated ................................................. 638
Authenticating the Image .............................................................................. 638
Additional Loader Requirements ................................................................. 639
Possible Loader Enhancements ................................................................. 639

Updates in a Multiprocessor System .............................................................. 639
The Image Management BIOS ...................................................................... 640
The Purpose of the Image Management BIOS ............................................ 640
The BIOS Interface ...................................................................................... 641
Detailed Function Call Description ............................................................. 641
The Presence Detect Function Call ............................................................... 641
The Write Microcode Update Data Function Call ....................................... 643
The Microcode Update Control Function Call .............................................. 648
The Read Microcode Update Data Function Call ........................................... 650

When Must the Image Upload Take Place? .................................................. 653
Determining if a New Update Supersedes a Previously-Loaded Update ........ 653
Effect of RESET# Or INIT# on a Previously-Loaded Update ....................... 653

Part 8: Pentium® II

Chapter 26: Pentium® II Hardware Overview
The Pentium® Pro and Pentium® II: Same CPU, Different Package ............... 658
Dual-Independent Bus Architecture (DIBA) ................................................ 658
IOQ Depth .............................................................................................. 658
Pentium® Pro/Pentium® II Differences ....................................................... 658
One Product Yields Three Product Lines ................................................... 660
The Pentium® II/Xeon/Celeron Roadmap ..................................................... 660
The Cartridge ............................................................................................. 661
The Pentium® and Pentium® Pro Sockets ..................................................... 661
The Problem ............................................................................................ 661
The Pentium® II Cartridge .......................................................................... 662
The SEC Substrate: the Processor Side ....................................................... 665
General ..................................................................................................... 665
Processor Core ......................................................................................... 665
The SEC Substrate: the Non-Processor Side ............................................... 666
Cartridge Block Diagram ............................................................................. 669
The L2 Cache ............................................................................................ 669
Contents

The Core ........................................................................................................................................ 670
  General ................................................................................................................................. 670
  L1 Caches .......................................................................................................................... 671
    L1 Code Cache Characteristics ...................................................................................... 671
    L1 Data Cache Characteristics ..................................................................................... 671
  L1 and L2 Cache Error Protection ................................................................................ 671
  16-bit Code Optimization ............................................................................................... 672
    The Pentium® Pro Was Not Optimized ................................................................ 672
    Pentium® II Shadows the Data Segment Registers .............................................. 672
The FSB and BSB ................................................................................................................ 675
  The FSB Protocol .............................................................................................................. 675
  The Processor Core and Bus Frequencies .................................................................... 676
  The FSB Arbitration Scheme .......................................................................................... 676
    The Pentium® Pro Processor FSB Arbitration ...................................................... 676
    Pentium® II Processor FSB Arbitration ................................................................. 677
  The BSB and the L2 Cache .............................................................................................. 678
    The BSB Frequency ................................................................................................... 678
    The L2 Cache ............................................................................................................. 678
The Introduction of the Celeron ......................................................................................... 679
Miscellaneous Hardware Stuff ............................................................................................. 679
  Pentium® II/Pentium® Pro Signal Differences ........................................................... 679
  Voltage Identification ...................................................................................................... 680

Chapter 27: Pentium® II Power Management Features
  The Pentium® Pro’s Power Conservation Modes ............................................................ 684
  The Pentium® II’s Power Conservation Modes ............................................................... 684
  The Normal State ................................................................................................................ 686
  The AutoHalt Power Down State ...................................................................................... 686
    Description .................................................................................................................... 686
    The Chipset’s Response to the Halt Message ............................................................ 687
  The Stop Grant State ......................................................................................................... 688
  The Halt/Grant Snoop State .............................................................................................. 691
  The Sleep State ............................................................................................................... 692
  The Deep Sleep State ...................................................................................................... 693

Chapter 28: Pentium® II Software Enhancements
  The Pentium® II and Pentium® III MSRs ......................................................................... 696
  Instruction Set Changes ................................................................................................... 707
    Introduction .................................................................................................................... 707
    Fast System Call/Return Instruction Pair .................................................................... 708
    Background .................................................................................................................... 708
    The OS Initialization of the Fast Call Facility ............................................................. 709
Contents

IOQ Depth ...................................................................................................................... 744
The L1 Caches .................................................................................................................. 745
L1 Code Cache Characteristics ..................................................................................... 745
L1 Data Cache Characteristics ..................................................................................... 745
The L2 Cache .................................................................................................................. 745
The L2 Cache on the Early Pentium® III ....................................................................... 745
The Advanced Transfer Cache ...................................................................................... 746
The Data Prefetcher ....................................................................................................... 747
SSE Introduced .............................................................................................................. 748
General ........................................................................................................................ 748
Detecting SSE Capability .............................................................................................. 749
Detailed Description of SSE .......................................................................................... 750
The SSE Execution Units ............................................................................................. 750
Introduction .................................................................................................................. 750
General ......................................................................................................................... 751
The FP Multiplier Unit ................................................................................................. 751
The Packed FP Add Unit ............................................................................................... 751
The Shuffle/Logical Unit .............................................................................................. 751
The Reciprocal/Reciprocal Square Root Unit ............................................................. 752
Optimized Data Copy Operations .............................................................................. 752
64-bit Paths Limited Performance ................................................................................ 753
The WCBs Were Enhanced .......................................................................................... 754
Additional Writeback Buffers ...................................................................................... 755
Background .................................................................................................................. 755
The Pentium® Pro and Pentium® II ............................................................................ 755
The Pentium® III ......................................................................................................... 755
SpeedStep Technology ................................................................................................. 755

Chapter 31: Pentium® III Software Enhancements

The Streaming SIMD Extensions (SSE) ......................................................................... 758
Why? ......................................................................................................................... 758
Detecting SSE Support ............................................................................................... 758
The SSE Elements ....................................................................................................... 759
The SSE Data Types ..................................................................................................... 760
General ....................................................................................................................... 760
The 32-bit SP FP Numeric Format ............................................................................... 761
Background ................................................................................................................. 761
A Quick IEEE FP Primer ............................................................................................. 761
The 32-bit SP FP Format .............................................................................................. 762
Representing Special Values ....................................................................................... 762
An Example .................................................................................................................. 763
Another Example ........................................................................................................ 764
## Contents

Accuracy vs. Fast Real-Time 3D Processing ................................................................. 765
The SSE Register Set ................................................................................................. 766
  The XMM Data Registers ....................................................................................... 766
  The MXCSR ........................................................................................................... 766
Loading and Storing the MXCSR .............................................................................. 769
Saving and Restoring the Register Set ..................................................................... 769
OS Support for FXSAVE/FXRSTOR, SSE and the SIMD FP Exception ..................... 770
  General .................................................................................................................. 770
  Enable SSE/SSE2 and SSE Register Set Save and Restore ............................... 770
  Enable the SSE SIMD FP Exception .................................................................. 771
SIMD (Packed) Operations ....................................................................................... 772
Scalar Operations ...................................................................................................... 772
Cache-Related Instructions ...................................................................................... 773
  Overlapping Data Prefetch with Program Execution ......................................... 773
  Streaming Store Instructions .............................................................................. 776
    Introduction ........................................................................................................ 776
    Some Questions Regarding Documentation ................................................. 778
    The MOVNTPS Instruction ........................................................................... 780
    The MOVNTQ Instruction ............................................................................. 781
    The MASKMOVQ Instruction ....................................................................... 782
Ensuring Delivery of Writes Before Proceeding ....................................................... 783
  An Example Scenario ......................................................................................... 783
  The SFENCE Instruction .................................................................................... 784
Elimination of Mispredicted Branches ...................................................................... 787
  Background .......................................................................................................... 787
  SSE Misprediction Enhancements ..................................................................... 787
    Comparisons and Bit Masks .......................................................................... 787
    Min/Max Determination .................................................................................. 788
    The Masked Move Operation ....................................................................... 788
Reciprocal and Reciprocal Square Root Operations ............................................... 788
MPEG-2 Motion Compensation .............................................................................. 789
Optimizing 3D Rasterization Performance .............................................................. 790
Optimizing Motion-Estimation Performance .......................................................... 790
Summary of the SSE Instruction Set ....................................................................... 791
SSE Alignment Checking ......................................................................................... 792
CPUID Enhanced ..................................................................................................... 793
  Serial Number Request Added ........................................................................... 793
  Brand Index Request Added .............................................................................. 793
Intro to the FSB Enhancements ............................................................... 841
IA Instructions Vary in Length and Are Complex ............................. 843
The Trace Cache ................................................................................ 843
There Are Two Pipeline Sections ......................................................... 844
The µop Pipeline ............................................................................ 844
  Introduction ............................................................................. 844
  The P6 Processor’s Instruction Pipeline ........................................... 845
  The Pentium® 4’s µop Pipeline ................................................... 845
  The 90nm Pentium® 4’s Instruction Pipeline ............................... 846
The IA32 Data Register Set Was Small .............................................. 846
  General .................................................................................. 846
  The P6 Had 40 General-Purpose Registers ................................... 847
  The Pentium® 4 Implements a Large Array of Data Registers .. 848
  The Compiler Manages Data Register Usage .............................. 848
  Elimination of False Register Dependencies .............................. 852
Speculative Execution ........................................................................ 853
Chapter 36: Pentium® 4 PowerOn Configuration
Configuration on Trailing-Edge of Reset ........................................... 856
Setup and Hold Time Requirements ...................................................... 858
Built-In Self-Test (BIST) Trigger ......................................................... 858
Assignment of IDs to the Processor ...................................................... 860
  Introduction ............................................................................ 860
  The Cluster ID ...................................................................... 860
    The Purpose of the Cluster ID ............................................. 860
    The Cluster ID Assignment ............................................. 860
  The Agent ID ........................................................................ 861
    The Purpose of the Agent ID ............................................. 861
    Physical versus Logical Processor ................................. 861
    The Agent ID Assignment ............................................. 862
      Example Xeon MP System with Hyper-Threading Disabled .... 862
      Example Xeon MP System with Hyper-Threading Enabled .... 862
      Dual Processor System with Hyper-Threading Enabled ...... 863
        A Single-Processor System with Hyper-Threading Enabled .... 863
  The Local APIC ID ............................................................ 864
    The Purpose of the Local APIC ID ..................................... 864
    The Local APIC ID Assignment .................................... 865
Error Observation Options .............................................................. 866
In-Order Queue Depth Selection ....................................................... 866
Power-On Restart Address .............................................................. 866
Tri-State Mode ............................................................................. 867
Processor Core Speed Selection ....................................................... 867
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus Parking Option</td>
<td>868</td>
</tr>
<tr>
<td>Description</td>
<td>868</td>
</tr>
<tr>
<td>Bus Parking Configuration</td>
<td>869</td>
</tr>
<tr>
<td>Hyper-Threading Option</td>
<td>869</td>
</tr>
<tr>
<td>Program-Accessible Startup Features</td>
<td>869</td>
</tr>
</tbody>
</table>

### Chapter 37: Pentium® 4 Processor Startup

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>876</td>
</tr>
<tr>
<td>The Processor's State After Reset</td>
<td>877</td>
</tr>
<tr>
<td>EAX, EDX Content After Reset Removal</td>
<td>883</td>
</tr>
<tr>
<td>The Core Is Starving and Caching is Disabled</td>
<td>884</td>
</tr>
<tr>
<td>Boot Strap Processor (BSP) Selection</td>
<td>885</td>
</tr>
<tr>
<td>Introduction</td>
<td>885</td>
</tr>
<tr>
<td>The BSP Selection Process</td>
<td>886</td>
</tr>
<tr>
<td>How the APs are Discovered and Configured</td>
<td>888</td>
</tr>
<tr>
<td>AP Detection and Configuration</td>
<td>888</td>
</tr>
<tr>
<td>Introduction</td>
<td>888</td>
</tr>
<tr>
<td>The BIOS's AP Discovery Procedure</td>
<td>889</td>
</tr>
<tr>
<td>Uni-Processor OS</td>
<td>890</td>
</tr>
<tr>
<td>MP OS</td>
<td>890</td>
</tr>
<tr>
<td>The FindAndInitAllCPUs Routine</td>
<td>893</td>
</tr>
</tbody>
</table>

### Chapter 38: Pentium® 4 Core Description

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>One µop Doesn't Necessarily = One IA32 Instruction</td>
<td>898</td>
</tr>
<tr>
<td>Upstream vs. Downstream</td>
<td>899</td>
</tr>
<tr>
<td>Introduction</td>
<td>899</td>
</tr>
<tr>
<td>The Big Picture</td>
<td>900</td>
</tr>
<tr>
<td>The Front-End Pipeline Stages</td>
<td>902</td>
</tr>
<tr>
<td>CS:EIP Address Generation</td>
<td>902</td>
</tr>
<tr>
<td>Linear to Physical Address Translation</td>
<td>904</td>
</tr>
<tr>
<td>The L2 Cache Lookup</td>
<td>906</td>
</tr>
<tr>
<td>On an L2 Miss, the Request Is Passed to the BSQ</td>
<td>907</td>
</tr>
<tr>
<td>The Code Block Is Placed in the Instruction Streaming Buffer</td>
<td>909</td>
</tr>
<tr>
<td>The Front-End BTB</td>
<td>910</td>
</tr>
<tr>
<td>The Static Branch Predictor</td>
<td>911</td>
</tr>
<tr>
<td>The Travels of a Conditional Branch Instruction</td>
<td>913</td>
</tr>
<tr>
<td>The IA32 Instruction Decoder</td>
<td>914</td>
</tr>
<tr>
<td>The P6 Instruction Decoder Was Complex</td>
<td>914</td>
</tr>
<tr>
<td>The Pentium® 4 Decoder Is Simple</td>
<td>915</td>
</tr>
<tr>
<td>The Trace Cache Can Keep Up with the Fast Execution Engine</td>
<td>915</td>
</tr>
<tr>
<td>µops Are Streamed into the Trace Cache and the µop Queue</td>
<td>916</td>
</tr>
</tbody>
</table>
## Contents

Complex Instructions Are Decoded by the Microcode Store ROM .......................... 917  
General ..................................................................................................................... 917  
The MS ROM and Interrupts or Exceptions ......................................................... 917  
The Trace Cache ......................................................................................................... 919  
General ..................................................................................................................... 919  
Build Mode .............................................................................................................. 920  
Deliver Mode............................................................................................................ 920  
Self-Modifying Code (SMC) .................................................................................. 923  
Introduction............................................................................................................. 923  
Your Code May Appear to be SMC ....................................................................... 923  
SMC and the Earlier IA32 Processors .................................................................... 923  
SMC and The Pentium® 4 ...................................................................................... 924  
The Trace Cache BTB and the Return Stack Buffer ............................................... 925  
The Trace Cache BTB ................................................................................................ 925  
The Return Stack Buffer (RSB) ............................................................................. 926  
The µop Queue ......................................................................................................... 927  

### Intro to the µop Pipeline

General ..................................................................................................................... 928  
The TC Next IP Stage .............................................................................................. 929  
The TC Fetch Stage .................................................................................................. 929  
The Drive 1 Stage ..................................................................................................... 930  
The Allocator Stage ................................................................................................. 930  
The Register Rename Stage .................................................................................... 931  
The Memory and General µop Queue Stage ....................................................... 933  
The Scheduler Stage ............................................................................................... 933  
The µop Dispatch Stage .......................................................................................... 934  
The Register File Stage ........................................................................................... 935  
The Execution Stage ............................................................................................... 935  
The Flags Stage ....................................................................................................... 936  
The Branch Check Stage ......................................................................................... 936  
The Drive 2 Stage .................................................................................................... 937  

### The µop Pipeline’s Major Elements

The Allocator .......................................................................................................... 938  
General ..................................................................................................................... 938  
The ReOrder Buffer (ROB) Entry ............................................................................ 939  
The Register File Allocation ..................................................................................... 940  
The Load and Store Buffer Allocation .................................................................... 940  
The Memory or General µop Queue Allocation .................................................... 941  
The Register Rename Unit ..................................................................................... 941  
General ..................................................................................................................... 941  
Renaming the Destination Register ......................................................................... 941  
Renaming the Source Register(s) ........................................................................... 942
## Contents

Renaming Eliminates False Register Dependencies ............................................ 950  
The Memory and General µop Queues ................................................................. 950  
The Schedulers Enable Out-of-Order Execution ....................................................... 951  
The Register Files Are Strategically Placed ........................................................... 953  
Dispatch Port 0 ........................................................................................................ 954  
Dispatch Port 1 ........................................................................................................ 956  
Dispatch Port 2 ........................................................................................................ 957  
Dispatch Port 3 ........................................................................................................ 958  
Instruction Dispatch Rate ...................................................................................... 959  
The Complex Execution Units Are Pipelined ......................................................... 960  
The Retirement Stage ............................................................................................ 960  
General .................................................................................................................. 960  
µop Retirement vs. IA32 Instruction Retirement .................................................... 961  
Additional, Core-Specific Terms ........................................................................... 962

### Chapter 39: Hyper-Threading

General .................................................................................................................. 966  
Background ............................................................................................................ 967  
Multithreading Overview ....................................................................................... 967  
How Threads Are Assigned in an SMP System ...................................................... 968  
CMP Is Another Solution ....................................................................................... 968  
Traditional Single-Processor Multithreading ....................................................... 968  
The HT Approach .................................................................................................. 969  
Instruction Level Parallelism (ILP) ......................................................................... 969  
But What If... ........................................................................................................... 970  
This Requires Two, Almost Complete Register Sets ............................................ 970  
HT = Simultaneous Multithreading ....................................................................... 970  
Terms: Cluster, Physical CPU, Logical CPU ......................................................... 971  
Detecting HT Capability ......................................................................................... 971  
Enabling/Disabling HT ........................................................................................... 972  
Each Logical Processor Has Its Own Local APIC ................................................. 972  
HT Processor Resource Types ................................................................................ 973  
General .................................................................................................................. 973  
Resources that Are Always Replicated ................................................................ 973  
Resources that Are Always Shared ....................................................................... 974  
Resources Wherein Sharing or Replication Is Design-Specific ............................ 974  
The HT States ......................................................................................................... 975  
Switching HT States .............................................................................................. 975  
Processor Enumeration ......................................................................................... 975  
The Primary and Secondary Logical Processor .................................................... 977  
OS Support for HT .................................................................................................. 977  
General .................................................................................................................. 977
Contents

OSs that Include Native HT Support ................................................................. 977
OSs that Are Compatible with HT ................................................................. 977
OSs with No HT Support .................................................................................. 978

Overview of HT Resource Usage ..................................................................... 978
TC Access ......................................................................................................... 978
L2 Cache Access ............................................................................................... 979
Code Block Is Placed in the Prefetch Streaming Buffer .................................. 980
Instruction Decode ............................................................................................ 981
Complex Instruction Decode ............................................................................. 982
The µops Are Placed in the Trace Cache ......................................................... 983
The Return Stack Buffer Is Replicated ............................................................ 984
The µops Are Placed in the µop Queue ............................................................. 985
In Each Clock, the Allocator Switches Queues ............................................... 986
The Register Rename Stage ............................................................................. 987
µop Queues Are Partitioned ........................................................................... 987
The Schedulers Are Agnostic ......................................................................... 988
Register File Access ......................................................................................... 989
The Retirement Stage ....................................................................................... 990

HT and the Data TLB ........................................................................................ 992
HT and the FSB ................................................................................................. 992

The IOQ Depth Was Increased ........................................................................ 993

HT Performance Issues .................................................................................... 993
Introduction ....................................................................................................... 993
Thread Distribution to Logical Processors ..................................................... 994
Load Balancing ................................................................................................. 995

HT and the Processor Caches ......................................................................... 996
Physical Processors Operating on Separate Data Sets ..................................... 996
Data Sharing by Physical Processors .............................................................. 997
Introduction ..................................................................................................... 997
Using a Semaphore to Access a Shared Data Area ....................................... 997
An Ideal Situation ......................................................................................... 997
A Bad Situation ......................................................................................... 998
If the Shared Data and the Semaphore Are in the Same Line .................... 999
Solution ....................................................................................................... 999
Data Sharing by Co-Resident Logical Processors .......................................... 1000
Co-Resident Logical Processors with Separate Data Sets ............................ 1000
Executing Identical Threads ......................................................................... 1000
Halt Usage ..................................................................................................... 1000
Thread Synchronization ............................................................................... 1001
Definition .................................................................................................... 1001
The Problem ................................................................................................. 1001
The Fix ....................................................................................................... 1002
Contents

When A Thread Is Idle ........................................................................................... 1003
Spin-Lock Optimization ......................................................................................... 1003
WCB Usage ........................................................................................................ 1004
HT and Serializing Instructions ............................................................................. 1004
HT and the Microcode Update Feature .............................................................. 1005
HT Cache-Related Issues .................................................................................... 1005
HT and the TLBs .................................................................................................. 1006
HT and the Thermal Monitor Feature ................................................................. 1006
HT and External Pin Usage .................................................................................. 1007
STPCLK# Pin ...................................................................................................... 1007
LINT0 AND LINT1 Pins ...................................................................................... 1007
A20M# Pin ......................................................................................................... 1008

Chapter 40: The Pentium® 4 Caches
A Cache Primer .................................................................................................... 1011
The L0 Cache ...................................................................................................... 1011
Upstream vs. Downstream ................................................................................. 1011
Overview ........................................................................................................... 1011
Determining the Processor's Cache Sizes and Structures ................................ 1012
Enabling/Disabling the Caches ............................................................................ 1013
The L1 Data Cache ............................................................................................... 1013
General ............................................................................................................... 1013
The L1 Data Cache Clients .................................................................................. 1014
The Data Cache Is a Write-Through Cache ....................................................... 1014
The Data Cache is Non-Blocking ....................................................................... 1018
Earlier Processor Caches Blocked, but So What ............................................. 1018
The L1 Data Cache is Non-Blocking, and That's Important! ......................... 1019
The L1 Data Cache Implements Squashing ....................................................... 1019
The L1 Data Cache Architecture ........................................................................ 1019
The Data Cache's View of Memory Space ....................................................... 1020
The Data Cache Lookup .................................................................................... 1022
The Line Number Selects the Directory Set .................................................... 1022
Simultaneously, a DTLB Lookup Is Performed .............................................. 1022
The Physical Page Address Formation ............................................................ 1023
The Physical Page Address Compare ............................................................... 1023
The Data Cache LRU Algorithm ....................................................................... 1024
The Data TLB (DTLB) ......................................................................................... 1024
The L2 ATC ......................................................................................................... 1025
Introduction ...................................................................................................... 1025
The L2 Cache's Clients ...................................................................................... 1025
The L2 Cache Architecture ............................................................................... 1025
The L2 Cache Is Non-Blocking ......................................................................... 1027
## Contents

The L2 Cache Implements Squashing ................................................................. 1027
The L2 Cache’s View of Memory Space ............................................................ 1028
The L2 Cache Lookup ....................................................................................... 1028
The L2 Cache LRU Algorithm ........................................................................... 1030
  General ........................................................................................................... 1030
  When an L2 Directory Entry Already Exists .............................................. 1030
  When an L2 Directory Entry Doesn’t Already Exist .................................. 1030
    If There Is an L3 Cache ........................................................................ 1030
    If There Isn’t an L3 Cache .................................................................. 1031
    Recording the New Sector in the L2 Cache ........................................ 1031
Loads and TC Requests and the L2 Cache ....................................................... 1031
Stores and the L2 Cache .................................................................................. 1034
Snoops and the L2 Cache ................................................................................ 1037
Other L2 Cache Sizes ...................................................................................... 1039
The Hardware Data Prefetcher ........................................................................ 1039
  Introduction .............................................................................................. 1039
  The Startup Penalty ................................................................................. 1040
  How the Data Prefetch Logic Works ...................................................... 1040
  Some Constraints .................................................................................... 1040
The L3 Cache .................................................................................................... 1041
  Introduction .............................................................................................. 1041
  The L3 Cache’s Client ............................................................................ 1041
  The L3 Cache Architecture ....................................................................... 1041
  The L3 Cache Is Non-Blocking .............................................................. 1043
  The L3 Cache Implements Squashing ...................................................... 1043
  The L3 Cache’s View of Memory Space .................................................. 1044
  The L3 Cache Lookup .......................................................................... 1044
  The L3 Cache LRU Algorithm ................................................................. 1045
    General ................................................................................................... 1045
    When an L3 Directory Entry Already Exists ....................................... 1046
    When an L3 Directory Entry Doesn’t Already Exist ............................ 1046
    Loads and TC Requests and the L3 Cache .......................................... 1047
    Stores and the L3 Cache ..................................................................... 1049
    Snoops and the L3 Cache ................................................................ 1051
    Other L3 Cache Sizes ....................................................................... 1054
FSB Transactions and the Caches .................................................................. 1055
  Background .............................................................................................. 1055
    A Single-Sector Fetch ........................................................................ 1055
    A Two Sector Fetch .......................................................................... 1055
    Writeback of a Modified Line ......................................................... 1056
The Cache Management Instructions ............................................................... 1056
## Chapter 41: Pentium® 4 Handling of Loads and Stores

**The Memory Type Defines Load/Store Characteristics** ................................................ 1062

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load µops</td>
<td>1063</td>
</tr>
<tr>
<td>The Load Buffers</td>
<td>1063</td>
</tr>
<tr>
<td>Loads from Cacheable Memory</td>
<td>1064</td>
</tr>
<tr>
<td>Loads Can Be Executed Out-of-Order</td>
<td>1065</td>
</tr>
<tr>
<td>The L1 Data Cache Implements Squashing</td>
<td>1066</td>
</tr>
<tr>
<td>Loads from Uncacheable Memory</td>
<td>1066</td>
</tr>
<tr>
<td>The Definition of a Speculatively Executed Load</td>
<td>1067</td>
</tr>
<tr>
<td>Replay</td>
<td>1067</td>
</tr>
<tr>
<td>Replay of µops Dependent on a Load</td>
<td>1067</td>
</tr>
<tr>
<td>Replay of Loads Dependent on a Store</td>
<td>1068</td>
</tr>
<tr>
<td>Loads and the Prefetch Instructions</td>
<td>1068</td>
</tr>
<tr>
<td>The LFENCE Instruction</td>
<td>1068</td>
</tr>
<tr>
<td>General</td>
<td>1068</td>
</tr>
<tr>
<td>LFENCE Ordering Rules</td>
<td>1069</td>
</tr>
<tr>
<td>Store-to-Load Forwarding</td>
<td>1070</td>
</tr>
<tr>
<td>Background</td>
<td>1070</td>
</tr>
<tr>
<td>Description</td>
<td>1070</td>
</tr>
<tr>
<td>Linear Address Mismatch Allows Load Before Store</td>
<td>1071</td>
</tr>
<tr>
<td>Linear Address Match Results in Store Forwarding</td>
<td>1072</td>
</tr>
<tr>
<td>Store Forwarding Rules</td>
<td>1072</td>
</tr>
<tr>
<td>Store µops</td>
<td>1072</td>
</tr>
<tr>
<td>Stores Are Handled by the Store Buffers</td>
<td>1073</td>
</tr>
<tr>
<td>Stores to UC Memory</td>
<td>1074</td>
</tr>
<tr>
<td>General</td>
<td>1074</td>
</tr>
<tr>
<td>UC Store Buffer Draining</td>
<td>1074</td>
</tr>
<tr>
<td>UC FSB Transactions</td>
<td>1074</td>
</tr>
<tr>
<td>Stores to WC Memory</td>
<td>1075</td>
</tr>
<tr>
<td>Determining if the WC Memory Type Is Supported</td>
<td>1075</td>
</tr>
<tr>
<td>The WC Memory Model</td>
<td>1076</td>
</tr>
<tr>
<td>WCB Evolution</td>
<td>1077</td>
</tr>
<tr>
<td>Filling the WCBs</td>
<td>1077</td>
</tr>
<tr>
<td>Draining the WCBs</td>
<td>1079</td>
</tr>
<tr>
<td>General</td>
<td>1079</td>
</tr>
<tr>
<td>Serializing Instructions</td>
<td>1079</td>
</tr>
<tr>
<td>A Special Use of the WCBs</td>
<td>1080</td>
</tr>
<tr>
<td>The WCBs and Hyper-Threading</td>
<td>1080</td>
</tr>
<tr>
<td>WCB FSB Transactions</td>
<td>1080</td>
</tr>
<tr>
<td>Stores to WP Memory</td>
<td>1081</td>
</tr>
<tr>
<td>General</td>
<td>1081</td>
</tr>
</tbody>
</table>
Chapter 42: The Pentium® 4 Prescott

Introduction ........................................................................................................... 1093
Increased Pipeline Depth .................................................................................... 1093
Trace Cache Improvements .................................................................................. 1093
  Increased Trace Cache BTB Size ..................................................................... 1093
  Enhanced Trace Cache µop Encoding .............................................................. 1093
Increased Number of WCBs ................................................................................ 1094
L1 Data Cache Changes ....................................................................................... 1094
Increased L2 Cache Size ..................................................................................... 1094
Enhanced Branch Prediction ................................................................................ 1095
  Enhanced Static Branch Predictor ................................................................. 1095
  Dynamic Branch Prediction Enhanced ........................................................... 1095
Store Forwarding Improved .............................................................................. 1095
  Increased Number of Store Buffers ............................................................... 1095
  Improved Load/Store Scheduling ................................................................. 1095
  Force Forwarding Background ................................................................. 1096
  Force Forwarding .......................................................................................... 1096
    The Solution to False Forwarding ......................................................... 1097
    The Address Misalignment Solution ......................................................... 1097
SSE3 Instruction Set ............................................................................................ 1098
  Introduction .................................................................................................... 1098
  Improved x87 FP-to-Integer Conversion Instruction .................................... 1099
    The Problem .................................................................................................. 1099
    The Solution ................................................................................................... 1099
  New Complex Arithmetic Instructions ......................................................... 1100
  Improved Motion Estimation Performance ................................................... 1101
    The Problem .................................................................................................. 1101
Contents

The Solution ............................................................................................................................................. 1101
The Downside ......................................................................................................................................... 1102
Instructions to Improve Processing of a Vertex Database ..................................................................... 1102
Thread Synchronization Instructions ..................................................................................................... 1104
Increased Elimination of Dependencies ................................................................................................ 1104
Enhanced Shifter/Rotator ....................................................................................................................... 1104
Integer Multiply Enhanced .................................................................................................................... 1105
Scheduler Enhancements ........................................................................................................................ 1105
Fixed the MXCSR Serialization Problem .............................................................................................. 1105
Data Prefetch Instruction Execution Enhanced ....................................................................................... 1106
Improved the Hardware Data Prefetcher ................................................................................................ 1106
Hyper-Threading Improved .................................................................................................................... 1106
  Decreased Possibility of L1 Data Cache Blocking ............................................................................. 1106
  Increased the Size of the µop Queue .................................................................................................. 1107
  Eliminated Page Table Walk/Split Line Access Conflict .................................................................. 1107
  Handling Multiple Page Table Walks that Miss All Caches ............................................................ 1108
  Trace Cache Responds Quicker to a Thread Stall ............................................................................ 1108
  The Data Cache and Hyper-Threading ............................................................................................. 1109
    Author’s Note ..................................................................................................................................... 1109
    Introduction ........................................................................................................................................ 1109
    Shared Mode ...................................................................................................................................... 1109
    Adaptive Mode ................................................................................................................................. 1109
  The MONITOR and MWAIT Instructions ......................................................................................... 1110
    Background ....................................................................................................................................... 1110
    The Monitor Instruction .................................................................................................................... 1111
    The Mwait Instruction ........................................................................................................................ 1111
    Example Code Usage ......................................................................................................................... 1112
    The Wake Up Call ............................................................................................................................. 1112

Chapter 43: Pentium® 4 FSB Electrical Characteristics

Introduction ............................................................................................................................................... 1116
The Bus and Processor Clocks ............................................................................................................... 1117
  The BSEL Outputs ............................................................................................................................. 1117
  The Processor’s Operational Clock Frequency ................................................................................... 1117
  BCLK Is a Differential Signal ............................................................................................................ 1117
The Address and Data Strobes ............................................................................................................. 1119
  Delivering the Request ....................................................................................................................... 1119
    The P6 Request Delivery Method ................................................................................................... 1120
    The Pentium® 4/M Request Delivery Method ............................................................................... 1120
  Delivering the Data ............................................................................................................................ 1121
    The P6 Data Delivery Method ........................................................................................................ 1121
    The Pentium® 4/M Data Delivery Method ..................................................................................... 1123
# Contents

Why Multiple Strobes? .......................................................... 1124
The Data Bus Inversion Signals ............................................. 1125
Address and Data Strobe Setup and Hold Specs ..................... 1126
The Voltage ID ........................................................................... 1126
Everything's Relative ............................................................... 1127
All AGTL+ Signals Are Active When Low ......................... 1127
All AGTL+ Signals Are Terminated ........................................... 1127
Deasserting an AGTL+ Signal Line ........................................... 1130
Each AGTL+ Input Has a Comparator ................................. 1131
The Reference Voltage ............................................................. 1131
The Sample Point ................................................................. 1131
The Pre-90nm Comparison ...................................................... 1131
The 90nm Comparison ............................................................ 1131
AGTL+ Setup and Hold Specs ................................................. 1134
Signals that Can Be Driven by Multiple FSB Agents .......... 1135
Minimum One BCLK Response Time ................................. 1135

## Chapter 44: Intro to the Pentium® 4 FSB

Enhanced Mode Scaleable Bus ............................................. 1138
FSB Agents ............................................................................. 1138
Agent Types ........................................................................... 1138
Multiple Personalities ............................................................ 1139
Uniprocessor vs. Multiprocessor Bus ................................. 1140
The Request Agent ............................................................... 1141
The Request Agent Types ...................................................... 1141
The Agent ID ........................................................................ 1141
The Purpose of the Agent ID ................................................ 1141
How the Agent ID Is Assigned ............................................. 1142
The Transaction Phases ........................................................ 1142
The P6 Transaction Phases .................................................... 1142
The Pentium® 4/M Transaction Phases .............................. 1142
Transaction Pipelining ......................................................... 1143
The FSB Is Subdivided into Signal Groups ......................... 1143
Step 1: Gain Ownership of the Request Phase Signal Group  ... 1143
Step 2: Issue the Transaction Request ................................ 1144
Step 3: Yield Request Phase Signal Group, Proceed to Next Signal Group 1144
The Phases Proceed in a Predefined Order ......................... 1144
The Request Phase ............................................................... 1144
The Snoop Phase ................................................................. 1145
The Response Phase ............................................................ 1145
The Data Phase(s) ................................................................. 1146
Contents

The Next Agent Can’t Use a Signal Group Until the Current
Agent Is Finished With It ................................................................. 1146

Transaction Tracking ........................................................................ 1147
  Request Agent Transaction Tracking .................................................. 1147
  Snoop Agent Transaction Tracking ....................................................... 1147
  Response Agent Transaction Tracking .................................................. 1148
  The IOQ .................................................................................................. 1148

Chapter 45: Pentium® 4 CPU Arbitration
  The Request Phase ............................................................................... 1150
  Logical versus Physical Processors ...................................................... 1150
  The Discussion Assumes a Quad Xeon MP System ................................. 1151
  Symmetric Agent Arbitration—Democracy at Work ............................... 1151
    No External Arbiter Required ............................................................ 1151
    The Arbitration Algorithm ................................................................. 1152
      One Arbiter Per Physical Processor ................................................. 1152
      The Rotating ID ............................................................................... 1152
      The Busy/Idle Indicator .................................................................. 1153
        General .......................................................................................... 1153
        Reset’s Effect on the Busy/Idle Indicator ....................................... 1154
        The Idle Loop ............................................................................... 1155
        Transition from Idle to Busy .......................................................... 1156
        Bus Parking .................................................................................. 1156
        Preemption by Another Physical Processor ..................................... 1158
        Transitioning Back to the Idle State ............................................... 1158
    Requesting Ownership ........................................................................ 1159
      Introduction ..................................................................................... 1159
      Example of One Symmetric Agent Requesting Ownership ................. 1160
      Example of Two Symmetric Agents Requesting Ownership ............... 1161
    Definition of an Arbitration Event ...................................................... 1163
      Once BREQn# Asserted, Keep Asserted Until Ownership Attained .......... 1164

Chapter 46: Pentium® 4 Priority Agent Arbitration
  Priority Agent Arbitration ................................................................. 1166
    Example Priority Agents ..................................................................... 1166
    Priority Agent Beats Symmetric Agents, Unless .................................. 1168
    Using Simple Approach, Priority Agent Suffers Penalty ......................... 1169
    Smarter Priority Agent Gets Ownership Faster ...................................... 1171
      Ownership Attained in 1 BCLK ......................................................... 1172
      Ownership Attained in 2 BCLKs ....................................................... 1173
Contents

Chapter 49: Pentium® 4 FSB Request Phase
Cautionary Note ................................................................................................................... 1202
Introduction to the Request Phase.................................................................................... 1202
The Source Synchronous Strobes ..................................................................................... 1203
The Request Phase Parity ................................................................................................... 1204
Request Phase Parity Checking......................................................................................... 1205
  ChipSet Request Phase Parity Checking and Reporting ............................................ 1205
  Processor Request Phase Parity Checking and Reporting ...................................... 1206
The Request Phase Signal Group is Multiplexed ......................................................... 1208
Introduction to the Transaction Types............................................................................. 1210
The Contents of Request Packet A ................................................................................... 1212
  Description..................................................................................................................... 1212
  32-bit vs. 36-bit Addresses ............................................................................................ 1217
The Contents of Request Packet B.................................................................................... 1219

Chapter 50: Pentium® 4 FSB Snoop Phase
Agents Involved in the Snoop Phase ............................................................................... 1226
The Snoop Phase Has Two Purposes ............................................................................... 1228
The Snoop Result Signals are Shared, DEFER# Isn’t ................................................... 1229
The Snoop Phase Duration Is Variable ........................................................................... 1229
There Is No Snoop Stall Duration Limit ......................................................................... 1234
Memory Transaction Snooping......................................................................................... 1234
  The Snoop’s Effects on Processor Caches ................................................................. 1234
  Self-Snooping .............................................................................................................. 1238
Non-Memory Transactions Have a Snoop Phase .......................................................... 1239

Chapter 51: Pentium® 4 FSB Response and Data Phases
A Note on Deferred Transactions ..................................................................................... 1242
The Purpose of the Response Phase................................................................................. 1242
The Response Phase Signal Group ................................................................................... 1243
The Response Phase Start Point ....................................................................................... 1243
The Response Phase End Point ......................................................................................... 1243
The Response Types ............................................................................................................ 1244
The Response Phase May Complete a Transaction....................................................... 1246
The Data Phase Signal Group ........................................................................................... 1246
Five Example Scenarios....................................................................................................... 1247
  A Transaction that Doesn’t Transfer Data................................................................. 1247
  A Read that Doesn’t Hit a Modified Line and is Not Deferred .................................. 1249
    The Basics .................................................................................................................. 1249
  A Detailed Description ............................................................................................... 1250
  How Does the Response Agent Know the Transfer Length? ............................. 1251
Contents

The Earliest Deassertion of DBSY# ................................................................. 1252
Special Case—Single BCLK, 0-Wait State Transfer ..................................... 1252
A Write that Doesn’t Hit a Modified Line and Isn’t Deferred ....................... 1252
Introduction .................................................................................................. 1252
Transaction 1’s Response ............................................................................. 1253
Transaction 1’s Target Is Ready to Accept Write Data ................................. 1253
Transaction 1’s Request Agent Gets the Go-Ahead ....................................... 1255
Condition that Permits 1 BCLK TRDY# Assertion ..................................... 1256
Transaction 1’s Request Agent Takes Ownership of the Data Bus ............... 1256
Transaction 1’s Completion ......................................................................... 1259
Transaction 2’s Description ....................................................................... 1260
A Hard Failure Response ............................................................................ 1261
The Snoop Agents Change the State of the Line from E to I or S to I ......... 1261
A Read that Hits a Modified Line .................................................................. 1262
The Basics .................................................................................................. 1262
Relaxed DBSY# Deassertion ........................................................................ 1264
A Write that Hits a Modified Line ................................................................ 1264
Data Phase Wait States ................................................................................ 1266
The Response Phase Parity ........................................................................ 1268
General ..................................................................................................... 1268
ChipSet Response Phase Parity Checking and Reporting ......................... 1268
Processor Response Phase Parity Checking and Reporting ....................... 1269
Data Bus Parity .......................................................................................... 1270
Introduction .............................................................................................. 1270
ChipSet Data Phase Parity Checking and Reporting .................................. 1273
Processor Data Phase Parity Checking and Reporting .............................. 1274
Parity When Transferring a Sub-Block ....................................................... 1275

Chapter 52: Pentium® 4 FSB Transaction Deferral

Example System Models ............................................................................... 1278
Example Multi-Cluster Model ...................................................................... 1279
The Problem .............................................................................................. 1279
Example Problem 1 .................................................................................... 1279
Example Problem 2 .................................................................................... 1280
Possible Solutions ....................................................................................... 1280
Example Read From a PCI Express Device ................................................ 1281
The Read Receives the Deferred Response ............................................... 1281
The Root Complex Performs the Read ...................................................... 1282
The Root Complex Issues a Deferred Reply Transaction ......................... 1283
General .................................................................................................... 1283
The Original Request Agent Is Selected .................................................... 1284
Chapter 55: Pentium® 4 FSB Miscellaneous Signals
The Signals ............................................................................................................................ 1314

Chapter 56: Pentium® 4 Software Enhancements
The Foundation .................................................................................................................... 1322
Miscellaneous New Instructions....................................................................................... 1325
  General........................................................................................................................ ..... 1325
  The Cache Line Flush Instruction............................................................................ 1325
  The Fence Instructions............................................................................................ 1325
    The Memory Fence Instruction ............................................................................ 1326
    The Load Fence Instruction ............................................................................... 1326
  The Non-Temporal Store Instructions ................................................................ 1326
    Introduction............................................................................................................. 1327
    The MOVNTDQ Instruction ........................................................................... 1327
    The MOVNTPD Instruction ............................................................................. 1328
    The MOVNTPD Instruction ............................................................................. 1329
    The MASKMOVDQU Instruction .................................................................... 1330
      General .............................................................................................................. 1330
    When a Mask of All Zeros Is Used................................................................ 1331
  The PAUSE Instruction ................................................................................................. 1331
  The Branch Hints............................................................................................................ 1331
Enhanced CPUID Instruction ............................................................................................ 1332
The SSE2 Instruction Set .................................................................................................... 1332
  General........................................................................................................................ ..... 1332
  DP FP Number Representation.................................................................................... 1334
  Packed and Scalar DP FP Instructions .................................................................... 1334
  SSE2 64-Bit and 128-Bit SIMD Integer Instructions................................................... 1335
  SSE2 128-Bit SIMD Integer Instruction Extensions ................................................... 1335
Your Choice: Accuracy or Speed ................................................................................. 1336
The SSE3 Instruction Set .................................................................................................... 1337
Local APIC Enhancements ................................................................................................ 1338
The Thermal Monitoring Facilities ................................................................................. 1340
  Introduction to Thermal Monitoring ................................................................. 1340
  Thermal Monitor Feature Detection ...................................................................... 1340
  Stop Clock Acts as a Gate for the Processor Clock............................................ 1340
  Catastrophic Shutdown Detector ........................................................................... 1341
  Automatic Thermal Monitoring ............................................................................ 1342
  Thermal Monitor and Interrupts ............................................................................ 1343
    Interrupts Are Blocked While Stop Clock Is Low............................................ 1343
    The Thermal Monitor Interrupt ....................................................................... 1343
  Software Controlled Clock Modulation ............................................................... 1344
Contents

Relationship of the Hardware- and Software-Based Mechanisms ......................... 1345
HTT and Thermal Monitoring ..................................................................................... 1345

FPU Enhancement ........................................................................................................ 1345
General ........................................................................................................................ ..... 1345
Fopcode Compatibility Mode ...................................................................................... 1346
The MSRs .............................................................................................................................. 1347

The Machine Check Architecture ..................................................................................... 1363
Introduction ....................................................................................................................1 3 6 3
The Pentium® 4 MCA Enhancements ........................................................................... 1363
The Extended MC State MSRs ..................................................................................... 1364

Last Branch, Interrupt, and Exception Recording .......................................................... 1365

The Debug Store (DS) Mechanism.............................................................................. 1366
Introduction ....................................................................................................................1 3 6 6
Feature Detection ........................................................................................................... 1367
Setting Up the DS Feature............................................................................................. 1367
Enabling the BTS Feature ........................................................................................... 1369
Enabling the PEBS Feature ........................................................................................... 1370
The PEBS Record Format .............................................................................................. 1370
The BTS Record Format .............................................................................................. 1370

New Exceptions .................................................................................................................... 1371

The Performance Monitoring Facility .............................................................................. 1371
Performance Monitoring Is Not Architecturally Defined ........................................ 1371
Author’s Note ................................................................................................................. 1371
An Overview ................................................................................................................... 1372
There Are Two Event Categories ................................................................................. 1373
There Are Three Sampling Methods ........................................................................... 1374
Relationship of a Counter, Its CCCR and the ESCRs ............................................... 1375
The Event Select Control Registers ............................................................................ 1376
The Counter Configuration Control Registers .......................................................... 1382

General ..................................................................................................................... 1397
Counter Cascading ................................................................................................. 1401
Interrupt on Overflow............................................................................................ 1402
Extended Cascading ............................................................................................... 1406
Accessing the Performance Counters ......................................................................... 1406
Halting Event Counting............................................................................................ 1407
Non-Retirement Event Counting................................................................................ 1407
Introduction................................................................................................................ 1407
The Set Up .................................................................................................................. 1408
The Event Filtering Mechanism ................................................................................ 1408
Introduction................................................................................................................ 1408
Threshold Comparison ......................................................................................... 1409
The Threshold Condition Transition Filter .............................................................. 1409
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>At-Retirement Event Counting</td>
<td>1409</td>
</tr>
<tr>
<td>First, Some Terminology</td>
<td>1409</td>
</tr>
<tr>
<td>Bogus, Non-Bogus, Retire</td>
<td>1409</td>
</tr>
<tr>
<td>Tagging</td>
<td>1409</td>
</tr>
<tr>
<td>Replay</td>
<td>1410</td>
</tr>
<tr>
<td>Assist</td>
<td>1410</td>
</tr>
<tr>
<td>General</td>
<td>1410</td>
</tr>
<tr>
<td>The Tagging Mechanisms</td>
<td>1411</td>
</tr>
<tr>
<td>Introduction</td>
<td>1411</td>
</tr>
<tr>
<td>Multi-Tagging</td>
<td>1411</td>
</tr>
<tr>
<td>PEBS and Multi-Tagging</td>
<td>1411</td>
</tr>
<tr>
<td>Some µops Cannot Be Tagged</td>
<td>1411</td>
</tr>
<tr>
<td>Front-End Tagging</td>
<td>1411</td>
</tr>
<tr>
<td>Execution Tagging</td>
<td>1412</td>
</tr>
<tr>
<td>Replay Tagging</td>
<td>1413</td>
</tr>
<tr>
<td>Precise Event-Based Sampling</td>
<td>1414</td>
</tr>
<tr>
<td>General</td>
<td>1414</td>
</tr>
<tr>
<td>Limited To a Single Counter</td>
<td>1415</td>
</tr>
<tr>
<td>Detecting the PEBS Capability</td>
<td>1415</td>
</tr>
<tr>
<td>Enabling PEBS</td>
<td>1415</td>
</tr>
<tr>
<td>The PEBS Interrupt Handler</td>
<td>1415</td>
</tr>
<tr>
<td>Sometimes, the DS Feature Is Disabled</td>
<td>1415</td>
</tr>
<tr>
<td>PEBS and Hyper-Threading</td>
<td>1416</td>
</tr>
<tr>
<td>Counting Clocks</td>
<td>1417</td>
</tr>
<tr>
<td>Introduction</td>
<td>1417</td>
</tr>
<tr>
<td>The Non-Halted Clockticks Measurement</td>
<td>1418</td>
</tr>
<tr>
<td>The Non-Sleep Clockticks Measurement</td>
<td>1419</td>
</tr>
<tr>
<td>The Time Stamp Counter</td>
<td>1419</td>
</tr>
</tbody>
</table>

### Chapter 57: Pentium® 4 Xeon Features

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>1422</td>
</tr>
<tr>
<td>The Pentium® 4 Xeon DP</td>
<td>1422</td>
</tr>
<tr>
<td>The Pentium® 4 Xeon MP</td>
<td>1422</td>
</tr>
</tbody>
</table>

### Part 11: Pentium® M

### Chapter 58: Pentium® M Processor

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>1426</td>
</tr>
<tr>
<td>The Pentium® M and Centrino</td>
<td>1426</td>
</tr>
<tr>
<td>Characteristics Overview</td>
<td>1427</td>
</tr>
</tbody>
</table>
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The FSB Characteristics</td>
<td>1427</td>
</tr>
<tr>
<td>Uses the Pentium® 4 FSB Protocol</td>
<td>1427</td>
</tr>
<tr>
<td>Pentium® M-Specific Signals</td>
<td>1428</td>
</tr>
<tr>
<td>FSB Power Utilization Enhancements</td>
<td>1429</td>
</tr>
<tr>
<td>Enhanced Power Management Characteristics</td>
<td>1429</td>
</tr>
<tr>
<td>Background</td>
<td>1429</td>
</tr>
<tr>
<td>Entry to the Deep Sleep State</td>
<td>1429</td>
</tr>
<tr>
<td>The Deeper Sleep State</td>
<td>1430</td>
</tr>
<tr>
<td>Enhanced SpeedStep</td>
<td>1433</td>
</tr>
<tr>
<td>Background</td>
<td>1433</td>
</tr>
<tr>
<td>Enhanced SpeedStep Description</td>
<td>1433</td>
</tr>
<tr>
<td>Three Different Packaging Models</td>
<td>1435</td>
</tr>
<tr>
<td>Improved Thermal Monitor Mode</td>
<td>1435</td>
</tr>
<tr>
<td>Enhanced Branch Prediction</td>
<td>1436</td>
</tr>
<tr>
<td>Introduction</td>
<td>1436</td>
</tr>
<tr>
<td>The Loop Detector</td>
<td>1436</td>
</tr>
<tr>
<td>The Indirect Branch Predictor</td>
<td>1436</td>
</tr>
<tr>
<td>The Problem</td>
<td>1436</td>
</tr>
<tr>
<td>Indirect Branch Predictor Description</td>
<td>1436</td>
</tr>
<tr>
<td>µop Fusion</td>
<td>1437</td>
</tr>
<tr>
<td>Background</td>
<td>1437</td>
</tr>
<tr>
<td>µop Fusion Description</td>
<td>1438</td>
</tr>
<tr>
<td>General</td>
<td>1438</td>
</tr>
<tr>
<td>The Fused Store</td>
<td>1438</td>
</tr>
<tr>
<td>The Fused Load and Operate</td>
<td>1438</td>
</tr>
<tr>
<td>Advanced Stack Management</td>
<td>1439</td>
</tr>
<tr>
<td>Background</td>
<td>1439</td>
</tr>
<tr>
<td>Advanced Stack Management Description</td>
<td>1439</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>1440</td>
</tr>
<tr>
<td>Hardware-Based Data Prefetcher</td>
<td>1440</td>
</tr>
<tr>
<td>The L2 Cache</td>
<td>1440</td>
</tr>
<tr>
<td>The Data Cache and Hyper-Threading</td>
<td>1440</td>
</tr>
<tr>
<td>The Next Pentium® M</td>
<td>1440</td>
</tr>
</tbody>
</table>

**Part 12: Additional Topics**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 59: CPU Identification</td>
<td>1444</td>
</tr>
<tr>
<td>Prior to the Advent of the CPUID Instruction</td>
<td>1444</td>
</tr>
<tr>
<td>Determining if the CPUID instruction Is Supported</td>
<td>1444</td>
</tr>
<tr>
<td>General</td>
<td>1446</td>
</tr>
<tr>
<td>Determining the Request Types Supported</td>
<td>1446</td>
</tr>
<tr>
<td>Determining Basic Request Types Supported</td>
<td>1446</td>
</tr>
</tbody>
</table>
Determining Extended Request Types Supported .......................................................... 1446

The Basic Request Types ............................................................................................... 1446
  Request Type 1 ............................................................................................................. 1446
  General ....................................................................................................................... 1446
  The Brand Index ....................................................................................................... 1447
  Request Type 2 ............................................................................................................. 1452
  General ....................................................................................................................... 1452
  Request for Cache and TLB Information ...................................................................... 1452
  Request Type 3 ............................................................................................................. 1457
  Request Type 4 ............................................................................................................. 1457
  Request Type 5 ............................................................................................................. 1458

The Extended Request Types .......................................................................................... 1459

Enhanced Processor Signature ....................................................................................... 1460

Chapter 60: System Management Mode (SMM)

What Falls Under the Heading of System Management? ................................................. 1465

The Genesis of SMM ....................................................................................................... 1465

SMM Has Its Own Private Memory Space ....................................................................... 1466

The Basic Elements of SMM .......................................................................................... 1466

A Very Simple Example Scenario .................................................................................. 1467

How the Processor Knows the SM Memory Start Address ............................................. 1468

Protected Mode, Paging and PAE-36 Mode Are Disabled ............................................. 1468

The Organization of SM RAM ....................................................................................... 1468

Entering SMM .................................................................................................................. 1473

The SMI Interrupt Is Generated ..................................................................................... 1473

No Interruptions Please .................................................................................................... 1474

General .......................................................................................................................... 1474

Exceptions and Software Interrupts Permitted but Not Recommended ......................... 1474

Servicing Maskable Interrupts While in the Handler ....................................................... 1475

Single-Stepping through the SM Handler ....................................................................... 1475

If Interrupts/Exceptions Permitted, Build an IDT ........................................................ 1475

SMM Uses Real Mode Address Formation ..................................................................... 1475

NMI Handling While in SMM ......................................................................................... 1476

Default NMI Handling ................................................................................................... 1476

How to Re-Enable NMI Recognition in the SM Handler .............................................. 1477

If an SMI Occurs within the NMI Handler ..................................................................... 1477

Informing the Chipset That SMM Has Been Entered ................................................... 1478

General .......................................................................................................................... 1478

A Note Concerning Memory-Mapped IO Ports ............................................................. 1478

The Context Save ......................................................................................................... 1478

General .......................................................................................................................... 1478

Although Saved, Some Register Images Are Forbidden Territory ................................ 1479

Special Actions Required on a Request for Power Down ............................................. 1479
Contents

The Register Settings on Initiation of the SM Handler ............................................. 1480
The SMM Revision ID .......................................................................................... 1482
The Body of the Handler ................................................................................. 1482
Exiting SMM ............................................................................................................. 1483
The Resume Instruction .................................................................................... 1483
Informing the Chipset That SMM Has Been Exited.......................................... 1483
The Auto Halt Restart Feature .......................................................................... 1484
Executing the HLT Instruction in the SM Handler ........................................... 1485
The IO Instruction Restart Feature ...................................................................... 1485
Introduction ......................................................................................................... 1485
An Example Scenario ......................................................................................... 1486
The Detail ............................................................................................................ 1486
Back-to-Back SMIs During IO Instruction Restart ........................................ 1486
Caching from SM Memory .................................................................................... 1487
Background........................................................................................................... 1487
The Physical Mapping of SM RAM Accesses .................................................. 1488
FLUSH# and SMI# .............................................................................................. 1493
Description........................................................................................................... 1493
A Cautionary Note Regarding the Pentium® .................................................. 1494
Setting Up the SMI Handler in SM Memory ..................................................... 1494
Relocating the SM RAM Base Address ............................................................. 1495
Description........................................................................................................... 1495
In an MP System, Each Processor Must Have a Separate State Save Area........ 1495
Accessing SM RAM Above the First MB .......................................................... 1496
SMM in an MP System ........................................................................................ 1496

Chapter 61: The Local and IO APICs
Before the Advent of the APIC............................................................................ 1498
MP Systems Need a Better Interrupt Distribution Mechanism.......................... 1501
Introduction ......................................................................................................... 1501
The APIC Interrupt Distribution Mechanism .................................................... 1502
Introduction ......................................................................................................... 1502
Message Transfer Mechanism Prior to the Pentium® 4 .................................... 1503
Message Transfer Mechanism Starting with the Pentium® 4 ............................ 1503
Inter-Processor Interrupt Messages ................................................................. 1503
Local Interrupts .................................................................................................... 1503
NMI, SMI and Init Messages ............................................................................. 1504
The Cluster and APIC ID ................................................................................... 1504
Physical Destination Mode ................................................................................ 1505
Logical Destination Mode .................................................................................. 1505
A Short History of the APIC ............................................................................... 1507
The Introduction of the APIC............................................................................. 1507
Contents

Pentium® Pro APIC Enhancements ................................................................. 1507
The Pentium® II and Pentium® III ............................................................. 1508
Pentium® 4 APIC Enhancements ................................................................. 1508
Detecting the Presence and Version of the Local APIC ............................ 1509
Enabling/Disabling the Local APIC ............................................................. 1510
General ........................................................................................................ 1510
Permanently Disabling the Local APIC .................................................... 1510
Temporarily Disabling the Local APIC ..................................................... 1511
Operational Characteristics of a Disabled Local APIC ............................ 1512
Local Cluster and APIC ID Assignment .................................................... 1513
Cluster ID Assignment ............................................................................. 1513
APIC ID Assignment .............................................................................. 1514
Maximum Number of Local APICs ........................................................ 1515
BIOS/OS Reassignment of Local APIC IDs ............................................ 1515
The Local APIC IDs Are Stored in the MP and ACPI Tables ................ 1516
Reading the Local APIC ID ..................................................................... 1516
An Introduction to the Interrupt Sources .................................................. 1516
Local Interrupt Sources .......................................................................... 1517
Remote Interrupt Sources ........................................................................ 1517
Introduction to Interrupt Priority ............................................................... 1517
General ........................................................................................................ 1517
Definition of a User-Defined Interrupt ................................................... 1518
The Priority Amongst the User-Defined Interrupts .................................. 1519
Definition of Fixed Interrupts ................................................................. 1522
Masking User-Defined Interrupts .............................................................. 1522
An Intro to Edge-Triggered Interrupts ....................................................... 1522
An Intro to Level-Sensitive Interrupts ....................................................... 1523
The Local APIC Register Set .................................................................... 1524
Local and IO APIC Register Areas Are Uncacheable ............................... 1524
Introduction to the Local APIC’s Register Set ........................................ 1524
The IRR, TMR and ISR Registers ............................................................ 1535
General ........................................................................................................ 1535
An Example .............................................................................................. 1536
The EOI and Its Effects ............................................................................ 1537
Interrupt Request Buffering ..................................................................... 1537
Register Access Alignment ...................................................................... 1539
Locally Generated Interrupts ..................................................................... 1539
Introduction .............................................................................................. 1539
The Local Vector Table ........................................................................... 1539
The Pentium® Family’s LVT ................................................................. 1540
The P6 Family’s LVT ............................................................................... 1540
The Pentium® 4 Family’s LVT ................................................................. 1540
## Contents

Local Interrupt 0 (LINT0) ................................................................. 1540  
  Introduction ....................................................................................... 1540  
  The Mask Bit ...................................................................................... 1541  
  The Trigger Mode and the Input Pin Polarity .................................... 1541  
  The Delivery Mode ........................................................................... 1541  
  The Vector Field ............................................................................... 1543  
  The Remote IRR Bit .......................................................................... 1543  
  The Delivery Status .......................................................................... 1543  

Local Interrupt 1 (LINT1) ................................................................. 1543  
  The Local APIC Timer ........................................................................ 1544  
  General ............................................................................................... 1544  
  The Divide Configuration Register .................................................. 1545  
  One Shot Mode .................................................................................. 1545  
  Periodic Mode ................................................................................... 1545  
  The Performance Counter Overflow Interrupt ................................ 1547  
  The Thermal Sensor Interrupt .......................................................... 1548  
  The Local APIC's Error Interrupt ...................................................... 1549  

### Task and Processor Priority .............................................. 1551  
  Introduction ...................................................................................... 1551  
  The Task Priority Register (TPR) ................................................... 1552  
  The Processor Priority Register (PPR) ............................................. 1552  
  The User-Defined Interrupt Eligibility Test .................................... 1553  

### Interrupt Messages .............................................................. 1555  
  Introduction ...................................................................................... 1555  
  Sending a Message From the Local APIC ....................................... 1556  
  Physical Destination Mode .............................................................. 1562  
  Logical Destination Mode ................................................................. 1562  
    Introduction .................................................................................. 1562  
    The Flat Model .............................................................................. 1563  
    The Cluster Model ....................................................................... 1563  
      The Flat Cluster Model ................................................................. 1563  
      The Hierarchical Cluster Model ................................................ 1564  
  Lowest-Priority Delivery Mode ...................................................... 1565  
    General ........................................................................................... 1565  
    Chipset-Assisted Lowest-Priority Delivery .................................. 1565  

### The IO APIC ................................................................. 1567  
  The Purpose of the IO APIC ............................................................ 1567  
  Overview of an Edge-Triggered Interrupt Delivery ...................... 1569  
  Overview of a Level-Sensitive Interrupt Delivery ......................... 1571  
  The IO APIC Register Set ................................................................. 1571  
    The IO APIC Register Set Base Address ...................................... 1573  
    The IO APIC Register Set .............................................................. 1573
## Contents

The IRQ Pin Assertion Register ................................................................. 1577  
The EOI Register .................................................................................. 1578  
Non-Shareable IRQ Lines .................................................................. 1578  
Shareable IRQ Lines ......................................................................... 1578  
Linked List of Interrupt Handlers ..................................................... 1578  
How It Works ....................................................................................... 1578  
The ID Register .................................................................................. 1581  
The Version Register ....................................................................... 1581  
The Redirection Table Register Set ................................................... 1581  
Interrupt Delivery Order Is Rotational ............................................. 1584  

**Message Signaled Interrupts (MSI)** .............................................. 1584  
General .............................................................................................. 1584  
Using the IO APIC as a Surrogate Message Sender ......................... 1585  
Direct-Delivery of the MSI ................................................................. 1586  
Memory Already Sync’d When Interrupt Handler Entered .......... 1590  
The Problem ....................................................................................... 1590  
The Old Way of Solving the Problem ............................................. 1590  
How MSI Solves the Problem ......................................................... 1590  

**Message Format** .......................................................................... 1591  
The FSB Message Format ................................................................. 1591  
The APIC Bus Message Format ....................................................... 1591  

**The Spurious Interrupt Vector** ..................................................... 1591  
The Problem ....................................................................................... 1591  
The Solution ....................................................................................... 1592  
Additional Spurious Vector Register Features ............................... 1592  

**The Agents in an Interrupt Message Transaction** ....................... 1593  
MCH Initiates an Interrupt Message Transaction ......................... 1593  
A Local APIC Initiates an Interrupt Message Transaction ........... 1594  

**BSP Selection Process** ................................................................. 1595  
Introduction to the BSP Selection Process ..................................... 1595  
The P6 Family BSP Selection Process ........................................... 1596  
The Pentium® 4 Family BSP Selection Process ......................... 1596  

**The APIC, the MPS and ACPI** ..................................................... 1597  

**Acronyms** .................................................................................... 1599  

**Index** .......................................................................................... 1619
1 Overview of the Processor Role

This Chapter

In order to have a full and complete understanding of a device, one must have a clear view of how it fits into the overall context. In the case of the processor, this means having an understanding of its role in the overall system and how it interacts with the overall machine environment. This chapter is intended to introduce the context that the processor exists in and interacts with.

The Next Chapter

As background material, this chapter provides a basic description of the single-task OS and application environment.

The IA32 Specification

The IA32 specification (also sometimes referred to as the IA32 ISA (Instruction Set Architecture) is comprised of the three volume documentation set consisting of:


Much of the actual processor implementation is outside the scope of the specification. The following are some examples:

- Whether or not a processor implements any caches and, if so, the number of, size of, and architecture of the caches is processor design-specific.
The Unabridged Pentium® 4

- Whether or not a processor implements any special-purpose caches to accelerate Paging and, if so, the number of, size of, and architecture of those caches is processor design-specific.
- The number of execution units is processor design-specific.
- The type of bus that connects the processor to the system is processor design-specific.

IA32 Processors

As used in this book, the term IA32 processors refers to all Intel® x86 processors starting with the first 32-bit processor, the 386, and ending with the Pentium® 4 and Pentium® M processors. The author makes this distinction because, while some Intel® documentation also includes the 8088 and 8086 processors in the IA32 category, they were not 32-bit processors.

IA32 Instructions vs. µops

The IA32 instructions are variable length instructions and, depending on the type of instruction and the number of special prefixes that precede the instruction, can be anywhere from one to 15 bytes in length. While all IA32 processors up to and including the Pentium® processor executed these instructions, all IA32 processors starting with the Pentium® Pro translate the IA32 instructions into primitive fixed-length instructions prior to executing them. These instructions are referred to as micro-ops, or µops. The number of µops that a single IA32 instruction translates into is specific to the type of IA32 instruction.

Processor = Instruction Fetch/Decode/Execute Engine

The processor’s role in the system is really quite simple: it is an engine designed to fetch instructions from memory, decode them, and execute them. Figure 1-1 on page 13 illustrates a minimalist processor design. It consists of the following entities:

- **Instruction Fetcher.** The fetcher is responsible for tracking where (in memory) the next instruction is to read from. It issues memory read transaction requests to the Front Side Bus (FSB) Interface Unit which then reads the instructions from system memory.
- **Instruction Decoder.** It is responsible for decoding the instructions fetched from memory. The decoder translates the instructions into a form that can be directly executed by the execution unit.
Chapter 1: Overview of the Processor Role

- **Instruction Dispatch Unit.** When one or more instructions requested by the Instruction Fetcher are returned from memory by the FSB Interface Unit, the dispatcher routes the instruction(s), one at time, to the instruction execution unit for execution.

- **Instruction Execution Unit.** It executes the instructions. In the course of doing so, it accesses the processor’s internal register set to obtain operands upon which the instruction acts. In addition, depending on the instruction type, it may cause the FSB Interface Unit to perform a transaction on the FSB. Some example cases wherein a transaction may have to be performed on the FSB are:
  - In order to execute an IO instruction (IN, OUT, INS, or OUTS), the processor must perform one or more transactions on the FSB.
  - If one of the operands that an instruction operates upon is in memory, a memory transaction must be performed on the FSB in order to access the memory-based operand.

- **Processor register set.** The register set basically consists of two groups of registers:
  - General Purpose Registers (GPRs) used by the currently executing program to examine and/or manipulate the data items being acted upon by the program.
  - Control and status registers used to control the processor’s fundamental behavior and to indicate the current state of the processor.

- **Front Side Bus (FSB) Interface Unit.** Upon receipt of a request to perform an access to an external memory or IO device, it arbitrates for ownership of the FSB and, upon gaining bus ownership, performs the requested transaction. If it’s a read transaction, the requested read data returned from memory or from an IO device is routed to the processor entity (e.g., the Instruction Fetcher) that requested the data.

Some Instructions Result in FSB Transactions

The execution of an instruction may or may not necessitate the performance of a transaction on the FSB. The following subsections introduce the various scenarios.

Many Instructions Do Not Require FSB Transactions

Many instructions perform an operation on the data currently contained in one or more processor registers and do not require the performance of a transaction on the processor’s FSB.
Instructions That Do Require FSB Transactions

IO Read and Write

Execution of an IO read or write instruction causes the execution unit to issue an IO read or IO write transaction request to the FSB Interface Unit. Ownership of the FSB is then requested and, when obtained, the IO read or IO write transaction is performed on the FSB.

**IO Read Instruction.** On an IO read, the FSB Interface Unit initiates the IO read transaction and the transaction cannot be completed until the requested read data is returned by the targeted external IO register. The external IO register addressed by the transaction eventually returns the requested read data in the Data Phase of the transaction. The read data obtained by the FSB Interface Unit is then routed to the Execution Unit and is placed in the General Purpose Register identified by the IO read instruction. That completes the execution of the instruction and the Execution Unit obtains the next instruction from the Dispatch Unit.

**IO Write Instruction.** On an IO Write, the FSB Interface Unit initiates the IO write transaction and drives the write data onto the data bus. The transaction is not completed until the external IO register addressed in the transaction accepts the write data. Once the data has been accepted, the FSB Interface Unit signals completion to the processor’s Execution Unit. That completes the execution of the instruction and the Execution Unit obtains the next instruction from the Dispatch Unit.

Memory Data Read

Some instructions require that a data item be read from external memory (or from a memory-mapped IO register). In this case, the Execution Unit issues a memory read transaction request to the FSB Interface Unit. The FSB Interface Unit initiates the memory read transaction and the transaction cannot be completed until the requested read data is returned by the targeted memory. The memory addressed by the transaction eventually returns the requested read data in the Data Phase of the transaction. The read data obtained by the FSB Interface Unit is then routed to the Execution Unit and is acted upon by the Execution Unit. That completes the execution of the instruction and the Execution Unit obtains the next instruction from the Dispatch Unit.
Chapter 1: Overview of the Processor Role

Memory Data Write

Some instructions require that a data value be written to an external memory location (or to a memory-mapped IO register). In this case, the Execution Unit issues a memory write transaction request to the FSB Interface Unit. The FSB Interface Unit initiates the memory write and drives the write data onto the data bus. Once the data has been accepted, the FSB Interface Unit signals completion to the processor’s Execution Unit. That completes the execution of the instruction and the Execution Unit obtains the next instruction from the Dispatch Unit.

Memory Instruction Read

The instruction pointer points to the next instruction in the currently executing program. The Instruction Fetcher issues a memory instruction read request to the FSB Interface Unit to fetch the next instruction from the memory address indicated by the instruction pointer. Using a memory instruction read transaction, the instruction is read from memory and is provided to the Instruction Dispatch Unit. As each instruction fetch is completed, the Instruction Fetcher auto-increments the instruction pointer to point to the next instruction in the currently executing program.

Figure 1-1: The Bare Bones Processor Is an Instruction Fetch, Decode, Execution Engine

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>FSB Transaction required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO Read</td>
<td>Yes; an IO Read transaction.</td>
</tr>
<tr>
<td>IO Write</td>
<td>Yes; an IO Write transaction.</td>
</tr>
<tr>
<td>Instruction requiring data from memory</td>
<td>Yes; a Memory Data Read transaction.</td>
</tr>
<tr>
<td>Instruction that must store data in memory</td>
<td>Yes; a Memory Data Write transaction.</td>
</tr>
</tbody>
</table>
2 Single-Task OS
and Application

The Previous Chapter
In order to have a full and complete understanding of a device, one must have a clear view of how it fits into the overall context. In the case of the processor, this means having an understanding of its role in the overall system and how it interacts with the overall machine environment. This chapter is intended to introduce the context that the processor exists in and interacts with.

This Chapter
As background material, this chapter provides a basic description of the single-task OS and application environment.

The Next Chapter
As background material, this chapter provides a very basic introduction to the multitask OS environment.

Operating System Overview
A single-task OS (e.g., MS DOS) basically consists of the following components:

- The command line interpreter (CLI).
- The program loader.
- The OS services.
Command Line Interpreter (CLI)

Once the OS has been loaded into memory by the startup firmware, control is passed to the OS initialization code which sets up any necessary data structures (e.g., the Interrupt Table) in memory, loads and initializes device drivers, etc., and then passes control to the CLI.

The CLI issues a prompt to the user requesting that the user identify the program to be run. The exact form that the prompt takes and the method utilized to make a selection is OS-dependent. In the case of DOS’s COMMAND.COM CLI, the prompt was not very user-friendly(>:. In response to the prompt, the user keys in the name of a program to be executed. In the case of DOS DOSSHELL, the user used the mouse to point and click on a file name.

Program Loader

Once the user selects a file name:

1. The OS reads the file’s directory entry and ascertains the amount of RAM memory necessary to hold the program. The OS locates a block of free (i.e., unused) memory into which it can load the program.
2. The OS either directly accesses the disk controller to initiate the read, or issues a disk read request to the disk BIOS routine in system memory or to the disk device driver. The BIOS routine or driver issues the request to the disk controller.
3. If the disk-to-memory transfer will be performed by the DMA controller, the BIOS routine or driver programs the disk controller’s associated DMA channel to transfer the data into the target memory. If the disk controller has bus master capability, the BIOS routine or driver programs the disk controller to transfer the data directly into the target memory.
4. The DMA controller or bus master-capable disk controller transfers the block of information into memory.
5. The disk controller then informs the BIOS or driver that the transfer has been completed. To do so, the disk controller generates its device-specific interrupt request, causing the processor to jump to the disk interrupt service routine.
6. The service routine checks the disk controller’s completion status to ensure that no errors were incurred during the transfer of the information into memory.
Chapter 2: Single-Task OS and Application

7. The service routine returns a good completion to the BIOS or driver and a good completion is returned to the OS.
8. Upon ascertaining that the program has been transferred into memory, the OS executes a far jump instruction to the program’s entry point (in a far jump instruction, the programmer specifies a target location in a different code segment; in a near jump instruction, the programmer specifies a target location in the same code segment). The application program then begins execution.

OS Services

In the course of accomplishing its task, the application program may have to communicate with a number of devices in the system. It may have to read/write disk files, perform data communications, interface with the display and keyboard, etc.

Rather than force the author of every application program to write routines to interface with these entities, the OS provides a variety of services to the application program. When the programmer wishes to establish a communications channel that can be used to access a disk file, for instance, he or she issues a “file open” request to the OS. The OS performs this function for the programmer. When the programmer needs to change the appearance of the display, a request can be issued to the OS. In short, the OS provides a toolbox of services useful to the application program. This increases the productivity of the application programmer by lessening the amount of code to be written. It also renders the application program platform hardware-independent (because it doesn’t communicate directly with the devices).

Direct IO Access

In order to achieve better performance, application programs sometimes access IO ports directly (rather than going through the OS services). As a side effect, this renders the program much more platform design-dependent. In addition, the OS is left outside the loop, so it doesn’t always “know” the current state of an IO device. In a single-task OS environment this usually will not cause problems because the OS only starts one application program at a time and lets it run to completion before starting another. Because an application program can manipulate IO ports directly, application programs (and the OS) cannot make any assumptions about the current state of an IO device when they begin execution, but must always initialize all of the device’s IO registers to a known state during each session.
Application Program Memory Usage

Because a single-task OS only runs one program at a time, there is no need to protect application programs from invading each other’s memory space. As long as the application program doesn’t trash itself or the OS that gave birth to it and nurtures it, everything should be fine.

Task Initiation, Execution and Termination

Figure 2-1 on page 26 illustrates (in an albeit primitive manner) the application program’s dependence on the OS while it’s executing. The OS loads the task (i.e., application program) into memory and executes it. While executing, the task may issue calls to the OS requesting performance of various functions. Upon completion, the task returns control back to the OS. The OS then deallocates the memory used by the program and prompts the user for the name of another program to be executed.

Figure 2-1: Task/OS Relationship
3 Definition of Multitasking

The Previous Chapter
As background material, this chapter provided a basic description of the single-task OS and application environment.

This Chapter
As background material, this chapter provides a very basic introduction to the multitask OS environment.

The Next Chapter
As background material, this chapter provides a very basic introduction to the problems that a multitask OS must be prepared to deal with.

Concept
It is incorrect to say that a multitasking OS runs multiple programs (i.e., tasks) simultaneously. In reality, it loads a task into memory, permits it to run for a while and then suspends it. It suspends the program by creating a snapshot, or image, of all or many of the processor’s registers in memory. In the IA32 architecture, the image is stored in a special data structure in memory referred to as a Task State Segment (TSS) and is accomplished by performing an automatic series of memory write transactions. In other words, the exact state of the processor at the point of suspension is saved in memory.

Having effectively saved a snapshot that indicates the point of suspension and the processor’s complete state at the time, the processor then initiates another task by loading it into memory and jumping to its entry point. Based on some OS-specific criteria, the OS at some point makes the decision to suspend this
task as well. As before, the state of the processor is saved in memory (in this task’s TSS) as a snapshot of the task’s state at its point of suspension.

At some point, the OS makes the decision to resume a previously-suspended task. This is accomplished by reloading the processor’s registers from the previously-saved register image (i.e., its TSS) by performing a series of memory read transactions. The processor then uses the address pointer stored in the CS:EIP register pair to fetch the next instruction, thereby resuming program execution at the point where it had been suspended earlier.

The criteria that an OS uses in making the decision to suspend a program is specific to that OS. It may simply use timeslicing—each program is permitted to execute for a fixed amount of time (e.g., 10ms). At the end of that period of time, the currently executing task is suspended and the next task in the queue is started or resumed. The OS may assign priority levels to programs, thereby permitting a higher priority program to “preempt” a lower priority program that may currently be running. This is referred to as preemptive multitasking. The OS would also choose to suspend the currently executing program if the program needs something that is not immediately available (e.g., when it attempts an access to a page of information that is currently not in memory, but resides on a mass storage device).

An Example—Timeslicing

Prior to starting or resuming execution of a task, the OS task scheduler would initialize a hardware timer to interrupt program execution after a defined period of time (e.g., 10ms). The scheduler then starts or resumes execution of the task. The processor proceeds to fetch and execute the instructions comprising the task for 10ms. When the hardware timer expires it generates an interrupt, causing the processor to suspend execution of the currently executing task and to switch to the OS’s task scheduler. The OS determines which task to run next.

Another Example—Awaiting an Event

Task Issues Call to OS for Disk Read

The application program calls the OS requesting that a block of data be read from a disk drive into memory. Once a disk read request is forwarded to the disk interface, the disk read/write head mechanism must be positioned over
the target disk cylinder. This is a lengthy mechanical process typically requiring milliseconds to complete. When the head mechanism has positioned the read/write heads over the target cylinder, the disk interface must then wait for the start sector of the requested block to be presented under the read head. The duration of this delay is defined by the rotational speed of the disk drive as well as the circumference of the cylinder. Once again, this is a lengthy delay that can be measured in milliseconds. Only then can the data transfer begin.

Rather than awaiting the completion of the disk read, the OS would better utilize the machine’s resources by suspending the task that originated the request and transferring control to another program so work can be accomplished while the disk operation is in progress.

**OS Suspends Task**

As described earlier, the processor saves its current state (i.e., its register image) in a special area of memory set aside for this task (the task’s TSS). Once this series of memory write transactions has completed, the task has been suspended.

**OS Initiates Disk Read**

The OS issues a disk read command to the disk controller. The disk controller begins to seek the heads to the target cylinder.

**OS Makes Entry in Event Queue**

The OS makes an entry in its event queue. This entry will be used to transfer control back to the suspended task when the disk interface completes the transfer.

**OS Starts or Resumes Another Task**

Rather than waiting for the completion of the disk read operation, the OS will start or resume another task.
Disk-Generated Interrupt Causes Jump to OS

When the disk controller (or, in older machines, its associated DMA channel) completes the transfer of the requested information into system memory, it generates an interrupt request. This causes the processor to jump to the disk driver’s interrupt service routine which checks the completion status of the disk operation to ensure a good completion.

Task Queue Checked

The OS then scans the event queue to determine which suspended task is awaiting this completion notification.

OS Resumes Task

The OS causes the processor to reload the suspended task’s stored register image (its TSS) into the processor’s registers. The processor then uses CS:EIP to determine what memory address to fetch its next instruction from. The resumed task then processes the data in memory that was read from the disk.
4 Multitasking Problems

The Previous Chapter

As background material, this chapter provided a very basic introduction to the multitask OS environment.

This Chapter

As background material, this chapter provides a very basic introduction to the problems that a multitask OS must be prepared to deal with.

The Next Chapter

This chapter provides a detailed description of the processor’s operation when in Real Mode. This description also applies to all IA32 processors subsequent to the 386 processor.

OS Protects Territorial Integrity

The multitasking OS loads multiple tasks into different areas of memory and permits each to run for a slice of time. As described in the previous chapter, it permits a task to run for a timeslice, suspends it, permits another task to run for a timeslice, suspends it, etc. If the OS is executing on a fast processor with fast access to memory, this task switching can be accomplished so quickly that all of the tasks appear to be executing simultaneously.

While the processor is executing a task, the OS kernel and all of the other dormant tasks are resident in memory. As each of the tasks (and the OS kernel) were suspended earlier in time, the processor created a snapshot of the processor’s register image in memory at the moment that task was suspended. In the IA32 environment, the OS sets up a separate Task State Segment (TSS) for each
task to be used during task switches. When it’s time to resume execution of a program, the processor can reload its register set from the task’s TSS and pick up right where it left off.

Stay in Your Own Memory Area

It’s obvious that the currently executing program utilizes certain areas of memory. Its program code resides in its code segment(s) within memory. Some of the data that it acts upon is stored within the processor’s registers and much of it in the areas of memory designated as its data segments. When the program needs to store the information from a register briefly so that it can use the register for something else, it typically stores the data in the area of memory designated as its stack segment.

The currently executing program is typically only aware of two entities—itself and the OS that created it. It is completely unaware of the existence of any other tasks that are currently suspended. The currently executing program should only access its own memory. If it were permitted to perform memory writes anywhere in memory, it is entirely probable that it will corrupt the code, stack or data areas of programs that reside in memory but are currently suspended. Consider what would happen when the OS resumes execution of a task that had been corrupted while in suspension. Its program and/or data would have been corrupted, causing it to behave unpredictably when it resumes execution.

The OS must protect suspended tasks (including itself!) from the currently executing task. If it doesn’t, multitasking will not work reliably.

IO Port Anarchy

Assume that the currently executing task needs to initiate a disk access. To do this directly, it would have to program the disk controller’s IO registers with the information defining the disk command type (e.g., disk read), the cylinder number, the head (i.e., surface) number, the start sector number and the number of sectors to be transferred. This is accomplished by executing a series of either memory-mapped write or IO write instructions that cause the processor to perform a series of memory or IO write transactions to transfer the command and associated parameters to the disk controller’s register set. Now assume that the task has programmed some, but not all of, the disk controller’s registers and the task’s timeslice expires. The OS suspends the current task and starts or resumes another task.
Chapter 4: Multitasking Problems

The new task, having no knowledge of tasks that are suspended, may decide that it also wants to issue a command to the disk controller. Assume that it does so and that the operation completes without error. Eventually, the OS suspends this task and reawakens the other task. When it resumes execution at the point of suspension, this task doesn’t know that it was put to sleep. In other words, it completes the series of memory or IO writes to transfer the remainder of the request parameters to the disk controller’s register set. It has no idea that the initial parameters that it sent to the disk controller (before the task was suspended) were overwritten by another task while it was asleep. The end result will be that this task’s disk operation will not occur correctly.

Generally speaking, the system’s IO devices should be considered a pool of shared resources to be managed by a central entity (the OS). Having one entity perform all communications with shared IO devices ensures that there will be no contention for IO devices between multiple tasks.

To accomplish this, the OS should not permit the tasks to talk directly to shared memory-mapped IO or IO ports that may result in problems such as that just mentioned. In other words, any attempt to execute an instruction that writes to one of these IO registers should cause the processor to trap (i.e., jump) to the OS. The OS then communicates with the IO device on behalf of the task.

The OS and/or processor could be configured to permit a task to access certain IO ports directly, but restrict access to other ports.

Unauthorized Use of OS’s Tools

The OS maintains the integrity of the system. It manages all shared resources and decides what task will run next and for how long. It should be fairly obvious that the person in charge must have more authority (greater privileges) than the other tasks currently resident in memory. It would be ill-conceived to permit normal tasks to access certain processor control registers, OS-related tables in memory, etc.

This can be accomplished in two ways: assignment of privilege levels to programs and assignment of ownership to areas of memory. The IA32 processors utilize both methods. There are four privilege levels:

- **Level zero.** Greatest amount of privilege. Assigned to the heart, or kernel, of the OS. It handles the task queues, memory management, etc.
- **Level one.** Typically assigned to OS services that provide services to the application programs and device drivers.
The Unabridged Pentium® 4

- **Level two.** Typically assigned to device drivers that the OS uses to communicate with peripheral devices.
- **Level three.** Assigned to application programs.

The application program operates at the lowest privilege level because its actions must be restricted. The OS has a very high privilege level so that it can accomplish its job of managing every aspect of the system. The integrity of the system would be compromised if an application program could call highly-privileged parts of the OS code to accomplish things it shouldn't be able to do. This implies that the processor must have some way of comparing the privilege level of the calling program to that of the program being called. To gain entry into the called program, the calling program's privilege level (CPL, or Current Privilege Level) must equal or exceed the privilege level of the program it is calling. IA32 processors incorporate this feature.

**No Interrupts, Please!**

An application program written to run under a single-tasking OS typically is master of all it surveys. It can communicate with any IO device, any memory location, disable interrupt recognition if it doesn't want to be interrupted, etc. In a single task environment, the program can disable recognition of interrupts if it will not adversely affect its own operation (the only program executing in the system).

If this same program is run under the management of a multitasking OS, however, it can cause severe problems. If permitted to execute a CLI (Clear Interrupt Enable) instruction, the EFlags[IF] bit is cleared to zero and, as result, the processor will no longer recognize interrupt requests originated by IO devices throughout the system. This means that these devices may not receive the servicing they require on a timely basis. As a result, they may suffer from buffer overflow or underflow conditions. This can result in anything from poor performance of a subsystem to completely flawed operation (data may be lost due to insufficient temporary buffer space within the subsystem). It should be noted that an IO device may generate an interrupt request to signal an event to another program that is currently suspended. The correct action may be for the processor to recognize the request, perform a task switch to the other program, service the request, and return to the interrupted task.

To summarize, the processor and the OS should not permit the application (written for a single-task OS environment) to execute the CLI instruction. An attempt to execute CLI should cause the processor to trap out to the OS. The OS would then set a bit indicating that this task prefers not to be interrupted. The
Chapter 4: Multitasking Problems

EFlags[IF] bit would not really be cleared, so the processor would still be able to recognize interrupt requests. The OS then resumes execution of the task. If an interrupt request is detected while this task is still executing, the processor jumps to a special routine to determine if this particular interrupt request is deemed important enough to interrupt the currently executing program. If not, the OS marks this request for subsequent servicing and resumes the interrupted task. The request is serviced after the current task has completed its time slice and has been suspended. If the request is considered important enough to be serviced immediately, the OS permits the processor to execute the IO device’s interrupt service routine and then resumes the interrupted task.

BIOS Calls

If an application program that was originally written to run under a single-tasking OS needs to communicate with an IO device, it may do this in one of the following manners:

- It can communicate with the device’s registers directly by executing an IN (IO read) or an OUT (IO write) instruction.
- It can communicate with the device’s registers directly by executing a memory read or a memory write instruction (if the device’s registers are mapped into memory rather than IO space).
- It can issue a request to the device’s BIOS routine. The BIOS routine, in turn, performs the necessary series of INs and OUTs to communicate the request to the IO device.

DOS programs call BIOS routines by executing software interrupt instructions. An example would be INT 13 to call the disk BIOS routine. In response, the processor indexes into entry 13h in the Interrupt Table in memory and jumps to the start address of the disk BIOS routine indicated in this entry. Since all, or most, accesses to IO devices should be routed through the multitasking OS, the processor should trap to the OS whenever an attempt is made by an application program to execute an INT instruction. The OS can then use the Interrupt Table entry number specified by the INT instruction to determine what BIOS routine the task is calling. The OS can then execute its own respective device driver to communicate the request to the target IO device.
386 Real Mode Operation

The Previous Chapter
As background material, this chapter provided a very basic introduction to the problems that a multitask OS must be prepared to deal with.

This Chapter
This chapter provides a detailed description of the processor’s operation when in Real Mode. This description also applies to all IA32 processors subsequent to the 386 processor.

The Next Chapter
This chapter provides a basic introduction to the following topics: Segmentation, Virtual Memory Paging, IO Protection, Privilege Levels, Virtual 8086 Mode, Task Switching and Interrupt Handling.

Special Note
This chapter contains a number of references to Protected Mode operation and terminology. A detailed description of 386 Protected Mode can be found in the chapters that follow this one.

An Overview of the 386 Internal Architecture
Figure 5-1 on page 41 illustrates the internal architecture of the 386 processor. It consisted of the following internal units:

- Bus Unit. Interfaces the processor to the FSB and the system in general.
- Prefetcher. Working on the presumption that the currently executing program never executes jumps, it instructs the Bus Unit to perform a series of memory code read transactions from ascending memory addresses.
The Unabridged Pentium® 4

- Prefetch Queue. The instructions prefetched from memory are placed in this queue.
- Instruction Decoder. Decodes each instruction into an executable form.
- Instruction Queue. The decoded instructions are placed in this queue.
- Execution Unit. Executes instructions one at a time as they are provided from the Instruction Queue.
- Register set. As each instruction is executed, the registers are accessed by the Execution Unit on an as-needed basis.
- Segment Unit. Whenever a memory access must be performed, the Segment Unit adds the offset of the item to be accessed (in the code, stack or data segment) to the base address of the target segment, thereby producing the 32-bit linear memory address. If Paging is disabled, the linear address is the physical memory address that is accessed by performing a transaction on the FSB.
- Paging Unit. If Paging is enabled and a memory access must be performed, the 32-bit linear memory address is submitted to the Paging Unit for a lookup in the Page Directory and a Page Table. The selected Page Table Entry (PTE) is then used to translate the 32-bit linear memory address into a 32-bit physical memory address. The resultant physical memory address is then accessed by performing a transaction on the FSB.
Figure 5-2 on page 44 illustrates the address- and data-related signals on the 386DX processor’s FSB. Although the 386 processor implemented a full 32-bit internal address bus, the two least-significant address lines, A[1:0], were not implemented as output pins on the FSB.
Address Bus Selects Dword

Whenever the processor initiated a transaction on the FSB, logic external to the processor behaved as if the least-significant two address lines are always zero. As a result, the processor could only output addresses divisible by four. As an example, it could address location 00000100h, but not 00000101h, 00000102h, or 00000103. In other words, the address output on A[31:2] selected a dword (i.e., a group of four locations starting at an address divisible by four) in either memory or IO address space (as defined by the transaction type).

Byte Enables Select Location(s) in Dword

In addition, the processor implemented four output pins designated as Byte Enable (BE) pins 3:0 (BE[3:0]#). The dword selector address is output on A[31:2] and the Byte Enable pins asserted by the processor indicate which of the four locations within the currently addressed dword are being selected for a read or a write (as defined by the transaction type). Refer to Table 5-1.

<table>
<thead>
<tr>
<th>Byte Enable Asserted</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BE0#</td>
<td>When asserted, indicates that location zero in the selected dword is being addressed and that the byte to be read or written will be transferred over data path 0 (D[7:0]).</td>
</tr>
<tr>
<td>BE1#</td>
<td>When asserted, indicates that location one in the selected dword is being addressed and that the byte to be read or written will be transferred over data path 1 (D[15:8]).</td>
</tr>
<tr>
<td>BE2#</td>
<td>When asserted, indicates that location two in the selected dword is being addressed and that the byte to be read or written will be transferred over data path 2 (D[23:16]).</td>
</tr>
<tr>
<td>BE3#</td>
<td>When asserted, indicates that location three in the selected dword is being addressed and that the byte to be read or written will be transferred over data path 3 (D[31:24]).</td>
</tr>
</tbody>
</table>
Chapter 5: 386 Real Mode Operation

Misaligned Transfers Affect Performance

It should be obvious that, in a single transaction, the processor can only address a single dword in which to perform a read or write. Consider the following example:

```
mov eax, [0101]
```

When executed, this instruction causes the processor to load the 32-bit EAX register with the four bytes from memory locations 00000101h through 00000104h. These are the last three locations in the dword that starts at 00000100h and the first location in the dword that starts at location 00000104h. In order to read these four locations, the processor must:

- Perform a memory data read transaction from the dword starting at location 00000100h. It asserts BE1#, BE2# and BE3#, indicating a read from locations 00000101h through 00000103h.
- Perform a memory data read transaction from the dword starting at location 00000104h. It asserts BE0# indicating a read from location 00000104h.

This scenario came about because the programmer (or the compiler) did not pay attention to alignment when this 32-bit data object was created in memory. Because it straddles two dwords, the processor must perform two transactions on its FSB whenever it must read or update this data object. This will negatively affect performance. The 386 processor did not provide the ability to flag this condition to the programmer as something that should be fixed in order to optimize execution speed. Starting with the 486 processor, all IA32 processors implement a mechanism to flag this condition (refer to “Alignment Check Exception (17)” on page 321).

Alignment Is Important!

As indicated in the previous section, misalignment of multi-byte data objects in memory can negatively affect performance. This is true in all IA32 processor implementations. If a multi-byte data object straddles a dword address boundary, it may also:

- **straddle a cache line boundary**: In a post-386 processor, this may result in a double cache miss causing the processor to perform two full cache line reads on its FSB. Not only is this time consuming for the processor that experienced the double miss, but it consumes FSB bandwidth making the FSB less available to other entities on the FSB.
6

Protected Mode

Introduction

The Previous Chapter
This chapter provided a detailed description of the processor’s operation when in Real Mode. This description also applies to all IA32 processors subsequent to the 386 processor.

This Chapter
This chapter provides a basic introduction to the following topics: Segmentation, Virtual Memory Paging, IO Protection, Privilege Levels, Virtual 8086 Mode, Task Switching and Interrupt Handling.

The Next Chapter
This chapter introduces segment register usage in Protected Mode, Segment Descriptors, the GDT, the LDTS, the IDT, and the general Segment Descriptor format.

General
This chapter provides a brief introduction to the various types of protection offered in the IA32 Protected Mode environment. The following topics are introduced:

- Memory Protection.
- IO Protection.
- Privilege Levels.
- Virtual Memory Paging.
- Virtual 8086 Mode (also referred to as VM86 mode, or VM mode).
The Unabridged Pentium® 4

- Task Switching.
- Interrupt Handling.

Each of the topics introduced in this chapter are discussed in detail in subsequent chapters.

Memory Protection

Segmentation

Using segmentation, the OS programmer defines the areas of memory that may be accessed by the currently executing program and how they may be accessed. In Real Mode, a segment has the following characteristics:

- Its start address must be in the first megabyte of memory space.
- The segment length is fixed at 64KB.
- The segment can be read from or written to by any program.

In a multitasking environment, the OS programmer must be able to define the following characteristics of a segment:

- A start address anywhere in the 4GB memory address space that can be addressed by the processor.
- A segment length ranging from one byte to 4GB.
- The privilege level that the currently executing program must equal or exceed to gain access to this segment of memory.
- Define the segment as read-only, execute-only or read/writable.
- Define the segment as a special segment used only by the OS, or as a code or data segment to be used by a task.
- Whether or not the segment has been accessed since it was created.
- Whether or not the segment of information is currently resident in memory (it may be out on a mass storage device).

A detailed description of segmentation can be found in the chapters entitled:

- “Intro to Segmentation in Protected Mode” on page 109.
- “Code Segments” on page 133.
- “Data and Stack Segments” on page 157.
- “The Flat Model” on page 247.
Chapter 6: Protected Mode Introduction

Virtual Memory Paging

When enabled and utilized by the OS, the processor’s Paging Unit can redirect a memory access to either:

- a physical address in memory other than the address generated by the currently executing program, or
- a page of data on a mass storage device.

Two programs may attempt to use the same area of memory. When one of the programs is active, the Paging Unit can redirect accesses to one physical area of memory. When the other program becomes active, the Paging Unit can alter its redirection mechanism to redirect memory accesses to an area of physical memory separate from that used by the first program. This ensures isolated data areas for the two programs (so they don’t interfere with each other). This process is transparent to the currently executing program.

It is especially useful when the OS is attempting to timeslice (i.e., multitask) multiple DOS tasks. Each will attempt accesses within the first megabyte of memory space. Paging can be used to direct each of their memory accesses to separate 1MB areas (other than the first megabyte). Also refer to the section entitled “Virtual 8086 Mode” on page 106. A detailed description of Paging can be found in the chapter entitled “386 Demand Mode Paging” on page 209.

IO Protection

When operating in Real Mode, any program can execute IO-oriented instructions and communicate directly with IO devices. For reasons described in the previous chapter, it can be dangerous to permit direct IO by tasks executing in a multitasking environment. To prevent this, the IA32 processors implement the IO privilege level (IOPL). By setting this two-bit field in the EFlags register image of a task’s TSS to the appropriate privilege level (a value between zero and three), the OS can ensure that only tasks with a privilege level that meets or exceeds that indicated in the EFlags[IOPL] field are permitted to communicate directly with IO devices.

An IO access attempt by a task with a privilege level less than the IOPL results in a General Protection exception. In other words, it’s not permitted.

When a DOS task is executing in Virtual 8086 (VM86) mode, the IOPL is not used. Rather, when the OS creates the task, it also creates an IO Permission Bit
Map (in the task’s TSS in memory). Each bit in this map corresponds to one of the 64K IO ports. When the task attempts to access any IO port, the processor first checks the task’s IO Permission Map to determine if access to the port(s) is permitted. A General Protection exception is generated if the access is prohibited.

Privilege Levels

As discussed in an earlier chapter, the IA32 processors provide four privilege levels when executing in Protected Mode:

- Level zero is the highest privilege level. Typically, only the OS kernel will run with privilege level zero. This permits it to perform any operation.
- Level one is the next privilege level. It is typically assigned to high-priority device drivers and OS services. It could also be assigned to debuggers to protect them from alteration by low-priority device drivers and applications programs.
- Level two is typically assigned to lower-priority device drivers.
- Level three is the lowest priority and is typically assigned to applications programs. This prevents them from performing actions that would be injurious to the OS, debuggers, device drivers, or each other.

Virtual 8086 Mode

Because programs written for DOS behave as if they own the entire machine, IA32 processors (starting with the 386) implement a mode known as Virtual 8086 (VM86) Mode. When a task is executed with this processor feature enabled (when EFlags[VM] = 1), the processor enables “watchdog” logic to monitor the program’s behavior on an instruction-by-instruction basis. When operating in VM86 mode, the processor traps out to a program referred to as a Virtual Machine Monitor (VMM) whenever the task attempts to perform an action inimical to the OS or the other currently-suspended programs. The VMM emulates the action required by the task in a fashion that is friendly to the OS and other programs. A detailed description of VM86 mode can be found in the chapter entitled “Virtual 8086 Mode” on page 329.

Task Switching

The IA32 processors provide automated mechanisms to handle the suspension of one task and the initiation of another. The OS creates a Task State Segment
Chapter 6: Protected Mode Introduction

(TSS) for each task to be run. In a task’s TSS, the OS programmer defines the following characteristics of the task:

- The initial settings of the processor’s registers.
- The task’s IO Permission Bit Map.

The task is launched by telling the processor the start address of its TSS. The processor then loads its register set from the TSS and begins execution of the program. When it’s time to suspend a task and to start or resume another task, the processor first stores the current state of most of its registers in the TSS of the task being suspended. It then loads most of its registers from the TSS associated with the next task and begins or resumes its execution. A detailed description of task switching can be found in the chapters entitled “Creating a Task” on page 171 and “Mechanics of a Task Switch” on page 191.

Interrupt Handling

Real Mode Interrupt Handling

In Real Mode, each entry in the Interrupt Table is four bytes long and represents the start address, in segment:offset format, of an interrupt handler. The handler is typically one of the following:

- a hardware interrupt service routine.
- a software error exception handler routine.
- a software interrupt handler (called via an INT nn instruction).
- a BIOS routine.
- a DOS request handler.

In Real Mode, any program can use the INT instruction to call a BIOS routine or to issue a request to the OS.

Protected Mode Interrupt Handling

In Protected Mode, the OS must restrict entry to some routines that can be called using the INT nn instruction. In addition, the OS programmer may wish to handle some interrupts or exceptions by suspending the current task and switching to another task designed to handle the event (rather than just jumping to an interrupt or exception service routine within the same task).
7 Intro to
Segmentation in Protected Mode

The Previous Chapter
This chapter provided a basic introduction to the following topics: Segmentation, Virtual Memory Paging, IO Protection, Privilege Levels, Virtual 8086 Mode, Task Switching and Interrupt Handling.

This Chapter
This chapter introduces segment register usage in Protected Mode, Segment Descriptors, the GDT, the LDTS, the IDT, and the general Segment Descriptor format.

The Next Chapter
This chapter provides a detailed description of Code Segments (both Conforming and Non-Conforming), privilege checking, and Call Gates.

Special Note
Please note that unless otherwise noted, the terms program, procedure, and routine are used interchangeably throughout the book.
Real Mode Limitations

In Real Mode, a segment has the following characteristics:

- Its start address must be in the first megabyte of memory space.
- The segment length is fixed at 64KB.
- The segment can be read or written by any program.

In order to have the maximum flexibility, the OS must be able to define a program’s segments as residing anywhere within the 4GB memory address range.

In Real Mode, segments cannot reside in extended memory (i.e., memory above the first megabyte).

Programs and the data they manipulate frequently occupy more than 64KB of memory space, but each segment has a fixed length of 64KB in Real Mode, neither shorter nor longer. If the OS only requires a very small segment for a program’s code, data or stack area, the smallest (and largest size) is 64KB. This can waste memory space. If the code or data utilized by a particular program is larger than 64KB, the programmer must set up and jump back and forth between multiple code segments. This is a very wasteful use of the programmer’s time and can be difficult to keep track of. It’s one of the major things programmers dislike about Real Mode segmentation.

In Real Mode, a segment can be accessed by any program. This is an invitation for one program to inadvertently trash another’s code, data or stack area. In addition, any program can call any other program. There is no concept of restricting access to certain programs.

Segment Descriptor Describes a Memory Area in Detail

In a multitasking environment, the OS programmer must be able to specify the following characteristics of each segment:

- The task that it belongs to.
- Its start address anywhere in the 4GB memory address range.
- Its length (anywhere from one byte to 4GB in length).
- How it may be accessed: read-only, execute-only, read/write.
- The minimum privilege level a program must have to access the segment.
Chapter 7: Intro to Segmentation in Protected Mode

- Whether it's a code or data segment, or a special segment that is only used by the OS.
- Whether the segment of information is currently present in memory or resides on a mass storage device.

Figure 7-1 on page 111 illustrates the manner in which the processor interprets the contents of a segment register while operating in Real Mode. The only thing it contains is the upper 16 bits of the 20-bit start address of the segment within the first megabyte of memory space. The processor automatically appends the lower four bits of the start address and always sets them to zero. As an example, if the programmer moved the value $1010h$ into the DS register

```
mov ax, 1010
mov ds, ax
```

this would set the start address of the data segment to $10100h$.

As stated earlier in this chapter, when in Protected Mode the OS programmer must be able to define many more properties of a segment in addition to its start memory address. It should be obvious that it would not be possible to define all of these characteristics in the 16-bit segment register.

In Protected Mode, it requires eight bytes of information to describe all of the characteristics associated with a particular segment of memory space. The protected mode OS must provide an eight byte descriptor for each memory segment to be used by each program (including those used by the OS itself). It would consume a great deal of processor real estate to keep descriptors for all segments used by all programs in registers on the processor chip itself. For this reason, the descriptors are stored in special tables in memory. The next section provides a description of the descriptor tables.
Segment Register—Selects Descriptor Table and Entry

When a programmer wishes to gain access to an area of memory, the respective segment register (the CS, SS, or one of the data segment registers—DS, ES, FS, or GS) must be loaded with a 16-bit value that identifies the area of memory. In Real Mode, the value loaded into the segment register represents the upper 16 bits of the 20-bit start address of the segment in memory. In Protected Mode, the value loaded into a segment register is referred to as the segment selector, illustrated in the upper part (i.e., the segment register’s visible part) of Figure 7-3 on page 114:

- The Requestor Privilege Level (RPL) field is described in “Code Segments” on page 133 and “Data and Stack Segments” on page 157.
- Bit [2] (the Table Indicator, or TI bit) of the segment register selects either the Global Descriptor Table (GDT) or the Local Descriptor Table (LDT). The descriptor tables are described in “Introduction to the Descriptor Tables” on page 114.
- The Index field is used to select an entry (i.e., a segment descriptor) in the indicated table.

Whenever a value is loaded into a segment register in Protected Mode, the processor multiplies the segment register’s index field value by eight (because there are eight bytes per entry) to create the offset into the indicated table. It then adds this offset to the respective table’s base address (supplied by either the GDT register—GDTR, or the LDT register—LDTR), yielding the start address of the selected segment descriptor in the specified table. The processor then performs a memory read to fetch the 8-byte descriptor from memory and places it into the invisible part of the specified segment register (see Figure 7-3 on page 114). The invisible part is referred to as the segment register’s cache register. There is a separate segment cache register for each of the six segment registers.

Figure 7-2 on page 113 illustrates the segment register, the Global and Local Descriptor Tables (GDT and LDT), the GDTR and the LDTR. Note that although there is only one GDT, there may be more than one LDT.
Chapter 7: Intro to Segmentation in Protected Mode

Figure 7-2: Relationship of a Segment Register and GDT, GDTR, LDT, and LDTR
8 Code Segments

The Previous Chapter
This chapter introduced segment register usage in Protected Mode, Segment Descriptors, the GDT, the LDTS, the IDT, and the general Segment Descriptor format.

This Chapter
This chapter provides a detailed description of Code Segments (both Conforming and Non-Conforming), privilege checking, and Call Gates.

The Next Chapter
This chapter provides a detailed description of Data and Stack segments (including Expand-Up and Expand-Down Stacks) and privilege checking.

Selecting the Code Segment to Execute

In order for it to fetch instructions from an area of memory, the programmer must inform the processor what code segment the instructions are to be fetched from. This is accomplished by loading a 16-bit value (a selector) into the Code Segment (CS) register. In Real Mode, this value represents the upper 16 bits of the 20 bit start address of the segment in memory. In Protected Mode, the value loaded into a segment register is interpreted as illustrated in Figure 8-1 on page 134.

Any of the following actions loads a value into the CS segment register, causing the processor to begin fetching instructions from the new code segment in memory:

- Execution of a far jump instruction. This loads both CS and EIP with new values.
- Execution of a far CALL instruction. This loads both CS and EIP with new values.
The Unabridged Pentium® 4

- A hardware interrupt or a software exception. In response, the processor reads values from the Interrupt Table entry into the CS and EIP registers.
- Execution of a software interrupt instruction (INT nn). In response, the processor reads values from the Interrupt Table entry into the CS and EIP registers.
- Initiation of a new task or resumption of a previously-suspended task. During the task switch, the processor loads all of its registers, including CS and EIP, with the values from the TSS associated with the new task.
- Execution of a far RET instruction. The return address is popped from the stack and placed in the CS and EIP registers.
- Execution of an Interrupt Return instruction (IRET). The return address is popped from the stack and placed in the CS and EIP registers.

Figure 8-1: Segment Selector

<table>
<thead>
<tr>
<th>Segment Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 3 2 1 0</td>
</tr>
</tbody>
</table>

Segment register's visible part: DT Index TI RPL

Segment register's invisible part: Attributes Segment Size Segment Base Address

<table>
<thead>
<tr>
<th>DT</th>
<th>Descriptor Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI</td>
<td>Table Indicator</td>
</tr>
<tr>
<td>RPL</td>
<td>Requestor Privilege Level</td>
</tr>
</tbody>
</table>

Code Segment Descriptor Format

The value loaded into the visible part of CS (Figure 8-1) identifies:

- the descriptor table that contains the code segment descriptor.
  — TI = 0 indicates that the entry resides in the GDT.
  — TI = 1 indicates that the entry resides in the LDT.
- the entry in the specified descriptor table. The Index field identifies one of 8192d entries in the selected table.

The processor multiplies the index by eight (eight bytes per entry) to obtain the offset in the table. A check is performed to ensure that the offset is not beyond the indicated table's limit (supplied from the GDTR or LDTR register). Otherwise, an exception results. The offset is then added to the table base address.
Chapter 8: Code Segments

(supplied from the GDTR or LDTR register) to form the start address of the descriptor in memory.

The processor reads the Code Segment descriptor from the selected segment descriptor table and checks to ensure that the currently executing program has sufficient privilege to access this code segment (the CPL of the current program meets or exceeds the DPL of the target Code Segment). If not, a General Protection (GP) exception is generated. If the privilege test is passed, the processor saves the descriptor information in its internal code segment cache register (the invisible part of the CS register).

Table 8-1 on page 135 and Figure 8-2 on page 136 illustrate the format of a code segment descriptor.

Table 8-1: Code Segment Descriptor Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1</td>
<td>S = 1 (because a code segment is not a special OS segment).</td>
</tr>
<tr>
<td>C/D</td>
<td>1</td>
<td>Code or Data bit = 1, indicating that the descriptor defines a code segment, rather than a data or a stack segment.</td>
</tr>
<tr>
<td>Conforming bit</td>
<td>0 or 1</td>
<td>Refer to the section entitled “Conforming and Non-Conforming Code Segments” on page 141 for a description of conforming versus non-conforming code segments.</td>
</tr>
</tbody>
</table>
| R              | 0 or 1| • If R = 0, only the instruction prefetcher may access this code segment (in other words, the segment is execute-only). Any attempt to access the code segment using data access instructions (e.g., a MOV) causes a GP exception.
• If R = 1, this segment may be read by both the instruction prefetcher and by using data access instructions. This is necessary if the code segment contains data items that must be read during the course of program execution. |
| Other fields   |       | The remaining bit fields in the code segment descriptor are defined in the section entitled “General Segment Descriptor Format” on page 121. |
Figure 8-2: Code Segment Descriptor Format

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4th Byte of Base Address</td>
<td>3rd Byte of Base Address</td>
<td>2nd Byte of Base Address</td>
<td>LSB of Base Address</td>
<td>2nd Byte of Segment Size</td>
<td>LSB of Segment Size</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**G Bit** Granularity bit defines meaning of limit value.  
0 = length of segment in bytes.  
1 = length of segment in 4KB pages.

**D Bit** In code segment, Default bit defines default size of operands and effective addresses.  
0 = 16-bit, 1 = 32-bit.

**AVL Bit** Available for use by system software

**P Bit** Segment Present bit (must be 1 if the code segment is present in memory).

**DPL Field** Descriptor Privilege Level (0-3)

**S Bit** System bit. When 0, indicates system segment.  
Must be 1 in a code segment descriptor.

**D/C** This could be called the Data/Code bit.  
A 0 indicates a data segment and a 1 indicates a code segment.

**C Bit** Conforming bit. Set to 1 if code segment is conforming. See text for a detailed description.

**R Bit** Readable bit. A 0 indicates an execute-only segment, while a 1 indicates the segment may be read from by both the prefetche and for data accesses.

**A Bit** Accessed bit. Set to 1 by the processor when a code segment is accessed.
Accessing the Code Segment

The processor accesses the code segment whenever it has to fetch an instruction from memory. Consider the following unconditional near jump instruction:

```
jmp 0009
```

The programmer has specified an offset, 0009h, within the current code segment as the target of this unconditional jump. In response, the processor compares the specified offset to the size, or limit, of the code segment currently in use to ensure that the programmer isn’t attempting to jump outside the bounds of the current code segment. The code segment’s start address, size and attributes are stored in the processor’s internal CS cache register. If the target location is within the bounds of the segment, the processor adds the specified offset to the segment’s base address to yield the memory address of the instruction to be jumped to. It then fetches the next instruction from that location.

In the following example, the programmer wishes the processor to perform an unconditional far jump instruction to fetch the next instruction from a location within a different code segment:

```
jmp 00d0:0003
```

Since this is an attempt to access a different code segment, the processor must first verify that the currently executing program is permitted to access the location in the new code segment. To do this, it must read the new code segment descriptor from memory and check its descriptor privilege level (DPL). The value 00d0h is placed into the CS register and is interpreted as indicated in Figure 8-4 on page 139 (the index field is binarily-weighted). The processor reads the 27th entry (d0h = 26d) from the GDT (TI = 0 selects the GDT). Figure 8-3 on page 138 illustrates the example code segment descriptor read from the GDT.

The processor verifies that the new segment is a code segment (System bit = 1, and C/D = 1) and is present in memory (P = 1). It must also determine if the currently executing program is sufficiently privileged to call or jump to the targeted code segment. This subject is covered in the next section (“Privilege Checking” on page 139). It checks the specified target offset, 0003h, to determine if it exceeds the limit (size) of the code segment (the segment size is 126525d bytes (the Granularity bit = 0, indicating that the size is specified in bytes, rather than in 4KB pages). If all tests are passed, it loads the new segment descriptor into its on-chip code segment cache register, adds the specified offset (0003h) to the code segment’s base address (00131BCCh) and fetches the next instruction from the target address—00131BCFh.
The Previous Chapter
This chapter provided a detailed description of Code Segments (both Conforming and Non-Conforming), privilege checking, and Call Gates.

This Chapter
This chapter provides a detailed description of Data and Stack segments (including Expand-Up and Expand-Down Stacks) and privilege checking.

The Next Chapter
This chapter provides a detailed description of the Task State Segment (TSS), the TSS segment descriptor, task creation, how the OS starts a task and what happens when a task starts.

A Note Regarding Stack Segments
Intel® considers the stack segment to be a data segment. However, it is treated separately in this chapter because it is used differently than the average data segment.
The Data Segments

Selecting and Accessing a Data Segment

The post-286 processors have four data segment registers: DS, ES, FS and GS. They identify up to four separate data segments (in memory) that can be accessed by the currently executing program.

To access data within one of the four data segments, the programmer must first load a 16-bit value into the respective data segment register. In Real Mode, the value in a data segment register specifies the upper 16 bits of the 20 bit memory start address of the data segment. In Protected Mode, the value selects a segment descriptor in either the GDT or LDT. Figure 9-1 on page 160 illustrates the format of a data segment descriptor. The example

```
    mov ax, 4f36 ;load ds register
    mov ds, ax ;
    mov al, [0100] ;read from data segment into al
    mov [2100], al ;write to data segment from al
```

has the following effect. The value 4F36h is moved into the DS data segment register and is interpreted by the processor as indicated in Figure 9-2 on page 160. The RPL = 2. The processor accesses entry 2534d in the LDT to obtain the data segment descriptor and performs an access rights check. Figure 9-3 on page 161 illustrates the example data segment descriptor fetched from the LDT. The segment is:

- a data segment (C/D = 0) 31,550d bytes in length.
- starting at memory location 00083EA0h.
- with a DPL = 2.
- and is read/writable.

Assuming that the privilege check is successful, the eight byte segment descriptor is loaded into the DS register’s invisible cache register on board the processor.

When the third instruction of the example (MOV AL,[0100]) is executed, the processor performs a limit check to ensure that the specified offset, 0100h, doesn’t exceed the length of the DS data segment. 0100h is compared to the segment size in the DS cache register. Since 0100h is less than 07B3Eh, the access is within the segment’s limit. The processor permits the access and the offset,
0100h, is added to the segment base address, 00083EA0h, yielding memory address 00083FA0h. One byte is read from this location and placed into the processor’s AL register. The next MOV instruction involves a memory write into the DS data segment. Before permitting this, the processor checks the descriptor’s W bit to ensure whether this segment is marked as writable (it is). Another limit check is performed to ensure that offset 2100h doesn’t exceed the segment length. The offset, 2100h, is then added to the segment’s base address, 00083EA0h, yielding memory address 00085FA0h. The byte in the AL register is written into this memory location.

The following code fragment is the same as the previous one except for the fact that it accesses the GS data segment instead of the DS data segment.

```
mov ax, 4f36 ;load gs register
mov gs, ax   ;
mov al, gs:[0100] ;read from gs data segment
mov gs:[2100], al ;write to gs data segment
```

**Data Segment Privilege Check**

The RPL, CPL and DPL are involved in the privilege check. The 16-bit value loaded into the respective data segment register is accepted if the lesser-privileged of the RPL and CPL has the same privilege level or is more privileged than the target data segment descriptor’s DPL. Another way of stating it is—a program can only access data in a segment with the same or a lesser privilege level.

Assuming that the currently executing program’s RPL and CPL are the same:

- a program with a CPL of zero can access data in a data segment with any DPL value.
- a program with a CPL of one can access data in data segments with a DPL of one, two, or three.
- a program with a CPL of two can access data in data segments with a DPL of two or three.
- a program with a CPL of three can only access data in data segments with a DPL of three.

Any violation of this criteria results in a GP exception.
Figure 9-1: Data Segment Descriptor Format

- **G Bit**: Granularity bit defines meaning of limit value: 0 = length of segment in bytes, 1 = length of segment in pages.
- **B Bit**: In data segment, Big bit defines SP size and upper bound of expand-down stack.
- **AVL Bit**: Available for use by system software.
- **P Bit**: Segment Present bit (must be 1 if the data segment is present in memory).
- **DPL Field**: Descriptor Privilege Level.
- **S Bit**: System bit. When 0, indicates system segment. Must be 1 in a data segment descriptor.
- **Bit 3**: This could be called the Data/Code bit. A 0 indicates a data segment and a 1 indicates a code segment.
- **E Bit**: Expand-Down bit. When set to 1, segment is an expand-down stack (rather than expand-up). See text.
- **W Bit**: Writable bit. A 0 indicates a read-only segment, while a 1 indicates a read-writable segment.
- **A Bit**: Accessed bit. Set to 1 by the processor when a data segment is accessed.

Figure 9-2: Example Value in DS Register

Index = 2534d  
TI  RPL  
1  2
Selecting and Accessing a Stack Segment

Introduction

A stack segment is a form of data segment. Its descriptor must identify it as a read/writable segment so that the processor may perform both pushes (i.e., writes to the stack) and pops (i.e., reads from the stack). The descriptor also describes the stack type. A stack may be designated as an expand-up stack (the most common type; described in “Accessing the Stack Segment” on page 76) or an expand-down stack. A description of the expand-down stack can be found in the section entitled “Expand-Down Stack” on page 164. It should be noted that most OSs implement expand-up stacks.
This chapter provides a detailed description of the Task State Segment (TSS), the TSS segment descriptor, task creation, how the OS starts a task and what happens when a task starts.

The Next Chapter

This chapter provides a detailed description of how the processor handles automatic task switching. It also covers Linked Tasks, Linkage Modification, the Busy Bit, and address mapping issues.

What Is a Task?

Each application consists of one more code segments, a group of one or more data segments, and a stack segment. In the course of executing, the current application must be able to access one or more code and data segments in memory, as well as one or more stack areas. All of these elements taken together comprise a task in a multitasking OS environment. Examples would be Microsoft Word, CorelDraw, etc.

Basics of Task Creation and Startup

The following sections describe the steps typically taken by the OS when it must start (or resume) a task.
Load All or Part of the Task into Memory

The OS loads all or part of the task (i.e., at a minimum, the startup code for the task) into memory.

Create a TSS and a TSS Descriptor for the Task

The OS creates a data structure in memory defining the context of the processor at the point when it first begins execution of the task. In other words, the data structure defines an exact image of the information that should be present in the processor’s register set when the processor initiates execution of the task. This data structure is referred to as the Task State Segment (TSS; see Figure 10-1 on page 175), and the OS must set up a separate TSS for each task.

In addition, the OS creates a special, 8-byte TSS segment descriptor in the GDT defining the base address, length, and DPL of the TSS.

Trigger the Timeslice Timer

A multitasking OS usually permits a task to execute for a predefined period of time, typically referred to as a timeslice. This is accomplished by starting a hardware timer prior to starting (or resuming) the task.

Scheduler Causes a Task Switch

The task is then started by the OS scheduler (see “How the OS Starts a Task” on page 186) and continues to execute until a hardware interrupt is generated by the timeslice timer (unless the task is suspended by the OS prior to this for some other reason).

The task is started by executing a far jump or a far CALL instruction wherein the 16-bit CS portion of the branch target address selects the task’s TSS descriptor in the GDT. In this case, the offset portion of the target address is discarded.

When the processor determines that a TSS descriptor has been selected, it suspends the current task (in this case, the OS scheduler) by storing the majority of the processor’s registers into the OS scheduler’s TSS. It then switches to the new task by loading the processor’s register set from the new task’s TSS (the one
Chapter 10: Creating a Task

pointed to by the GDT entry selected by the far jump or far CALL). The processor uses the pointer placed in CS:EIP (from the new task’s TSS) and begins fetching code from the new application.

Interrupt on Timer Expiration

When the hardware timeslice timer expires it generates an interrupt that selects an entry in the Interrupt Descriptor Table (IDT) containing a Task Gate that points to the OS’s task scheduler. The task that was executing is suspended (the processor automatically copies most of the processor’s registers into the task’s TSS). The next task (i.e., the OS task scheduler) is restarted by loading the processor’s register set from the new task’s TSS before resuming program execution.

Unlike many other processors (e.g., the PowerPC processor family, as well as later IA32 processors), the 386 processor did not incorporate a hardware timeslice timer to facilitate the timeslice approach to multitasking. Rather, the system designer had to incorporate a hardware timer external to the processor. This timer was implemented as an IO device that the OS scheduler programmed for the desired interval and then enabled. The timer was started by software and generated a maskable interrupt to the 386 (on its INTR input pin) when the timer expired.

With the advent of the P54C version of the Pentium® processor, all subsequent IA32 processors implement a programmable timer capable of generating an interrupt on expiration or at set intervals. This timer is part of the processor’s Local APIC.

TSS Structure

General

The 286 implemented a different TSS structure than that defined for the post-286 processors. This is referred to as a 16-bit TSS and is not covered in this book.

All post-286 processors implement the TSS structure illustrated in Figure 10-1 on page 175. This is referred to as a 32-bit TSS. Note that the 386 and the early 486 processors did not implement the Interrupt Redirection Map. It was first implemented in the Pentium® processor and was then migrated to the later versions of the 486 processor, as well as all subsequent IA32 processors. It is described in “Efficient Handling of the INT Instruction” on page 495.
At a minimum, the TSS must include locations 00h through 67h (104d locations). This required portion consists of three type of fields:

- Those locations shown as zeros are reserved by Intel® and must not be used.
- The dynamic fields are read by the processor whenever the task is started or resumed and are automatically updated by the processor whenever the task is suspended (hence the term “dynamic” because these fields change dynamically during system operation).
- The static fields are read by the processor but are not written to (in other words, they remain static).

The portion of the TSS that resides above location 67h consists of three areas:

- The OS may utilize the optional area starting at location 68h for OS-specific data related to the task. The size and interpretation of this area is OS-specific. As an example, the OS could use the FSAVE instruction to save the contents of the FPU’s registers in this area after the task has been suspended due to a task switch.
- The Interrupt Redirection Bit Map (not implemented until the advent of the Pentium® processor) consists of 32 bytes (eight dwords) and is only necessary if the OS supports the VM86 Mode extensions that are enabled with the CR4[VME] bit (not implemented in the 386).
- The IO Permission Bit Map can be up to 8KB in size and is necessary if the OS supports IO protection.

The sections that follow describe each field in the TSS.
Chapter 10: Creating a Task

Figure 10-1: 32-bit Task State Segment (TSS) Format

IO Port Access Protection

IO Protection in Real Mode

When the processor is operating in Real Mode, there isn’t any IO protection. In other words, any program may execute the IA32 processor’s IO instructions at any time. As stated earlier in “IO Port Anarchy” on page 32, the inability of the OS to restrict the ability of applications programs to talk directly to IO ports can result in problems when multitasking. When operating in Protected Mode, the

Visit MindShare Training at www.mindshare.com
The Previous Chapter
This chapter provided a detailed description of the Task State Segment (TSS), the TSS segment descriptor, task creation, how the OS starts a task and what happens when a task starts.

This Chapter
This chapter provides a detailed description of how the processor handles automatic task switching. It also covers Linked Tasks, Linkage Modification, the Busy Bit, and address mapping issues.

The Next Chapter
This chapter provides a complete description of 386-style demand mode paging. This discussion is also directly applicable to all subsequent IA processors. Table 12-5 on page 244 provides linkage to all of the paging-related enhancements that appeared in subsequent IA32 processors.

Events that Initiate a Task Switch
There are a number of events that can cause the processor to suspend the current task and start or resume another task. Table 11-1 on page 192 provides a description of each event. The sections that follow detail the sequence of actions taken by the processor when suspending the current task and starting or resuming another one.
### Table 11-1: Events that Cause a Task Switch

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Far CALL/Far jump to TSS descriptor</td>
<td>If the 16-bit segment portion of a far jump or far CALL selects a TSS descriptor in the GDT, a task switch occurs. The offset portion of the target address is discarded. The processor loads the 16-bit segment selector into the visible portion of the TR and then loads the selected TSS descriptor from the GDT into the invisible part of the TR. A privilege check is performed and, if the currently executing program has sufficient privilege (CPL ≤ DPL), the state of the current task is stored in its TSS and the register values from the new TSS (identified by the TSS descriptor) are loaded into the processor's register set. More detailed information can be found in the sections entitled “Switch as a Result of a Far Call” on page 197 and “Switch as the Result of a Far Jump” on page 197.</td>
</tr>
<tr>
<td>Far CALL/Far jump to Task Gate descriptor</td>
<td>All TSS descriptors must reside in the GDT. The DPL of a TSS descriptor is typically set to zero. This means that a program that resides at a less-privileged level could not switch to the task defined by the TSS. If the currently executing program has access to a Task Gate in its LDT, it can switch to a task (if the less-privileged of the currently executing program's CPL and RPL is at least as privileged as the Task Gate's DPL). The TSS DPL is ignored. The Task Gate has the format specified in Figure 11-1 on page 195 and is described in the section entitled “Task Gate Descriptor” on page 194. Also refer to the sections entitled “Switch as a Result of a Far Call” on page 197 and “Switch as the Result of a Far Jump” on page 197.</td>
</tr>
</tbody>
</table>
Chapter 11: Mechanics of a Task Switch

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT ( mn ) execution that selects a Task Gate in IDT</td>
<td>When the processor executes an INT ( mn ) instruction, the value ( mn ) acts as an index into the IDT. If the selected IDT entry contains a Task Gate descriptor and the program executing the INT instruction has sufficient privilege, a task switch results. Additional information can be found in the sections entitled “Task Gate Descriptor” on page 194 and “Switch Due to a BOUND/INT/INTO/INT3 Instruction” on page 198, and in the chapter entitled “Interrupts and Exceptions” on page 251.</td>
</tr>
<tr>
<td>Hardware interrupt that selects a Task Gate in IDT</td>
<td>When a hardware interrupt request is detected by the processor, the interrupt vector obtained from the interrupt controller is used as an index into the IDT. If the selected IDT entry contains a Task Gate descriptor, a task switch results (exceptions, interrupts and IRET cause a task switch regardless of the Task Gate’s DPL). Additional information can be found in the sections entitled “Task Gate Descriptor” on page 194 and “Task Switch Details” on page 196, and in the chapter entitled “Interrupts and Exceptions” on page 251. Also refer to “Scheduler Causes a Task Switch” on page 172.</td>
</tr>
<tr>
<td>Software exception that selects a Task Gate in IDT</td>
<td>When a software exception condition is detected by the processor, the type of exception condition determines the index into the IDT. If the selected IDT entry contains a Task Gate descriptor, a task switch results (exceptions, interrupts and IRET cause a task switch regardless of the Task Gate’s DPL). Additional information can be found in the sections entitled “Switch Due To an Interrupt or Exception” on page 196 and “Task Switch Details” on page 196, and in the chapter entitled “Interrupts and Exceptions” on page 251.</td>
</tr>
<tr>
<td>IRET execution with EFlags[NT] bit set</td>
<td>Refer to the sections entitled “Link Field (to Old TSS Selector)” on page 184 and “Linked Tasks” on page 201 for a detailed description.</td>
</tr>
</tbody>
</table>
Switch Via a TSS Descriptor

A far CALL or far jump can cause a task switch if the 16-bit segment portion of the target address selects a TSS descriptor in the GDT. However, a GP exception results if the following privilege check isn’t passed:

The less-privileged of the RPL (the lower two bits of the 16-bit segment portion of the target address) or CPL must be at least as privileged as the TSS descriptor’s DPL. Since TSS descriptors typically have a DPL of zero, this means that only privilege level zero programs can CALL or jump to another task using a TSS descriptor.

Task Gate Descriptor

TSS descriptors must reside in the GDT. Task Gate descriptors, on the other hand, may reside in the GDT, an LDT, or the IDT (Interrupt Descriptor Table). Figure 11-1 on page 195 illustrates the format of a Task Gate descriptor. It contains a 16-bit value that selects an entry in the GDT containing a TSS descriptor.

Task Gate Selected by a Far Call/Jump

When a far CALL or a far jump selects a Task Gate descriptor, the DPL of the Task Gate, rather than the DPL of the TSS descriptor, is checked during the privilege level check (the DPL of the TSS is ignored). A task switch occurs if the less-privileged of the RPL or CPL is at least as privileged as the Task Gate’s DPL value. As examples:

- A Task Gate with a DPL of three permits any program to jump to or call the task pointed to by the TSS descriptor.
- A Task Gate with a DPL of two permits programs with privilege levels of zero through two to cause a task switch, while a program with a privilege level of three would cause a GP exception.

It should be noted that the offset portion of the branch target address is irrelevant and is discarded.
Chapter 11: Mechanics of a Task Switch

Task Gate Selected by a Hardware Interrupt or a Software Exception

When a Task Gate is placed in the IDT (see Figure 11-2 on page 196), any hardware interrupt or software exception that selects the IDT entry containing the Task Gate causes a task switch. Both the Task Gate’s and the TSS descriptor’s DPL are ignored. In other words, the privilege check isn’t performed. More detail can be found in “Task Switch Details” on page 196.

Task Gate Selected by an INT Instruction

If an INT nn/INTO/INT3 or a BOUND instruction selects an IDT entry containing a Task Gate, the privilege check is performed. The DPL of the Task Gate, rather than that of the TSS descriptor, is checked during the privilege level check (the DPL of the TSS is ignored). A task switch occurs if the less-privileged of the RPL or CPL is at least as privileged as the Task Gate’s DPL value.

Figure 11-1: The Task Gate Format

The table describes the fields of the Task Gate format:

- **P Bit**: Segment Present bit.
- **DPL Field**: Descriptor Privilege Level.
- **S Bit**: System bit. When 0, indicates system segment. Must be 0 in a Task Gate descriptor.
- **TSS Selector**: Identifies the TSS descriptor that holds the base address, limit and attributes of the TSS for the task being switched to.

Note: A Task Gate descriptor may reside in the Global, Local or Interrupt Descriptor Tables.
This chapter provides a complete description of 386-style demand mode paging. This discussion is also directly applicable to all subsequent IA processors. Table 12-5 on page 244 provides linkage to all of the paging-related enhancements that appeared in subsequent IA32 processors.

The Next Chapter

This chapter describes how usage of the Flat Model can effectively eliminate segmentation from the picture. It should be noted that virtually all modern OSs utilize the Flat Model.

Problem—Loading Entire Task into Memory is Wasteful

Consider the following scenario:

1. A machine has 256MB of RAM memory (a ridiculously small amount, but this is just an example, after all).
2. The ROM-based Power-On Self-Test (POST) completes execution and the boot program reads (i.e., boots) the OS loader program into memory.
3. The OS loader reads the entire OS into memory, consuming 250MB of memory (this is just an example; in this day and age, it would be amazing if an
OS were this small). The OS is a multitasking OS, permitting the end user to start multiple programs. The OS rapidly timeslices between them, giving the appearance that all of the programs run simultaneously.

4. The user tells the OS to start a word processing program. In response, the OS loads the entire program into memory, consuming 3MB of memory (leaving only 3MB of available RAM memory).

5. The user starts another program, which is loaded in its entirety into memory, consuming an additional 2.5MB of memory.

6. 255.5MB of memory is now in use and only .5MB remains free. The user attempts to start another program, causing the OS to respond that there is insufficient memory.

In this scenario, both the OS loader and the OS task manager manage the pool of free memory in a very inefficient fashion. The entire OS is loaded into memory even though large portions of the OS code may never be required during the current work session. Every time the user starts a program, the OS loads the entire program into memory. Once again, large portions of the application’s code may never be required during the current work session. As an example, Microsoft Word implements hundreds of features, most of which are never called upon during a typical work session.

Solution—Load Part and Keep Remainder on Disk

Load on Demand

The OS loader should be designed to load only the portions of the OS:

- that are necessary to initiate application programs;
- that are used very frequently and must always reside in memory in order to yield good performance.

The remainder of the OS should be kept on disk until it is required.

Likewise, the OS application program loader should be designed to load only enough of an application program into memory to get it started. Additional portions of the application program should only be read into memory upon demand.
Chapter 12: 386 Demand Mode Paging

Track Usage

After a portion of the OS or an application program has been loaded into memory, the OS should track how long it has been since the information was last used. If it hasn't been used for quite a while, the OS should eliminate it from memory. In the event that some of the information has been updated since it was read from disk, the OS should swap it back to disk before eliminating it from memory.

Capabilities Required

In order to implement the capabilities just discussed, the OS must have the following capabilities:

- Whenever an instruction (or the instruction prefetcher) initiates a memory code or data access, the processor must in some manner quickly determine if the target information is already in memory (and, if so, where). If it isn’t in memory, the processor must be able to quickly determine the mass storage address of the required information so it can load it into memory to be accessed by the current program.
- The processor must have some way of determining if the block of information has been accessed since it was placed in memory, and, if so, was it changed (i.e., written to).
- Although not mentioned in the preceding discussion, it would also be nice if the processor could determine:
  — if the currently executing program is permitted access to the information (i.e., it has sufficient privilege).
  — if the currently executing program is permitted to write to the targeted area.

Problem—Running Two (or more) DOS Programs

Application programs designed for the DOS environment are written using 8088 code and only access information in the first 1MB of memory space (i.e., from 00000000h through 000FFFFFh). Furthermore, each DOS application believes itself to be the only program executing and, as long as it doesn’t mangle the OS (which also resides in the first 1MB area), it can access any location within the first 1MB of memory space.
If a multitasking OS were to load two or more DOS application programs into the first 1MB of memory, the second one loaded would almost certainly overwrite a portion of the first one (thereby rendering it useless). Even if they occupied mutually-exclusive areas of the first 1MB (highly unlikely), each of the programs would feel free to build (i.e., write) data structures in the memory areas occupied by the other program(s). In a word, anarchy!

Solution—Redirect Memory Accesses to Separate Memory Areas

The OS can multitask multiple DOS applications by taking the following precautions:

- Load each DOS application program into a separate 1MB area of memory.
- When a DOS program is executing, it only generates memory accesses within the first 1MB of memory. Since it actually resides in a different 1MB area other than the first MB, the processor must in some manner automatically redirect each of its memory accesses to the area that it really resides in. Figure 12-1 on page 213 illustrates a scenario wherein two DOS application programs have each been placed in a separate 1MB memory area along with a complete copy of what each expects in the 1st MB of memory space.
Chapter 12: 386 Demand Mode Paging

Figure 12-1: Paging Redirects DOS Accesses to a Discrete 1MB Area

Global Solution—Map Linear Address to Disk Address or to a Different Physical Memory Address

Both of the problems discussed earlier are solved by treating the memory address generated for each code or data access as a logical, or virtual, address. The processor then translates (or redirects) the address into one of the following:
13 The Flat Model

The Previous Chapter
This chapter provided a complete description of 386-style demand mode paging. This discussion is also directly applicable to all subsequent IA processors. Table 12-5 on page 244 provides linkage to all of the paging-related enhancements that appeared in subsequent IA32 processors.

This Chapter
This chapter describes how usage of the Flat Model can effectively eliminate segmentation from the picture. It should be noted that virtually all modern OSs utilize the Flat Model.

The Next Chapter
This chapter provides a detailed description of all of the various types of interrupts and exceptions. A detailed description of the Local and IO APICs can be found in “The Local and IO APICs” on page 1497.

Segments Complicate Things
The use of segments complicates the programmer’s life. The programmer should only have to think of what 32-bit memory location to access and not have to worry about what segment it’s in.

Paging Can Do It All
If segmentation is eliminated and Paging is used, the Paging Unit can provide complete protection, as well as the paging capability described in the previous chapter. The Paging Unit provides the following checks on each memory access attempt:
A privilege check using the PTE’s U/S bit.
Read/write permission checking using the PTE’s R/W bit.

When a memory access is attempted, the Paging Unit deals with one of three cases:

1. The target page is currently in memory (P = 1 in the PTE). Assuming that the currently executing program has sufficient privilege to access the page and that it’s not attempting to write to a read-only page, the access is permitted.

2. The target page isn’t in memory (P = 0 in the PTE). This results in a Page Fault exception. The Page Fault handler examines the 32-bit linear address and determines whether or not the target page belongs to the currently executing program. If it does, the page is read into memory and the PTE is updated with the page location and the P bit is set to one. The access that caused the fault is then restarted and completes successfully.

3. The target page isn’t in memory (P = 0 in the PTE). This results in a Page Fault exception. The Page Fault handler examines the 32-bit linear address and determines whether or not the target page belongs to the currently executing program. If the page doesn’t belong to the program, the OS alerts the end user that the program has attempted an unauthorized memory access and shuts the offending program down.

Eliminating Segmentation

There is no way to disable the IA32 processor’s segmentation logic. However, if all segments are described (in the GDT) as read/writable, starting at location 00000000h and as 4GB in length, segmentation is effectively eliminated.

The code segment is defined as a 32-bit code segment (the C/D bit in the segment descriptor is set to one), with a base address of 00000000h and a length of 4GB. Defining it as a 32-bit code segment has the following effects:

- All memory addresses generated by the EIP register are 32-bits wide, permitting access to any location in the 4GB code segment.
- All memory addresses generated by instructions for data accesses are 32-bits wide, permitting the program to access operands anywhere within the 4GB data segment.
Chapter 13: The Flat Model

The Privilege Check

The code segment descriptor used by the OS would have its DPL set to 0, while the code segment descriptor used by all application programs would have its DPL set to 3. As described in the previous chapter, the CPL of the currently executing program must first pass the segment descriptor’s privilege check and then the page’s privilege check.

Since an application program’s code segment DPL is set to 3 (and the DPL becomes its CPL), it can successfully access any page that has its U/S (User/Supervisor) bit set to one indicating that user access is permitted. However, if it attempts to access a page with U/S = 0, a GP exception results (because only programs with a privilege level of 0, 1, or 2 are permitted access to Supervisor pages).

The code segment for the OS, however, has a DPL of 0 and the OS therefore executes at privilege level 0. It can access both User and Supervisor pages.

The Read/Write Check

Assuming that the currently executing program has sufficient privilege to access a page, it is not permitted write access to a page if the page is write-protected. It should be noted, however, that on the 386, a program executing at privilege level 0, 1, or 2 can write to a write-protected page. This issue was addressed on subsequent IA32 processors starting with the advent of the 486 (see “The Write Protect Feature” on page 450).

Each Task (including the OS) Has Its Own TSS

When a task switch occurs, the processor automatically loads its segment registers with the values from the new task’s TSS. The GDTR register is not loaded with a new value, however. This means that all tasks share the same GDT, but each can select a different set of segment descriptors within the GDT when it is started or resumed (via a task switch).

Switch to an Application Task

If the new task is an application program, the value loaded into the CS register from its TSS selects a code segment descriptor with a DPL of 3. This means the CPL of the task is 3.
A new value is also loaded into CR3, selecting the Page Directory used while the application task is executing. The task’s Page Directory and its associated set of Page Tables describes the pages that the task is permitted to access and how it may access them (i.e., read/write or read-only). The task may be permitted to access up to $2^{20}$ pages of information (4GB) some of which are present in memory while others remain on mass storage until they are needed.

**Switch to an OS Kernel Task**

If the new task is the OS, the value loaded into the CS register selects a code segment descriptor with a DPL of 0. This means the CPL of the task 0. A new value is also loaded into CR3, selecting the Page Directory used while the OS task is executing. The task’s Page Directory and its associated set of Page Tables describes the pages that the task is permitted to access and how it may access them (but remember that on the 386, a program executing at privilege level 0, 1, or 2 can write to a write-protected page). The task may be permitted to access up to $2^{20}$ pages of information (4GB) some of which are present in memory while others remain on mass storage.
14 Interrupts and Exceptions

The Previous Chapter
This chapter describes how usage of the Flat Model can effectively eliminate segmentation from the picture. It should be noted that virtually all modern OSs utilize the Flat Model.

This Chapter
This chapter provides a detailed description of all of the various types of interrupts and exceptions. A detailed description of the Local and IO APICs can be found in “The Local and IO APICs” on page 1497.

The Next Chapter
This chapter provides a detailed description Virtual 8086 Mode (VM86 Mode). This description is directly applicable to all subsequent IA32 processors. VM86 Mode was enhanced starting with the advent of the Pentium® processor and a detailed description of those enhancements can be found in “VM86 Extensions” on page 490.

Special Note
The program executed to service a hardware interrupt or a software exception is frequently referred to as a handler in this chapter. Alternately, it may be referred to as an interrupt service routine (sometimes abbreviated as ISR).
General

There are four types of interrupt-related events that can cause the currently executing program to be interrupted:

- An interrupt request from a hardware device external to the processor is recognized if recognition of external interrupts is enabled (EFlags[IF] = 1).
- The assertion of the processor’s NMI input.
- Execution of a software interrupt (INT) instruction.
- Processor detection of a software exception error condition.

When any of these events occurs, the currently executing program is interrupted. In other words, the processor must:

1. Suspend execution of the program.
2. Mark its place for later resumption.
3. Determine the type of request.
4. Jump to an event-specific interrupt service routine (or task) to service the request.
5. Return to the interrupted program and resume execution at the point of interruption.

Hardware Interrupts

There are two types of interrupt requests that can be initiated by hardware external to the processor:

- Maskable interrupt requests initiated by hardware devices. These requests are delivered to the interrupt controller which, in turn, delivers the interrupt to the processor by asserting the INTR signal to the processor. They are referred to as maskable interrupts because the programmer may disable the processor’s ability to recognize the INTR signal and may also program the interrupt controller to selectively disable recognition of interrupt requests from certain devices.
- Non-Maskable Interrupt (NMI) requests issued by the chipset to signal that a serious hardware condition has been detected on the system board. These interrupts are delivered to the processor by asserting the processor’s NMI input signal. The programmer cannot disable the processor’s ability to recognize and respond to the assertion of its NMI input.
Chapter 14: Interrupts and Exceptions

For more detailed coverage of hardware interrupt generation and servicing in the PC-compatible environment, refer to “The Local and IO APICs” on page 1497. It should be stressed that the current chapter describes externally generated hardware interrupt servicing as it is handled by a processor without a Local APIC or when the Local APIC is disabled.

Maskable Interrupt Requests

IO devices typically generate an interrupt request to signal conditions such as:

- an action required on the part of the program in order to continue operation.
- a previously-initiated operation has been completed with no errors encountered
- a previously-initiated operation has encountered an error condition and cannot continue.

In any of these cases, the IO device asserts an interrupt request signal to the interrupt controller, which in turn asserts INTR (maskable interrupt request) to the processor.

An interrupt request may be temporarily ignored by the processor if the programmer has disabled recognition of requests from IO devices by executing a Clear Interrupt Enable (CLI) instruction. This clears the EFlags[IF] bit to zero, causing the processor to ignore its INTR input until a Set Interrupt Enable (STI) instruction is executed.

This feature must be used cautiously as it delays the processor’s servicing of interrupt requests generated by external hardware devices. Many IO devices are sensitive to lengthy delays while awaiting service and may suffer data overrun or underrun conditions if their interrupt requests are not serviced on a timely basis.

In Protected Mode, the processor is sensitive to the value in the EFlags[IOPL] field when executing the CLI and STI instructions. They may only be successfully executed when the current program’s CPL meets or exceeds the IOPL (IO Privilege Level). Any attempt to execute them with insufficient privilege results in a GP exception.

Other operations that affect EFlags[IF] are:

- Reset clears EFlags[IF], inhibiting recognition of maskable interrupts.
- The PUSHF (Push Flags) instruction copies the contents of the EFlags register to the stack and then clears the EFlags[IF] bit. The EFlags bits, including IF, can then be examined and modified in stack memory.
The Unabridged Pentium® 4

- The POPF instruction copies the EFlags image from stack memory into the EFlags register.
- A task switch modifies the EFlags register when it copies the EFlags field from the new TSS into EFlags. Task switching is covered in the chapter entitled “Mechanics of a Task Switch” on page 191.
- The IRET instruction copies the EFlags image from stack memory into the EFlags register.
- An interrupt that selects an IDT entry containing an Interrupt Gate descriptor clears EFlags[IF] after EFlags has been copied to stack memory.

Maskable Interrupt Servicing

Automatic Actions

If interrupt recognition is enabled and the processor’s INTR input is sampled asserted, the processor begins to service the hardware request upon completion of the currently executing instruction. This discussion assumes that the system interrupt controller consists of an 8259A Programmable Interrupt Controller (PIC; the APIC did not make its appearance until the P54C version of the Pentium® processor). In response to the assertion of INTR, the following sequence of actions is performed by the processor:

1. Two, back-to-back Interrupt Acknowledge transactions are generated on the FSB (starting with the advent of the Pentium® Pro processor, this was reduced to a single Interrupt Acknowledge transaction). The first one tells the 8259A interrupt controller to prioritize the currently pending interrupt requests from IO devices. The second one is a request to the PIC for the interrupt vector number associated with the highest-priority request (the 8-bit vector is used as an index into the IDT in memory).
2. Using the vector to select an IDT entry, the processor reads the contents of the indicated IDT descriptor from memory.
3. The processor pushes the contents of its CS, EIP and EFlags registers onto the stack. This is necessary to save its place in the interrupted program.
4. EFlags[IF] is cleared to disable recognition of subsequent interrupt requests.
5. The processor jumps to the device-specific interrupt service routine indicated in the IDT entry. If the IDT entry contains a Task Gate descriptor, the processor performs a task switch and begins execution of the interrupt service task.

The actions just described are the ones that the processor performs automatically in order to start an interrupt service routine. The following discussion assumes that the IDT entry did not contain a Task Gate descriptor (and therefore an interrupt handler in the same task will be executed).
Chapter 14: Interrupts and Exceptions

Actions Performed by the Software Handler

After entering the interrupt service routine, the programmer must perform the following actions:

1. Save (in stack memory) the contents of any registers that will be altered in this routine. When control is returned to the interrupted program, all registers must contain their original contents in order to ensure proper operation of the interrupted program.

2. Check the device’s status and perform any device-specific servicing requested by the device.

3. Issue an End-of-Interrupt (EOI) command to the 8259A interrupt controller to clear the request.

4. Execute an Interrupt Return (IRET) instruction. This causes the processor to pop the original CS, EIP and EFlags values from the stack and load them into their respective registers (reenabling recognition of external, hardware interrupts).

5. The processor resumes execution of the interrupted program.

PC-Compatible Vector Assignment

Table 14-1 on page 256 defines the typical hardware interrupt request line assignment in a PC-compatible machine. It identifies the IDT entry number associated with each.

The table also highlights a particularly aberrant characteristic of the PC-compatible architecture. The original IBM PC was based on the Intel® 8088 processor. As with any of the x86 processors, the 8088 generates software exceptions when certain special conditions are detected. Intel® dedicated IDT entries 0 through 7 for these software exception conditions. The PC BIOS programmed the 8259A interrupt controller to associate IDT entries 8 through 15d (Fh) with the hardware interrupt lines IRQ0 through IRQ7. In order to be backward-compatible, the IBM PC-AT’s interrupt controller was also programmed to use IDT entries 8 through 15d for these hardware interrupts. However, the PC-AT was designed around the 286 processor and that processor generates more types of software exceptions than did the 8088. These new exceptions used IDT entries 8 through 13d. Later machines based on the post-286 processors and they added additional exceptions using IDT entries 14d and 15d. In other words, IDT entries 8 through 15d can be selected when either a hardware interrupt or a software exception event occurs. Table 14-1 on page 256 explains the actions software must take in order to ensure that all hardware and software events are serviced correctly.
15 Virtual 8086 Mode

The Previous Chapter
This chapter provided a detailed description of all of the various types of interrupts and exceptions. A detailed description of the Local and IO APICs can be found in “The Local and IO APICs” on page 1497.

This Chapter
This chapter provides a detailed description Virtual 8086 Mode (VM86 Mode). This description is directly applicable to all subsequent IA32 processors. VM86 Mode was enhanced starting with the advent of the Pentium® processor and a detailed description of those enhancements can be found in “VM86 Extensions” on page 490.

The Next Chapter
This chapter provides a detailed description of the Debug register set. This description is directly applicable to all subsequent IA32 processors. This feature was enhanced starting with the advent of the Pentium® processor and a detailed description of the enhancement can be found in “Debug Extension” on page 497.

A Special Note
The terms “DOS task” and “VM86 task” are used interchangeably in this chapter (because the vast majority of VM86 tasks are DOS tasks). It should not be construed, however, that only DOS tasks are candidates to be treated as VM86 tasks. Any Real Mode task that must be executed by a multitasking OS should be set up as a VM86 task.
DOS Application—Portrait of an Anarchist

The chapter entitled “Multitasking Problems” on page 31 introduced some of the ways in which DOS programs are disruptive in a multitasking environment. They:

- may attempt to access memory belonging to currently-suspended programs,
- may communicate directly with IO ports,
- can call OS code (even routines they shouldn’t be able to),
- may disable interrupt recognition when they don’t wish to be interrupted,
- frequently call BIOS routines to indirectly communicate with IO devices (thereby bypassing the OS).

In addition, the task assumes that DOS is the OS it is interacting with when it may be a completely different OS (e.g., Windows XP). In this case, all OS calls initiated by the DOS task must be intercepted and passed to the host OS (or another program that substitutes for the DOS OS).

Solution—Set a Watchdog on the DOS Application

Starting with the 386 processor, Intel®’s solution to this problem is to provide a hardware/software combination tasked with monitoring the behavior of a DOS program on an instruction-by-instruction basis and intercepting all actions which may prove injurious to the overall multitasking environment.

The OS creates a separate 32-bit TSS (see Figure 15-1 on page 331) associated with each DOS task. It cannot be a 16-bit, 286-style TSS because:

- The 286 TSS only has a 16-bit field for the Flag register image.
- It doesn’t have a 32-bit EFlags register field containing the VM bit.

When the OS creates the TSS for a DOS task, it sets the VM bit to one in the EFlags register image within the TSS. Whenever a task switch to a DOS task occurs, the processor copies the EFlags image from the task’s TSS into the EFlags register, setting EFlags[VM] = 1. EFlags[VM] = 1 informs the processor that the current task is a DOS task and enables the processor’s watchdog logic that monitors for anarchistic behavior. Note that “watchdog” is the author’s term, not Intel®’s.
Chapter 15: Virtual 8086 Mode

Figure 15-1: Task State Segment (TSS)

The Virtual Machine Monitor (VMM)

When the processor’s internal hardware associated with VM86 mode detects that the currently executing DOS task is attempting a potentially disruptive action, it suspends the VM86 task and jumps to the GP (General Protection) exception handler. As with any exception, before jumping to the exception handler, the processor first stores the current EFlags register contents (along with CS and EIP) on the stack. It then clears the EFlags[VM] bit, disabling VM86.
mode. Upon entry to the GP exception handler, the programmer examines the VM bit in the EFlags image stored on the stack to determine if the exception was generated by a DOS task (i.e., EFlags[VM] = 1). If it was, the GP exception handler jumps to the watchdog program. If it wasn’t, the body of the normal, Protected Mode GP exception handler is executed.

The watchdog program associated with a DOS task is referred to as the Virtual Machine Monitor (VMM). The VMM’s job is to identify the action attempted by the DOS task and to accomplish it in a manner that is not disruptive to the multitasking OS or to the other, currently suspended tasks. In order to have full access to all of the processor’s facilities to deal with problems, the VMM executes at privilege level 0.

Having emulated the potentially disruptive action in a benign fashion, the VMM program then resumes execution of the DOS task at the instruction after the one that caused the exception.

The discussion in this chapter indicates that the GP exception handler code determines whether a VM86 task was executing when the exception occurred and that it jumps to the VMM program if this is the case. Please note that rather than having the GP handler jump to the VMM program, the VMM program itself could serve as the GP exception handler.

---

**The Unabridged Pentium® 4**

---

**Entering or Reentering VM86 Mode**

---

**Task Creation, Startup and Suspension**

**Create a TSS**

Before the multitasking OS initially starts a DOS task, it creates a 32-bit TSS for the task, setting the EFlags[VM] bit to one in the TSS’s EFlags field. It also creates a TSS descriptor (in the GDT) that points to the task’s TSS in memory.

**Each Task Gets a Timeslice**

A multitasking OS usually permits a task to execute for a predefined period of time, typically referred to as a timeslice. This is accomplished by triggering a hardware timer prior to starting (or resuming) the task. The task is then started by the OS scheduler and continues to execute until a hardware interrupt is generated by the timeslice timer (unless the task is interrupted prior to this for some other reason). The timer interrupt selects an IDT entry containing a Task
Gate that points to the OS’s task scheduler. The task that was executing is suspended and the new task (i.e., the OS task scheduler) is resumed.

Unlike many other processors (e.g., the PowerPC processor family), the 386 processor did not incorporate a hardware “timeslice” timer to facilitate the timeslice approach to multitasking. Instead, the system designer had to incorporate a hardware timer external to the processor. This timer was implemented as an IO device that could be programmed for the desired interval and then enabled. The timer generates a maskable interrupt to the processor when it expires. Since the advent of the P54C version of the Pentium® processor, however, each IA32 processor implements the Local APIC which includes a programmable timer capable of generating an interrupt on expiration or at set intervals.

Select DOS Task via a Far Call or a Far Jump

The task is started by executing a far jump or a far CALL instruction with a CS value that selects the TSS descriptor (associated with the task) in the GDT. The offset portion of the target address is discarded.

When the processor determines that a TSS descriptor has been selected, it suspends the current task (in this case, the OS task scheduler) by copying the majority of the processor’s registers into the OS scheduler’s TSS. It then switches to the DOS task by loading the processor’s register set from the DOS task’s TSS. When the EFlags register is loaded from the TSS, EFlags[VM] is set to one, automatically placing the processor into VM86 mode. In other words, the watchdog logic is activated just before the task starts (or resumes) execution.

An Interrupt or Exception Causes an Exit From VM86 Mode

General

The processor temporarily exits VM86 mode when an interrupt or exception occurs. The IDT entry selected by the interrupt or exception can contain one of the following descriptor types:

- **A Task Gate descriptor.** When the interrupt or exception selects an IDT entry that contains a Task Gate, a task switch occurs—the current task is suspended and another task is initiated.

- **A Trap Gate or an Interrupt Gate.** A task switch does not occur when an entry containing a Trap Gate or an Interrupt Gate is selected. Rather, the processor executes the interrupt or exception handler pointed to by the selected IDT descriptor.
The Debug Registers

The Previous Chapter
This chapter provided a detailed description Virtual 8086 Mode (VM86 Mode). This description is directly applicable to all subsequent IA32 processors. VM86 Mode was enhanced starting with the advent of the Pentium® processor and a detailed description of those enhancements can be found in “VM86 Extensions” on page 490.

This Chapter
This chapter provides a detailed description of the Debug register set. This description is directly applicable to all subsequent IA32 processors. This feature was enhanced starting with the advent of the Pentium® processor and a detailed description of the enhancement can be found in “Debug Extension” on page 497.

The Next Chapter
The next chapter is for those who feel the need for a primer on cache memory. For those who don’t feel the need for it, please move on to the next chapter. It occupies this place in the book because the next two chapters cover the 486 processor, the first IA32 processor to incorporate an integrated cache.

The Debug Registers
Starting with the 386, all IA32 processors provide hardware breakpoint detection. This is implemented using the processor’s debug registers. These registers are illustrated in Figure 16-1 on page 377. Other processor functions associated with debug include:
The debug exception (Exception 1). This exception is generated when the processor encounters a breakpoint match on a condition specified in the debug registers.

- The breakpoint instruction exception (Exception 3). This exception is generated when the processor executes the breakpoint (INT3) instruction.
- The Trap bit in a task's TSS. Causes a debug exception when a task switch occurs to a task with this bit (the T bit) set to one.
- The EFlags[RF]. When set to one by the debugger, the subsequent execution of the IRETD instruction prevents the processor from generating a debug exception again when it returns to an instruction that already caused a debug exception.
- The EFlags[TF]. When set to one, the processor generates a debug exception before the execution of each instruction. This permits single-stepping through a program.

The Debug Control register, DR7, is used to enable one or more breakpoints. Table 16-1 on page 378 describes the bits in DR7.
Using the processor’s DR7, DR0, DR1, DR2 and DR3 registers, the programmer may enable the processor to detect any of four different types of accesses to up to four different memory or IO addresses specified in DR0-DR3. The Pentium® processor added the capability to monitor for read or write accesses to IO ports. The 386 and 486 processors did not possess this capability.

When a breakpoint is detected, the processor generates a debug exception (Exception 1) and jumps to the debug exception handler routine. In addition, the processor sets the appropriate bits in the Debug Status register, DR6. Table 16-4 on page 381 describes the bits in DR6.
### Table 16-1: Definition of DR7 Bits Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W0</td>
<td>Defines the type of access to the address specified in DR0 that the processor will look for a match on. Table 16-2 on page 380 defines the interpretation of the value in this field.</td>
</tr>
<tr>
<td>LEN0</td>
<td>Defines the size of the access to the address specified in DR0 that the debug logic will monitor for. The interpretation of the value in this field is defined in Table 16-3 on page 381.</td>
</tr>
<tr>
<td>L0</td>
<td>Enable local breakpoint 0. When set to one, a debug exception will be generated if the debug logic detects a match on an access of the type and length specified in the R/W0 and LEN0 fields to the address specified in DR0 while in the current task. This bit is automatically cleared when a task switch occurs. This prevents the generation of a debug exception on an access match while in another task.</td>
</tr>
<tr>
<td>G0</td>
<td>Enable global breakpoint 0. When set to one, a debug exception will be generated if the debug logic detects a match on an access of the type and length specified in the R/W0 and LEN0 fields to the address specified in DR0 while in any task.</td>
</tr>
<tr>
<td>R/W1</td>
<td>Defines the type of access to the address specified in DR1 that the processor will look for a match on. Table 16-2 on page 380 defines the interpretation of the value in this field.</td>
</tr>
<tr>
<td>LEN1</td>
<td>Defines the size of the access to the address specified in DR1 that the debug logic will monitor for. The interpretation of the value in this field is defined in Table 16-3 on page 381.</td>
</tr>
<tr>
<td>L1</td>
<td>Enable local breakpoint 1. When set to one, a debug exception will be generated if the debug logic detects a match on an access of the type and length specified in the R/W1 and LEN1 fields to the address specified in DR1 while in the current task. This bit is automatically cleared when a task switch occurs. This prevents the generation of a debug exception on an access match while in another task.</td>
</tr>
<tr>
<td>G1</td>
<td>Enable global breakpoint 1. When set to one, a debug exception will be generated if the debug logic detects a match on an access of the type and length specified in the R/W1 and LEN1 fields to the address specified in DR1 while in any task.</td>
</tr>
</tbody>
</table>
Chapter 16: The Debug Registers

Table 16-1: Definition of DR7 Bits Fields (Continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W2</td>
<td>Defines the type of access to the address specified in DR2 that the processor will look for a match on. Table 16-2 on page 380 defines the interpretation of the value in this field.</td>
</tr>
<tr>
<td>LEN2</td>
<td>Defines the size of the access to the address specified in DR2 that the debug logic will monitor for. The interpretation of the value in this field is defined in Table 16-3 on page 381.</td>
</tr>
<tr>
<td>L2</td>
<td>Enable local breakpoint 2. When set to one, a debug exception will be generated if the debug logic detects a match on an access of the type and length specified in the R/W2 and LEN2 fields to the address specified in DR2 while in the current task. This bit is automatically cleared when a task switch occurs. This prevents the generation of a debug exception on an access match while in another task.</td>
</tr>
<tr>
<td>G2</td>
<td>Enable global breakpoint 2. When set to one, a debug exception will be generated if the debug logic detects a match on an access of the type and length specified in the R/W2 and LEN2 fields to the address specified in DR2 while in any task.</td>
</tr>
<tr>
<td>R/W3</td>
<td>Defines the type of access to the address specified in DR3 that the processor will look for a match on. Table 16-2 on page 380 defines the interpretation of the value in this field.</td>
</tr>
<tr>
<td>LEN3</td>
<td>Defines the size of the access to the address specified in DR3 that the debug logic will monitor for. The interpretation of the value in this field is defined in Table 16-3 on page 381.</td>
</tr>
<tr>
<td>L3</td>
<td>Enable local breakpoint 3. When set to one, a debug exception will be generated if the debug logic detects a match on an access of the type and length specified in the R/W3 and LEN3 fields to the address specified in DR3 while in the current task. This bit is automatically cleared when a task switch occurs. This prevents the generation of a debug exception on an access match while in another task.</td>
</tr>
<tr>
<td>G3</td>
<td>Enable global breakpoint 3. When set to one, a debug exception will be generated if the debug logic detects a match on an access of the type and length specified in the R/W3 and LEN3 fields to the address specified in DR3 while in any task.</td>
</tr>
</tbody>
</table>
Caching Overview

The Previous Chapter
This chapter provided a detailed description of the Debug register set. This description is directly applicable to all subsequent IA32 processors. This feature was enhanced starting with the advent of the Pentium® processor and a detailed description of the enhancement can be found in “Debug Extension” on page 497.

This Chapter
This chapter is for those who feel the need for a primer on cache memory. For those who don’t feel the need for it, please move on to the next chapter. It occupies this place in the book because the next two chapters cover the 486 processor, the first IA32 processor to incorporate an integrated cache.

The Next Chapter
The next chapter provides a description the 486 processor’s hardware-related characteristics. This includes the 486 roadmap, an overview of the 486 internal architecture, an overview of the 486 FSB, the A20 Mask signal, the on-die cache, and the on-die FPU.

Definition of a Load and a Store
This chapter (and many other throughout the book contains many references to loads and stores:

• A load is a memory data read.
• A store is a memory data write.
The Unabridged Pentium® 4

The Cache’s Purpose

Without a Cache, Core Stalls Were Common

When an IA32 processor prior to the 486 had to perform a memory access (i.e., an instruction fetch, a memory data read, or a memory data write), the processor had to arbitrate for ownership of its FSB in order to perform the memory read or write on the FSB. The processor core’s ability to continue program execution (and therefore its performance) was affected as follows:

• If the processor was performing a memory code read to prefetch the next instruction from memory, it could affect the processor’s ability to continue with program execution. The earlier processors had a very shallow instruction prefetch buffer to supply instructions to the processor’s execution unit. The processor core may have been executing code (i.e., instructions) at a fairly high frequency rate and the external memory from which the prefetch was being performed may have been quite slow to provide the requested instruction. In this case, the execution unit might have completed the execution of all of the instructions currently in the prefetch queue before the prefetch of the next instruction was completed on the FSB. The processor core would then experience starvation and would have stalled until the code fetch from memory completed on the FSB.

• If the processor was performing a memory code read due to the execution of a branch instruction, an interrupt, or an exception, the instructions currently in the prefetch queue are not the instructions that need to be executed next. The processor would purge them all from the prefetch queue, arbitrate for FSB ownership and initiate a code fetch from memory to obtain the instruction that is being branched to and therefore must be executed next. The processor core would then experience starvation and would have stalled until the code fetch from memory completed on the FSB.

• If the processor was performing a memory data read (i.e., a load operation), the core could not move on to the next instruction until the requested data had been obtained from external memory and was placed in the specified target register. The processor core would stall until the data read from memory completed on the FSB.

• If the processor was performing a memory data write (i.e., a store operation), it could be handled in one of two ways:
  — The core could not move on to the next instruction until the store data had been successfully written to external memory. The processor core would stall until the memory data write transaction was completed on the FSB.
Chapter 17: Caching Overview

— Alternatively, the processor design could include a Posted Write Buffer (PWB) buffer. When a store was executed, the address to be stored to and the data to be written would be latched into the next entry in the PWB. From the perspective of the processor core, the memory write was complete and it could move on to the next instruction in the program. This approach certainly yielded better performance. However, the depth of the processor’s PWB was typically fairly shallow (due to real estate constraints on the processor die). When executing a program that performed a fair number of stores, the PWB could fill up rather rapidly and the processor core would then be forced to stall program execution until one or more of the PWB entries had been written to external memory over the FSB.

An On-Die Cache Eliminates Many Core Stalls

Introduction

With the advent of the 486 processor, all IA32 processors have cache memory integrated onto the processor die. The cache essentially consists of one or more banks of fast access SRAM (Static RAM) memory and an equal number of directories (implemented in SRAM) that keep track of the information (code and data) that currently resides in the on-die cache.

The cache is designed to copy lines of information (code and data) from external memory into the fast access cache memory. Whenever the processor core must perform a code or data access, the memory address to be read from or stored to is submitted to the cache directory (or directories) to determine if a copy of the target memory location(s) is currently in the cache.

On a Cache Miss

The first time that the processor core requests a data or code item (an item could consists of one or more bytes), the cache lookup results in a miss and the processor has to arbitrate for ownership the FSB and initiate a memory transaction on the FSB to fetch the item from memory.

The Cache Line

The cache is not designed to just fetch the requested byte or bytes and place them in the cache. Rather, on a cache miss, the cache is designed to fetch the block of information that contains the requested information that caused the miss. The block is referred to as a line and the size of the line that is fetched from memory and placed in the cache is cache design-specific. Some examples are:
On the 486, the cache line size was 16 bytes.

On the Pentium® and the P6 family processors (i.e., the Pentium® Pro, Pentium® II, Pentium® II Xeon, Pentium® II Celeron, the Pentium® III, Pentium® III Xeon and Pentium® III Celeron), the cache line size was 32 bytes.

On the Pentium® 4, Pentium® 4 Xeon and Pentium® 4 Celeron processors, the cache line size is 128 bytes.

On the Pentium® M processor, the cache line size is 64 bytes.

A line of information in memory always starts at an address boundary that is evenly divisible by the cache line size. The line containing the requested byte or bytes is fetched from memory (this is referred to as a cache line fill operation) and is placed in the cache. In addition, if the line fetch was caused by a load miss, the requested byte or bytes are immediately routed to the processor’s execution unit so it can complete the load instruction. If the line fetch was caused by a store miss, the core stores into the line to complete the store instruction.

The Directory Entry

When the desired line has been fetched from memory and is stored in the cache, the cache also creates a directory entry that records what area of memory the line was fetched from and also keeps track of the current state of the line (more on this in “The Write-Through Cache” on page 388 and “The Write Back Cache” on page 391).

Repeat Accesses to the Same Areas Result in Cache Hits

After a line has been placed in the cache, any subsequent memory accesses to any location(s) within the same line result in a cache hit and the load or store can complete very quickly. This obviously results in dramatically increased performance.

The Write-Through Cache

Introduction

A cache can be designed either as a Write-Through (WT) cache or as a Write Back (WB) cache. The following subsections provide a description of a WT cache’s basic operation.
Chapter 17: Caching Overview

On a Load Miss

When a load is executed, the processor takes the following actions (this example assumes that the cache lookup results in a cache miss):

1. The load cannot be completed until the requested data has been obtained and placed in the specified target register.
2. The processor submits the start memory address specified by the load to the on-die cache for a lookup. This example assumes that the lookup results in a cache miss.
3. The cache forwards the load request upstream to the next level of memory. In an IA32 processor that only implemented an L1 Cache (e.g., the 486 or the Pentium®), the request would have to be submitted to external memory by performing a memory data read transaction on the FSB to fetch the line from memory. On an IA32 processor that implements an L2 Cache on board the processor (e.g., any IA32 processor after the Pentium®), the request would be forwarded to the processor L2 Cache over the BSB (Back Side Bus; a private bus connecting the core to the L2 Cache). If the lookup in the L2 also resulted in miss, the request would have to be forwarded upstream to the next level of memory (either an L3 Cache or system memory).
4. When the requested line is received (either from system memory or from an upstream cache), the critical data (i.e., the originally requested byte or bytes) are immediately forwarded to the execution unit so it can complete the load instruction.
5. The line of information is recorded in one of the cache banks (referred to as Ways).
6. The following information is recorded in an entry of the directory that is associated with the Way in which the line was stored:
   — The address of the memory page (referred as the Tag address) from which the line was fetched.
   — The state of the line is marked as Valid. In a WT cache, a line in the cache can only be in one of two possible states: Valid or Invalid. In all of Intel®’s WT cache designs, they refer to these as the Shared (Valid) and Invalid states. Don’t let the termed Shared in this context confuse you. It just means that the line is valid.

On a Load Hit

When a load is executed, the processor takes the following actions (this example assumes that the cache lookup results in a cache hit):
18 486 Hardware Overview

The Previous Chapter

The previous chapter was for those who feel the need for a primer on cache memory. It occupies this place in the book because the next two chapters cover the 486 processor, the first IA32 processor to incorporate an integrated cache.

This Chapter

This chapter provides a description the 486 processor’s hardware-related characteristics. This includes the 486 roadmap, an overview of the 486 internal architecture, an overview of the 486 FSB, the A20 Mask signal, the on-die cache, and the on-die FPU. The discussion of the A20 Mask signal is directly applicable to all subsequent IA32 processors.

The Next Chapter

This chapter provides a detailed description of the software enhancement included in the 486 processor. This discussion is directly applicable to all subsequent IA32 processors and covers:

- The on-die FPU.
- The Alignment Checking Feature.
- Paging-Related Changes.
- Caching-Related Changes to the Programming Environment.
- The Test Registers.
- Instruction Set Changes.
- New/Altered Exceptions.
486 Flavors

The 486 processor was produced in the following flavors (shown in order of introduction; note that all of them incorporated an internal cache):

- **486SX/487SX.** The original incarnation of the 486 did not have an integrated FPU. Rather, the system board included a socket into which the 487 numeric coprocessor could be installed. In fact, the 487 was a full-blown 486 processor with an integrated x87 FPU. When installed, it asserted a signal to the 486 that caused it to float all of its output drivers so the 487 could take over the role of the system processor. This processor integrated an internal, unified, write-through code/data cache.
- **486SX2.** This version of the 486SX was the first IA32 processor to use an internal clock multiplier. An internal PLL (Phase Locked Loop) multiplied the bus clock frequency by two to yield the internal processor clock speed. This processor integrated an internal, unified, write-through code/data cache. It did not implement an integrated FPU.
- **486DX.** This processor implemented an integrated FPU and an on-die, unified, write-through code/data cache. All subsequent IA32 processors implemented an on-die FPU.
- **486DX2 (WT).** This version of the 486 integrated the FPU and a clock multiplier (x2). This processor integrated an internal, unified, write-through code/data cache.
- **486DX2 (WB).** This version of the 486 integrated the FPU and a clock multiplier (x2). This processor integrated an internal, unified, write-back code/data cache. It was the first IA32 processor to implement a WB (write back) cache.
- **486DX4.** Same as the 486DX2 except it implemented a x4 clock multiplier.

Note that the earlier versions of the 486 did not implement SMM, but all later versions did.

An Overview of the 486 Internal Architecture

Figure 18-1 on page 414 illustrates the internal architecture of the 486 processor. It should be noted that the initial version of the 486 did not incorporate the FPU. The 486 processor core consisted of the following units:

- **Bus Unit.** Interfaces the processor to the FSB and the system in general.
- **Instruction Prefetch Unit.** Working on the presumption that the currently executing program never executes jumps, it instructs the Bus Unit to perform a series of memory code read transactions from ascending memory addresses.
Chapter 18: 486 Hardware Overview

- **Prefetch Queue (not shown).** The instructions prefetched from memory are placed in this queue. The queue was 16 bytes deep on the 386 and was increased to 32 bytes on the 486.

- **Instruction Decoder.** Consisted of a two-stage decoder. Decodes each instruction into an executable form.
  - Decode Stage 1. Performed the preliminary instruction decode.
  - Decode Stage 2. Accomplishes the following:
    - If the instruction will involve a memory data access, the segment offset is provided to the Segment Unit to be added to the segment start address, yielding the 32-bit linear memory address.
    - If the instruction is a FP instruction, it is forwarded directly to the FPU for execution.
    - Non-FP instructions are provided to the Control Unit for further decode.

- **Control Unit.** Non-FP instructions are submitted to the Microcode ROM which produces a stream of one or more internal operations that, taken together, accomplish the IA instruction. These micro-operations are streamed to the Datapath Unit for execution.

- **Datapath (Execution) Unit.** Executes instructions one at a time as they are provided from the Instruction Queue.

- **Register set (not shown).** As each instruction is executed, the registers are accessed by the Execution Unit on an as-needed basis.

- **Segment Unit.** Whenever a memory access must be performed, the Segment Unit adds the offset of the item to be accessed to the base address of the target segment (code, stack or data segment), thereby producing the 32-bit linear memory address. If Paging is disabled, the linear address is the physical memory address that is accessed by performing a transaction on the FSB.

- **Paging Unit.** If Paging is enabled and a memory access must be performed, the 32-bit linear memory address is submitted to the Paging Unit for a lookup in the Page Directory and a Page Table. The selected Page Table Entry (PTE) is then used to translate the 32-bit linear memory address into a 32-bit physical memory address. The resultant physical memory address is then accessed by performing a transaction on the FSB.

- **Cache Unit.** The 486 was the first IA32 processor with an integrated Cache. It was implemented as a unified code/data cache (i.e., it caches both code and data and does not discriminate between the two). A unified cache has two disadvantages:
  - It services requests from both the Execution Unit as well as the instruction prefetcher. If simultaneous requests are submitted to the cache, it stalls the prefetcher’s request and services the Execution Unit’s request. This causes stutters in performance.
— Fetching a line of data into the cache can cause a previously fetched code line to be evicted to make room for the new data line. Conversely, fetching a line of code into the cache can cause a previously fetched data line to be evicted to make room for the new code line.

Figure 18-1: 486 Internal Architecture
An Overview of the 486 FSB

Address/Data Bus Structure

The 486DX processor implemented the same address/data bus structure as that found on the 386DX processor (see Figure 5-2 on page 44).

On a Cache Miss, an Entire Line Must Be Fetched

When the Instruction Prefetcher or the Execution Unit submits a memory access request to the internal cache, the line that contains the critical data (i.e., the requested data) may not be in the cache. In this event, the processor uses the FSB to fetch the line containing the critical data from memory. The cache line size for the 486 processor was 16 bytes (four dwords). If the 486 FSB were implemented in the same manner as the 386DX processor, the processor would have to perform four separate dword reads from memory to obtain the requested line. This would take a considerable amount of time and the requester (i.e., the Instruction Prefetcher or the Execution Unit) would be stalled during this time.

486 Implemented a Burst Line Fill Transaction

Background

The 486 processor was the first IA32 processor to implement the burst transaction. Rather than performing four separate memory read transactions each comprised of an Address Phase and a Data Phase on a cache miss, the 486 performed a burst memory read transaction consisting of a single Address Phase and four Data Phases.

Each 16-byte cache line starts on an address boundary divisible by 16 and consisting of four dwords. On a cache miss, the processor would address the critical dword (i.e., the one containing the requested code or data) at the start of the transaction. The system memory controller was designed to provide the critical dword to the processor in the first Data Phase, followed by the remaining three dwords in a predefined order. Providing the critical dword first permits the processor core to uninstall the requesting unit as quickly as possible.
The Previous Chapter
This chapter provided a description the 486 processor’s hardware-related characteristics. This included the 486 roadmap, an overview of the 486 internal architecture, an overview of the 486 FSB, the A20 Mask signal, the on-die cache, and the on-die FPU. The discussion of the A20 Mask signal is directly applicable to all subsequent IA32 processors.

This Chapter
This chapter provides a detailed description of the software enhancement included in the 486 processor. This discussion is directly applicable to all subsequent IA32 processors and covers:

- The on-die FPU.
- The Alignment Checking Feature.
- Paging-Related Changes
- Caching-Related Changes to the Programming Environment.
- The Test Registers.
- Instruction Set Changes.
- New/Altered Exceptions.

The Next Chapter
This chapter provides an overview of the Pentium® processor’s hardware design characteristics. This includes:

- The Pentium® roadmap.
- An overview of the Pentium® internal architecture.
- An overview of the Pentium® FSB.
- The Caches.
- The Local APIC.
The Unabridged Pentium® 4

- The Test Access Port (TAP). This discussion is directly applicable to all subsequent IA32 processors.
- FRC Mode. This discussion is directly applicable to all subsequent IA32 processors up to and including the Pentium® III processor.
- Soft Reset (INIT#). This discussion is directly applicable to all subsequent IA32 processors.

FPU Added On-Die

Introduction

Prior to the advent of the 486DX processor, IA32 processors did not include an on-die FPU. Rather, the end user had to add an external x87 FPU chip to the system and the processor treated it as a specialized IO device. Whenever the processor encountered a FPU instruction while fetching the current program from memory, it had to perform a series of one or more IO writes to send the instruction to the off-chip FPU to be executed. Obviously, this was very inefficient.

The 486DX processor incorporated the x87 FPU on the processor die (see Figure 19-1 on page 433) as another execution unit and all subsequent IA32 processors include the on-die FPU. The sections that follow provide a description of the FPU’s register set as well as the format in which FP numbers are represented.
Chapter 19: 486 Software Enhancements

Figure 19-1: 486 with Integrated FPU
FPU-Related Register Set Changes

Refer to Figure 19-2 on page 434. The 486DX processor was the first IA32 processor to incorporate the FPU onto the processor die (the 486SX did not incorporate the x87 FPU; rather, it required the 487SX to perform FP operations).

In addition to the addition of the FPU register set to the processor, one bit was altered in CR0 and another was added:

- In the earlier processors, CR0[ET] was a read/write bit used by software to indicate the type of numeric coprocessor installed on the system board. The ET bit is now hardwired to 1 to indicate the presence of a 387 style x87 FPU.
- CR0[NE] was added. Refer to “DOS-Compatible FP Error Reporting” on page 445 and “FP Error Reporting Via Exception 16” on page 446 for a description of this bit.
Chapter 19: 486 Software Enhancements

The CR0 FPU Control Bits

Refer to Figure 19-2 on page 434. CR0[EM] and CR0[MP] control the processor’s x87 FPU. Table 19-1 on page 435 defines how software uses these two control bits.

Table 19-1: CR0 FPU Control Bits

<table>
<thead>
<tr>
<th>CR0[EM]</th>
<th>CR0[MP]</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0       | 0       | This setting is used when an x87 FPU is present and the OS is not a multitasking OS (e.g., when running DOS):  
  • CR0[EM] = 0 indicates that the x87 FPU is present and enables the x87 FPU to execute FP instructions. If an IA32 processor incorporates MMX technology, this setting enables execution of MMX instructions. If an IA32 processor incorporates SSE/SSE2/SSE3 technology, this setting enables execution of these instructions. The SSE and SSE2 instructions that are not affected by the EM flag are the PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, and CLFLUSH instructions.  
  • CR0[MP] = 0 causes the processor to ignore the state of CR0[TS] when executing the WAIT/FWAIT instruction. |
| 0       | 1       | This setting is used when an x87 FPU is present and the OS is a multitasking OS:  
  • CR0[EM] = 0 indicates that the x87 FPU is present and enables the x87 FPU to execute FP instructions. If an IA32 processor incorporates MMX technology, this setting enables execution of MMX instructions. If an IA32 processor incorporates SSE/SSE2/SSE3 technology, this setting enables execution of these instructions. The SSE and SSE2 instructions that are not affected by the EM flag are the PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, and CLFLUSH instructions.  
  • CR0[MP] = 1 causes the processor to test the state of CR0[TS] when executing the WAIT/FWAIT instruction and to generate a DNA exception if CR0[TS] = 1. |
The Previous Chapter
This chapter provided a detailed description of the software enhancement included in the 486 processor. This discussion is directly applicable to all subsequent IA32 processors and covered:

- The on-die FPU.
- The Alignment Checking Feature.
- Paging-Related Changes.
- Caching-Related Changes to the Programming Environment.
- The Test Registers.
- Instruction Set Changes.
- New/Altered Exceptions.

This Chapter
This chapter provides an overview of the Pentium® processor’s hardware design characteristics. This includes:

- The Pentium® roadmap.
- An overview of the Pentium® internal architecture.
- An overview of the Pentium® FSB.
- The Caches.
- The Local APIC.
- The Test Access Port (TAP). This discussion is directly applicable to all subsequent IA32 processors.
- FRC Mode. This discussion is directly applicable to all subsequent IA32 processors up to and including the Pentium® III processor.
- Soft Reset (INIT#). This discussion is directly applicable to all subsequent IA32 processors.
The Next Chapter

This chapter provides a description of the software enhancements incorporated in the Pentium® processor. This discussion is directly applicable to all subsequent IA32 processors. It includes:

- The VM86 Extensions.
- Protected Mode Virtual Interrupts.
- The Debug Extension.
- The Time Stamp Counter.
- 4MB Pages.
- the Machine Check Architecture (MCA).
- Performance Monitoring.
- The Local APIC Register Set.
- The MSRs Added.
- Instruction Set Changes.
- New/Altered Exceptions.

Pentium® Flavors

The Pentium® processor evolved through three basic incarnations (there were many speed variations of each):

- The initial version was the P5. It implemented neither the Local APIC nor the MMX instruction set. It was the first IA32 processor that had separate code and data caches (each 8KB in size).
- The P54C version was the first IA32 processor to implement the Local APIC. Its FSB arbitration scheme supported dual P54C processors on the FSB.
- The P55C version was the first IA32 processor to implement MMX. It also doubled the size of the code and data caches from 8KB to 16KB each.

An Overview of the Pentium® Internal Architecture

The First Superscalar IA32 Processor

Processors prior to the Pentium® had a single instruction pipeline and could only execute one instruction per clock cycle. Refer to Figure 20-1 on page 466. The Pentium® was the first IA32 processor that employed parallel execution units capable of executing multiple instructions simultaneously. The Pentium® processor had dual instruction pipelines and could therefore execute up to two instructions per clock cycle.
Chapter 20: Pentium® Hardware Overview

The two instruction pipelines were called the “u” and the “v” pipelines and they originate directly beneath the code cache in the illustration. The two pipelines were comprised of the stages described in Table 20-1 on page 465.

Table 20-1: The Pentium® Instruction Pipeline Stages

<table>
<thead>
<tr>
<th>“u” Stages</th>
<th>“v” Stages</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefetch</td>
<td></td>
<td>The instructions that comprise the currently executing program are prefetched from the code cache (or over the FSB if it’s uncacheable memory or there’s a cache miss). The instructions are distributed into the two 64-byte Prefetch Buffers associated with the two instruction pipelines.</td>
</tr>
<tr>
<td>Decode 1</td>
<td></td>
<td>During D1, the opcodes are decoded in both pipelines to determine whether the two instructions can be paired according to the Pentium® processor’s pairing rules. If pairing is possible, the two instructions are sent in unison to the stage two decode.</td>
</tr>
<tr>
<td>Decode 2</td>
<td></td>
<td>During D2 the address of memory resident operands are calculated.</td>
</tr>
</tbody>
</table>
| Complex Decode | The “v” pipeline does not implement this stage. All complex instructions must be routed through the “u” pipeline. | Also referred to as the Microcode Unit, the Control Unit consists of the following sub-units:  
  * the Microcode Sequencer  
  * the Microcode Control ROM  
  It interprets the instruction word and microcode entry points fed to it by the Decode 2 stage. It handles exceptions, breakpoints and interrupts. In addition, it controls the integer pipelines and FP sequences. |
| Integer Execution | The two ALUs perform the arithmetic and logical operations specified by the instructions in their respective pipeline. The ALU for the “u” pipeline can complete an operation prior to the ALU in the “v” pipeline, but the opposite is not true. | |
| Register Writeback | The results of the instruction’s execution are committed to the processor’s register set. | |
Figure 20-1: The P5 Internal Architecture
Chapter 20: Pentium® Hardware Overview

Brief Core Description

The Pentium® P5 processor core consisted of the following units:

- **FSB Unit.** The FSB unit provided the physical interface between the Pentium® processor and the system.
- **Data Cache.** The Data Cache kept copies of the most frequently used data requested by the two integer pipelines and the FPU. The data cache was an 8KB write-back cache, organized as a 2-way set associative cache with a cache line size of 32-bytes. The Data Cache directory was triple ported to allow simultaneous access from each of the pipelines and to support snooping.
- **Code Cache.** The code cache (instruction cache) kept copies of the most frequently used instructions. The code cache was an 8KB cache dedicated to supplying instructions to each of the processor's execution pipelines. The cache was organized as a 2-way set associative cache with a line size of 32 bytes. The cache directory was triple ported to allow two simultaneous accesses from the prefetcher and to support snooping.
- **Prefetcher.** Instructions were requested from the code cache by the prefetcher. If the requested line was not in the cache, a burst memory transaction was performed on the FSB to fetch the line from system memory. Prefetches were made sequentially until a branch instruction was fetched. The Prefetcher accessed two lines simultaneously when the starting prefetch address fell in the middle of a cache line. In this way, a split-line access could be made to fetch an instruction that resides partially in two separate lines within the cache.
- **Branch Target Buffer (BTB).** The Pentium® was the first IA32 processor that included branch prediction logic. This consisted of a special, high speed look-aside cache that kept history on the execution of branch instructions. Whenever a branch instruction entered the pipeline, the BTB used the memory address that the branch was fetched from to perform a lookup. A BTB miss meant that the processor had no history on the branch. As a result, the processor would not predict the branch as taken. A BTB hit indicated that the BTB had seen the branch executed one or more times in the past and had recorded whether the branch was taken or not. The processor would therefore use the BTB history to predict whether or not the branch would be taken when it arrived at the ALU and was executed. If the branch was predicted to be taken, any instruction already in the pipeline that came after the branch instruction were deleted and the Prefetcher would be instructed to start fetching instructions from the predicted branch target address.
The Previous Chapter

This chapter provided an overview of the Pentium® processor’s hardware design characteristics. This included:

- The Pentium® roadmap.
- An overview of the Pentium® internal architecture.
- An overview of the Pentium® FSB.
- The Caches.
- The Local APIC.
- The Test Access Port (TAP). This discussion is directly applicable to all subsequent IA32 processors.
- FRC Mode. This discussion is directly applicable to all subsequent IA32 processors up to and including the Pentium® III processor.
- Soft Reset (INIT#). This discussion is directly applicable to all subsequent IA32 processors.

This Chapter

This chapter provides a description of the software enhancements incorporated in the Pentium® processor. This discussion is directly applicable to all subsequent IA32 processors. It includes:

- The VM86 Extensions.
- Protected Mode Virtual Interrupts.
- The Debug Extension.
- The Time Stamp Counter.
The Unabridged Pentium® 4

- 4MB Pages.
- the Machine Check Architecture (MCA).
- Performance Monitoring.
- The Local APIC Register Set.
- The MSR Added.
- Instruction Set Changes (including MMX).
- New / Altered Exceptions.

The Next Chapter

This chapter provides the P6 processor roadmap.

VM86 Extensions

The VME feature was first implemented in the Pentium® processor. It was migrated backwards into the later versions of the 486 and is present in all IA32 processor subsequent to the Pentium®.

Introduction

The chapter entitled “Virtual 8086 Mode” on page 329 provided a detailed description of VM86 Mode as implemented on the 386 processor. VM86 Mode operation on the early versions of the 486 was identical to operation on the 386. The Pentium® processor introduced some improvements to VM86 Mode. Whether or not these improvements are activated is controlled by CR4[VME] (VM86 Mode Extensions; see Figure 21-1 on page 491):

- When CR4[VME] = 0, an IA32 processor’s VM86 Mode is 100% compatible with the 386 version of VM86 Mode.
- If the OS sets CR4[VME] = 1, the improved VM86 features are activated.

CR4 was implemented in the later versions of the 486 processor and is implemented in all subsequent IA32 processors. Executing a CPUID request type 1 returns the processor’s capabilities in the EDX register (see Figure 21-2 on page 491). Bit 1 indicates whether or a processor supports the VME feature.
Efficient CLI/STI Instruction Handling

Background

In 386 VM86 Mode, the attempted execution of a CLI or STI instruction in a VM86 task (i.e., a DOS task) resulted in the generation of a GP exception. This caused the VMM to be executed. The VMM would have to use the pointer that had been pushed into stack memory to determine that the instruction that caused the exception was a CLI or an STI. The handling of an attempted CLI or STI execution was described in:

- “Attempted Execution of a CLI Instruction” on page 365.
- “Attempted Execution of the STI Instruction” on page 368.

This involves quite a bit of software/processor overhead and results in degradation of the performance of the VM86 task.

When the VM86 extensions have been enabled, a VM86 task’s attempt to execute CLI or STI is handled with considerably more grace and without incurring any software/processor overhead.

CLI Handling

Refer to Figure 21-3 on page 494. When a VM86 task attempts execution of the CLI instruction and CR4[VME] = 1, the state of the EFlags[IF] bit is not affected. Rather, the processor sets EFlags[VIF] = 0 (VIF is a virtual copy of the IF bit). Its state has absolutely no effect on the processor’s operation and merely records whether or not the VM86 task prefers not to be interrupted.

Assuming that EFlags[IF] = 1, it remains so and the processor’s ability to recognize an externally generated hardware interrupted remains enabled.

Refer to Figure 21-4 on page 494. If an external hardware should subsequently be detected on the processor’s INTR input (or is delivered to the processor’s core by its Local APIC), it is recognized on the next instruction boundary. As a result, the following actions are taken:

1. The processor ceases executing the interrupted program.
2. The processor core obtains the 8-bit interrupt vector from either the external 8259A interrupt controller or from the Local APIC.
3. It uses the vector to index into the IDT and reads the 8-byte descriptor. Assuming that it’s an Interrupt Gate or a Trap Gate (not a Task Gate), the
Chapter 21: Pentium® Software Enhancements

processor pushes the CS, EIP and EFlags registers onto the stack and jumps to the handler the gate points to. This will be the Protected Mode handler for that level of interrupt.

4. The Protected Mode handler examines the EFlags image saved on the stack and determines that the interrupted program was a VM86 task. As a result, the handler passes control to the VMM in the OS for handling. It also passes its vector number to the VMM.

5. The VMM sees that EFlags[VIF] = 0, indicating that the interrupted VM86 task prefers not to be interrupted. The VMM then evaluates the vector number delivered to it by the Protected Mode handler and makes one of two determinations:
   — If, in the VMM’s estimation, the interrupting device can stand some delay in being serviced, it takes the following actions:
     - The VMM sets a bit in a bit mask in a deferred interrupt table in memory indicating the IRQ number of the handler whose execution is being deferred until the end of the VM86 task’s timeslice.
     - The VMM sets EFlags[VIP] (Virtual Interrupt Pending) bit = 1 to indicate that the execution of one or more handlers have been deferred until the end of the VM86 task’s timeslice.
     - The VMM then returns to the next instruction of the interrupted VM86 task and resumes its execution.
   — If, in the VMM’s estimation, the interrupting device requires rather more timely servicing, it calls the respective handler and instructs it to service the device now. The body of the handler is executed, thereby satisfying the device’s request for servicing. The handler then returns control to the interrupted VM86 task.

When the VM86 task’s timeslice expires, the hardware timer interrupts the processor. If the selected entry in the IDT contains a Task Gate descriptor, the interrupt causes the processor to suspend the VM86 task and switch to the OS’s task scheduler. When the task scheduler determines that a VM86 task has just completed its timeslice, it examines the state of the EFlags[VIP] bit in the EFlags image saved in the now suspended task’s TSS. If VIP = 1, this indicates that the execution of one or more interrupt handlers were deferred until the end of the suspended task’s timeslice. The scheduler then examines the deferred interrupt table in memory, determines the handlers that need to be executed and calls each of them to service their respective devices.
The Previous Chapter
This chapter provided a description of the software enhancements incorporated in the Pentium® processor. This discussion is directly applicable to all subsequent IA32 processors. It included:

- The VM86 Extensions.
- Protected Mode Virtual Interrupts.
- The Debug Extension.
- The Time Stamp Counter.
- 4MB Pages.
- the Machine Check Architecture (MCA).
- Performance Monitoring.
- The Local APIC Register Set.
- The MSRs Added.
- Instruction Set Changes (including MMX).
- New/Altered Exceptions.

This Chapter
This chapter provides the P6 processor roadmap.

The Next Chapter
The next chapter provides a brief introduction to the Pentium® Pro processor’s hardware design characteristics.

The P6 Processor Family
All Intel® IA32 processors in the Pentium® Pro, Pentium® II and Pentium® III product lines (including Celerons and Xeons) are referred to as P6 family processors because they were all based on variants of the P6 processor core. The three core variants were code named (in chronological order) as follows:
The Unabridged Pentium® 4

- The Klamath core.
- The Deschutes core.
- The Katmai core.

The next three sections provide an overview of these three cores and the products that were based on each of them.

The Klamath Core

Figure 22-1 on page 540 illustrates (in chronological order) the Intel® products that were based on the Klamath processor core.

Figure 22-1: P6 Klamath Core Roadmap
The Deschutes Core

Figure 22-2 on page 541 illustrates (in chronological order) the Intel® products that were based on the Deschutes processor core.

Figure 22-2: P6 Deschutes Core Roadmap
The Katmai Core

Figure 22-3 on page 542 illustrates (in chronological order) the Intel® products that were based on the Katmai processor core. Basically, all versions of the Pentium® III processor were based on the Katmai core.

Figure 22-3: P6 Katmai Core Roadmap
23 P6 Hardware Overview

The Previous Chapter
This chapter provided the P6 processor roadmap.

This Chapter
This chapter provides a brief introduction to the Pentium® Pro processor’s hardware design characteristics.

The Next Chapter
This chapter provides a detailed description of the software enhancements incorporated in the Pentium® Pro processor. This discussion is directly applicable to all subsequent IA32 processors. It includes:

- PAE-36 Mode.
- Global Pages.
- APIC Enhancements.
- SMM Enhancement.
- The Memory Type and Range Registers (MTRRs).
- The MCA.
- The Performance Counters.
- The MSR.
- Instruction Set Changes.
- New/Altered Exceptions.
The Unabridged Pentium® 4

For More Detail

A more detailed introduction to the P6 processor core and FSB can be found on the CD included with this book. For a detailed description of the P6 processor, refer to the MindShare book entitled Pentium® Pro and Pentium® II System Architecture, Second Edition.

Introduction

Starting with the advent of the Pentium® Pro processor, IA32 processors no longer execute the complex, multi-byte, IA32 instruction set. Rather, the front end logic within the processor decodes each IA32 instruction into a series of one or more fixed-length, primitive instructions referred to as µops (micro-ops). The resulting µops are the instructions executed by the processor core.

The Pentium® Pro processor also introduced the FSB that is utilized on subsequent IA32 processors up to and including the Pentium® 4 and Pentium® M processor families.

Figure 23-1 on page 545 illustrates the basic elements that comprise a P6 family processor.
Chapter 23: P6 Hardware Overview

The P6 Processor Core

The P6 processor core consists of (the pipeline stages are shown in Figure 23-2 on page 546):

- The front-end logic that, guided by the processor’s Branch Prediction logic, fetches IA32 instructions from memory and stages them in the L1 Code Cache to be supplied to the processor’s instruction pipeline.
- The decode logic that decodes the IA32 instructions that comprise the program into a series of primitive, fixed-length instruction referred to as μops (micro-ops).
The Unabridged Pentium® 4

- The µop pipeline stages that perform the following functions:
  - The µop Queue stage that accepts µops from the decoders.
  - The RAT (Register Alias Table) stage that allocates physical registers to be utilized in lieu of the GPRs.
  - The ROB stage wherein the µops are placed in the ReOrder Buffer until they complete execution and are retired.
  - The Dispatch stage wherein the µops are dispatched for execution.
  - The Execute stage.
  - The Retirement stages.

*Figure 23-2: The P6 µop Pipelines Stages*

The FSB Interface Unit

The Agent Types

There are three types of agents involved in a FSB transaction:

- The Request Agent issues the transaction request.
- The Response Agent is the device that acts as the target of the transaction.
- The Snoop Agents are the entities that contain caches (typically, the processors). If the transaction is a memory transaction, they perform a lookup in their caches using the transaction’s address and report the snoop result (to the Request and Response Agents) in the transaction’s Snoop Phase.

The Request Agent Types

There are two types of Request Agents:

- The Symmetric Request Agents are the processors. They use a symmetric (rotational) bus arbitration scheme.
- The Priority Agents are agents other than processors that perform transactions on the FSB. An example would be the North bridge, MCH, or Root Complex (in other words, the chipset).
The Transaction Phases

Each transaction performed on the FSB consists of the following phases:

- The Request Phase. The transaction request is issued.
- The Error Phase. If any FSB agents detected a parity error, they assert AERR# to the Request Agent and the transaction is aborted. *This phase was eliminated with the advent of the Pentium® 4 processor.*
- The Snoop Phase. The Snoop Agents report the snoop result.
- The Response Phase. The Response Agent indicates how it will treat the transaction (Retry, Deferred, Hard Failure, Supply Data, Accept Data, or Hit on Modified Line).
- The Data Phase.

The Transaction Types

The FSB interface Unit performs FSB transactions when requested to do so by the L2 Cache or the processor core. The transactions types the processor performs on the FSB are:

- IO Read or Write Transaction.
- Memory Read Transaction.
- Memory Write Transaction.
  - Memory Write. This is the regular memory write transaction that is used for most memory writes.
  - Memory Line Writeback. Used to write a modified line back to memory.
- Memory Read and Invalidate Transaction. Used to kill a line in the caches of other processors, or to read a line with the intent to modify it.
- Special Transaction. Used to broadcast a message to the platform.
- Interrupt Acknowledge Transaction. Used to obtain the interrupt vector from the interrupt controller.
- Branch Trace Message Transaction. Used as debug aid.
- Deferred Reply Transaction.

The Backside Bus (BSB) Interface Unit

The BSB Unit interfaces the processor core to the unified L2 Cache.
The Previous Chapter
This chapter provided a brief introduction to the Pentium® Pro processor’s hardware design characteristics.

This Chapter
This chapter provides a detailed description of the software enhancements incorporated in the Pentium® Pro processor. This discussion is directly applicable to all subsequent IA32 processors. It includes:

- PAE-36 Mode.
- Global Pages.
- APIC Enhancements.
- SMM Enhancement.
- The Memory Type and Range Registers (MTRRs).
- The MCA.
- The Performance Counters.
- The MSRs.
- Instruction Set Changes.
- New/Altered Exceptions.

The Next Chapter
This chapter provides a detailed description of the Microcode Update feature (also referred to as the BIO Update feature). This discussion is directly applicable to all subsequent IA32 processors.
Paging Enhancements

PAE-36 Mode

The Problem

Refer to Figure 24-1 on page 555. When any IA32 processor is using the 386-compatible Paging mechanism (described in “386 Demand Mode Paging” on page 209), a 2-level lookup is performed to translate the 32-bit linear address into the 32-bit physical memory address. The linear memory address to be accessed is, by definition, a 32-bit address identifying the target location to be accessed within the currently executing task’s 4GB virtual memory address space. The 2-level lookup selects a PTE and, assuming that the PTE’s Present bit = 1, the PTE’s upper 20 bits supplies the upper 20 bits of the 32-bit physical memory address that will be accessed. The lower 12 bits of the linear address is also used as the lower 12 bits of the physical address.

Since the resulting physical memory address is only 32 bits wide, the 32-bit virtual memory address can only be mapped to a location in the lower 4GB of physical memory address space. There is no way to map the supplied 32-bit virtual memory address to a physical memory location above the 4GB address boundary.

The Pentium® Pro, Pentium® II, Pentium® III, Pentium® 4 and all Xeon processors implement external address pins A[35:3]#, permitting the processor to address a total of 64GB of physical memory (note that the Celeron and Pentium® M processors only implement address pins A[31:3]# and are therefore limited to addressing the lower 4GB of physical memory). When an IA32 processor is using the 386-compatible Paging mechanism, however, it is not capable of asserting address pins A[35:32]#.
Chapter 24: Pentium® Pro Software Enhancements

Figure 24-1: 386-Compatible Paging Address Translation

The Solution: PAE-36 Mode

With the advent of the Pentium® Pro processor, a new feature was introduced that permits the supplied 32-bit virtual memory address to be mapped to a physical memory location that is either below or above the 4GB address boundary anywhere within the 64GB addressable address space. This feature is
Enabling PAE-36 Mode

PAE-36 Mode is enabled by setting CR4[PAE] = 1 (see Figure 24-2 on page 556). Note that the processor must also be operating in Protected Mode—CR0[PE] = 1, with Paging enabled—CR0[PG] = 1.

The Application Is Still Limited to a 4GB Virtual Address Space

The currently executing program is still limited to a 32-bit (i.e., 4GB) virtual address space consisting of a total of 1M (2^20) 4KB pages, but the Paging Unit can now map (i.e., translate) the specified 32-bit linear address to a destination physical page anywhere in a 64GB (rather than 4GB) physical address space. The translation is performed by using a 3-level, rather than a 2-level, directory lookup.
Chapter 24: Pentium® Pro Software Enhancements

The OS Creates the Application’s Address Translation Tables

Just as with the 386-compatible mechanism, the OS builds the paging-related tables in system memory and places the base address of the top level directory in CR3 (see Figure 24-3 on page 557). The top level directory is referred to as the Page Directory Pointer Table (PDPT).

CR3 Is Loaded with the Top Level Address Translation Table Pointer

Whenever a task switch occurs, the processor loads CR3 (see Figure 24-4 on page 558) with the pointer to the top level address translation table associated with the current task. CR3[31:5] specifies the upper 27 bits of the PDPT’s 32-byte aligned physical base address. The processor assumes that the lower five bits of the address are zeros, thereby forcing the base address to be aligned on an address boundary evenly divisible by 32.

The OS uses CR3[PWT] and CR3[PCD] to tell the processor whether or not the PDPT entries can be cached and, if they can, whether to treat the area of memory containing the table as cacheable write-through or cacheable write back memory. See Table 24-1 on page 558.

Figure 24-3: CR3 Contains Pointer to PDPT

All entries in the PDPT, PD and PT are 64 bits wide, permitting PDs, PTs, or pages to be located anywhere in 64GB memory address space.
The Previous Chapter
This chapter provided a detailed description of the software enhancements incorporated in the Pentium® Pro processor. This discussion is directly applicable to all subsequent IA32 processors. It included:

- PAE-36 Mode.
- Global Pages.
- APIC Enhancements.
- SMM Enhancement.
- The Memory Type and Range Registers (MTRRs).
- The MCA.
- The Performance Counters.
- The MSR.
- Instruction Set Changes.
- New/Altered Exceptions.

This Chapter
This chapter provides a detailed description of the Microcode Update feature (also referred to as the BIO Update feature). This discussion is directly applicable to all subsequent IA32 processors.

The Next Chapter
This chapter provides an overview of the Pentium® II processor’s hardware design characteristics. This includes:

- The Pentium® Pro/Pentium® II Differences.
- One Product Yields Three Product Lines.
- The Pentium® II/Xeon/Celeron Roadmap.
- The Cartridge.
The Unabridged Pentium® 4

- The Core.
- The FSB and BSB.
- The Introduction of the Celeron.

The Problem

Today’s processors may be the most complex machines ever designed by man. When dealing with such a level of complexity, it’s impossible to avoid errors. In other words, every complex processor is shipped with bugs—some known, some not.

When the first version of a processor is shipped, that is referred to as the first stepping, or revision, of the silicon. As time goes on and bugs are uncovered, the manufacturer redesigns the silicon to eliminate the problems recognized at the time. This comprises the next stepping of the silicon. During the life of a processor, it typically passes through a number of steppings as improvements/fixes are included in the design.

If a machine is purchased with an earlier stepping of the processor and it is later decided to update to a later stepping (to eliminate problems or, possibly, to improve performance), the user would have to purchase a new processor—an expensive proposition.

The Solution

At the heart of the P6 (and Pentium® 4 and Pentium® M) processors, microcode instructions (referred to as µops) are executed to accomplish the processor’s internal operations. The processor’s microcode is contained in ROM memory that resides within the processor core. In earlier processors, this ROM was truly read-only—the microcode burned into the ROM at the time of manufacture could not be changed.

Using a special procedure, revised microcode can be automatically loaded into a P6, or Pentium® 4, or Pentium® M processor each time that the system is powered up (or even after it has been powered up). The new microcode can eliminate bugs (what Intel® refers to as errata). When a new revision of microcode is loaded into the processor after the machine is powered up, the silicon level, or stepping, of the processor is effectively raised to match a new stepping of the silicon that is currently being shipped from Intel®’s manufacturing plants. This is a very powerful and extremely cost-effective solution for Intel® and system board manufacturers, as well as the end-user.
The Microcode Update Image

Introduction

When a Microcode Update must be applied to processors in the field, Intel® supplies system board manufacturers and/or BIOS vendors with a binary image referred to as a Microcode Update. This image is incorporated in the BIOS and has the format shown in Figure 25-1 on page 633. A Microcode Update image is exactly 2048 bytes in length and has the following basic composition:

- The first 48 bytes comprises the Update Header data structure. The header contains information that identifies the target processor to which the update should be applied, as well as other information (refer to “The Microcode Update Header” on page 634).
- The 2000 byte microcode binary image immediately follows the header data structure. This is the image that is updated into the processor to upgrade it to a new stepping level.

Figure 25-1: The Microcode Image Format
The Microcode Update Header

Figure 25-1 on page 633 illustrates the format of a Microcode Update Image and Table 25-1 on page 634 provides a description of each of the Header fields.

Table 25-1: Format of the Microcode Update Header Data Structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length (in bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header Version</td>
<td>0d</td>
<td>4d</td>
<td>Version number of the update header data structure (i.e., the 48-byte data structure at the start of the image). The current version number is 00000001h and has the format shown in Figure 25-1 on page 633 and in this table. The same format is used for all processors starting with the Pentium® Pro and including the Pentium® 4 and Pentium® M family processors. Additional header data structure formats may be defined in the future (one more new field may be defined within an area currently defined as Reserved).</td>
</tr>
<tr>
<td>Update Revision</td>
<td>4d</td>
<td>4d</td>
<td>This represents the revision of the Microcode Update contained within the 2000d byte image that immediately follows this header. After the update has been loaded into the processor, this field can be compared to the signature returned by the CPUID instruction to verify a good load. For more information, refer to “Matching the Image to a Processor” on page 636 and to “Determining if a New Update Supersedes a Previously-Loaded Update” on page 653.</td>
</tr>
<tr>
<td>Date</td>
<td>8d</td>
<td>4d</td>
<td>Date of creation of this update, in hex format. As an example, a creation date of 07/30/95 is represented as 07301995h.</td>
</tr>
</tbody>
</table>
### Chapter 25: MicroCode Update Feature

#### Table 25-1: Format of the Microcode Update Header Data Structure (Continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length (in bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>12d</td>
<td>4d</td>
<td>Family, model and stepping of the processor that requires this update. The format of this field is identical to that returned by the CPUID instruction (see Figure 25-2 on page 635).</td>
</tr>
<tr>
<td>Checksum</td>
<td>16d</td>
<td>4d</td>
<td>Checksum of the entire 2048d bytes consisting of the header and the Microcode Update image. The checksum is correct if the sum of the 512 dwords of the image is zero.</td>
</tr>
<tr>
<td>Loader Revision</td>
<td>20d</td>
<td>4d</td>
<td>The version number of the loader program required to load this update. The initial version is 00000001h and is the loader version used for all processors in the P6, Pentium® 4, and Pentium® M processor families.</td>
</tr>
<tr>
<td>Reserved</td>
<td>24d</td>
<td>24d</td>
<td>Reserved for future field definition.</td>
</tr>
</tbody>
</table>

#### Figure 25-2: EAX After a CPUID Request Type 1

![Figure 25-2: EAX After a CPUID Request Type 1](image-url)
The Previous Chapter
This chapter provided a detailed description of the Microcode Update feature (also referred to as the BIO Update feature). This discussion is directly applicable to all subsequent IA32 processor.

This Chapter
This chapter provides an overview of the Pentium® II processor’s hardware design characteristics. This includes:

- The Pentium® Pro/Pentium® II Differences.
- One Product Yields Three Product Lines.
- The Pentium® II/Xeon/Celeron Roadmap.
- The Cartridge.
- The Core.
- The FSB and BSB.
- The Introduction of the Celeron.

The Next Chapter
This chapter provides a detailed description of the power management modes found in all IA32 processors starting with the Pentium® II processor. Note that the Pentium® M processor added one additional mode, Deeper Sleep, and a description can be found in “Enhanced Power Management Characteristics” on page 1429.
The Unabridged Pentium® 4

The Pentium® Pro and Pentium® II: Same CPU, Different Package

To a goodly degree, the Pentium® II represented a repackaging of the Pentium® Pro. The package changed from the PGA to the cartridge. Throw in MMX capability, four new instructions, change the size of the caches and, ouila, you have a Pentium® II.

Dual-Independent Bus Architecture (DIBA)

Like all IA32 processors starting with the Pentium® Pro, the Pentium® II had a dedicated BSB used to transfer data between the processor core and the L2 Cache and a FSB used to communicate with other system devices. The processor can use both of these buses simultaneously, thereby yielding better overall performance. With the advent of the Pentium® II processor, Intel® started referring to this as DIBA.

IOQ Depth

Like the Pentium® Pro, the Pentium® II processor implemented an In Order Queue with a depth of eight. See “Transaction Tracking” on page 1147 for more information on the IOQ.

Pentium® Pro/Pentium® II Differences

The following is a list of differences between the Pentium® Pro the Pentium® processors:

- Intel® switched from using PGA (Pin Grid Array) packaging to the new cartridge package.
- The Pentium® Pro topped out at a 200MHz core speed. The Pentium® II was introduced at a core speed of 233MHz and topped out at 450MHz.
- While the Pentium® Pro did not include the MMX register set or instruction set (with one exception; see “Pentium® II Overdrive Processor” on page 159 on the CD), the Pentium® II added them back in and all subsequent IA32 processors include the MMX capability (see “MMX Capability” on page 519).
Chapter 26: Pentium® II Hardware Overview

- The Pentium® II included power conservation modes that were not implemented on the Pentium® Pro. This topic is covered in “Pentium® II Power Management Features” on page 683.
- The Pentium® Pro yielded less than stellar performance when executing legacy 16-bit code (and Windows 95 included a LOT of legacy code). The Pentium® II was optimized to improve the execution speed of 16-bit code.
- Two new instructions were added to the instruction set. They were the Fast System Call/Return instruction pair (described in “Fast System Call/Return Instruction Pair” on page 708).
- The processor’s Backside Bus (BSB) speed was reduced to 50% of the processor core speed.
- The earlier models of the Pentium® II had a FSB speed of 66MHz (the same as the later models of the Pentium® Pro), while the later models increased the FSB speed to 100MHz.
- The Pentium® Pro’s FSB arbitration scheme supported up to four processors on the FSB. The Pentium® II supported one or two processors.
- The Pentium® Pro’s L1 Code and Data Caches were each 8KB in size. The Pentium® II’s L1 cache sizes were increased to 16KB each to make up for slow BSB speed.
- All versions of the Pentium® II had an L2 Cache size of 512KB. There were three variants (in all cases, the BSB ran at 50% of the processor’s core speed):
  - The L2 cache only cached from the first 512MB of memory address space and the BSB was not ECC protected.
  - The L2 cache only cached from the first 512MB of memory address space and the BSB was ECC protected.
  - The L2 cache cached from the first 4GB of memory address space and the BSB was ECC protected.
- Later models of the Pentium® II had a hardwired core/FSB frequency ratio, while the earlier models were auto-configured via A20M#, IGNE#, LINT1, and LINT2 sampling on the deassertion of the reset signal (see chapter 3 of the MindShare book entitled Pentium® Pro and Pentium® II System Architecture, Second Edition).

While the hardware-related differences are described in the remainder of this chapter, the power management modes are described in “Pentium® II Power Management Features” on page 683, and the software differences in “Pentium® II Software Enhancements” on page 695.
One Product Yields Three Product Lines

Soon after the advent of the Pentium® II product, Intel® divided the P6 into three product lines, two of which are just variations on the Pentium® II processor.

- The Pentium® II processor targeted the mid- to high-end desktop market and supported either one or two processors on the FSB.
- The Xeon processor. The Xeon targeted the workstation and server market and supported either two (in a Xeon DP product), or four (in a Xeon MP product) processors on the FSB. With the following exceptions, the Xeon was identical to the Pentium® II:
  - While all models of the Pentium® II had a 512KB L2 Cache, the Pentium® II Xeon was available with a 512KB, 1MB, or 2MB L2 Cache.
  - While the Pentium® II’s BSB operated at 50% of the processor’s core speed, the Pentium® II Xeon’s BSB operated at 100% of the core speed.
  - The Xeon implemented a Processor Information ROM, a scratch EEPROM and a thermal diode that could all be read from over the serial SMBus.
  - The Pentium® II implemented two pins (BR[1:0]#) for FSB arbitration, permitting two processors on the FSB. The Xeon MP implemented four pins (BR[3:0]#), permitting four processors on the FSB.
- The Celeron processor. The Pentium® II Celeron targeted the low-end desktop market and only supported one processor on the FSB. With the following exceptions, the Celeron was identical to the Pentium® II:
  - The Pentium® II implemented two pins (BR[1:0]#) for FSB arbitration, permitting two processors on the FSB. The Celeron implemented one pin (BR0#), permitting one processor on the FSB.
  - The FSB speed (i.e., the BCLK frequency) of the Celeron was slower than that of the Pentium® II.

The Pentium® II/Xeon/Celeron Roadmap

The following is a list of the major milestones in the Pentium® II’s evolution:

- The original Pentium® II was based on the .28µm Klamath core and had a FSB speed of 66MHz.
- Deschutes was based on the 0.25 µm process and had a 100MHz FSB (as did all subsequent Pentium® II processors).
- Tonga was the mobile version of the Pentium® II and was based on the 0.25µm process.
Chapter 26: Pentium® II Hardware Overview

- **Covington** was the first Celeron processor. Intel® found themselves in the position of losing market share to AMD because it did not have a small form factor, low-cost processor that would fit well into small footprint machines. While engineering worked on the development of such a processor, Covington was introduced as the short-term solution. Basically, the L2 Cache was removed from the Pentium® II processor allowing the cost and the cartridge size to be reduced. Unfortunately, the removal of the L2 Cache resulted in a processor with reduced performance.

- **Mendocino** was the first real Celeron processor. It was socketed rather than a cartridge, had a 128KB L2 Cache integrated on the processor die, a full-speed BSB between the processor core and the L2 Cache, and the FSB arbitration scheme only supported one processor (rather than two as was the case with the Pentium® II).

- The **first Xeon** processor represented a repackaging of the Pentium® II processor. For more information, refer to “Pentium® II Xeon Features” on page 719.

- **Dixon** was the final Celeron based on the Pentium® II technology. It was socketed rather than a cartridge, had a 256KB L2 Cache integrated on the processor die, a full-speed BSB between the processor core and the L2 Cache, and the FSB arbitration scheme only supported one processor (rather than two as was the case with the Pentium® II).

---

**The Cartridge**

**The Pentium® and Pentium® Pro Sockets**

The Pentium® and Pentium® Pro processors utilized PGA (i.e., Pin Grid Array) packages and were installed into a PGA socket on the system board. The Pentium® sockets were referred to as sockets 1 through 7, with socket 7 being the de facto standard on most Pentium® system boards. The Pentium® Pro processor implemented the socket 8 PGA. System board designers could license Socket 8 from Intel®, but processor designers could not.

---

**The Problem**

It stands to reason that Intel® must have had a good reason for switching from the compact PGA package to the rather large cartridge. At that time, the silicon wafer and die layout processes in use were rather coarse when compared to today’s technologies. This made it impossible to include a large number of transistors on the processor die. That was Intel®’s original impetus for using the
The Previous Chapter
This chapter provided an overview of the Pentium® II processor’s hardware design characteristics. This included:

- The Pentium® Pro/Pentium® II Differences.
- One Product Yields Three Product Lines.
- The Pentium® II/Xeon/Celeron Roadmap.
- The Cartridge.
- The Core.
- The FSB and BSB.
- The Introduction of the Celeron.

This Chapter
This chapter provides a detailed description of the power management modes found in all IA32 processors starting with the Pentium® II processor. Note that the Pentium® M processor added one additional mode, Deeper Sleep, and a description can be found in “Enhanced Power Management Characteristics” on page 1429.

The Next Chapter
This chapter provides a detailed description of the software enhancements first implemented in the Pentium® II processor. This discussion is directly applicable to all subsequent IA32 processors. This includes:
The Unabridged Pentium® 4

- The Pentium® II and Pentium® III MSRs.
- The Fast System Call/Return Instruction Pair.
- The FP/SSE Save/Restore Instruction Pair.
- New/Altered Exceptions.

The Pentium® Pro’s Power Conservation Modes

The Pentium® Pro processor only implemented the STPCLK# pin and the following power conservation states:

- The Normal state.
- The AutoHalt Power Down state.
- The Stop Grant state.
- The Halt/Grant Snoop state.

The issue of power management was not covered in the Pentium® Pro section of the book, but the Pentium® Pro’s power management states are covered in this chapter.

The Pentium® II’s Power Conservation Modes

In addition to the four states implemented in the Pentium® Pro, the Pentium® II processor (as well as the Pentium® III and Pentium® 4) implemented two more states and another pin related to power conservation (the SLP# input pin—Sleep). The additional power conservation states are:

- The Sleep state.
- The Deep Sleep state.

The sections that follow provide a detailed description of the six power conservation states. Refer to Figure 27-1 on page 685 during the discussion of the power conservation states.
Chapter 27: Pentium® II Power Management Features

Figure 27-1: Power Conservation States Flowchart

- AutoHalt Power Down State
  - BCLK running
  - Snoops/Interrupts recognized
  - Halt instruction executed
  - Special transaction generated
  - INIT#, BINIT#, INTR, NMI, SMI#, RESET#

- Normal State (normal execution)
  - STOPCLK# asserted

- Halt Grant Snoop State
  - BCLK running
  - Service a Snoop
  - Snoop event occurs
  - Snoop complete

- Stop Grant State
  - BCLK running
  - Snoops/Interrupts recognized
  - Snoop event occurs
  - Snoop complete

- Sleep State
  - BCLK running
  - Snoops/Interrupts not allowed

- Deep Sleep State
  - BCLK stopped
  - PLL stopped
  - Snoops/Interrupts not allowed
The Normal State

This is the processor’s normal operating state. No power conservation strategies are in effect and the processor is operating at full speed.

The AutoHalt Power Down State

Description

Refer to Figure 27-2 on page 688. When a HLT (Halt) instruction is executed, the processor generates a Special transaction (see “The Special Transaction” on page 1306) on the FSB to broadcast a Halt message to the system. It then leaves the Normal state and enters the AutoHalt Power Down state. This state has the following characteristics:

- The processor powers down all logic except the logic that is necessary for the recognition of interrupts and the snooping of memory accesses generated by other FSB agents.
- The BCLK signal on the FSB continues to run.
- The processor services any snoop events (i.e., memory transactions generated by other agents on the FSB) and then returns to the AutoHalt Power Down state. To do this, the processor temporarily transitions from the AutoHalt Power Down state to the Halt/Grant Snoop state. While in this state, it presents the memory address received from the other agent to the three caches within the processor for a lookup. It then presents the snoop result to the other agent as well as to the system memory controller on the HIT# and HITM# signals. After the snoop is complete, the processor returns to the AutoHalt Power Down state.
- Upon the occurrence of an interrupt event (RESET#, SMI#, BINIT#, INIT#, or LINT[1:0]—NMI or INTR), the processor exits the AutoHalt Power Down state and returns to the Normal state to service the interrupt.
- Upon return from the SMI interrupt handler, the processor either enters the Normal state (if the instruction returned to is an instruction other than a HLT) or the AutoHalt Power Down state (if the instruction returned to is a HLT instruction).
- If the processor’s FLUSH# input is asserted while the processor is in the AutoHalt Power Down state, the flush is serviced (i.e., all modified lines are written back to system memory and the processor caches are then invalidated). Upon completion of the writeback operation, the processor re-enters the AutoHalt Power Down state.
Chapter 27: Pentium® II Power Management Features

- The system board logic (specifically, the chipset) may assert the STPCLK# signal to the processor while the processor is in the AutoHalt Power Down state. This causes the processor to leave the AutoHalt Power Down state and enter the Stop Grant state. The processor remains in the Stop Grant state until STPCLK# is removed and then re-enters the AutoHalt Power Down state.

The Chipset’s Response to the Halt Message

When the chipset receives the Halt message from the processor (in the Special transaction), the action(s) taken by the processor are design-specific. As an example, the chipset might be designed to power down some of the system board logic during the period of time that the processor remains inactive. It could then reapply power to that logic when the processor arbitrates for ownership of the FSB to initiate a transaction on the FSB.
The Previous Chapter
This chapter provided a detailed description of the power management modes found in all IA32 processors starting with the Pentium® II processor. Note that the Pentium® M processor added one additional mode, Deeper Sleep, and a description can be found in “Enhanced Power Management Characteristics” on page 1429.

This Chapter
This chapter provides a detailed description of the software enhancements first implemented in the Pentium® II processor. This discussion is directly applicable to all subsequent IA32 processors. This includes:

- The Pentium® II and Pentium® III MSRs.
- The Fast System Call/Return Instruction Pair.
- The FP/SSE Save/Restore Instruction Pair.
- New/Altered Exceptions.

The Next Chapter
This chapter describes the first Xeon processor. It was based on the Pentium® II processor. This includes:

- The Cartridge.
- FSB Protocol Alteration (GTL+ to AGTL+).
- FSB Arbitration.
- SMBus (System Management Bus). This is an introduction to the SMBus. A detailed description of the SMBus is outside the scope of this book.
- PSE-36 Mode. This discussion is directly applicable to all subsequent IA32 processors.
The Unabridged Pentium® II and Pentium® III MSRs

Table 28-1 on page 696 defines the MSRs implemented in the Pentium® II and the Pentium® III processors. The newly added registers are the ones in the BBL (Backside Bus Logic) and the Fast System Enter/Exit register groups (all of the others were present in the Pentium® Pro).

### Table 28-1: Pentium® II and III MSRs

<table>
<thead>
<tr>
<th>Reg Address (specified in ECX before executing RDMSR or WRMSR)</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>000h</td>
<td>0</td>
<td>P5_MC_ADDR</td>
</tr>
<tr>
<td>001h</td>
<td>1</td>
<td>P5_MC_TYPE</td>
</tr>
<tr>
<td>010h</td>
<td>16</td>
<td>TSC</td>
</tr>
<tr>
<td>017h</td>
<td>23</td>
<td>IA32_PLATFORM_ID</td>
</tr>
<tr>
<td>01Bh</td>
<td>27</td>
<td>APIC_BASE</td>
</tr>
</tbody>
</table>

The Pentium® II and Pentium® III MSRs
## Chapter 28: Pentium® II Software Enhancements

Table 28-1: Pentium® II and III MSRs (Continued)

<table>
<thead>
<tr>
<th>Reg Address (specified in ECX before executing RDMSR or WRMSR)</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>033h 51</td>
<td>TEST_CTL</td>
<td>The TEST_CTL register implements two miscellaneous control bits. See “Test Control Register (TEST_CTL)” on page 620 for a detailed description of this register.</td>
</tr>
<tr>
<td>1D9h 473</td>
<td>DEBUGCTLMSR</td>
<td>The DEBUGCTL register implements bits that control Branch Trace Messaging, Branch Recording, and the usage of the processor’s PB output pins. See “DebugCtl MSR” on page 621 for a detailed description of this register.</td>
</tr>
<tr>
<td>1E0h 480</td>
<td>ROB_CR_BKUPT MPDR6</td>
<td>ROB_CR_BKUPTMPDR6 register implements the Fast String Enable bit. When copying large blocks of data from one area of memory to another on the P6 processors, setting this bit can improve the speed of the copy. See “ROB_CR_BKUPTMPDR6 MSR” on page 621 for a detailed description of this register.</td>
</tr>
</tbody>
</table>
### The Unabridged Pentium® 4

#### Table 28-1: Pentium® II and III MSRs (Continued)

<table>
<thead>
<tr>
<th>Reg Address (specified in ECX before executing RDMSR or WRMSR)</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td><strong>Microcode Update MSRs (See “MicroCode Update Feature” on page 631 for a detailed description of the Microcode Update feature).</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 079h | 121 | BIOS_UPDT_TRIG | • BIOS Update Trigger register.  
• BIOS Signature register. The Microcode Update-related MSRs. Please note that, depending on how it is used, register 139d serves double-duty as the BIOS_SIGN register or as the BBL_CR_D3[63:0] register (see the BBL—Backside Bus Logic—section of this table for more information). |
| 08Bh | 139 | BIOS_SIGN or BBL_CR_D3[63:0] | |
| **Performance Monitoring MSRs (see “The Performance Counters” on page 606 for a detailed description).** |
| 0C1h | 193 | PERFCTR0 | The Performance Monitoring registers are implemented identically in the P6 processors (but differently than they were in the Pentium®). See “The Performance Counters” on page 606 for a detailed description of the Pentium® Pro’s Performance Monitoring facility. |
| 0C2h | 194 | PERFCTR1 | |
| 186h | 390 | EVNTSEL0 | |
| 187h | 391 | EVNTSEL1 | |
## Memory Type and Range Registers (MTRRs)

The MTRRs are optional in the architecture, but are implemented in the P6 processors. See “MTRRs Added” on page 572 for a detailed description of the MTRRs.

<table>
<thead>
<tr>
<th>Reg Address (specified in ECX before executing RDMSR or WRMSR)</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0FEh 254 MTRRCap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200h 512 MTRRphysBase0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>201h 513 MTRRphysMask0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>202h 514 MTRRphysBase1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>203h 515 MTRRphysMask1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>204h 516 MTRRphysBase2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>205h 517 MTRRphysMask2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>206h 518 MTRRphysBase3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>207h 519 MTRRphysMask3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>208h 520 MTRRphysBase4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>209h 521 MTRRphysMask4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20Ah 522 MTRRphysBase5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20Bh 523 MTRRphysMask5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The MTRR registers permit the BIOS and/or the OS to define the rules of conduct that the processor must use when accessing various areas of memory.
Pentium® II Xeon

Features

The Previous Chapter
This chapter provided a detailed description of the software enhancements first implemented in the Pentium® II processor. This discussion is directly applicable to all subsequent IA32 processors. This included:

- The Pentium® II and Pentium® III MSRs.
- The Fast System Call/Return Instruction Pair.
- The FP/SSE Save/Restore Instruction Pair.
- New/Altered Exceptions.

This Chapter
This chapter describes the first Xeon processor. It was based on the Pentium® II processor. This includes:

- The Cartridge.
- FSB Protocol Alteration (GTL+ to AGTL+).
- FSB Arbitration.
- SMBus (System Management Bus). This is an introduction to the SMBus. A detailed description of the SMBus is outside the scope of this book.
- PSE-36 Mode. This discussion is directly applicable to all subsequent IA32 processors.

The Next Chapter
This chapter provides a description of the Pentium® III processor’s hardware design characteristics. This includes:

- One product = three product lines.
- Pentium® II/Pentium® III differences.
- The Pentium® III/Xeon/Celeron roadmap.
The Unabridged Pentium® 4

- The L1 Caches.
- The L2 Cache.
- The Data Prefetcher.
- SSE introduced.
- The WCBs were enhanced.
- Additional Writeback Buffers.
- SpeedStep Technology.

Introduction

The Xeon based on the Pentium® II processor was the first Xeon. Xeon processors target the workstation and server market and support either two (in a Xeon DP product), or four (in a Xeon MP product) processors on the FSB. With the following exceptions, the Pentium® II Xeon was identical to the Pentium® II:

- While all models of the Pentium® II had a 512KB L2 Cache, the Pentium® II Xeon was available with a 512KB, 1MB, or 2MB L2 Cache.
- While the Pentium® II’s BSB operated at 50% of the processor’s core speed, the Xeon’s BSB operates at 100% of the core speed.
- The Xeon implemented a Processor Information ROM, a scratch EEPROM and a thermal diode that could all be read via the serial SM Bus.

The Pentium® II implemented two pins (BR[1:0]#) for FSB arbitration, permitting two processors on the FSB. The Xeon MP implemented four pins (BR[3:0]#), permitting four processors on the FSB.

To Avoid Confusion...

Over the years since the introduction of the Xeon processor, Intel® documentation always refers to any particular version of the Xeon as “the Xeon” processor. This may seem like a no brainer, but it’s confusing. As an example, an Intel® document may state “this feature was introduced in the Xeon processor”. This does not clearly define which version of the Xeon processor first introduced the feature. It could have been the Pentium® II, Pentium® III, or Pentium® 4 version of the Xeon processor.

When necessary to avoid confusion, this book will state which version (Pentium® II, Pentium® III, or Pentium® 4 version) of the Xeon is being referred to.
Chapter 29: Pentium® II Xeon Features

Basic Characteristics

The following is a list of the Pentium® II Xeon’s basic characteristics (in no particular order):

- The Xeon was first introduced as a variant on the Pentium® II processor technology.
- It used the cartridge packaging, but it was a larger cartridge and used a different connector, referred to as the Slot 2 connector. This connector had more power and ground pins (because the larger, faster L2 Cache drew more power than the Pentium® II’s L2 Cache).
- The Pentium® II Xeon’s core speeds ranged from 400MHz to 450MHz.
- The electrical characteristics of the FSB signals were modified to accommodate a higher FSB speed. The GTL+ FSB electrical specification evolved into the AGTL+ specification.
- The Xeon used the same core as the Pentium® II.
- The FSB speed ran at a frequency of 100MHz (as did the later versions of the Pentium® II).
- The BSB ran at 100% of the processor’s full core speed.
- The L2 Cache was available in three sizes: 512KB, 1MB and 2MB.
- The L2 was ECC protected.
- The L2 Cache could cache from the full 64GB of memory address space (versus 512MB or 4GB for the Pentium® II).
- The PSE-36 feature was added as an alternative to the PAE-36 feature.
- The SMBus was added and the processor incorporated a Processor Information ROM, a scratch EEPROM and a thermal diode that could all be read via the serial SMBus.
- The MP version implemented four Bus Request pins (BR[3:0]) permitting the FSB arbitration scheme to support up to four processors (versus four for the Pro, two for the Pentium® II, and one for the Celeron).
- Due to the inability of processors to recognize FSB transactions during the Sleep power conservation state, multiprocessor Intel® Xeon systems are not allowed to simultaneously have one processor in the Sleep power conservation state while the other processors are in the Normal or the Stop Grant power conservation state (see “Pentium® II Power Management Features” on page 683).
- Xeons do not implement the Deep Sleep power conservation state.
Hardware Characteristics

The Cartridge

Figure 29-1 on page 722 illustrates the Pentium® Xeon cartridge. It represented a different form factor than the Pentium® II Slot 1 cartridge. It was physically larger and had different pinouts—more power and ground pins (because the larger, faster L2 Cache drew more power than the Pentium® II’s L2 Cache).

Figure 29-1: The Pentium® II Xeon Cartridge
FSB Protocol Alteration (GTL+ to AGTL+)

Due to the faster FSB (100MHz versus 66MHz), Intel® changed the GTL+ specification and referred to the modified specification as the AGTL+ specification (Assisted GTL+).

- On a GTL+ FSB, a signal that had been driven low (i.e., it had been asserted) was returned to the electrically high state (i.e., the deasserted state) passively: the agent driving a signal low would cease driving it low and the termination resistors on either end of the signal trace would return the signal to the electrically high state. This would take some time and the signal would ring for a while before settling down to a stable electrical high.
- On the AGTL+ FSB, a signal that had been driven low (i.e., it was asserted) was returned to the electrically high state (i.e., the deasserted state) actively: the agent must actively drive the signal high for one BCLK cycle to return the signal to the deasserted state quickly (and settling it in the electrically high state quickly).

FSB Arbitration

The Pentium® II implemented two pins (BR[1:0]#) for FSB arbitration, permitting two processors on the FSB. The Xeon MP implemented four pins (BR[3:0]#), permitting four processors on the FSB. It uses the same bus arbitration algorithm as the Pentium® Pro (see “Pentium® 4 CPU Arbitration” on page 1149).

SMBus (System Management Bus)

Note

A detailed discussion of the SMBus is outside the scope of this book.

General

Starting with the Pentium® II Xeon, all Xeon processors implement the SMBus. This is a serial bus derived from the I²C bus. A Xeon-based system may incorporate an SMBus controller in the chipset. Using this controller to send requests to the processor over the SMBus, system management software has access to the following entities within the processor (note that they cannot be accessed in any other way):
30 Pentium® III

Hardware Overview

The Previous Chapter
This chapter described the first Xeon processor. It was based on the Pentium® II processor. This included:

- The Cartridge.
- FSB Protocol Alteration (GTL+ to AGTL+).
- FSB Arbitration.
- SMBus (System Management Bus). This is an introduction to the SMBus. A detailed description of the SMBus is outside the scope of this book.
- PSE-36 Mode. This discussion is directly applicable to all subsequent IA32 processors.

This Chapter
This chapter provides a description of the Pentium® III processor’s hardware design characteristics. This includes:

- One product = three product lines.
- Pentium® II/Pentium® III differences.
- The Pentium® III/Xeon/Celeron roadmap.
- The L1 Caches.
- The L2 Cache.
- The Data Prefetcher.
- SSE introduced.
- The WCBs were enhanced.
- Additional Writeback Buffers.
- SpeedStep Technology.
The Unabridged Pentium® 4

The Next Chapter

This chapter provides a detailed description of the software enhancements introduced in the Pentium® III processor. This discussion is directly applicable to all subsequent IA32 processors. It includes:

- The Streaming SIMD Extensions (SSE).
- The SIMD FP exception.
- The Serial Number feature. This feature was discontinued with the advent of the Pentium® 4 processor.
- CPUID Enhanced.
- Brand Index feature.

One Product = Three Product Lines

Like the Pentium® II before it, three separate product lines were derived from the Pentium® III processor:

- The Pentium® III processor targeted the mid- to high-end desktop market and supported either one or two processors on the FSB.
- The Xeon processor (see “Pentium® III Xeon Features” on page 795 for more information on the Pentium® III-based Xeon). The Xeon targeted the workstation and server market and supported either two (a Xeon DP), or four (a Xeon MP) processors on the FSB. With the following exceptions, the Xeon was identical to the Pentium® III:
  - While various models of the Pentium® III had L2 Cache sizes of 128KB or 512KB, the Pentium® III Xeon was available with a 256KB, 512KB, 1MB, or 2MB L2 Cache.
  - While the Pentium® III’s BSB operated at 50% of the processor’s core speed, the Pentium® III Xeon’s BSB operated at 100% of the core speed.
  - The Xeon implemented a Processor Information ROM, a scratch EEPROM and a thermal diode that could all be read from over the serial SMBus.
  - The Pentium® III implemented two pins (BR[1:0]#) for FSB arbitration, permitting two processors on the FSB. The Xeon MP implemented four pins (BR[3:0]#), permitting four processors on the FSB.
- The Celeron processor. The Pentium® III Celeron targeted the low-end desktop market and only supported one processor on the FSB. With the following exceptions, the Celeron was identical to the Pentium® III:
  - The Pentium® III implemented two pins (BR[1:0]#) for FSB arbitration, permitting two processors on the FSB. The Celeron implemented one pin (BR0#), permitting one processor on the FSB.
Chapter 30: Pentium® III Hardware Overview

— The FSB speed (i.e., the BCLK frequency) of the Celeron was slower than that of the Pentium® III.

Pentium® II/Pentium® III Differences

The following is a list of differences between the Pentium® III the Pentium® II processors:

- The Pentium® III added the Streaming SIMD Extension (SSE) instruction set (initially referred to as KNI: Katmai New Instructions) consisting of 70 new instructions. See “The Streaming SIMD Extensions (SSE)” on page 758 for a description of SSE.
- In support of SSE, the Pentium® III added eight, 128-bit registers (XMM[7:0]) and the 32-bit MXCSR (MX Command Status Register). Additional execution units were also added (see “The SSE Execution Units” on page 750). See “The Streaming SIMD Extensions (SSE)” on page 758 for a description of the SSE capability.
- While the Pentium® II added the FPU/SSE Save/Restore instruction pair (FXRSTOR and FXSAVE), that processor did not implement SSE capability and the instructions only saved or restored the FPU’s register set. When executed on any IA32 processor starting with the Pentium® III, however, these two instructions save and restore both the FPU and SSE register sets. A detailed description can be found in “FP/SSE Save/Restore Instruction Pair” on page 712.
- The Pentium® III implemented the poorly-received processor serial number feature which was removed from subsequent processors (see “CPUID Enhanced” on page 793 for more information).
- In its various incarnations, the Pentium® III was available with core speeds ranging from 450MHz to 1.33GHz.
- BSB speed: originally, the BSB speed was 50% of the core speed on the cartridge-based versions of the Pentium® III, and 100% of the core speed on the Xeon models. With the introduction of the .18µm Coppermine version, the L2 Cache was incorporated on the processor die and the BSB speed was raised to 100% of the core speed on all subsequent models.
- FSB speeds: 66MHz, 100Mhz, and 133MHz.
- FSB protocol: identical to that used on the Pentium® Pro and Pentium® II.
- The number of processors supported on the FSB:
  — Celeron: 1.
  — Desktop: 2.
  — Xeon: 4.
- The Pentium® III added the Page Attribute Table (PAT) feature to permit more memory type designations on a page basis. See “PAT Feature (Page Attribute Table)” on page 797 for more information.
The Unabridged Pentium® 4

- While the Pentium® Pro and Pentium® II each had only one writeback buffer, the Pentium® III implemented four writeback buffers to hold modified lines that need to be cast back to memory. See “Additional Writeback Buffers” on page 755 for more information.
- The Pentium® Pro and Pentium® II processors had four cache line fill buffers to handle outstanding FSB line reads caused by cache misses. One of the buffers could be used as a WCB. The Pentium® III design makes dual use of the four cache line fill buffers, using them, when necessary, as WCBs. See “The WCBs Were Enhanced” on page 754 for more information.
- When a WCB becomes full, it is automatically written to external memory (unlike the Pentium® Pro and Pentium® II which did not dump the one and only WCB until a synchronizing event occurred). See “The WCBs Were Enhanced” on page 754 for more information. See “Forcing a Buffer Drain” on page 1083 for an explanation of synchronizing events.
- Hardware-based Data Prefetching was first implemented in the following .13µm models: Pentium® III, mobile Pentium® III, and mobile Celeron. See “The Data Prefetcher” on page 747 for more information.
- The ATC (Advanced Transfer Cache) L2 Cache was introduced in the later versions of the processor. See “The Advanced Transfer Cache” on page 746 for more information.
- The earlier models used the cartridge packaging, while the later versions reverted to the socket format.
- SpeedStep technology was first introduced on the mobile version of the Pentium® III processor. See “SpeedStep Technology” on page 755 for more information.

The Pentium® III/Xeon/Celeron Roadmap

The following is a list of the major milestones in the Pentium® II’s evolution:

- The .25µm Pentium® III was based on the Katmai core.
- The .25µm Xeon was code named Tanner.
- The .18µm Celeron and Pentium® III were code named Coppermine.
- The .18µm Xeon was code named Cascades.
- Geyserville was the code name for the SpeedStep Technology which was introduced in the mobile Pentium® III.
- The .13µm Celeron and Pentium® III were code named Tualatin.

IOQ Depth

Like the Pentium® II and the Pentium® Pro, the Pentium® III processor implemented an In Order Queue with a depth of eight. See “Transaction Tracking” on page 1147 for more information on the IOQ.
The L1 Caches

L1 Code Cache Characteristics
The characteristics of the Pentium® III’s L1 Code Cache are:

- It is 16KB in size.
- It is implemented as a 4-way set-associative cache.
- The cache line size is 32 bytes.
- It implements an SI subset of the MESI coherency protocol.

L1 Data Cache Characteristics
The characteristics of the Pentium® III’s L1 Data Cache are:

- It is 16KB in size.
- It is implemented as a 4-way set-associative cache.
- Each cache bank is subdivided into two subbanks.
- The cache line size is 32 bytes.
- It implemented the full MESI coherency protocol.
- It was a non-blocking cache.

The L2 Cache
Refer to Figure 30-1 on page 747.

The L2 Cache on the Early Pentium® III
The earlier versions of the Pentium® III processor were implemented using the cartridge format and the L2 Cache was implemented using discrete SRAM chips (just as in the Pentium® II). The BSB ran at 50% of the processor core speed and the data bus portion of the BSB was 64-bits wide, permitting a single qword to be transferred per clock cycle from the L2 Cache to the L1 Cache. The L2 Cache had the following additional characteristics:

- It was a unified code/data cache.
- It was available in two versions, one of which could cache from the first 512MB of memory space, and the other could cache from the first 4GB of memory space.
The Previous Chapter
This chapter provided a description of the Pentium® III processor’s hardware design characteristics. This included:

- One product = three product lines.
- Pentium® II/Pentium® III differences.
- The Pentium® III/Xeon/Celeron roadmap.
- The L1 Caches.
- The L2 Cache.
- The Data Prefetcher.
- SSE introduced.
- The WCBs were enhanced.
- Additional Writeback Buffers.
- SpeedStep Technology.

This Chapter
This chapter provides a detailed description of the software enhancements introduced in the Pentium® III processor. This discussion is directly applicable to all subsequent IA32 processors. It includes:

- The Streaming SIMD Extensions (SSE).
- The SIMD FP exception.
- The Serial Number feature. This feature was discontinued with the advent of the Pentium® 4 processor.
- CPUID Enhanced.
- Brand Index feature.
The Unabridged Pentium® 4

The Next Chapter

This chapter provides a description of the Xeon processor based on the Pentium® III technology. It includes:

- The processor’s basic characteristics.
- The Page Attribute Table (PAT) feature.

The Streaming SIMD Extensions (SSE)

Why?

The single most important impetus behind the Streaming SIMD Extensions (SSE) was to achieve a significant performance boost when executing multimedia applications. To this end, Intel® needed to:

- Extend the SIMD model to include SIMD FP capability (as MMX made SIMD integer operations possible).
- Provide new instructions specifically tailored to boost the performance of multimedia applications.
- Enhance memory write operations and to make more efficient use of the FSB.

It should be noted that applications other than multimedia applications can also realize significant benefit from the new SSE feature set.

Detecting SSE Support

Refer to Figure 31-1 on page 759. The programmer can determine if a processor supports the SSE instruction and register set by performing a CPUID request type 1 and checking that EDX[SSE] = 1.
The implementation of SSE was accomplished by adding the following elements to the processor architecture:

- 70 new instructions (SSE instruction set) were added to the instruction set.
- Eight, 128-bit data registers were added to the architecture (see Figure 31-2 on page 760). Unlike the MMX registers which are aliased over the lower 64 bits of each of the x87 FPU’s data registers, the SSE data registers are implemented as separate registers.
- A Control/Status register (MXCSR; Figure 31-2 on page 760) to control the SSE FP SIMD capability and to indicate its status via the error status bits.
- A new SIMD FP exception was added to report SSE SIMD FP errors to the OS.
The Unabridged Pentium® 4

The SSE instructions can be divided into the following categories:

- SIMD FP instructions that simultaneously operate on four, 32-bit, Single Precision (SP) FP numbers.
- Scalar FP instructions. First, a definition of scalar: a single number, as opposed to a vector or matrix of numbers. As an example, scalar multiplication refers to the operation of multiplying one number (one scalar) by another and the term scalar is used to differentiate this from matrix math operations.
- Cacheability instructions including prefetches into different levels of the cache hierarchy.
- Control instructions.
- Data conversion instructions.
- New media extension instructions such as the PSAD and the PAVG that accelerate encoding and decoding, respectively.

Figure 31-2: The SSE Register Set

The SSE Data Types

General

Each 128-bit XMM register can hold:

- 16 bytes packed into an XMM register or into a memory variable, or
- 8 words packed into an XMM register or into a memory variable, or
Chapter 31: Pentium® III Software Enhancements

- 4 dwords packed into an XMM register or into a memory variable, or
- 2 qwords packed into an XMM register or into a memory variable, or
- Four 32-bit Single Precision (SP) FP numbers (see Figure 31-3 on page 765) packed into an XMM register or into a memory variable.

The 32-bit SP FP Numeric Format

Background. The new data type introduced with the advent of SSE is the 32-bit SP FP numeric format and it fully complies with the definition found in the IEEE Standard 754 for Binary FP Arithmetic. It should be noted that although this data type is new to the IA32 SIMD model, it is not new. It was defined in the 1980s and has been supported by the Intel® x87 FPU for many years. The x87 FPU, however, stores all FP numeric values in memory in the 80-bit (10 byte) DEP (Double Extended Precision) format (see “The FP Data Operand Format” on page 443). On reading a value from memory, the x87 can perform computations on the value in its native DEP form or, prior to performing a computation, can internally convert it into the 32-bit SP or the 64-bit DP form (see “DP FP Number Representation” on page 1334). When a numeric value is stored back to memory, however, the x87 FPU automatically converts it to the DEP form before storing it. The following is a brief tutorial on the 32-bit SP FP format.

A Quick IEEE FP Primer. The author would like to stress that this is not meant to be a comprehensive tutorial on the IEEE FP specification. Rather, it is meant to familiarize a reader who is not conversant in the FP vernacular with the major concepts and terms necessary to understand the basics.

A FP value represented in the IEEE FP format is computed as follows:

\[ x.yyyyy \times 2^{z_{th}} \]

where the digit to the left of the decimal point (x) is implied and is assumed to be one for all numbers (positive or negative) except for:

- Zero
- Numbers (irrespective of their sign, either positive or negative) that are less than 1 but greater than 0 (e.g., +0.1242, +0.98, -0.548, -0.13, etc.). These are referred to as denormal numbers (and are also referred to as tiny numbers).
- In both of these cases, the implied digit is assumed to be 0.

The range of all possible real numbers that can be represented using this format are limited by the width of the y field (referred to as the mantissa or significand field) and the z field (the exponent field). As shown in Figure 31-3 on page 765, the 32-bit format uses an 8-bit exponent field and a 23-bit mantissa field.
The Previous Chapter
This chapter provided a detailed description of the software enhancements introduced in the Pentium® III processor. This discussion is directly applicable to all subsequent IA32 processors. It included:

- The Streaming SIMD Extensions (SSE).
- The SIMD FP exception.
- The Serial Number feature. This feature was discontinued with the advent of the Pentium® 4 processor.
- CPUID Enhanced.
- Brand Index feature.

This Chapter
This chapter provides a description of the Xeon processor based on the Pentium® III technology. It includes:

- The processor’s basic characteristics.
- The Page Attribute Table (PAT) feature. This discussion is directly applicable to all subsequent IA32 processors.

The Next Chapter
This chapter provides the roadmap of Pentium® 4-based products.
The Pentium® III Xeon was based on the Pentium® III processor and had the following basic characteristics (in no particular order):

- It was available with core speeds up to 1GHz.
- It was available in two forms:
  - The FSB arbitration scheme of the Pentium® III Xeon MP supported up to four processors.
  - The FSB arbitration scheme of the Pentium® III Xeon DP supported up to two processors.
- Depending on the model, the BCLK frequency of the FSB was either 100MHz (on the earlier models), or 133 MHz (on the later models).
- The processor’s caches were capable of caching information from the entire 64GB memory address space.
- On all models, the processor’s L2 Cache had the following characteristics:
  - It was a unified code/data cache.
  - It was a non-Blocking (a miss did not cause it to stop servicing additional requests forwarded from the L1 caches).
  - The BSB ran at 100% of the processor’s core speed.
- The additional characteristics of the L2 Cache were model-specific:
  - The Pentium® III Xeon based on the .25µm process did not implement an on-die L2 ATC Cache. It was implemented using discrete SRAM chips, was connected to the core by a 64-bit data path, and was a 4-way, set-associative cache.
  - The model based on the .18µm process was the first Xeon to implement an on-die, ATC L2 Cache. It was 256KB in size. All subsequent Xeons implemented the on-die ATC.
  - The model based on the .13µm process had an ATC L2 Cache available in sizes of 1MB and 2MB.
  - All ATC’s have the following characteristics:
    - The ATC L2 Cache is integrated onto the processor die.
    - The cache is architected as an 8-way set associative cache.
    - The ATC is connected to the core via a 256-bit data path.
- ECC protection was available on the L2 Cache and on the FSB’s data paths. It was capable of detecting and correcting single-bit errors and detecting but not correcting multi-bit errors.
- This was the first IA32 processor to implement the PAT (Page Attribute Table) feature [see “PAT Feature (Page Attribute Table)” on page 797].
Chapter 32: Pentium® III Xeon Features

PAT Feature (Page Attribute Table)

What’s the Problem?

As previously discussed in “MTRRs Added” on page 572, it is imperative that the processor core know the proper way to behave when performing a memory access within any given region of memory space. The BIOS can program the memory type for each memory range into the MTTRs at startup time.

When the OS sets up the Page Directory and the Page Tables associated with each task, it uses the PCD and PWT bits in each PDE and PTE to define the memory type for the page of memory space:

- In the PTE that defines the mapping of a 4KB memory page, the PCD and PWT bits are used to define the page’s memory type.
- In a PDE that defines the mapping of a 4MB memory page, the PCD and PWT bits are used to define the page’s memory type.

Using a 2-bit field to define the page’s memory type imposes an obvious limit of no more than four possible memory types to choose from (in reality, PCD and PWT only permit three memory types). The PAT feature addresses this issue.

Detecting PAT Support

The programmer can determine whether or not a processor supports the PAT feature by performing a CPUID request type 1 and verifying that EDX[PAT] = 1 (see Figure 32-1 on page 798).

If a processor supports PAT, the PAT feature is always enabled.
PAT Allows More Memory Types

In a processor that supports the PAT feature, each of the following paging-related entries contains a bit (formally reserved) referred to as the PATᵢ (PAT Index) bit:

- Each PTE that maps to a 4KB page (see Figure 32-3 on page 799).
- Each PDE that maps to a 2MB page (if PAE-36 Mode is enabled; refer to “PAE-36 Mode” on page 554). Refer to Figure 32-4 on page 800.
- Each PDE that maps to a 4MB page (if 4MB pages are enabled—or if PSE-36 Mode is enabled—see “PSE-36 Mode” on page 731). Refer to Figure 32-5 on page 800.

The 3-bit field comprised of PATᵢ, PCD, and PWT allows 1-of-8 possible memory types to be specified. However, this bit field does not specify the memory type.
type directly. Rather, these three bits select one of eight possible fields in the IA32_CR_PAT MSR (see Figure 32-2 on page 799 and Table 32-1 on page 801). This MSR resides at MSR address 27710 (and is guaranteed to remain at this address). The value (see Table 32-2 on page 801) in the selected field of the MSR defines the memory type assigned to the page. It should be noted that Table 10-10 on page 10-39, section 10.12.2 of IA32 Intel® Architecture Software Developer’s Manual Volume 3: System Programming Guide indicates that each entry (i.e., field) in the IA32_CR_PAT MSR contains an 8-bit value. This is incorrect. Each entry contains a 3-bit value.

Figure 32-2: IA32_CR_PAT MSR

![Figure 32-2: IA32_CR_PAT MSR](image)

* = Reserved

Figure 32-3: PTE Mapped to a 4KB Page

![Figure 32-3: PTE Mapped to a 4KB Page](image)
The Previous Chapter
This chapter provided a description of the Xeon processor based on the Pentium® III technology. It included:

- The processor’s basic characteristics.
- The Page Attribute Table (PAT) feature. This discussion is directly applicable to all subsequent IA32 processors.

This Chapter
This chapter provides the roadmap of Pentium® 4-based products.

The Next Chapter
This chapter introduces the Pentium® 4 processor’s relationships with the various system subsystems.

The Roadmap
Table 33-1 on page 814 provides a brief description of the Pentium® 4 roadmap from its introduction and projecting into 2005. The reader should keep in mind that Intel®’s future roadmap is always subject to change, so don’t consider the future roadmap as carved in bronze.
Table 33-1: The Pentium® 4 Roadmap

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Willamette</td>
<td>11/20/00</td>
<td>Released at 1.4/1.5GHz with the following major features (in no particular order and not a complete list):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SSE2 instructions added ability to perform matrix math on packed DP FP numbers, and ability to perform MMX operations on data packed into 128-bit XMM registers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Completely re-designed core with 20 pipeline stages (see Figure 35-3 on page 846), versus 10 in the P6 processor family (see Figure 35-2 on page 845).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Improved branch prediction to avoid mispredictions whenever possible and the deep performance degradation that results from the pipeline flush.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The L1 Code Cache was redesigned to cache µops rather than legacy IA32 instructions. It is referred to as the Trace Cache (TC) and only caches µops corresponding to IA32 instructions along the predicted execution path.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The two integer execution units enhanced to complete the execution of an instruction in half a processor cycle (as opposed to one clock cycle in the P6 processors; referred to as double pumping the execution units). They are referred to as the Rapid Execution Engine.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The die also contains two FP units, one of which deals with x87 FP instructions, MMX and SSE-2 while the other manages FP moves and stores.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The cache line size was increased from 32 to 128 bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The on-die L2 cache is 256KB in size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The L2 Cache can deliver data in every clock cycle (versus the Coppermine L2 cache’s ability to deliver in every other clock cycle).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• It is based on the 0.18µm technology.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The FSB has a double-pumped Request Phase and a quad-pumped Data Phase.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The Error Phase has been eliminated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Interrupts are delivered over the processor’s FSB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The 3-wire APIC bus has been eliminated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• This version did not implement the Hyper-Threading feature (code named Jackson).</td>
</tr>
</tbody>
</table>
# Chapter 33: Pentium® 4 Road Map

Table 33-1: The Pentium® 4 Roadmap (Continued)

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
</table>
| Northwood                                  | 01/07/02 | This model was released at 2 and 2.2GHz and had the following additional major features (in no particular order and not a complete list):  
  • Based on 0.13 micron technology.  
  • 512KB on-die L2 Cache. |
| Northwood B with Hyper-Threading           | 11/14/02 | This model was released at 3.06GHz and had the following major features (in no particular order and not a complete list):  
  • First desktop Pentium® 4 with Hyper-Threading. |
| Northwood with 800MHz FSB                  | 04/14/03 | This model was released at 3GHz and had the following major features (in no particular order and not a complete list):  
  • This is the first IA32 processor with the quad-pumped, 800MHz FSB. |
| Pentium® 4 Extreme Edition (Gallatin)      | 09/16/03 | This model was released at 3.2GHz and had the following major features (in no particular order and not a complete list):  
  • Based on the Xeon MP's Gallatin core.  
  • Quad-pumped, 800MHz FSB.  
  • 512KB L2 Cache.  
  • 2MB on-die L3 Cache. |
Table 33-1: The Pentium® 4 Roadmap (Continued)

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prescott</td>
<td>02/01/04</td>
<td>This model was released at 2.4, 2.8, 3.0, 3.2 and 3.4GHz and had the following major features (in no particular order and not a complete list):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The first IA32 processor based on 90nm (nanometer) technology.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added the SSE3 instruction set (13 instructions).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Included a 2.8GHz model without Hyper-Threading and with a 533MHz, quad-pumped FSB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1MB L2 Cache.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 16KB L1 Data Cache.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Improved branch prediction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Improved Data Prefetcher.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Two new instructions were added to improve thread synchronization when using Hyper-Threading.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The instruction pipeline was expanded from 20 to 31 stages.</td>
</tr>
<tr>
<td>Tejas</td>
<td>Q2 of 2005</td>
<td>It is expected that this model will be released at 3.6GHz and eventually achieve 6GHz (or, as some believe, 9.2GHz). The following major features (in no particular order and not a complete list) are anticipated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Base on 90nm technology.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Packaged in a 775 contact LGA (might be called Socket T). Processors could be snapped in and out of a system board using a waffle-iron like device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Improved Hyper-Threading.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Eight new multimedia instructions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 24KB L1 Data Cache.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Initial FSB speed of 800MHz FSB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Eventual FSB speed of 1.066GHz.</td>
</tr>
<tr>
<td>Nehalem</td>
<td>2005</td>
<td>This processor is expected to have a completely new core design and is expected to be initially based on 90nm technology.</td>
</tr>
</tbody>
</table>
### Table 33-1: The Pentium® 4 Roadmap (Continued)

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
</table>
| Celeron Willamette | 05/15/01 | This model was released at 1.7GHz and has the following major features (in no particular order and not a complete list):  
- Based on the Willamette core.  
- On-die L2 cache is 128KB in size (versus 256KB on Willamette core).  
- Based on 0.18 micron technology.  
- 400MHz, quad-pumped FSB. |
| Celeron Northwood   | 9/18/02  | This model was released at 2GHz and had the following major features (in no particular order and not a complete list):  
- Based on the Northwood core.  
- 128KB L2 Cache.  
- 400MHz, quad-pumped FSB. |
| Celeron Prescott    | Q2, 2004 | Expected to be released at 2.53, 2.66, 2.8 and 3.06GHz. The following major features (in no particular order and not a complete list) are anticipated:  
- 256KB on-die L2 Cache.  
- 533MHz, quad-pumped FSB. |
| Foster DP          | 05/21/01 | This model was released at 1.4, 1.5 and 1.7GHz and had the following major features (in no particular order and not a complete list):  
- Supported two processors on the FSB.  
- Did not implement Hyper-Threading. |
| Foster MP          | 03/12/02 | This model was released at 1.4, 1.5 and 1.6GHz and had the following major features (in no particular order and not a complete list):  
- Supports up to four processors on the FSB.  
- 256KB on-die L2 Cache.  
- On-die 512KB or 1MB L3 Cache.  
- Did not implement Hyper-Threading. |
The Previous Chapter

This chapter provided the roadmap of Pentium® 4-based products.

This Chapter

This chapter introduces the Pentium® 4 processor’s relationships with the various system subsystems.

The Next Chapter

This chapter provides an overview of the Pentium® 4 processor. This includes:

- The Pentium® 4 Processor Family.
- Pentium® III/Pentium® 4 Differences.
- Pentium® 4/Pentium® 4 Prescott Differences.
- Pentium® 4 Processor Basic Organization.
- The FSB is Tuned for Multiprocessing.
- Intro to the FSB Enhancements.
- IA Instructions and µops.
- The Trace Cache.
- The µop Pipeline.
- The Alias Registers.
- Speculative Execution.
General

The chapter entitled “Overview of the Processor Role” on page 9 provides a good introduction to the processor’s role in the system.

Refer to Figure 34-1 on page 825. In a system that incorporates PCI Express, the Root Complex plays the role that used to be played by the Memory Control Hub (MCH), or the North Bridge. It is a bridge between the processor(s) and the remainder of the system. It also incorporates the system memory controller. For a complete description of the PCI Express spec, refer to the MindShare book entitled *PCI Express System Architecture* (published by Addison-Wesley).

A system may incorporate one or more processors. The maximum number of processors currently supported on a FSB are four [in a system that utilizes Pentium® 4 Xeon MP (Multi Processor) processors]. A system using the Pentium® 4 Xeon DP (Dual Processor) processor supports up to two processors on the FSB.

The processors are tasked with fetching instructions from system memory (labelled DDR RAM in the illustration), decoding them and executing them. In a PC-compatible system, the processors are only permitted to cache information that is read from system memory. They are not permitted to cache from memory other than system memory.

The Graphics Adapter

The graphics adapter (labeled GFX) is connected to the Root Complex, giving it a direct path to access system memory. The graphics controller can store graphics information that it requires very fast access to in its own, local memory, and can store additional information in an area of system memory set aside for its use (referred to as the graphics aperture). As of this writing, the graphics adapter is connected to the system memory controller via AGP (for a complete description of AGP, refer to the MindShare book entitled *AGP System Architecture*—published by Addison-Wesley), but with the advent of PCI Express systems in the latter half of 2004, AGP is being replaced by a PCI Express link to the Root Complex. The processor(s) or device adapters can access the graphics adapter’s register set and local memory through the Root Complex.
Device Adapters

Device adapters beneath the Root Complex frequently require access to system memory. They can do so by injecting memory read or write request packets into the fabric. These packets are guided to the system memory controller via switches along the path to the Root Complex (which contains the system memory controller).
The Unabridged Pentium® 4

Snooping

General

The system processor(s) cache information from system memory and may modify the data in the cache but not perform a memory write on the FSB to update the original line in memory. The line in memory is then stale. For this reason, whenever any device (a processor or a device adapter) attempts to access system memory, the memory access must be made visible to the processors so they may snoop the memory address in their caches. In the Snoop Phase of the transaction, the processors provide the request initiator with the snoop result. If the snoop results in a hit on a modified line, the processor with the modified copy of the line asserts the HITM# signal in the Snoop Phase and then provides its modified copy of the line in the transaction’s Data Phase.

A Memory Access Initiated by a Processor

When a processor has a miss on its internal caches and initiates a memory access on the FSB, the other processors latch the transaction, determine that it’s a memory access, and submit the memory address to their internal caches for a lookup. In the transaction’s Snoop Phase, they provide the Request Agent (i.e., the processor that experienced the miss) with one of the snoop results shown in Table 34-1 on page 827 (the values shown for the signals are electrical values). Please note that this section is not intended to be a detailed description including all aspects of snooping (for a detailed description, refer to “Pentium® 4 FSB Snoop Phase” on page 1225).
Table 34-1: Possible Snoop Results on a Memory Access by a Processor

<table>
<thead>
<tr>
<th>Access Type</th>
<th>HIT#</th>
<th>HITM#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Read</td>
<td>1</td>
<td>1</td>
<td>A snoop miss on all processor caches. The Request Agent is permitted to read the line from system memory.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>A snoop hit on one or more copies of the line that is still the same as the line originally read from system memory. Once again, the Request Agent is permitted to read the line from system memory.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>A snoop hit on a copy of the line in the modified (M) state. When the system memory controller detects this snoop result, it cancels its read of the line from system memory. In the transaction’s Data Phase, the processor that asserted HITM# supplies the line to both the Request Agent (i.e., the other processor) and to the system memory controller. The system memory controller latches the line and uses it to update the stale copy of the line in memory. The processor that supplied the modified copy of the line changes the state of the line in its cache to indicate that it now the same as the one in memory.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>One or more of the Snoop Agents (i.e., the processors) need a little more time before supplying the actual snoop result. As a result, the Request Agent inserts wait states in the transaction’s Snoop Phase until the actual snoop result is provided.</td>
</tr>
</tbody>
</table>
This chapter introduced the Pentium® 4 processor’s relationships with the various system subsystems.

This chapter provides an overview of the Pentium® 4 processor. This includes:

- The Pentium® 4 Processor Family.
- Pentium® III/Pentium® 4 Differences.
- Pentium® 4/Pentium® 4 Prescott Differences.
- Pentium® 4 Processor Basic Organization.
- The FSB is Tuned for Multiprocessing.
- Intro to the FSB Enhancements.
- IA Instructions and µops.
- The Trace Cache.
- The µop Pipeline.
- The Alias Registers.
- Speculative Execution.

This chapter provides a detailed description of the automatic configuration of the Pentium® 4 processor when the system is first powered up. This includes:

- Setup and Hold Time Requirements.
- Built-In Self-Test (BIST) Trigger.
- The Cluster ID Assignment.
- The Agent ID Assignment.
The Unabridged Pentium® 4

- The Local APIC ID Assignment.
- Error Observation Options.
- In-Order Queue Depth Selection.
- Power-On Restart Address.
- Tri-State Mode.
- Processor Core Speed Selection.
- Bus Parking Option.
- Hyper-Threading Option.
- Program-Accessible Startup Features.

The Pentium® 4 Processor Family

As was the case with the earlier IA32 processors starting with the Pentium® II, Intel® sells three product lines based on the Pentium® 4 processor:

- The Pentium® 4 desktop processors.
- The Pentium® 4 Celeron for low-cost, single-processor systems.
- The Pentium® 4M product line for notebooks. This product line is being eliminated and the Pentium® M processor is taking its place.
- The Pentium® 4 Xeon product line which is divided into two products: the Xeon DP and the Xeon MP.

All of them are derivatives of the basic Pentium® 4 processor.

Pentium® III/Pentium® 4 Differences

The following is a list of major differences between the Pentium® III and the Pentium® 4 processors:

- Hyper-Pipelined micro-architecture. The processor core was completely redesigned and is now referred to as the NetBurst Architecture. While the P6 processor family had a 10-stage pipeline, the Pentium® 4 has a 20-stage pipeline.
- The branch prediction mechanism was enhanced. This was necessary because a mispredicted branch in a processor with such a deep pipeline causes the creation of a large bubble and a deep dip in performance.
- The ability to control the processor’s internal temperature via software-controlled clock modulation was added.
- The Last Branch, Interrupt, and Exception Recording was enhanced with the addition of the Branch Trace Store (BTS) facility.
- The Debug Store (DS) mechanism was added.
- SMT (Simultaneous MultiThreading) implemented via Hyper-Threading.
- A data prefetcher was implemented in hardware.
Chapter 35: Pentium® 4 Processor Overview

- The SSE2 instruction set was added.
- The Pentium® 4 Xeon MP was the first model to add an on-die L3 cache. Subsequently, the Pentium® 4 Extreme and later versions of the Pentium® 4 Xeon DP were introduced with an on-die L3 cache.
- The Trace Cache replaced the L1 Code Cache.
- The FSB was enhanced in a number of ways.
- The ability to send interrupt message transactions over the FSB was added and the 3-wire APIC bus was eliminated.
- The array of three instruction decoders (Complex-Simple-Simple) were replaced by a decoder that decodes one IA32 instruction at a time.
- A number of MSRs were declared part of the architecture specification. Their names, addresses and bit field functions are guaranteed not to change in future IA32 processors.
- The processor’s two integer execution units can each execute two instructions per clock cycle (this is referred to as double-pumping) and is referred to as the Rapid Execution Engine.
- The cache line size in the L1 Data Cache was increased from 32 bytes in the P6 family to 64 bytes. The cache line size in the L2 and L3 Caches is 128 bytes with each line divided into two sectors of 64 bytes each.
- The Pentium® III processor had four 32-byte buffers that could be used either as fill buffers when a line had been requested from system memory, or as WCBs. The Pentium® 4 processor has 6 dedicated WCBs and each of them is 64 bytes in size (as opposed to 32 bytes in the P6 processor family).
- The processor’s IOQ depth was increased from 8 to 12 entries.
- While the P6 processors implemented two performance counters, the Pentium® 4 increased the number of counters to 18.
- The ability to generate an interrupt when an internal temperature trip point is crossed was added.

Pentium® 4/Pentium® 4 Prescott Differences

The following is a list of differences between the Pentium® 4 and the Pentium® 4 Prescott (i.e., the Pentium® 4 based on the 90nm process technology; a detailed description of Prescott can be found in “The Pentium® 4 Prescott” on page 1091):

- This is the first IA32 processor based on Intel®’s 90nm process technology.
- The instruction pipeline depth increased from 20 stages to 31 stages.
- Branch prediction was improved again.
- The number of WCBs was increased from six to eight.
- The SSE3 instruction set, consisting of 13 new instructions, was added.
- Two new instructions added to improve thread synchronization when using Hyper-Threading.
The Unabridged Pentium® 4

- The Trace Cache BTB size was increased from 512 entries to 2K entries.
- The L1 Data Cache was increased from 8KB to 16KB and its architecture was changed from 4-way set-associative to 8-way set-associative.
- The on-die L2 Cache size was increased from 512KB to 1MB.
- The number of Store Buffers was increased from 24 to 32.
- Improvements were made to the store forwarding mechanism.
- The Static Branch Predictor was improved.
- The dynamic branch predictor was enhanced by adding an indirect branch predictor.
- More types of µops can be stored in the Trace Cache than in the earlier Pentium® 4 implementations.
- Added a shifter/rotator block to one of the ALUs (i.e., integer execution units). This allows the most common forms of shift and rotate instructions to be executed on one of the double-pumped ALUs. On earlier Pentium® 4 processor implementations, these operations were executed as complex integer operations and took multiple cycles to execute.
- On the earlier Pentium® 4 implementations, integer multiply operations were executed by the FP multiplier. The source operands had to be moved to the FP side of the execution engine and then the result had to be moved back to the integer side. The 90m Pentium® 4 implements a dedicated integer multiplier.
- Improvements were made to the instruction schedulers.
- The software PREFETCHh instruction has been enhanced.
- The hardware-based data prefetch mechanism has been improved.
- The L1 Data Cache does not stop processing loads until there are six outstanding misses that have been forwarded upstream to the L2 Cache for fulfillment. On the earlier Pentium® 4 processors, the L1 Data Cache started blocking after four misses.
- The Hyper-Threading capability has been enhanced.

Pentium® 4 Processor Basic Organization

Refer to Figure 35-1 on page 840. The Pentium® 4 processor includes the following major subsystems:

- The processor core. This is the heart of the beast: the instruction fetch, decode, execute engine. It is responsible for the following:
  - Instruction fetch.
  - Branch prediction.
  - Parsing of the IA instruction stream.
  - Decoding of IA instructions into primitive, fixed-length instructions (referred to as micro-ops, or µops).
Chapter 35: Pentium® 4 Processor Overview

— Mapping accesses for the small IA data register set to a larger physical register set.
— Dispatch, execution and retirement of µops.

• The **Local APIC**. This unit receives interrupt requests from sources local to the processor as well as from remote sources (e.g., other processors and the IO APIC) and is also capable of sending interrupt messages to other processors, or to the IO APIC(s) within the chipset. Unlike the Pentium® and P6 processor families (which implemented the 3-wire APIC bus), the Local APIC communicates with the Local APICs in other processors and with the IO APIC(s) in the chipset via interrupt message transactions performed on the FSB.

• The **L1 Data Cache**. This unit caches data from system memory to expedite the execution of load and store operations. In the event of a cache miss, the request is forwarded upstream to the L2 Cache over the BSB for fulfillment.

• The **Trace Cache**. Unlike the earlier IA32 processors, the L1 Code Cache in the Pentium® 4 does not cache legacy IA32 instructions. Rather, it caches the instructions after they have been decoded into µops. In the event of a cache miss, the request is forwarded upstream to the L2 Cache over the BSB for fulfillment.

• The Back Side Bus (BSB) interface connects the L2 Cache to the L1 caches as well as to the FSB interface unit (and to the on-die L3 Cache if the processor implements one).

• The on-die **L2 Cache**. The L2 Cache is on the processor die in all models of the Pentium® 4 processor family. This unit services loads and stores that miss the processor’s L1 caches. In the event of a cache miss, the request is forwarded upstream to the FSB (or to the on-die L3 Cache if the processor implements one).

• Optionally, an on-die **L3 Cache**. As of this writing, an on-die L3 cache is implemented on the Pentium® 4 Extreme, all Xeon MP processors, and later model Xeon DP processors.

• The Front Side Bus (FSB) interface. The FSB connects the processor to the outside world. The processor uses it to communicate with other devices, to snoop memory transactions initiated by other entities, and to send and receive interrupt messages.

In the event of a miss on either of the L1 caches, the L2 Cache can be accessed via the BSB at the same time that the processor (or another FSB agent) is using the FSB. This is referred to as the Dual Independent Bus Architecture, or DIBA.
36  Pentium® 4  
PowerOn  
Configuration

The Previous Chapter
This chapter provided an overview of the Pentium® 4 processor. This included:

- The Pentium® 4 Processor Family.
- Pentium® III/Pentium® 4 Differences.
- Pentium® 4/Pentium® 4 Prescott Differences.
- Pentium® 4 Processor Basic Organization.
- The FSB is Tuned for Multiprocessing.
- Intro to the FSB Enhancements.
- IA Instructions and µops.
- The Trace Cache.
- The µop Pipeline.
- The Alias Registers.
- Speculative Execution.

This Chapter
This chapter provides a detailed description of the automatic configuration of the Pentium® 4 processor when the system is first powered up. This includes:

- Setup and Hold Time Requirements.
- Built-In Self-Test (BIST) Trigger.
- The Cluster ID Assignment.
- The Agent ID Assignment.
- The Local APIC ID Assignment.
- Error Observation Options.
- In-Order Queue Depth Selection.
- Power-On Restart Address.
The Unabridged Pentium® 4

- Tri-State Mode.
- Processor Core Speed Selection.
- Bus Parking Option.
- Hyper-Threading Option.
- Program-Accessible Startup Features.

The Next Chapter

This chapter provides a detailed description of the processor’s state immediately after reset is removed. It also describes how the Boot Strap Processor (BSP) is selected, as well as the Application Processor discovery and configuration process. This discussion includes:

- The Processor’s State After Reset.
- EAX, EDX Content After Reset Removal.
- The Core Is Starving and Caching is Disabled.
- Boot Strap Processor (BSP) Selection.
- How the APs are Discovered and Configured.

Configuration on Trailing-Edge of Reset

The processor samples a subset of its signal pins on the trailing-edge of reset (see Figure 36-1 on page 857) to configure some of its operational characteristics. Figure 36-2 on page 857 illustrates the signals that are sampled and the features associated with each.

This raises the question of where the configuration information comes from. The chipset (specifically, the Root Complex, MCH, or North Bridge) contains a chipset-specific register that supplies this information to the processors. The default content of the register is chipset design-specific. When the machine is powered up, the chipset asserts reset to all system devices including the processor(s) on the FSB. It then drives the content of this register onto the configuration signals on the FSB and then deasserts the reset signal. All of the processors latch the configuration information on the trailing-edge of reset.

If the programmer wishes to change the way the processors have been configured, a new value is written into the chipset register and the chipset is then commanded to reassert reset and then deassert it. In response, the chipset asserts reset, drives out the new contents of the register, and then deasserts reset.

The sections that follow describe each of the power-on auto-configuration options.
Chapter 36: Pentium® 4 PowerOn Configuration

Figure 36-1: The Power-On Auto-Configuration

Figure 36-2: The Pentium® 4’s Power-On Configuration Pins

<table>
<thead>
<tr>
<th>PIN</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT#</td>
<td>BIST</td>
</tr>
<tr>
<td>A12#</td>
<td>ACPI Cluster ID</td>
</tr>
<tr>
<td>A11#</td>
<td>BINIT# Observation</td>
</tr>
<tr>
<td>A10#</td>
<td>MCERR# Observation</td>
</tr>
<tr>
<td>A9#</td>
<td>Disable Bus Parking</td>
</tr>
<tr>
<td>A15#</td>
<td>IOQ Depth</td>
</tr>
<tr>
<td>A7#</td>
<td>Disable Hyper-Threading</td>
</tr>
<tr>
<td>A31#</td>
<td>Tri-State Mode</td>
</tr>
<tr>
<td>SMI#</td>
<td>Agent ID</td>
</tr>
<tr>
<td>BR3#</td>
<td></td>
</tr>
<tr>
<td>BR2#</td>
<td></td>
</tr>
<tr>
<td>BR1#</td>
<td></td>
</tr>
</tbody>
</table>
Setup and Hold Time Requirements

Refer to Figure 36-3 on page 858. In order to reliably sample the values presented on its pins on the trailing-edge of the reset signal, the following setup and hold times must be met:

- the signal to be sampled must be in the appropriate state for at least four bus clocks before the trailing-edge of RESET#. This is the setup time requirement.
- the signal must be held in that state for at least two but not greater than 20 bus clocks after RESET# is deasserted. This is the hold time requirement.

**Figure 36-3: Setup and Hold Times**

Built-In Self-Test (BIST) Trigger

Refer to Figure 36-4 on page 859. If the INIT# pin is sampled in the low state at the trailing-edge of RESET#, the processor will execute its internal Built-In Self-Test prior to the initiation of program fetch and execution. The duration of the BIST is processor design-specific. The processor cannot monitor transactions initiated by other FSB agents while it is executing its BIST. For this reason, the processor will continually toggle the Block Next Request (BNR#) signal for the duration of the BIST. This prevents any other FSB agent from initiating a transaction until the BIST has been completed.

If the BIST completes successfully, EAX contains zero. If an error is incurred during the BIST, however, EAX contains a non-zero error code. Intel® does not provide a breakdown of the error codes. When the BIST is not invoked, EAX
contains zero when program execution is initiated after reset’s removal. In either case, the programmer should check to ensure that EAX is clear at the start of the Power-On Self-Test (POST) and not proceed with the POST if it contains a non-zero value. In a strong sense, this is a moot point because the BIST will more than likely hang if it fails (but EAX can still be read by a debug tool using the processor’s Test Access Port (TAP; i.e., its boundary scan interface).

Figure 36-4: The BIST Trigger

- The Duration of the BIST is processor design-specific.
- BNR# is asserted for the duration of the BIST to block the issuance of transactions by any other FSB agents until the BIST completes.
- The BIST result is placed in EAX at the conclusion of the BIST. EAX = 0 indicates good completion, while a non-zero value indicates a processor-specific error code.
The Previous Chapter
This chapter provided a detailed description of the automatic configuration of the Pentium® 4 processor when the system is first powered up. This included:

- Setup and Hold Time Requirements.
- Built-In Self-Test (BIST) Trigger.
- The Cluster ID Assignment.
- The Agent ID Assignment.
- The Local APIC ID Assignment.
- Error Observation Options.
- In-Order Queue Depth Selection.
- Power-On Restart Address.
- Tri-State Mode.
- Processor Core Speed Selection.
- Bus Parking Option.
- Hyper-Threading Option.
- Program-Accessible Startup Features.

This Chapter
This chapter provides a detailed description of the processor’s state immediately after reset is removed. It also describes how the Boot Strap Processor (BSP) is selected, as well as the Application Processor discovery and configuration process. This discussion includes:

- The Processor’s State After Reset.
- EAX, EDX Content After Reset Removal.
- The Core Is Starving and Caching is Disabled.
- Boot Strap Processor (BSP) Selection.
- How the APs are Discovered and Configured.
The Unabridged Pentium® 4

The Next Chapter

This chapter provides a detailed description of the Pentium® 4 processor core. This includes:

- The Big Picture.
- The Front-End Pipeline Stages.
- Intro to the µop Pipeline.
- The µop Pipeline’s Major Elements.
- Additional, Core-Specific Terms.

Introduction

The initial steps in the system startup are as follows:

1. The power is off.
2. When the power is turned on, the power supply keeps the PowerGood signal deasserted to the chipset until the supply voltages are up and stable. During this period of time, the chipset keeps reset asserted to the processors and to all other devices until the power is stable and for a period of time afterwards (to allow sufficient time for clock generators to spin up, etc.).
3. While reset is still asserted to the processors, the chipset (specifically, the Root Complex, MCH, or North Bridge) drives the contents of its processor configuration register onto the FSB signal lines that provide startup configuration information to the processors (see “Pentium® 4 PowerOn Configuration” on page 855).
4. The chipset then deasserts the reset signal to the processors and the processors latch the configuration signals on the trailing-edge of reset.
5. The effects that reset’s assertion has on the processor(s) is covered in “The Processor’s State After Reset” on page 877.
6. If the configuration information instructed the processors to execute the BIST, the BIST runs to completion before the processor starts normal operation. The duration of the BIST is processor design-specific. During the BIST execution, the processor cannot monitor transactions initiated by other FSB agents. For this reason, the processor will continually toggle the Block Next Request (BNR#) signal for the duration of the BIST. This prevents any other FSB agent from initiating a transaction until the BIST has been completed.
7. Before any of the processors start fetching instructions from memory, the processor that is going to perform the initial system startup and the OS boot (it is referred to as the Boot Strap Processor, or BSP) must be selected. The processors perform transactions on the FSB to decide which will be the BSP.
Chapter 37: Pentium® 4 Processor Startup

The other processors are referred to as Application Processors, or APs and they remain in the halt state (drawing a minimum of power) until they are instructed to execute a program by the program executing on the BSP.

8. At this point, the BSP starts fetching instructions from memory at the power-on restart address (FFFFFFF0h). This program is fetched from the Boot ROM (also called the system BIOS ROM).

9. The BIOS startup code accomplishes the following:
   — It executes the Power-On Self-Test (POST) code to test the processor’s basic functionality as well as the basic functionality of the system board components and device adapters that will be necessary to boot the OS into memory.
   — It configures the processor in preparation for booting the OS into memory.
   — It creates an entry in the Multiprocessing Table and in the ACPI Table indicating the BSP’s type and capabilities.
   — It places a program in memory to be executed by each of the APs (which are currently in the halt state).
   — It instructs its Local APIC to send a Startup IPI (Inter Processor Interrupt; SIPI) to all of the other Local APICs in the system.
   — Upon receipt of the SIPI, each of the APs, in turn, executes the program they have been instructed to execute. The execution of this program causes the AP to be configured and causes it to make an entry in the Multiprocessing Table and in the ACPI Table indicating the AP’s type and capabilities. At the end of the startup program, the AP halts.

10. The program executing on the BSP then reads the OS startup code into memory and causes the processor to execute it.

11. The OS startup program boots the remainder of the OS kernel into memory, sets up any data structures necessary for the OS’s use (the GDT, the LDTs, Page Directories, Page Tables, etc.). The OS loads all of the loadable device drivers associated with the device adapters installed in the system. The OS also finishes the configuration of the device adapters throughout the system.

12. That’s it! The system is ready for normal operation.

The sections in this chapter describe many of the steps just introduced.

The Processor’s State After Reset

The assertion of the processor’s reset input has the effects indicated in Table 37-1 on page 878.
Table 37-1: Effects of Reset on the CPU

<table>
<thead>
<tr>
<th>Effect</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3 Cache</td>
<td>If the processor implements an L3 Cache, all entries in the L3 Cache are invalidated.</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>All entries in the L2 Cache are invalidated.</td>
</tr>
<tr>
<td>Trace Cache</td>
<td>All entries in the Trace Cache are invalidated.</td>
</tr>
<tr>
<td>L1 Data Cache</td>
<td>All entries in the L1 Data Cache are invalidated.</td>
</tr>
<tr>
<td>Branch Target Buffers (BTBs)</td>
<td>All entries in the BTBs are invalidated, causing all initial branches to be predicted by the static, rather than dynamic, branch prediction units. For additional information, refer to “The Front-End BTB” on page 910, “The Static Branch Predictor” on page 911, and “The Trace Cache BTB and the Return Stack Buffer” on page 925.</td>
</tr>
<tr>
<td>Instruction Prefetch Queue</td>
<td>The instruction prefetch queue is cleared, so there are no instructions available to the instruction pipeline.</td>
</tr>
<tr>
<td>µop queue</td>
<td>The instruction decode queue is invalidated, so there are no µops available to be executed.</td>
</tr>
<tr>
<td>The Re-Order Buffer (ROB)</td>
<td>The ROB is cleared, so there are no µops available for execution.</td>
</tr>
<tr>
<td>CR0</td>
<td>Contains 60000010h. This has the following effects:</td>
</tr>
<tr>
<td></td>
<td>• The processor is in Real Mode.</td>
</tr>
<tr>
<td></td>
<td>• Paging is disabled.</td>
</tr>
<tr>
<td></td>
<td>• Alignment Checking is disabled.</td>
</tr>
<tr>
<td></td>
<td>• Caching is disabled.</td>
</tr>
<tr>
<td></td>
<td>• The Write Protect feature is disabled.</td>
</tr>
<tr>
<td></td>
<td>• FP emulation is disabled.</td>
</tr>
<tr>
<td>CR4</td>
<td><strong>Software Features register.</strong> Contains 00000000h. All post-486 (Pentium®, P6, and Pentium® 4) software features are disabled.</td>
</tr>
</tbody>
</table>
## Chapter 37: Pentium® 4 Processor Startup

**Table 37-1: Effects of Reset on the CPU (Continued)**

<table>
<thead>
<tr>
<th>Effect</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR2</td>
<td><strong>Page Fault Address register.</strong> Contains 00000000h. No effect.</td>
</tr>
<tr>
<td>CR3</td>
<td><strong>Page Directory Base Address register.</strong> Contains 00000000h. No effect (because Paging is disabled).</td>
</tr>
<tr>
<td>DTLB and ITLB</td>
<td><strong>Data and Instruction Translation Lookaside Buffers.</strong> All DTLB and ITLB entries are invalidated. This has no initial effect because Paging is disabled.</td>
</tr>
<tr>
<td>The Local APIC</td>
<td><strong>Advanced Programmable Interrupt Controller.</strong> Has been assigned a Local APIC ID (see “The Local APIC ID Assignment” on page 865). If the BIST was triggered, it runs to completion. This processor’s Local APIC along with the Local APICs associated with the other processors (including the other logical processor within the same package), begins arbitrating for ownership of the FSB’s Request Phase signal group. The processor that wins ownership first starts the Special transaction and outputs the NOP message. It also sets the BSP bit in its IA32_APIC_BASE MSR. Upon receipt of the first processor’s NOP message, all of the other Local APICs clear the BSP bit in their respective IA32_APIC_BASE MSRs (this marks them as Application Processors, or APs) and their respective processors remain in the halt state. The AP Local APICs enter the “Wait For SIPI” state. See “How the APs are Discovered and Configured” on page 888 for more information. Recognition of all external interrupts is disabled.</td>
</tr>
</tbody>
</table>
This chapter provides a detailed description of the Pentium® 4 processor core. This includes:

- The Big Picture.
- The Front-End Pipeline Stages.
- Intro to the µop Pipeline.
- The µop Pipeline’s Major Elements.
- Additional, Core-Specific Terms.

The Next Chapter

This chapter provides a detailed description of Hyper-Threading and includes:

- Multithreading Overview.
- How Threads Are Assigned in an SMP System.
- CMP Is Another Solution.
- Traditional Single-Processor Multithreading.
One µop Doesn’t Necessarily = One IA32 Instruction

Like the P6 processor, the Pentium® 4 processor family does not execute the variable-length IA32 instructions. Rather, each IA32 instruction is decoded into a series of one or more µops (primitive, fixed-length instructions) which, when executed by the processor core, have the same effect on the processor’s state as would the IA32 instruction.

Just because a µop is retired doesn’t necessarily mean that an IA32 instruction is being retired. While most IA32 instructions translate into a single µop, some decode into a number of µops (perhaps even hundreds or more). There are two implications related to this:

- The processor only recognizes interrupts and exceptions on IA32 instruction boundaries, not on µop boundaries.
- The processor’s registers are only updated on IA32 instruction boundaries (when all of the µops associated with an IA32 instruction have completed execution).
Chapter 38: Pentium® 4 Core Description

Upstream vs. Downstream

Any references to the terms “upstream” or “downstream” should be interpreted as follows:

- **Upstream.** As in “the request is forwarded upstream to the L2 Cache”. It means to the next higher level in the memory hierarchy. The L1 Data Cache represents the lowest level of the memory hierarchy (i.e., the closest to the processor core). In order, the remaining upstream levels are: the L2 Cache, the L3 Cache (if there is one), and system memory.

- **Downstream.** As in “the data is forwarded downstream to the L1 Data Cache”. It means to the next lower level in the memory hierarchy. System memory represents the highest level of the memory hierarchy (i.e., the furthest from the processor core). In order, the remaining downstream levels are: the L3 Cache (if there is one), the L2 Cache, and the L1 Data Cache.

Introduction

This chapter describes the Pentium® 4 processor core and it assumes that Hyper-Threading is disabled. The chapter entitled “Hyper-Threading” on page 965 expands upon this chapter to describe how the core works when Hyper-Threading is enabled. It should be stressed that not every aspect of the processor core is covered in this chapter:

- The chapter entitled “The Pentium® 4 Caches” on page 1009 covers the L1 Data Cache, the L2 Cache and the L3 Cache. The Trace Cache is covered in the current chapter.
- The chapter entitled “Hyper-Threading” on page 965 broadens the processor core discussion to cover Hyper-Threading.
- The chapter entitled “Pentium® 4 Handling of Loads and Stores” on page 1061 describes how the processor core handles loads (i.e., memory data reads) and stores (i.e., memory data writes).
- The chapter entitled “The Pentium® 4 Prescott” on page 1091 describes how the 90nm version of the Pentium® 4 (code named Prescott) improved on various aspects of the processor design.

Intel® refers to the overall core design as the NetBurst Architecture.
The Big Picture

Although the entire machine (i.e., processor) is pipelined, this chapter conceptually segments the discussion into the front-end pipeline stages and the µop pipeline stages (Intel’s public domain documentation commonly refers to the µop pipeline stages as the instruction pipeline stages). The reader should not confuse the IA32 instructions with the equivalent µops into which they are decoded.

The processor’s core logic is pictured in the following illustrations:

- Figure 38-2 on page 901 pictures the front-end pipeline stages. These are the stages that fetch legacy IA32 instructions from memory, decode the instructions into µops, caches the µops in the Trace Cache, queues them up, and feeds them to the µop pipeline (pictured in Figure 38-1). As noted in Figure 38-2, the L1 Data Cache is not shown because the emphasis in this discussion is on the fetching, decoding and execution of instructions. A detailed description of the L1 Data Cache can be found in “The Pentium® 4 Caches” on page 1009.
- The front-end pipeline section’s final stage (the µop Queue) in Figure 38-2 is connected to the first stage (the Allocator) in Figure 38-3 on page 902. Figure 38-3 illustrates some of the major units that comprise the µop pipeline.
- Figure 38-1 on page 900 illustrates the 20 stages that comprise the µop pipeline.

This chapter discusses the processor core in three sections:

- “The Front-End Pipeline Stages” on page 902.
- “Intro to the µop Pipeline” on page 928.
- “The µop Pipeline’s Major Elements” on page 938.

Figure 38-1: The 20-Stage Instruction Pipeline
39 Hyper-Threading

The Previous Chapter
This chapter provided a detailed description of the Pentium® 4 processor core. This included:

- The Big Picture.
- The Front-End Pipeline Stages.
- Intro to the μop Pipeline.
- The μop Pipeline’s Major Elements.
- Additional, Core-Specific Terms.

This Chapter
This chapter provides a detailed description of Hyper-Threading and includes:

- Multithreading Overview.
- How Threads Are Assigned in an SMP System.
- CMP Is Another Solution.
- Traditional Single-Processor Multithreading.
- Detecting HT Capability.
- Enabling/Disabling HT.
- Each Logical Processor Has Its Own Local APIC.
- HT Processor Resource Types.
- The HT States.
- Processor Enumeration.
- OS Support for HT.
- Overview of HT Resource Usage.
- HT and the Data TLB.
- HT and the FSB.
- The IOQ Depth Was Increased.
- Thread Distribution to Logical Processors.
- Load Balancing.
- HT and the Processor Caches.
- Executing Identical Threads.
- Halt Usage.
Thread Synchronization.
WCB Usage.
HT and Serializing Instructions.
HT and the Microcode Update Feature.
HT and the TLBs.
HT and the Thermal Monitor Feature.
HT and External Pin Usage.

The Next Chapter

This chapter provides a detailed description of the Pentium® 4 caches. This includes:

- Determining the Processor’s Cache Sizes and Structures.
- Enabling/Disabling the Caches.
- The L1 Data Cache.
- The L2 ATC.
- The Hardware Data Prefetcher.
- The L3 Cache.
- FSB Transactions and the Caches.
- The Cache Management Instructions.

General

For the remainder of this chapter, Hyper-Threading is abbreviated as HT.

The code name for HT was Jackson and it was first implemented in the Prestonia version of the Pentium® 4 Xeon processor on 02/25/02. It first appeared in a desktop Pentium® 4 processor in the Northwood B version on 11/14/02. It has been in all Pentium® 4 models since that time, with the exception of the Pentium® M (which is based on the Pentium® III core rather than the Pentium® 4 core), and the 2.8GHz model of the 90nm Prescott Pentium® 4.

See “The Pentium® 4 Prescott” on page 1091 for enhancements made to HT in the 90nm Prescott versions of the Pentium® 4 processor.
Chapter 39: Hyper-Threading

Background

Multithreading Overview

In a multitasking OS (or an application written specifically for a multiprocessor system) a job may be subdivided into multiple tasks (also referred to as threads). In an SMP (Symmetric Multiprocessing) system (see Figure 39-1 on page 967), multiple physical processors reside on the FSB. Each processor in an SMP system can be commanded to execute a separate thread.

The threads comprising the overall task are simultaneously executed by the array of processors, yielding increased performance. This is commonly referred to as Thread-Level Parallelism (TLP).

Figure 39-1: An Example Multiprocessor (MP) System
How Threads Are Assigned in an SMP System

The OS scheduler assigns a task to an IA32 processor in the following manner:

- The OS places the thread in memory.
- One of the following actions is taken:
  - An IDT entry is created that points to the start address of the thread. The OS scheduler commands its processor’s Local APIC to send an IPI (Inter-Processor Interrupt) message to the Local APIC within the processor that is to execute the thread. Upon IPI receipt, using the vector in the message, the receiving Local APIC accesses the IDT and starts fetching and executing the thread pointed to by the IDT entry.
  - A SIPI message is sent to the target processor’s Local APIC containing the start address of the thread (in the IPI’s Vector field).

Implementing multi-threading using this approach cost more than the HT approach.

CMP Is Another Solution

Another approach is to place multiple processors cores on the same die. This takes up less system board real estate but, relatively speaking, this approach also cost more than the HT approach. This approach is commonly referred to as Chip-level Multiprocessing (CMP). As of this writing, an example processor that uses this approach is the IBM Power4 PowerPC chip. A number of multi-core processors are expected to be introduced by several other vendors (including Intel®) in the not too distant future.

Traditional Single-Processor Multithreading

There are two ways that an OS can cause a single processor core to switch between multiple threads:

- Time-sliced multithreading. This is really just multitasking—switching from one task to another after a fixed amount of time has passed (see “Definition of Multitasking” on page 27).
- Switch-on-event multithreading. As an example, a processor could be designed to switch to another task when a cache miss occurs.
Chapter 39: Hyper-Threading

The HT Approach

Instruction Level Parallelism (ILP)

Refer to Figure 39-2 on page 969. Instruction Level Parallelism (ILP) refers to a superscalar processor’s ability to dispatch and execute multiple instructions simultaneously (using an array of execution units). Optimized compilers attempt to keep as many of the execution units busy in each clock cycle as possible, but, in almost every clock cycle, one or more execution units are typically idle.

The number of execution units that are actually productive in each clock cycle is a function of the instruction mix that comprises the currently running program and even the finest program will have difficulty keep everyone productive all of the time.

Such a waste!

Figure 39-2: It’s Difficult Keeping All of the Execution Units Busy
The Previous Chapter
This chapter provided a detailed description of Hyper-Threading and included:

- Multithreading Overview.
- How Threads Are Assigned in an SMP System.
- CMP Is Another Solution.
- Traditional Single-Processor Multithreading.
- Detecting HT Capability.
- Enabling/Disabling HT.
- Each Logical Processor Has Its Own Local APIC.
- HT Processor Resource Types.
- The HT States.
- Processor Enumeration.
- OS Support for HT.
- Overview of HT Resource Usage.
- HT and the Data TLB.
- HT and the FSB.
- The IOQ Depth Was Increased.
- Thread Distribution to Logical Processors.
- Load Balancing.
- HT and the Processor Caches.
- Executing Identical Threads.
- Halt Usage.
- Thread Synchronization.
- WCB Usage.
- HT and Serializing Instructions.
- HT and the Microcode Update Feature.
- HT and the TLBs.
- HT and the Thermal Monitor Feature.
- HT and External Pin Usage.
The Unabridged Pentium® 4

This Chapter
This chapter provides a detailed description of the Pentium® 4 caches. This includes:

- Determining the Processor’s Cache Sizes and Structures.
- Enabling/Disabling the Caches.
- The L1 Data Cache.
- The L2 ATC.
- The Hardware Data Prefetcher.
- The L3 Cache.
- FSB Transactions and the Caches.
- The Cache Management Instructions.

The Next Chapter
This chapter provides a detailed description of load and store operations and includes:

- The Memory Type Defines Load/Store Characteristics.
- The Load Buffers.
- Loads from Cacheable Memory.
- Loads Can Be Executed Out-of-Order.
- The L1 Data Cache Implements Squashing.
- Loads from Uncacheable Memory.
- The Definition of a Speculatively Executed Load.
- Replay.
- Loads and the Prefetch Instructions.
- The LFENCE Instruction.
- Store-to-Load Forwarding.
- Stores Are Handled by the Store Buffers.
- Stores to UC Memory.
- Stores to WC Memory.
- Stores to WP Memory.
- Stores to WT Memory.
- Forcing a Buffer Drain.
- The SFENCE Instruction.
- Sharing Access to a UC, WC, WP or WT Memory Region.
- Stores to WB Memory.
- Out-of-Order String Stores.
- Stores and Hyper-Threading.
- The MFENCE Instruction.
- Non-Temporal Stores.
Chapter 40: The Pentium® 4 Caches

A Cache Primer

If the reader feels the need for primer on cache memory, refer to the chapter entitled “Caching Overview” on page 385.

The L0 Cache

Just a note that some Intel® documents (VERY few) make reference to the L0 cache. This is a reference to the L1 Data Cache (the lowest level cache that is closest to the processor core).

Upstream vs. Downstream

Any references to the terms “upstream” or “downstream” should be interpreted as follows:

- **Upstream.** As in “the request is forwarded upstream to the L2 Cache”. It means that it’s forwarded to the next higher level in the memory hierarchy. The L1 Data Cache represents the lowest level of the memory hierarchy (i.e., the closest to the processor core). In order, the remaining upstream levels are: the L2 Cache, the L3 Cache (if there is one), and system memory.

- **Downstream.** As in “the data is forwarded downstream to the L1 Data Cache”. It means that it’s forwarded to the next lower level in the memory hierarchy. System memory represents the highest level of the memory hierarchy (i.e., the furthest from the processor core). In order, the remaining downstream levels are: the L3 Cache (if there is one), the L2 Cache, and the L1 Data Cache.

Overview

All current implementations of the Pentium® 4 processor family include an on-die L1 Data Cache, an on-die Trace Cache (TC), and an on-die L2 ATC (Advanced Transfer Cache; i.e., the L2 Cache). Some implementations also include an on-die L3 Cache (e.g., the Pentium® 4 Extreme Edition and the Pentium® 4 Xeon MP).

This chapter provides a detailed description of the L1 Data Cache, the L2 Cache and the L3 Cache. The Trace Cache was described in “The Trace Cache” on page 919. Figure 40-1 on page 1012 shows the basic relationships of the caches to each other as well as the major characteristics of each of the caches.
Determining the Processor’s Cache Sizes and Structures

The OS can tune its use of memory to yield optimal processor performance if it understands the geometry of the processor’s caches and TLBs. The CPUID instruction may be executed with a request type 2 to return information regarding the size and organization of:

- the L2 Cache.
- the L3 Cache (if there is one).
- the L1 Data Cache.
- the L1 Code Cache (the Trace Cache in the Pentium® 4 family).
- the Code TLB.
- the Data TLBs.

For detailed information on the CPUID instruction, refer to “CPU Identification” on page 1443.
Chapter 40: The Pentium® 4 Caches

Enabling/Disabling the Caches

The caches are enabled or disabled using the CD and NW bits in CR0 (see Table 40-1 on page 1013).

Table 40-1: Enable/Disable the Caches

<table>
<thead>
<tr>
<th>CR0[CD]</th>
<th>CR0[NW]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Caching is fully enabled.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Invalid and Reserved.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The cache is locked. No new lines are loaded into the cache, but cache lookups are performed.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Caching is fully disabled.</td>
</tr>
</tbody>
</table>

The L1 Data Cache

The description of the L1 Data Cache in this section assumes that the L1 Data Cache is virtually addressed and that each cache directory entry contains a physical page address tag. This assumption is based on the following statement from an Intel® Technology Journal article entitled Hyper-Threading Technology Architecture and Microarchitecture:

“The L1 data cache is 4-way set associative with 64-byte lines. It is a write-through cache, meaning that writes are always copied to the L2 cache. The L1 data cache is virtually addressed and physically tagged.”

General

The Pentium® 4 processor family’s L1 Data Cache has the following major characteristics:

- It is a dedicated data cache. Unlike a unified cache which caches both code and data, the Data Cache treats all information as data. If an instruction from the Code Segment is loaded into a register using a load μop, the processor treats it as a data access and performs a lookup in the Data Cache.
The Previous Chapter
This chapter provided a detailed description of the Pentium® 4 caches. This included:

- Determining the Processor’s Cache Sizes and Structures.
- Enabling/Disabling the Caches.
- The L1 Data Cache.
- The L2 ATC.
- The Hardware Data Prefetcher.
- The L3 Cache.
- FSB Transactions and the Caches.
- The Cache Management Instructions.

This Chapter
This chapter provides a detailed description of load and store operations and includes:

- The Memory Type Defines Load/Store Characteristics.
- The Load Buffers.
- Loads from Cacheable Memory.
- Loads Can Be Executed Out-of-Order.
- The L1 Data Cache Implements Squashing.
- Loads from Uncacheable Memory.
- The Definition of a Speculatively Executed Load.
- Replay.
- Loads and the Prefetch Instructions.
- The LFENCE Instruction.
The Next Chapter

This chapter provides a complete description of the 90nm Prescott Pentium® 4 processor. This includes:

- Increased Pipeline Depth.
- Trace Cache Improvements.
- Increased Number of WCBs.
- L1 Data Cache Changes.
- Increased L2 Cache Size.
- Enhanced Branch Prediction.
- Store Forwarding Improved.
- SSE3 Instruction Set.
- Increased Elimination of Dependencies.
- Enhanced Shifter/Rotator.
- Integer Multiply Enhanced.
- Scheduler Enhancements.
- Fixed the MXCSR Serialization Problem.
- Data Prefetch Instruction Execution Enhanced.
- Improved the Hardware Data Prefetcher.
- Hyper-Threading Improved.

The Memory Type Defines Load/Store Characteristics

µops that read data from memory into a processor register are referred to as loads. µops that write to memory are referred to as stores.
Chapter 41: Pentium® 4 Handling of Loads and Stores

The manner in which the processor handles a load or a store is defined by the type of memory being written to. When a memory data access is initiated, the 32-bit linear memory address is submitted to the DTLB and the Paging Unit to translate the linear address into a physical memory access. The physical memory address is submitted to the MTRRs to determine the memory type. In addition, the PTE or PDE selected by the linear address also defines the memory type. If there is a memory type conflict between the two, the processor makes its decision based on Table 32-4 on page 803.

This chapter provides a detailed description of how loads and stores are handled in each of the various memory types:

- UC is uncachable memory. “Uncacheable (UC) Memory” on page 582 provides an introduction to the UC memory type.
- WC is uncacheable, Write-Combining memory. “Write-Combining (WC) Memory” on page 582 provides an introduction to the WC memory type.
- WP is cacheable, Write-Protected memory. “Write-Protect (WP) Memory” on page 584 provides an introduction to the WP memory type.
- WT is cacheable, Write-Through memory. “Write-Through (WT) Memory” on page 583 provides an introduction to the WT memory type.
- WB is cacheable, Write-Back memory. “Write-Back (WB) Memory” on page 584 provides an introduction to the WB memory type.

Load µops

The Load Buffers

When a load µop arrives at the Allocator stage of the instruction pipeline (see “The Allocator” on page 938), the Allocator reserves one of the processor’s 48 Load Buffers to handle the load when it is subsequently dispatched for execution. If Hyper-Threading is enabled, the 48 Load Buffers are partitioned into two groups of 24 buffers each and each group is reserved for the use of one of the logical processors.

See Figure 41-1 on page 1064. Port 2 supports the dispatch of one load operation per cycle. When a load µop is executed by the Load execution unit, the load request is placed in one of Load Buffers and remains there until one of the following becomes true:

- The load µop is completed, retired, and deallocated.
- Loads from WC, WP, WT and WB memory can be speculatively executed (a speculative load is a load that lies beyond a conditional branch µop that has
not yet been executed). If, when the conditional branch µop is subsequently executed it is determined that one or more speculative loads that lie beyond the branch should not have been executed, the contents of those Load Buffers are discarded and those Load Buffers become available to handle additional load µops. See “The Definition of a Speculatively Executed Load” on page 1067.

**Figure 41-1: The Load Execution Unit**

---

**Loads from Cacheable Memory**

The types of memory that the processor is permitted to cache from are WP, WT and WB memory (as defined by the MTRRs and the PTE or PDE).

When the core dispatches a load µop, the µop is placed in the Load Buffer that was reserved for it in the Allocator stage. The memory data read request is then issued to the L1 Data Cache for fulfillment:
Chapter 41: Pentium® 4 Handling of Loads and Stores

1. If the cache has a copy of the line that contains the requested read data, the read data is placed in the Load Buffer.
2. If the cache lookup results in a miss, the request is forwarded upstream to the L2 Cache.
3. If the L2 Cache has a copy of the sector that contains the requested read data, the read data is immediately placed in the Load Buffer and the sector is copied into the L1 Data Cache.
4. If the cache lookup results in a miss, the request is forwarded upstream to either the L3 Cache (if there is one) or to the FSB Interface Unit.
5. If the L3 Cache has a copy of the sector that contains the requested read data, the read data is immediately placed in the Load Buffer and the sector is copied into the L2 Cache and the L1 Data Cache.
6. If the lookup in the top-level cache results in a miss, the request is forwarded to the FSB Interface Unit.
7. When the sector is returned from memory, the read data is immediately placed in the Load Buffer and the sector is copied into the L3 Cache (if there is one), the L2 Cache, and the L1 Data Cache.

The processor core is permitted to speculatively execute loads that read data from WC, WP, WT or WB memory space (see “The Definition of a Speculatively Executed Load” on page 1067).

Loads Can Be Executed Out-of-Order

The following code fragment reads the contents of four memory-based variables into four of the processor’s registers. The description that follows assumes that all four of the memory variables are in cacheable memory:

```assembly
mov eax, mem1
mov ebx, mem2
mov ecx, mem3
mov edx, mem4
---
```

Prior to the advent of the P6 processor family, these instructions would be executed in strict program order. The P6 and Pentium® 4 family processors, however, utilize out-of-order execution strategies:

1. The first load µop is dispatched to its assigned Load Buffer and that Load Buffer submits the read request to the L1 Data Cache.
2. If the first load µop resulted in a cache miss, the read request is forwarded upstream to the L2 Cache for fulfillment.
The Pentium® 4 Prescott

The Previous Chapter
This chapter provided a detailed description of load and store operations and included:

- The Memory Type Defines Load/Store Characteristics.
- The Load Buffers.
- Loads from Cacheable Memory.
- Loads Can Be Executed Out-of-Order.
- The L1 Data Cache Implements Squashing.
- Loads from Uncacheable Memory.
- The Definition of a Speculatively Executed Load.
- Replay.
- Loads and the Prefetch Instructions.
- The LFENCE Instruction.
- Store-to-Load Forwarding.
- Stores Are Handled by the Store Buffers.
- Stores to UC Memory.
- Stores to WC Memory.
- Stores to WP Memory.
- Stores to WT Memory.
- Forcing a Buffer Drain.
- The SFENCE Instruction.
- Sharing Access to a UC, WC, WP or WT Memory Region.
- Stores to WB Memory.
- Out-of-Order String Stores.
- Stores and Hyper-Threading.
- The MFENCE Instruction.
- Non-Temporal Stores.
This Chapter

This chapter provides a complete description of the 90nm Prescott Pentium® 4 processor. This includes:

- Increased Pipeline Depth.
- Trace Cache Improvements.
- Increased Number of WCBs.
- L1 Data Cache Changes.
- Increased L2 Cache Size.
- Enhanced Branch Prediction.
- Store Forwarding Improved.
- SSE3 Instruction Set.
- Increased Elimination of Dependencies.
- Enhanced Shifter/Rotator.
- Integer Multiply Enhanced.
- Scheduler Enhancements.
- Fixed the MXCSR Serialization Problem.
- Data Prefetch Instruction Execution Enhanced.
- Improved the Hardware Data Prefetcher.
- Hyper-Threading Improved.

The Next Chapter

This chapter provides a detailed description of the FSB’s electrical characteristics. This includes:

- The BSEL Outputs.
- The Processor’s Operational Clock Frequency.
- BCLK Is a Differential Signal.
- The Address and Data Strobes.
- The Voltage ID.
- All AGTL+ Signals Are Active When Low.
- All AGTL+ Signals Are Terminated.
- Deasserting an AGTL+ Signal Line.
- Each AGTL+ Input Has a Comparator.
- The Reference Voltage.
- The Sample Point.
- The Pre-90nm Comparison.
- The 90nm Comparison.
- AGTL+ Setup and Hold Specs.
- Signals that Can Be Driven by Multiple FSB Agents.
- Minimum One BCLK Response Time.
Introduction

At the time of this writing, the first 90nm (nanometer) version of the Pentium® 4 processor (code named Prescott) has just been introduced. The previous versions were based on the 130nm (0.13 micron) process technology. This chapter describes the improvements found in this new processor.

Increased Pipeline Depth

In order to support higher clock rates, the 20 pipeline stages found in the earlier Pentium® 4 processors have been further divided into 31 stages. Intel® has not provided any information in the public domain regarding the stage names or functions, but it is widely believed that the processor stages remain unchanged (other than being subdivided into sub-stages that each contain less logic and which can therefore be clocked at a faster rate).

Trace Cache Improvements

Increased Trace Cache BTB Size

The Trace Cache BTB size was increased from 512 entries to 2K entries, permitting the processor to maintain execution history on up to 2K conditional branches contained in the Trace cache.

Enhanced Trace Cache µop Encoding

When a complex IA32 instruction is encountered (one that decodes into more than four µops), it is submitted to the Microcode Store ROM which streams the equivalent µops into the pipeline. In addition, a token (consisting of a microcode instruction pointer) representing the complex instruction is placed in the Trace Cache. Whenever it has to be executed, it is sent to the ROM which then streams the resultant µops to the µop Queue.

The 90nm processor’s Trace Cache has been improved in that some instructions that had to go to the ROM in the earlier processors can now be stored in the Trace Cache. Two examples are:

- Indirect calls with a register source operand.
- The software PREFETCHh instructions.
Increased Number of WCBs

While the earlier processors implemented a total of six WCBs, the 90nm version implements eight WCBs.

L1 Data Cache Changes

The L1 Data Cache has improved in the following ways:

- Its size has increased from 8KB to 16KB.
- It architecture was 4-way set-associative. It is now 8-way set-associative.
- In the earlier versions, the L1 Data Cache would not block the servicing of load/store requests until four cache misses had occurred. This number has been increased to eight. While this has little effect on a processor executing a single thread, it enhances performance when both logical processors are executing threads.
- As previously covered in “The Data Cache Lookup” on page 1022, the documentation for the earlier versions of the processor specifically state that the L1 Data Cache is virtually-addressed and physically-tagged. The following statement is from an Intel® Technology Journal article on the 90nm microarchitecture:

  — “On top of the changes to the execution units, we also changed the L1 data cache. As with all implementations of the NetBurst microarchitecture, the cache is designed to minimize the load-to-use latency by using a partial virtual address match to detect early in the pipeline whether a load is likely to hit or miss in the cache. On this processor, we significantly increased the size of the partial address match from previous implementations, thus reducing the number of false aliasing cases.”

Increased L2 Cache Size

The unified L2 Cache size has been increased from 512KB to 1MB. It is still 8-way set-associative with a cache line size of 128 bytes and each line is subdivided into two sectors of 64 bytes each.
Enhanced Branch Prediction

Enhanced Static Branch Predictor

The Static Branch Predictor (see “The Static Branch Predictor” on page 911) is consulted when a miss occurs on the Front-End BTB (i.e., the BTB doesn’t have any execution history on a conditional branch instruction). In the earlier versions of the processor, it would predict a backward relative conditional branch as taken and a forward branch as not taken. This approach works well for a backward branch at the end of a loop, but not all backward-relative branches reside at the bottom of a loop.

The 90nm processor’s Static Branch Predictor uses the distance that the branch jumps backward as well as the condition on which the branch depends to try and determine if the branch resides at the bottom of a loop or not:

- Intel®’s studies indicated that there is a threshold for the distance between a backward branch and its branch target address. If the distance of the branch is more than the threshold value, the branch is deemed unlikely to reside at the bottom of a loop. The Static Branch Predictor only predicts a branch as taken if the branch distance is less than the threshold value.
- Intel®’s studies also indicated that branches based on certain conditions are, more often than not, not taken (regardless of the branch’s direction and/or distance). These conditions are not common loop-ending conditions, so the Static Branch Predictor predicts them as not taken.

Dynamic Branch Prediction Enhanced

The dynamic branch predictor (i.e., the BTB) added an indirect branch predictor. Note that the Pentium® M processor also implements the indirect branch predictor (see “The Indirect Branch Predictor” on page 1436 for more information).

Store Forwarding Improved

Increased Number of Store Buffers

The number of Store Buffers has been increased from 24 to 32.
The Previous Chapter
This chapter provides a complete description of the 90nm Prescott Pentium® 4 processor. This included:

- Increased Pipeline Depth.
- Trace Cache Improvements.
- Increased Number of WCBs.
- L1 Data Cache Changes.
- Increased L2 Cache Size.
- Enhanced Branch Prediction.
- Store Forwarding Improved.
- SSE3 Instruction Set.
- Increased Elimination of Dependencies.
- Enhanced Shifter/Rotator.
- Integer Multiply Enhanced.
- Scheduler Enhancements.
- Fixed the MXCSR Serialization Problem.
- Data Prefetch Instruction Execution Enhanced.
- Improved the Hardware Data Prefetcher.
- Hyper-Threading Improved.

This Chapter
This chapter provides a detailed description of the FSB’s electrical characteristics. This includes:

- The BSEL Outputs.
- The Processor’s Operational Clock Frequency.
**The Unabridged Pentium® 4**

- BCLK Is a Differential Signal.
- The Address and Data Strobes.
- The Voltage ID.
- All AGTL+ Signals Are Active When Low.
- All AGTL+ Signals Are Terminated.
- Deasserting an AGTL+ Signal Line.
- Each AGTL+ Input Has a Comparator.
- The Reference Voltage.
- The Sample Point.
- The Pre-90nm Comparison.
- The 90nm Comparison.
- AGTL+ Setup and Hold Specs.
- Signals that Can Be Driven by Multiple FSB Agents.
- Minimum One BCLK Response Time.

**The Next Chapter**

This chapter introduces the Pentium® 4 FSB. It includes:

- Enhanced Mode Scalable Bus.
- FSB Agents.
- The Request Agent.
- The Transaction Phases.
- Transaction Pipelining.
- Transaction Tracking.

**Introduction**

One of the keys to a high-speed signaling environment is to utilize a low-voltage swing (LVS) to change the state of a signal from one state to the other. The P6 and Pentium® 4/M (i.e., Pentium® 4 and Pentium® M) FSB falls into this category. It permits the operation of the FSB at speeds of 200MHz or higher. The FSB is implemented using a modified version of the industry standard GTL (Gunning Transceiver Logic) specification, referred to by Intel® as AGTL+ (Assisted GTL+). The spec has been modified to provide larger noise margins and reduce ringing. This was accomplished by using a higher termination voltage and controlling the edge rates. The net result is that the FSB supports more electrical loads (currently up to eight devices) than it would if implemented using the standard GTL spec. The sections that follow introduce the basic concepts behind FSB operation. A detailed AGTL+ spec can be obtained from Intel®.
Chapter 43: Pentium® 4 FSB Electrical Characteristics

The Bus and Processor Clocks

The BSEL Outputs

Each model of Pentium® 4 processor is designed to operate at a certain internal clock frequency as well as a FSB frequency. A clock generator on the system board generates the Bus Clock (BCLK) to the processor(s) and all other FSB agents. The processor provides two outputs, BSEL[1:0], that are connected to the system board’s clock generator and the 2-bit pattern that is output on these two signals tells the clock generator the frequency of the BCLK to be supplied to all FSB agents. Table 43-1 on page 1117 defines the possible settings on the processor’s BSEL[1:0] outputs. Currently, the Pentium® 4’s FSB has a BCLK speed of 200MHz.

Table 43-1: BSEL Truth Table

<table>
<thead>
<tr>
<th>BSEL1</th>
<th>BSEL0</th>
<th>BCLK Frequency Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>100MHz.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>133MHz.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>200MHz.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

The Processor’s Operational Clock Frequency

The processor derives its internal clock from the BCLK frequency. BCLK is provided as an input to a PLL (Phase-Locked Loop) within the processor. The PLL multiplies the BCLK frequency by a factory preset multiplier value to yield the internal processor clock.

BCLK Is a Differential Signal

All signaling on the FSB is synchronized to the Bus Clock (BCLK). While this function was fulfilled by one signal line (BCLK) on the P6 FSB, it is now a differential signal pair comprised of the BCLK[1:0] signals (see Figure 43-1 on page 1118).
The Unabridged Pentium® 4

All FSB timing parameters are specified with respect to the rising-edge of BCLK0 crossing V CROSS (i.e., the point where the voltage level on BCLK0 and BCLK1 are equal).

Common clock signals are driven or are sampled when the rising-edge of BCLK0 crosses V CROSS. They are listed in Table 43-2 on page 1118.

Table 43-2: Signals that Are Synchronous to BCLK[1:0]

<table>
<thead>
<tr>
<th>Signal Name(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPRI#</td>
<td>Bus Priority Agent Request.</td>
</tr>
<tr>
<td>DEFER#</td>
<td>The Defer or Retry signal.</td>
</tr>
<tr>
<td>RESET#</td>
<td>The Hard Reset signal.</td>
</tr>
<tr>
<td>RS[2:0]#</td>
<td>The Response bus.</td>
</tr>
<tr>
<td>RSP#</td>
<td>The parity bit for the Response bus.</td>
</tr>
<tr>
<td>TRDY#</td>
<td>Target Ready.</td>
</tr>
<tr>
<td>AP[1:0]#</td>
<td>The Request Phase parity bits for packets A and B.</td>
</tr>
<tr>
<td>ADS#</td>
<td>Address Strobe.</td>
</tr>
<tr>
<td>BINIT#</td>
<td>Bus Initialization.</td>
</tr>
<tr>
<td>BNR#</td>
<td>Block Next Request.</td>
</tr>
</tbody>
</table>
Chapter 43: Pentium® 4 FSB Electrical Characteristics

Table 43-2: Signals that Are Synchronous to BCLK[1:0] (Continued)

<table>
<thead>
<tr>
<th>Signal Name(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPM[5:0]#</td>
<td>Breakpoint/Performance Monitor output pins.</td>
</tr>
<tr>
<td>BR0#</td>
<td>The Bus Request output.</td>
</tr>
<tr>
<td>DBSY#</td>
<td>Data Bus Busy.</td>
</tr>
<tr>
<td>DP[3:0]#</td>
<td>The Data Bus parity bits.</td>
</tr>
<tr>
<td>DRDY#</td>
<td>Data Ready.</td>
</tr>
<tr>
<td>HIT#</td>
<td>• If HIT# is asserted but HITM# is not, signals a Hit on an unmodified line.</td>
</tr>
<tr>
<td></td>
<td>• If both HIT# and HITM# are asserted, signals a Snoop Stall condition.</td>
</tr>
<tr>
<td>HITM#</td>
<td>• If HITM# is asserted but HIT# is not, signals a Hit on a modified line.</td>
</tr>
<tr>
<td></td>
<td>• If both HIT# and HITM# are asserted, signals a Snoop Stall condition.</td>
</tr>
<tr>
<td>LOCK#</td>
<td>Asserted during a locked read/modify write operation.</td>
</tr>
<tr>
<td>MCERR#</td>
<td>The Machine Check Error output.</td>
</tr>
<tr>
<td>ADSTB[1:0]#</td>
<td>The Request Phase strobes.</td>
</tr>
<tr>
<td>DSTBP[3:0]#, DSTBN[3:0]#</td>
<td>The Data Phase strobes.</td>
</tr>
</tbody>
</table>

The Address and Data Strobes

Delivering the Request

When a transaction is initiated, the initiating agent outputs two packets of information that completely describe the transaction.
The Previous Chapter
This chapter provided a detailed description of the FSB’s electrical characteristics. This included:

- The BSEL Outputs.
- The Processor’s Operational Clock Frequency.
- BCLK Is a Differential Signal.
- The Address and Data Strobes.
- The Voltage ID.
- All AGTL+ Signals Are Active When Low.
- All AGTL+ Signals Are Terminated.
- Deasserting an AGTL+ Signal Line.
- Each AGTL+ Input Has a Comparator.
- The Reference Voltage.
- The Sample Point.
- The Pre-90nm Comparison.
- The 90nm Comparison.
- AGTL+ Setup and Hold Specs.
- Signals that Can Be Driven by Multiple FSB Agents.
- Minimum One BCLK Response Time.

This Chapter
This chapter introduces the Pentium® 4 FSB. It includes:

- Enhanced Mode Scalable Bus.
- FSB Agents.
- The Request Agent.
- The Transaction Phases.
- Transaction Pipelining.
- Transaction Tracking.
The Next Chapter

This chapter provides a detailed description of how the processors arbitrate for ownership of the FSB. It includes:

- The Request Phase.
- Logical versus Physical Processors.
- No External Arbiter Required.
- The Rotating ID.
- The Busy/Idle Indicator.
- Requesting Ownership.
- Definition of an Arbitration Event.

Enhanced Mode Scaleable Bus

The FSB implemented on the P6, the Pentium® 4 and the Pentium® M processor families is referred to as the EMSB (Enhanced Mode Scalable Bus). The FSB protocol has been enhanced in a number of ways in making the transition from the P6 to the Pentium® 4.

It should be stressed that the Pentium® 4/M FSB is a derivative of the P6 FSB and, as such, is very similar.

FSB Agents

Agent Types

All devices that reside on the processor's FSB are referred to as agents. Basically, there are three type of agents:

- The Request Agent is the device that initiates a transaction by issuing a transaction request (e.g., a memory read or write, an IO read or write, etc.). It is also referred to as the transaction initiator.
- The Response Agent is the target of the transaction (e.g., an IO target or a memory target).
- The Snoop Agents (aka the snoopers) are any devices on the FSB that have memory caches (usually processors, but, as an example, in addition to the processors there could be an external cache that resides on the FSB). Whenever any initiator starts a transaction, the transaction request is latched by all FSB agents including the snoopers. If it is a memory transaction, the
memory address is then submitted to the snoopers’ caches for a lookup (a
snoop) and the results of the snoop are reported back to the Request Agent
and to the system memory controller. The results will be one of the follow-
ing:
• A **snoop miss**—indicates that none of the snoopers has a copy of the
  addressed line.
• A **snoop hit on a clean line**—indicates that one or more of the snoopers
  has a copy of the addressed line in the E or S state and it hasn’t been
  changed since being read from memory.
• A **snoop hit on a modified line**—indicates that one of the snoopers has
  a copy of the line and one or more of the bytes in the line have been
  written to by the processor core since the line was copied into the cache
  from memory. The line in memory is stale (i.e., it does not contain up-
to-date information).

**Multiple Personalities**

An agent may only be capable of acting as a Response Agent (i.e., as the target
of a transaction). As an example, the system memory controller typically acts as
the target of memory reads and writes. It never initiates transactions, nor does it
ever act as a Snoop Agent in a transaction.

An agent may be capable of acting as the Response Agent in some transactions
and as the Request Agent for other transactions. As an example, in Figure 44-1
on page 1140 the Root Complex may:

• act as the Response Agent (i.e., the target) of a processor-initiated transac-
tion to read data from an IO port in a PCI Express device that resides
beyond the bridge.
• act as the Request Agent of a memory snoop transaction when a device
adapter that resides beneath the Root Complex is writing data to or reading
data from system memory.

An agent may act as the Request Agent for transactions that it initiates and as
the Snoop Agent for memory transactions initiated by others. An example
would be a processor. It not only initiates transactions on an as-needed basis,
but also snoops memory transactions that are initiated by the other processors
or by the Root Complex (on behalf of device adapters).
Uniprocessor vs. Multiprocessor Bus

The FSB utilized on IA32 processors prior to the advent of the P6 processor family was ill-suited in a platform wherein multiple processors reside on the FSB (see Figure 44-1 on page 1140).

The Pentium® Pro FSB was specifically designed to support multiple processors on the same bus, and the Pentium® 4/M FSB is a derivative of the P6 FSB. The following major changes were made:

- In a typical Pentium® 4/M FSB environment, up to 12 transactions can simultaneously be in progress at various stages of completion.
- If the target of a transaction (i.e., the Response Agent) cannot deal with a new transaction right now (e.g., due to a temporary logic busy condition), rather than tie up the bus by inserting wait states, it will issue a Retry response to the initiator. This causes the Request Agent to rearbitrate for ownership of the FSB and retry the transaction again at a later time. This frees up the FSB for other initiators.
Chapter 44: Intro to the Pentium® 4 FSB

- If the target of a read or write transaction determines that it will take a fairly long time to complete the data transfer (i.e., to provide read data or to accept write data), it can issue a Deferred response to the Request Agent. This instructs the Request Agent to terminate the transaction without transferring any data. When the Response Agent has obtained the requested read data or has delivered the write data, it arbitrates for ownership of the FSB and initiates a Deferred Reply transaction to complete the transfer. This is referred to as transaction deferral.

These mechanisms prevent any properly-designed FSB agent from tying up the FSB for extended periods of time. A detailed description of the processor’s FSB is presented in the subsequent chapters of the book.

The Request Agent

The Request Agent Types

There are two types of Request Agents:

- **Symmetric Request Agents**—Most typically, these are the processors. With regard to FSB arbitration, the symmetric Request Agents have equal importance with respect to each other and use a rotational (symmetrical) priority scheme for FSB arbitration. Note that a custom-designed Request Agent other than a processor could be designed to operate as a symmetric agent. The symmetric agent FSB arbitration scheme supports up to but no more than four symmetric Request Agents in the rotation (eight if Hyper-Threading is enabled in four physical processors on the FSB).

- **Priority Request Agents**—The system designer may include one or more Request Agents that are not processors (and that don’t emulate a symmetric FSB agent). If a Priority Agent is competing against the symmetric agents for bus ownership, it wins and they lose (with one exception that is highlighted in a later chapter).

The Agent ID

The Purpose of the Agent ID

When a Request Agent issues a transaction request, two of the items of information that it provides to the addressed Response Agent are:

- The Request Agent’s unique Agent ID.
- A unique transaction ID assigned by the Request Agent.
This chapter provides a detailed description of how the processors arbitrate for ownership of the FSB. It includes:

- The Request Phase.
- Logical versus Physical Processors.
- No External Arbiter Required.
- The Rotating ID.
- The Busy/Idle Indicator.
- Requesting Ownership.
- Definition of an Arbitration Event.

The Next Chapter

This chapter provides a detailed description of how priority agents arbitrate for ownership of the FSB. It includes:

- Priority Agent Arbitration—Despotism.
- Example Priority Agents.
- Priority Agent Beats Symmetric Agents, Unless...
The Unabridged Pentium® 4

- Using Simple Approach, Priority Agent Suffers Penalty.
- Smarter Priority Agent Gets Ownership Faster.
- Ownership Attained in 1 BCLK.
- Ownership Attained in 2 BCLks.
- Be Fair to the Common People.
- Priority Agent Parking.

The Request Phase

There are a number of references to the Request Phase of the transaction in this chapter. After a Request Agent has arbitrated for and won ownership of the Request Phase signal group, it may then initiate a transaction by issuing a transaction request during the Request Phase of the transaction. This consists of the output of two packets of information and the assertion of ADS# (Address Strobe) during the first BCLK cycle of the transaction. For a detailed description of the Request Phase, refer to the chapter entitled “Pentium® 4 FSB Request Phase” on page 1201.

Logical versus Physical Processors

As previously described in “Assignment of IDs to the Processor” on page 860, if Hyper-Threading is enabled a unique Agent ID is assigned to each of the logical processors within each of the physical processors on the trailing-edge of reset. Table 45-1 on page 1150 defines the agent IDs assigned to each of the logical processors in a cluster consisting of four Xeon MP processors.

Table 45-1: Quad Xeon MP System with Hyper-Threading Enabled

<table>
<thead>
<tr>
<th>BR1#</th>
<th>BR2#</th>
<th>BR3#</th>
<th>Physical Processor ID</th>
<th>ID of Logical Processor 0</th>
<th>ID of Logical Processor 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Visit MindShare Training at www.mindshare.com
Chapter 45: Pentium® 4 CPU Arbitration

The Discussion Assumes a Quad Xeon MP System

Unless stated otherwise, the remainder of this chapter assumes that there are four Xeon MP processors on the FSB and all of them have Hyper-Threading enabled.

Symmetric Agent Arbitration—Democracy at Work

A symmetric system is one in which any processor is capable of handling (i.e., executing) any task. The job of the SMP (symmetrical multiprocessing) OS is to attempt to keep all of the processors equally busy at all times (in other words, executing various tasks). At a given instant in time, one or more of the logical processors may require ownership of the Request Phase signal group in order to communicate with an external device. In a well-balanced system, the bus arbitration scheme used to decide which of the processors gets ownership next is based on rotational (symmetric) priority—each of the processors has equal importance.

No External Arbiter Required

Refer to Figure 45-1 on page 1152. The Pentium® 4 processors that make up a cluster (i.e., the group of processors that reside on the FSB) have a built-in rotational priority scheme. No external arbitration logic is necessary to determine which of the logical processors require ownership of the Request Phase signal group and which should acquire ownership next. Each of the physical processors always keeps track of:

- whether any of them currently owns the Request Phase signal group,
- which of them owned the Request Phase signal group last (or still owns it),
- and which of them gets to use it next (assuming any of them are requesting ownership).

In order for them to track this information, each physical processor must know its own Physical Processor ID as well as the ID of the physical processor that last gained ownership of the Request Phase signal group. If a physical processor knows who had ownership last (or still has it), then it knows the physical processor whose turn it is next (because it’s a rotational scheme).
The Arbitration Algorithm

One Arbiter Per Physical Processor

Each physical processor’s FSB Interface Unit contains an arbiter that services requests received from each of the logical processors within the physical processor. The arbiter, in turn, then asserts the physical processor’s BR0# output pin to request ownership of the Request Phase signal group.

When a physical processor acquires ownership of the FSB, it services requests from the two logical processors in round-robin order.

The Rotating ID

As stated earlier, each physical processor must keep track of which of the physical processors was the last to acquire Request Phase signal group ownership. This is referred to as the Rotating ID. When reset is asserted, the Rotating ID is reset to three in all of the logical processors. This means that all of the physical processors believe that physical processor three owned the Request Phase signal group last and therefore physical processor zero should acquire ownership next (if it does in fact request ownership). The sequence in which the physical processors acquire ownership (if all of the physical processors were asking for ownership when reset was deasserted) is 0, 1, 2, 3, 0, etc.

The example just cited assumed a system with four Xeon MPs with or without Hyper-Threading enabled. Table 45-2 on page 1153 provides a list of some additional configurations.
Chapter 45: Pentium® 4 CPU Arbitration

Table 45-2: Some Agent ID Assignment Scenarios

<table>
<thead>
<tr>
<th>Processor Type</th>
<th>Hyper-Threading Enabled?</th>
<th>Number of Physical Processors</th>
<th>Logical Processor ID Assignments</th>
<th>Initial Rotating ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon MP</td>
<td>N</td>
<td>4</td>
<td>0, 1, 2, and 3.</td>
<td>3</td>
</tr>
<tr>
<td>Xeon MP</td>
<td>Y</td>
<td>4</td>
<td>0, 1, 2, 3, 4, 5, 6, and 7.</td>
<td>3</td>
</tr>
<tr>
<td>Xeon DP</td>
<td>N</td>
<td>2</td>
<td>0 and 1.</td>
<td>1</td>
</tr>
<tr>
<td>Xeon DP</td>
<td>Y</td>
<td>2</td>
<td>0, 1, 2 and 3.</td>
<td>1</td>
</tr>
<tr>
<td>Extreme Edition</td>
<td>N</td>
<td>1</td>
<td>0.</td>
<td>0</td>
</tr>
<tr>
<td>Extreme Edition</td>
<td>Y</td>
<td>1</td>
<td>0 and 1.</td>
<td>0</td>
</tr>
<tr>
<td>Desktop Pentium® 4</td>
<td>N</td>
<td>1</td>
<td>0.</td>
<td>0</td>
</tr>
<tr>
<td>Desktop Pentium® 4</td>
<td>Y</td>
<td>1</td>
<td>0 and 1.</td>
<td>0</td>
</tr>
<tr>
<td>Celeron</td>
<td>N</td>
<td>1</td>
<td>0.</td>
<td>0</td>
</tr>
<tr>
<td>Celeron</td>
<td>Y</td>
<td>1</td>
<td>0 and 1.</td>
<td>0</td>
</tr>
</tbody>
</table>

The Busy/Idle Indicator

**General.** Refer to Figure 45-2 on page 1154. In addition to the Rotating ID maintained by each of the physical processors, the arbiter within each of the physical processors must also keep track of whether the last physical processor that acquired ownership of the Request Phase signal group retained ownership or has released it (and therefore none of them currently owns it). When the last to acquire ownership retains ownership, the ownership state is said to be *Busy*. If the previous owner surrendered ownership and none of them currently owns the Request Phase signal group, the ownership state is said to be *Idle*. Each of the physical processors maintains an internal Busy/Idle state indicator to indicate whether the Request Phase signal group ownership state is currently Busy or Idle.
The Previous Chapter
This chapter provided a detailed description of how the processors arbitrate for ownership of the FSB. It included:

- The Request Phase.
- Logical versus Physical Processors.
- No External Arbiter Required.
- The Rotating ID.
- The Busy/Idle Indicator.
- Requesting Ownership.
- Definition of an Arbitration Event.

This Chapter
This chapter provides a detailed description of how priority agents arbitrate for ownership of the FSB. It includes:

- Priority Agent Arbitration—Despotism.
- Example Priority Agents.
- Priority Agent Beats Symmetric Agents, Unless...
- Using Simple Approach, Priority Agent Suffers Penalty.
- Smarter Priority Agent Gets Ownership Faster.
- Ownership Attained in 1 BCLK.
- Ownership Attained in 2 BCLKs.
- Be Fair to the Common People.
- Priority Agent Parking.
This chapter describes the FSB locking mechanism, the reason for its existence, and the instructions that invoke it. It includes:

- The Shared Resource Concept.
- Testing the Availability of and Gaining Ownership of Shared Resources.
- A Race Condition Can Present a Problem.
- Guaranteeing the Atomicity of a Read/Modify/Write.
- Locking a Cache Line.

Priority Agent Arbitration

Example Priority Agents

While the physical processors are very polite to each other, the system may include one or more agents that play by different rules. They are referred to as Priority Agents. In Figure 46-1 on page 1167, whenever a PCI Express device adapter initiates a read from or write to system memory, the Root Complex must arbitrate for ownership of the FSB to initiate a snoop transaction. To do so, it uses the BPRI# signal (Bus Priority agent request; note that there is only one BPRI# signal). The Root Complex acts as the surrogate FSB Request Agent when a PCI Express device adapter requires access to system memory. BPRI# is an input to each processor’s arbiter and, when asserted, it informs the processors that the Priority Agent would like to break into the rotation in order to initiate a transaction.

Figure 46-2 on page 1167 illustrates two Xeon MP clusters interconnected via an Cluster Bridge. When a processor on one FSB initiates an access to cacheable memory on the other FSB, the Cluster Bridge must arbitrate for ownership of the other FSB and it does so by asserting BPRI# to the array of processors on the target FSB.

Only one device is permitted to assert BPRI# at a time. In the case where multiple Priority Agents reside on the FSB, there must therefore be some method for the Priority Agents to arbitrate amongst themselves to determine which of them gets to use BPRI# to request ownership (if more than one of them needs to issue a transaction request at the same time).
Chapter 46: Pentium® 4 Priority Agent Arbitration

Figure 46-1: System Block Diagram 1

PCI Express Server System

Figure 46-2: System Block Diagram 2

Visit MindShare Training at www.mindshare.com
Priority Agent Beats Symmetric Agents, Unless...

When a Priority Agent is requesting ownership at the same time that one or more of the symmetric agents are also requesting ownership, the Priority Agent normally wins.

The only case where the Priority Agent will be unsuccessful in winning ownership of the FSB is the case where a physical processor has already acquired ownership and has asserted the LOCK# signal (because it has initiated a locked transaction series when performing a locked read/modify/write operation). This prevents the Priority Agent from acquiring ownership until the physical processor completes the locked transaction series and deasserts the LOCK# signal. The reasons why a symmetric agent might assert LOCK# are covered in the section entitled “Pentium® 4 Locked Transaction Series” on page 1177. The Priority Agent must deal with the cases described in Table 46-1 on page 1168.

Table 46-1: Possible Priority Agent Arbitration Scenarios

<table>
<thead>
<tr>
<th>Case</th>
<th>Resulting Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A physical processor initiates a transaction request in the same clock that the Priority Agent asserts BPRI#, but does not assert LOCK#</td>
<td>In this case, the Priority Agent assumes ownership after the physical processor finishes delivery of its transaction request. This will be 3 clocks after BPRI# assertion.</td>
</tr>
<tr>
<td>A physical processor initiates a transaction request and asserts LOCK# in the same clock that the Priority Agent is asserting BPRI#</td>
<td>The Priority Agent cannot assume ownership until the physical processor deasserts LOCK#.</td>
</tr>
</tbody>
</table>
| A physical processor has acquired ownership on the same rising-edge of the clock that BPRI# is sampled asserted. In this case, the physical processor proceeds with its transaction request and may or may not assert LOCK# | • If LOCK# is asserted, the Priority Agent doesn’t acquire ownership until LOCK# is deasserted by the physical processor.  
• If LOCK# isn’t asserted, the Priority Agent acquires ownership as soon as the physical processor completes issuing its transaction request. This will be 2 clocks after BPRI# is asserted. |
Using Simple Approach, Priority Agent Suffers Penalty

Refer to Figure 46-3 on page 1171. A Priority Agent may be designed in such a manner that it doesn’t check to see if a physical processor has started a transaction request (in other words, it doesn’t check the state of the ADS# signal) in order to determine when (and if) it can take ownership of the Request Phase signal group. Rather, it checks in the two clocks immediately following its assertion of BPRI# to see if LOCK# is asserted. If LOCK# is sampled asserted on the rising-edge of BCLK0 in either of the two clocks immediately after BPRI# is asserted, then a physical processor had already asserted LOCK# and the Priority Agent can’t take ownership until LOCK# is deasserted. If LOCK# is sampled deasserted during both of these two clocks, however, one of three conditions is true (but the Priority Agent doesn’t know which):

1. No physical processor has initiated a transaction request during these two clocks and LOCK# is not being held asserted by a physical processor that issued an earlier transaction request.
2. A physical processor started a transaction request on the same clock that BPRI# was driven asserted, but it did not assert LOCK#.
3. A physical processor started a transaction request on the clock after BPRI# was asserted, but it did not assert LOCK#.

In any of these cases, the Priority Agent has gained ownership. However, because it doesn’t check ADS# to determine which of the three cases is true, it must assume the worst case—case number 3. In this case, the physical processor has sampled BPRI# asserted at the start of the clock in which the physical processor initiated its transaction request. It does not assert LOCK#, and it will therefore honor the BPRI# assertion. The Priority Agent cannot assume ownership, however, until 3 clocks after the physical processor starts its transaction request. This is a total of 3 clocks after BPRI# is asserted.

1. An arbitration event occurs on clock 2 in Figure 46-3 on page 1171 and physical processor 0 acquires ownership in clock 3 (BPRI# was not asserted at the start of clock 2, so physical processor 0 is not prevented from taking ownership). Also at the start of clock 2, the Priority Agent asserts BPRI# to request ownership, but this isn’t detected by physical processor 0 until clock 3, the clock in which it starts to drive out a transaction request. This means that physical processor 0 has successfully acquired ownership and will proceed with the issuance of its transaction request.
The Previous Chapter

This chapter provided a detailed description of how priority agents arbitrate for ownership of the FSB. It included:

- Priority Agent Arbitration—Despotism.
- Example Priority Agents.
- Priority Agent Beats Symmetric Agents, Unless...
- Using Simple Approach, Priority Agent Suffers Penalty.
- Smarter Priority Agent Gets Ownership Faster.
- Ownership Attained in 1 BCLK.
- Ownership Attained in 2 BCLKs.
- Be Fair to the Common People.
- Priority Agent Parking.

This Chapter

This chapter describes the FSB locking mechanism, the reason for its existence, and the instructions that invoke it. It includes:

- The Shared Resource Concept.
- Testing the Availability of and Gaining Ownership of Shared Resources.
- A Race Condition Can Present a Problem.
- Guaranteeing the Atomicity of a Read/Modify/Write.
- Locking a Cache Line.
The Unabridged Pentium® 4

The Next Chapter

This chapter describes the mechanism that permits FSB agents to limit the number of transactions that can be injected into the FSB. It includes:

- Assert BNR# When One Entry Remains.
- BNR# Can Be Used by a Debug Tool.
- Who Monitors BNR#.
- BNR# is a Shared Signal.
- The Stalled/Throttled/Free Indicator.
- Initial Entry to the Stalled State.
- The Throttled State.
- The Free State.
- As an Agent Approaches Full, It Signals BNR# to Stall Everyone.
- BNR# Behavior at Powerup.
- BNR# Behavior During Runtime.

Introduction

The previous chapter, “Pentium® 4 Priority Agent Arbitration” on page 1165, described how the assertion of LOCK# by a symmetric agent prevents a Priority Agent from acquiring ownership. This section describes the reasons why a symmetric agent might need to perform a series of transactions without fear of any other agent performing an access in between its own transactions.

The Shared Resource Concept

Assume that the OS sets aside an area of memory to be used by tasks executing on multiple processors (or even by different tasks executed by the same processor) as a shared memory buffer. It is intended to be used as follows:

1. Before using the buffer (i.e., reading from or writing to it), a task must first test a memory-based flag to ensure that the buffer isn’t currently owned by another task. If the buffer is currently unavailable, the task wishing to gain ownership should periodically check back to see when it becomes available.
2. When the flag indicates that the buffer is available, the task sets the flag, indicating that it has exclusive ownership of the buffer. The buffer is then unavailable if any other task should attempt to gain ownership of it.
3. Having gained exclusive ownership of the buffer, the task can now read and write the buffer.
4. If the buffer is in an area of memory designated as WT, WC, WP, or UC memory (refer to “Store µops” on page 1072), writes are absorbed into the
processor’s Store Buffers. These buffers are not snooped when other agents access memory. In this case, when the task is done using the buffer, it should ensure that all of its updates (i.e., memory writes) have been flushed all the way to memory.

5. After ensuring that the buffer has received all updates, the task should release ownership of the buffer so it can be used by other tasks.

Testing the Availability of and Gaining Ownership of Shared Resources

The OS typically uses a memory location (or series of memory locations) as the flag (see the previous section) indicating the availability or unavailability of a particular shared resource. This is referred to as a memory semaphore.

A Race Condition Can Present a Problem

Consider the following possibility:

1. The task executing on processor 0 reads the semaphore to determine the buffer’s availability.
2. The task tests the semaphore’s value and determines that the buffer is available (the semaphore value = zero).
3. Immediately after the task on processor 0 has completed the memory read to obtain and test the state of the semaphore value, a task executing on processor 1 has initiated a memory read request on the FSB to test the state of the same semaphore. It completes the read and begins testing the value.
4. The processor 0 task initiates a memory write on the FSB to update the semaphore to a non-zero value to mark the shared buffer as unavailable. After it completes the write, it considers itself the sole owner of the buffer.
5. The processor 1 task also determined the buffer is available and it now performs a memory write on the FSB to update the semaphore to a non-zero value to mark the shared buffer as unavailable. It completes the write and it also now considers itself the sole owner of the buffer.

Two tasks executing on two separate processors now each believe that they have exclusive ownership of the buffer.
Guaranteeing the Atomicity of a Read/Modify/Write

This problem came about because processor 1 was able to read the semaphore immediately after processor 0 read it. The two processors were in a race condition. Processor 0 then wrote to it immediately, followed by processor 1 writing to it. The tasks on the two processors each ended up believing it had sole ownership of the buffer.

This problem can be prevented if processor 0 could prevent other initiators from using the FSB from the time it initiates its read until the time it completes the write to update the semaphore to a non-zero value. In other words, it should lock the FSB while it performs the read/modify/write (frequently referred to as a RMW) of the semaphore.

To do this, the programmer uses one of several special instructions to perform the RMW operation. When using these instructions, the processor (refer to Figure 47-1 on page 1181) takes the following actions:

1. The processor asserts the LOCK# signal when it initiates the memory read, keeps LOCK# asserted while it performs the internal semaphore test and then performs the memory write to update the semaphore before releasing the LOCK# signal. The assertion of LOCK# prevents any Priority Agent from obtaining FSB ownership during this period.

2. The processor also keeps its BR0# output asserted throughout this period to keep any of the other processors from obtaining ownership of the Request Phase signal group.
Chapter 47: Pentium® 4 Locked Transaction Series

Figure 47-1: Example of Symmetric Agent Performing Locked Transaction Series
The Previous Chapter
This chapter described the FSB locking mechanism, the reason for its existence, and the instructions that invoke it. It included:

- The Shared Resource Concept.
- Testing the Availability of and Gaining Ownership of Shared Resources.
- A Race Condition Can Present a Problem.
- Guaranteeing the Atomicity of a Read/Modify/Write.
- Locking a Cache Line.

This Chapter
This chapter describes the mechanism that permits FSB agents to limit the number of transactions that can be injected into the FSB. It includes:

- Assert BNR# When One Entry Remains.
- BNR# Can Be Used by a Debug Tool.
- Who Monitors BNR#.
- BNR# is a Shared Signal.
- The Stalled/Throttled/Free Indicator.
- Initial Entry to the Stalled State.
- The Throttled State.
- The Free State.
- As an Agent Approaches Full, It Signals BNR# to Stall Everyone.
- BNR# Behavior at Powerup.
- BNR# Behavior During Runtime.
The Next Chapter

This chapter provides a detailed description of the Request Phase of a FSB transaction. It includes:

- Introduction to the Request Phase.
- The Source Synchronous Strobes.
- The Request Phase Parity.
- Request Phase Parity Checking.
- ChipSet Request Phase Parity Checking and Reporting.
- Processor Request Phase Parity Checking and Reporting.
- The Request Phase Signal Group is Multiplexed.
- Introduction to the Transaction Types.
- The Contents of Request Packet A.
- 32-bit vs. 36-bit Addresses.
- The Contents of Request Packet B.

Blocking New Requests—Stop! I'm Full!

The section entitled “Transaction Tracking” on page 1147 introduced the concept of transaction tracking and the IOQ (In-Order Queue). Each agent has an IOQ that it uses to keep track of each transaction that is currently outstanding on the FSB. The depth of an agent’s IOQ is device-specific. The P6 processors had a selectable queue depth of either one or eight. The Pentium® 4 processors have a selectable queue depth of one or 12. The queue depths of the various North Bridges, MCHs, or Root Complexes are design-specific. Their queue depth will be either \( \leq \) the processor’s queue depth.

Assert BNR# When One Entry Remains

Refer to Figure 48-1 on page 1191. When the maximum number of transactions (minus one) that a device can track are currently outstanding on the FSB at various stages of completion, the agent cannot permit any other agent to initiate a new transaction. If a new transaction were initiated, the agent would be incapable of tracking it and consequently would lose track of all activity on the FSB.

For this reason, agents must have the ability to throttle the ability of other agents to initiate new transactions. That is the purpose of the BNR# (Block Next Request) signal. An agent must assert BNR# when its In Order Queue (IOQ) has one entry remaining empty. This is necessary because a new transaction request
could be issued by another agent at the same time that an agent begins to assert BNR#. The one entry that is remaining can then be used to latch and track the newly-issued transaction. There is no danger that another transaction will be issued to the FSB because all agents have detected BNR# by this time.

**BNR# Can Be Used by a Debug Tool**

BNR# could also be used by a debug tool to create a controlled situation where no additional transactions can be issued to the FSB until the current transaction has been completed. In other words, transactions could be single-stepped onto the bus to simplify the debug process.

*Figure 48-1: Don’t Wait Until It’s Too Late!*
Who Monitors BNR#?

Refer to Figure 48-2 on page 1192. All FSB agents that are waiting to issue new transactions must monitor the state of BNR#.

*Figure 48-2: Who Monitors BNR#?*

BNR# is a Shared Signal

BNR# is a shared, open-drain signal because multiple FSB agents may assert it simultaneously to indicate that they are not ready to deal with a new transaction.

The Stalled/Throttled/Free Indicator

Each FSB agent that is capable of initiating transactions must maintain an internal indicator referred to as the Stalled/Throttled/Free indicator.
Initial Entry to the Stalled State

Refer to Figure 48-3 on page 1193. At power-up or when reset is asserted, all initiators (i.e., Request Agents) reset the indicator to the Stalled State. All Request Agents are required to start sampling BNR# on a periodic basis starting soon after reset is removed (see “BNR# Behavior at Powerup” on page 1197) and must remain in the Stalled State until BNR# is sampled deasserted. This prevents any agent from issuing a transaction until BNR# is sampled deasserted, indicating that all agents are prepared to deal with a new transaction. The following are some example situations wherein a FSB agent might continually assert BNR# for some period of time after reset is removed:

- If a processor has been instructed to execute its BIST [refer to “Built-In Self-Test (BIST) Trigger” on page 858], it cannot observe the FSB and track FSB activity while the BIST is executing. For this reason, the processor continually toggles BNR# after reset is removed until its BIST has been completed.
- A FSB agent (e.g., the North Bridge, MCH, or Root Complex) might require a period of time after the removal of reset to initialize its internal logic before it is ready to track FSB activity generated by other FSB agents. For this reason, the device could continually assert BNR# after reset is removed until it has completed its internal initialization. It then stops asserting BNR#.

Figure 48-3: The Stalled State

Definition of the Stalled state:

While the indicator is in the Stalled state, no request agents may issue new transactions to the bus.
The Previous Chapter
This chapter described the mechanism that permits FSB agents to limit the number of transactions that can be injected into the FSB. It included:

- Assert BNR# When One Entry Remains.
- BNR# Can Be Used by a Debug Tool.
- Who Monitors BNR#.
- BNR# is a Shared Signal.
- The Stalled/Throttled/Free Indicator.
- Initial Entry to the Stalled State.
- The Throttled State.
- The Free State.
- As an Agent Approaches Full, It Signals BNR# to Stall Everyone.
- BNR# Behavior at Powerup.
- BNR# Behavior During Runtime.

This Chapter
This chapter provides a detailed description of the Request Phase of a FSB transaction. It includes:

- Introduction to the Request Phase.
- The Source Synchronous Strobes.
- The Request Phase Parity.
- Request Phase Parity Checking.
- ChipSet Request Phase Parity Checking and Reporting.
- Processor Request Phase Parity Checking and Reporting.
- The Request Phase Signal Group is Multiplexed.
- Introduction to the Transaction Types.
- The Contents of Request Packet A.
- 32-bit vs. 36-bit Addresses.
- The Contents of Request Packet B.
This chapter provides a detailed description of the Snoop Phase of a FSB transaction. It includes:

- Agents Involved in the Snoop Phase.
- The Snoop Phase Has Two Purposes.
- The Snoop Result Signals are Shared, DEFER# Isn’t.
- The Snoop Phase Duration Is Variable.
- There Is No Snoop Stall Duration Limit.
- Memory Transaction Snooping.
- The Snoop’s Effects on Processor Caches.
- Self-Snooping.
- Non-Memory Transactions Have a Snoop Phase.

Cautionary Note

Unless noted otherwise, the representation of all signal states in tables is in logical, not electrical format. As an example, the first row in Table 49-5 on page 1216 shows a 00000b on REQ[4:0]#, indicating a Deferred Reply transaction type. This means that REQ[4:0]# are deasserted (electrical ones) when driven onto REQ[4:0]#.

Introduction to the Request Phase

Once ownership of the Request Phase signal group has been acquired (see “Pentium® 4 CPU Arbitration” on page 1149 and “Pentium® 4 Priority Agent Arbitration” on page 1165), the Request Agent uses the Request Phase signal group to broadcast the transaction request. This includes the address and transaction type, as well as additional information about the transaction. The Request Phase signal group consists of the signals introduced in Table 49-1 on page 1202.

Table 49-1: The Request Phase Signal Group

<table>
<thead>
<tr>
<th>Signal(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[35:3]#</td>
<td>These signals are used to output the address as well as additional information about the transaction.</td>
</tr>
<tr>
<td>AP[1:0]#</td>
<td>The Address/Request parity bits.</td>
</tr>
</tbody>
</table>
Chapter 49: Pentium® 4 FSB Request Phase

Table 49-1: The Request Phase Signal Group (Continued)

<table>
<thead>
<tr>
<th>Signal(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ[4:0]#</td>
<td>The Request Type bus is used to output the transaction type (e.g., a Memory Data Read), as well as additional information about the transaction.</td>
</tr>
<tr>
<td>ADS#</td>
<td>The Address Strobe signal is asserted to indicate that a new transaction has been initiated.</td>
</tr>
<tr>
<td>ADSTB[1:0]#</td>
<td>The source-synchronous strobes that the Request Agent drives along with the request. The input receiver within each FSB agent uses the falling- and rising-edges of the strobes to latch the address and the request type.</td>
</tr>
</tbody>
</table>

The Request Phase is one BCLK in duration during which the information describing the transaction is output in two packets (see Figure 49-1 on page 1205) and ADS# is asserted. The assertion of ADS# indicates that a new transaction request is being issued. The packets are referred to as Packets A and B and both of them are latched by all FSB agents (not just by Response Agents).

As discussed earlier, all FSB agents must track the transaction as it passes through each phase from inception to completion. In addition, if it is a memory transaction, Snoop Agents (typically, processors with internal caches) must submit the transaction’s memory address to their caches for a lookup and must deliver the snoop result during the transaction’s Snoop Phase. It should be noted that all of the information necessary for the snoop are output in Packet A (i.e., the memory address and the transaction type).

All of the Response Agents on the FSB must decode the address and transaction type to determine which of them is the target of the transaction.

The Source Synchronous Strobes

The Request Agent starts driving a transaction request on the rising-edge of BCLK0. In Figure 49-1 on page 1205, a request is issued at the start of BCLK cycle 1 and another at the start of BCLK cycle 3. The signals that comprise the Request Phase signal group are divided into subgroups on the system board:

- The Address Strobe 0 signal trace is routed with the A[16:3]# and REQ[4:0]# signal traces.
- The Address Strobe 1 signal trace is routed with the A[35:17]# signal traces.
The Unabridged Pentium® 4

1. In the first half of the BCLK cycle, the Request Agent:
   — drives out address bits [35:3] on A[35:3]#,
   — drives out the transaction type on REQ[4:0]#,
   — drives both of the Address Strobe signals (ADSTB[1:0]) low.
   The information on A[35:3]# and REQ[4:0]# comprises request Packet A.
2. All of the FSB agents use the falling-edge of the two Address Strobes to latch Packet A into their input receivers. At this point, processors on the FSB have all of the information necessary to determine whether or not it is a memory transaction (i.e., the memory address and the transaction type) and, if it is, they initiate a snoop in their internal caches. The snoop result will be delivered when the transaction has entered its Snoop Phase.
3. In the second half of the BCLK cycle, the Request Agent:
   — drives out additional transaction information on A[35:3]#,
   — drives out additional transaction information on REQ[4:0]#,
   — drives both of the Address Strobe signals (ADSTB[1:0]) high.
4. The information on A[35:3]# and REQ[4:0]# comprises request Packet B.
5. All of the FSB agents use the rising-edge of the two Address Strobes to latch Packet B into their input receivers.

The Request Phase Parity

The Request Agent drives the two Request Phase parity bits, AP[1:0]#, at the start of the BCLK cycle immediately following the Request Phase:

- It drives AP0# to either an electrical high or low to force an even number of electrical lows in the overall pattern consisting of A[35:24]# in Packet A, and A[23:3]# and REQ[4:0]# in Packet B.
- It drives AP1# to either an electrical high or low to force an even number of electrical lows in the overall pattern consisting of A[23:3]# and REQ[4:0]# in Packet A, and A[35:24]# in Packet B.
Chapter 49: Pentium® 4 FSB Request Phase

Request Phase Parity Checking

ChipSet Request Phase Parity Checking and Reporting

Refer to Figure 49-2 on page 1207. The example system shown is a PCI Express-based system and the device that connects the processors to the remainder of the system is referred to as the Root Complex. In a PCI or a PCI-X based system, it is referred to as the North Bridge or as the Memory Control Hub (MCH). The device that connects the processors to the remainder of the system is part of the chipset.

When a FSB agent other than the chipset (e.g., a processor) initiates a transaction on the FSB, the chipset may or may not check the Request Phase parity bits for correctness. Many chipsets designed for low- and medium-range systems do...
50 Pentium® 4 FSB

Snoop Phase

The Previous Chapter
This chapter provides a detailed description of the Request Phase of a FSB transaction. It includes:

- Introduction to the Request Phase.
- The Source Synchronous Strobes.
- The Request Phase Parity.
- Request Phase Parity Checking.
- ChipSet Request Phase Parity Checking and Reporting.
- Processor Request Phase Parity Checking and Reporting.
- The Request Phase Signal Group is Multiplexed.
- Introduction to the Transaction Types.
- The Contents of Request Packet A.
- 32-bit vs. 36-bit Addresses.
- The Contents of Request Packet B.

This Chapter
This chapter provides a detailed description of the Snoop Phase of a FSB transaction. It includes:

- Agents Involved in the Snoop Phase.
- The Snoop Phase Has Two Purposes.
- The Snoop Result Signals are Shared, DEFER# Isn’t.
- The Snoop Phase Duration Is Variable.
- There Is No Snoop Stall Duration Limit.
- Memory Transaction Snooping.
- The Snoop’s Effects on Processor Caches.
- Self-Snooping.
- Non-Memory Transactions Have a Snoop Phase.
The Next Chapter

This chapter provides a detailed description of the Response and Data Phases of a FSB transaction. It includes:

- The Purpose of the Response Phase.
- The Response Phase Signal Group.
- The Response Phase Start Point.
- The Response Phase End Point.
- The Response Types.
- The Response Phase May Complete a Transaction.
- The Data Phase Signal Group.
- Five Example Scenarios.
- Data Phase Wait States.
- The Response Phase Parity.
- Data Bus Parity.

Agents Involved in the Snoop Phase

Refer to Figure 50-1 on page 1227. The following agents are involved in the Snoop Phase of the transaction:

- The **Request Agent** issues the transaction request. This can be one of the processors or the chipset. If the transaction is a memory transaction, the Request Agent checks the snoop result presented in the Snoop Phase.

- The **Snoop Agents** are the processors. They latch the transaction and, if it’s a memory transaction, submit the memory address to their internal caches for a lookup. They present the snoop result to the Request Agent and to the system memory controller (located in the Root Complex, North Bridge or MCH). If the snoop resulted in a hit on a modified line in a processor’s cache, that Snoop Agent (i.e., processor) supplies the modified line in the Data Phase (see the next bullet item). If it’s a non-memory transaction, the processors do not snoop the transaction in their caches.

- The **Response Agent** is the currently-addressed target. This could be the system memory controller, the configuration registers or IO ports within the chipset, a target residing on one of the PCI Express links, or a target residing on another bus in the system (e.g., a PCI or PCI-X bus). If the Response Agent is the system memory controller, it must observe the snoop response presented by the Snoop Agents:
  - If the access results in a miss on all of the caches, or in a hit on a clean line (i.e., a line that is in the E or S state), the system memory controller supplies the read data or, if a write, accepts the write data presented by the Request Agent.
— If the transaction is a read that hits on a modified line, the Snoop Agent with the modified copy of the line supplies the modified line directly to the Request Agent and also to the system memory controller. The transaction started out as a read from system memory’s perspective, but the hit on a modified line cancels the read from system memory. Instead, the system memory controller accepts the modified line that the Snoop Agent supplies to the Request Agent and uses it to update the stale line in memory.

— Refer to Figure 50-2 on page 1228. If the transaction is a write (performed by a device adapter) that hits on a modified line in a processor’s cache (a Snoop Agent), the memory controller accepts the write data from the Request Agent (i.e., the Bridge), then accepts the modified line from the Snoop Agent (the processor), and finally merges the write data into the modified line and writes the updated line into memory.

Figure 50-1: System Block Diagram
The Snoop Phase Has Two Purposes

In the Snoop Phase, the Request Agent samples the snoop result signals to determine two things:

1. If the currently-addressed Response Agent (i.e., the target) intends to complete the transaction now or it intends to issue a Retry or a Deferred response when the transaction reaches its Response Phase.
2. If the transaction is a memory read or write that the Response Agent will complete now (i.e., it doesn’t intend to defer its completion), does any other cache have a copy of the line and, if so, in what state will its line be at the completion of the transaction (clean or modified)?

The reader should also remember that the snoop signals are being sampled by all FSB agents so as to remain synchronized with the state of the FSB.
The Snoop Result Signals are Shared, DEFER# Isn’t

There are three snoop result signals (divided into two groups) that are sampled by the Request Agent during the Snoop Phase:

- HIT# and HITM# are the signals used by the Snoop Agents (i.e., the processor caches) to deliver the cache snoop result, or to stall the completion of the Snoop Phase if one or more of the Snoop Agents isn’t ready to deliver the snoop result (see “Line Containing a Semaphore Is in the E or M State” on page 1185 for an example of snoop stall). Both HIT# and HITM# might be driven by multiple snoopers if any of them need to stall the completion of the Snoop Phase until the snoop result is available. At that time, each of the snoopers that have been stalling would stop driving both lines and either assert neither of them (if it’s a miss), just the HIT# signal (if it’s a hit on an E or S line), or just the HITM# signal (if it’s a hit on a modified line). HIT# and HITM# are shared, open-drain signals that may be driven by more than one device at a time.

- Only the currently-addressed Response Agent (i.e., the target) is permitted to assert the DEFER# signal during the Snoop Phase, so it is not a shared, open-drain signal. The Response Agent only asserts DEFER# if it intends to issue a Retry or a Deferred response to the Request Agent when the transaction enters the Response Phase (it would more correctly be called the Defe or Retry signal). These topics are discussed towards the end of this chapter.

It would be a violation of the MESI protocol for one or more processors to assert HIT# while another asserts HITM# indicating it has a modified copy of a line. In order for a processor to update a line and mark it modified, it must first acquire exclusive ownership of that line. If the line is currently marked Shared in its cache, before updating the line it must first perform a “kill” transaction (i.e., an MRI for 0 bytes) on its FSB to invalidate the line in the caches of all other processors. Then and only then can it store into the line and mark it Modified.

The Snoop Phase Duration Is Variable

Refer to Figure 50-3 on page 1232. The Snoop Phase begins immediately after the Request Phase completes (clocks 2, 4, and 6) and completes when a valid snoop result (something other than both HIT# and HITM# asserted) is presented to the Request and Response Agents by the Snoop Agents. Table 50-1 on page 1232 defines the meaning of the various snoop results. The following provides a clock-by-clock description of Figure 50-3 on page 1232 (an example of three back-to-back memory transactions):
The Previous Chapter

This chapter provided a detailed description of the Snoop Phase of a FSB transaction. It included:

- Agents Involved in the Snoop Phase.
- The Snoop Phase Has Two Purposes.
- The Snoop Result Signals are Shared, DEFER# Isn’t.
- The Snoop Phase Duration Is Variable.
- There Is No Snoop Stall Duration Limit.
- Memory Transaction Snooping.
- The Snoop’s Effects on Processor Caches.
- Self-Snooping.
- Non-Memory Transactions Have a Snoop Phase.

This Chapter

This chapter provides a detailed description of the Response and Data Phases of a FSB transaction. It includes:

- The Purpose of the Response Phase.
- The Response Phase Signal Group.
- The Response Phase Start Point.
- The Response Phase End Point.
- The Response Types.
- The Response Phase May Complete a Transaction.
- The Data Phase Signal Group.
- Five Example Scenarios.
The Unabridged Pentium® 4

- Data Phase Wait States.
- The Response Phase Parity.
- Data Bus Parity.

The Next Chapter

This chapter provides a detailed description of the Deferred Transaction mechanism. It includes:

- The Problem.
- Example Read From a PCI Express Device.
- The Read Receives the Deferred Response.
- The Root Complex Performs the Read.
- The Root Complex Issues a Deferred Reply Transaction.
- Example Write To a PCI Express Device.
- The Write Receives the Defefer Response.
- The Root Complex Delivers the Write Data to the Target.
- The Root Complex Issues a Deferred Reply Transaction.

A Note on Deferred Transactions

Please note that a detailed description of deferred transactions can be found in the chapter entitled “Pentium® 4 FSB Transaction Deferral” on page 1277.

The Purpose of the Response Phase

The possible responses that the Response Agent may supply in the transaction’s Response Phase are:

- The Response Agent may command the Request Agent to retry the transaction repeatedly until it succeeds (or fails). The Response Agent can’t service the request now, but will be able to later.
- The Response Agent may inform the Request Agent that it will defer completion of the transaction until a later time. The Response Agent will service the request (read or write) off-line and will deliver the results to the Request Agent in a subsequent Deferred Reply transaction.
- The Response Agent may indicate a hard failure to the Request Agent. The Response Agent is broken and can’t service the request at all.
- If the transaction is one that doesn’t require the Response Agent to send data to the Request Agent (i.e., it is a write transaction, a Special transaction,
a Memory Read, or Memory Read and Invalidate transaction for 0 bytes),
the Response Agent indicates that, as requested, no data will be returned to
the Request Agent.
• If the transaction is a memory read or write that results in a hit on a modi-
fied line in the Snoop Phase, the Response Agent indicates that the Snoop
Agent will transfer the entire modified line to memory (referred to as an
implicit writeback operation) in the Data Phase of the transaction (and, if it’s
a read transaction, to the Request Agent at the same time).
• If the transaction is any form of a read transaction (i.e., a Memory Read, a
Memory Read and Invalidate for 64 bytes, an IO Read, or an Interrupt
Acknowledge), the Response Agent indicates that it will return the
requested data in the Data Phase (alternatively, it may choose to defer deliv-
ery of the read data until a later time). This is referred to as the normal data
response.

The Response Phase Signal Group

The following signals are used in the Response Phase:

• RS[2:0]#. The Response Bus. This 3-bit bus is used to deliver the response
to the Request Agent.
• RSP#. The Response Bus parity bit. This is the parity signal that covers
RS[2:0]#. It is an even parity signal that is driven low or high to force an
even number of electrical lows in the overall 4-bit pattern that includes
RS[2:0]# and RSP#.
• TRDY# (Target Ready). TRDY# is only asserted by the Response Agent if
data is to be written to it by either the Request Agent, a Snoop Agent (i.e.,
there will be an implicit writeback of a modified line due to the assertion of
HITM#), or both. The assertion of TRDY# indicates the Response Agent’s
readiness to accept the write data.

The Response Phase Start Point

The Response Phase starts immediately after the Snoop Phase completes.

The Response Phase End Point

The Response Phase ends when the Response Agent delivers a valid response to
the Request Agent. This implies that the Response Agent can stall the Response
Phase (i.e., insert wait states) until it is ready to present its response. One BCLK
cycle after entry to the Response Phase, the Request Agent starts sampling
RS[2:0]# at the start of each BCLK cycle. As long as the Idle response is detected, the Request Agent continues sampling RS[2:0]# at the start of each BCLK cycle until a response other than the Idle response is detected. Detection of a non-Idle response completes the transaction’s Response Phase.

The spec doesn’t place a limit on the number of wait states that may be inserted into the Response Phase of a transaction. However, the system designer may choose to monitor the behavior of agents to ensure that none of them inserts excessive wait states. This would adversely affect all subsequently-issued transactions that are awaiting delivery of their respective Responses.

### The Response Types

Table 51-1 on page 1244 lists the possible responses that can be presented on RS[2:0]#.

<table>
<thead>
<tr>
<th>RS[2:0]#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000b</td>
<td><strong>Idle Response.</strong> RS[2:0]# are deasserted. This is the state of RS[2:0]# before and after the response has been delivered to the Request Agent. In other words, immediately upon entry into the Response Phase, RS[2:0]# are in this state and will remain in this state until a valid response is presented. When any of the valid (i.e., non-idle) responses are driven (for one clock), one or more of the RS[2:0]# signals are driven low. All of the valid response patterns have at least one of the RS signals asserted (remember, in this table, a 0 = deasserted = electrically high). After one clock, the response is removed. The RS signals are then returned to the deasserted state (in other words, back to the Idle state).</td>
</tr>
</tbody>
</table>
| 001b     | **Retry Response.** The Response Agent is commanding the Request Agent to retry the transaction repeatedly until the transaction succeeds (or fails). The Response Agent can’t service the request now, but will be able to later. A classic case wherein a Response Agent would issue the Retry response would be as follows:  
  - A device (e.g., the Root Complex, North Bridge, or MCH) handles a memory write by posting it in a Posted Memory Write buffer.  
  - If the buffer is currently full, the device would return the Retry response. |
### Chapter 51: Pentium® 4 FSB Response and Data Phases

#### Table 51-1: Response List (0 = deasserted, 1 asserted) (Continued)

<table>
<thead>
<tr>
<th>RS[2:0]#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010b</td>
<td><strong>Deferred Response.</strong> The Response Agent informs the Request Agent that it is deferring completion of the transaction until a later time. In other words, it will service the request off-line and will deliver the results to the Request Agent in a subsequent Deferred Reply transaction.</td>
</tr>
<tr>
<td>011b</td>
<td><strong>Reserved.</strong></td>
</tr>
<tr>
<td>100b</td>
<td><strong>Hard Failure Response.</strong> The Response Agent is indicating a hard failure to the Request Agent. The Response Agent is broken and can’t service the request at all.</td>
</tr>
</tbody>
</table>
| 101b     | **No Data Response.** This response indicates that no data was requested by the Request Agent and therefore no data will be delivered.  
• This is the proper response to a write (although data is written to the device, none is requested from it).  
• It is also the proper response to a transaction that doesn’t require any data to be transferred—the Special transaction, the Memory Read and Invalidate for 0 bytes, the Memory Code or Data Read for 0 bytes, or the IO Read for 0 bytes. |
| 110b     | **Implicit Writeback Response.** This response is issued by the Response Agent if a memory transaction resulted in a hit on a modified line (i.e., HITM# was asserted in the Snoop Phase). The Snoop Agent that has the modified line will supply the modified line to the Response Agent (i.e., the memory controller) as well as to the Request Agent (if it’s a read transaction). The author thinks of this as the “don’t be startled” response. A non-processor Request Agent (e.g., a bridge on behalf of a device adapter) may be attempting to read less than a line of information and, if the Snoop Agent has a hit on a modified line, it always supplies the full line. The implicit writeback response tells the Request Agent that eight qwords (64 bytes) will be transferred, rather than the smaller data packet actually requested. The eight qwords are transferred in toggle mode order, critical qword first. This means that the first qword sent back by the Snoop Agent will be the first one requested by the Request Agent and, if a second qword was also requested (assume, for example, that this is a 16 byte read request), the second qword sent back is the first qword’s toggle mode partner. The Request Agent should just accept the qword(s) requested and ignore the rest. The memory controller (i.e., the Response Agent), on the other hand, will accept the full line. |
The Previous Chapter

This chapter provided a detailed description of the Response and Data Phases of a FSB transaction. It included:

- The Purpose of the Response Phase.
- The Response Phase Signal Group.
- The Response Phase Start Point.
- The Response Phase End Point.
- The Response Types.
- The Response Phase May Complete a Transaction.
- The Data Phase Signal Group.
- Five Example Scenarios.
- Data Phase Wait States.
- The Response Phase Parity.
- Data Bus Parity.

This Chapter

This chapter provides a detailed description of the Deferred Transaction mechanism. It includes:

- The Problem.
- Example Read From a PCI Express Device.
- The Read Receives the Deferred Response.
- The Root Complex Performs the Read.
- The Root Complex Issues a Deferred Reply Transaction.
- Example Write To a PCI Express Device.
The Unabridged Pentium® 4

- The Write Receives the Defer Response.
- The Root Complex Delivers the Write Data to the Target.
- The Root Complex Issues a Deferred Reply Transaction.

The Next Chapter

This chapter describes the characteristics of FSB IO transactions. It includes:

- The IO Address Range.
- The Data Transfer Length.
- Behavior Permitted by the Spec.

Example System Models

The sections in this chapter describe transactions that are deferred and the subsequent Deferred Reply transactions using the example system models pictured in Figure 52-1 on page 1278 and Figure 52-2 on page 1279.

Figure 52-1: Example Multi-Cluster System
Example Multi-Cluster Model

This discussion focuses on the use of transaction deferral and Deferred Reply transactions to increase the overall performance of the system pictured in Figure 52-1 on page 1278.

The Problem

Example Problem 1

In Figure 52-1 on page 1278, when any of the local processors on one FSB attempts to perform a read or write that targets memory residing on the remote FSB, it can result in very long latency during the Data Phase of the transaction.
When a memory read transaction is initiated, all of the agents on the local FSB latch the transaction and each of the local Response Agents examines the address and transaction type to determine which of them is the target. Assuming that the local processor is targeting memory on the remote FSB, the Cluster Bridge must act as the local Response Agent for the transaction. Essentially, it acts as the surrogate for the remote target which resides on the remote FSB. If the transaction is a read, the bridge takes ownership of the local Data Bus (by asserting DBSY#), but keeps DRDY# deasserted until it can present the requested read data.

While the bridge continues to stretch the Data Phase by keeping DRDY# deasserted, it asserts BPRI# to the array of processors that reside on the target FSB. When it has acquired ownership of the remote Request Phase signal group, it initiates the memory read transaction. Eventually the memory target on the target FSB supplies the read data to the bridge. Only then can the bridge assert DRDY# to the local processor that is acting as the Request Agent and present the data to the local processor.

The local FSB has a Data Bus busy condition during this entire process. This will cause any subsequently issued local FSB transactions to stall when they reach their Data Phases.

Example Problem 2

In Figure 52-2 on page 1279, assume that a processor initiates an IO or a memory-mapped IO read from a register within the IEEE 1394 FireWire controller. This device resides on the PCI bus at the bottom of hierarchy. If the Root Complex handled the read by keeping the FSB Data Bus busy (i.e., keeping DBSY# asserted until the requested read data is finally returned by the FireWire controller), the Data Bus portion of the FSB will be tied up for an extensive period of time. This will cause any subsequently issued FSB transactions to stall when they reach their Data Phases.

Possible Solutions

The designers of the Root Complex can take one of three possible approaches:
1. The Root Complex could keep the FSB Data Bus busy for extensive periods of time. This is certainly the least-desirable approach.
2. The Root Complex can memorize the transaction and issue a retry response to the processor. This obligates the processor to re-arbitrate for ownership of the Request Phase signal group and retry the transaction on a
Chapter 52: Pentium® 4 FSB Transaction Deferral

periodic basis until it gets a good response and the read or write completes. When the Root Complex has finally completed the requested transaction on the PCI Express side, it waits for the processor’s next retry. When it latches a transaction issued on the FSB, it compares the agent ID and transaction ID to see if they match the IDs memorized in the transaction that was issued a retry response earlier. When it has a match, it permits the transaction to complete. It supplies the read data that it obtained from the PCI Express side and the Normal Data response.

Although better than option one, the retried processor’s repeated intrusions into the symmetric arbitration and its usage of the Request, Snoop, and Response signal groups will significantly diminish the performance of the other processors.

3. The optimal approach is for the Root Complex to memorize the transaction and issue a Deferred Response to the processor. The processor will not retry the transaction. Rather, the processor terminates the transaction, places the request in its Deferred Transaction Queue and suspends it. It will wait for the Response Agent to initiate a Deferred Reply transaction to provide the completion notice and the read data. The processor therefore doesn’t waste valuable FSB bandwidth with fruitless retries of the transaction and the FSB remains available for the processors to use (including the same processor).

Example Read From a PCI Express Device

The Read Receives the Deferred Response

Refer to Figure 52-2 on page 1279 and Figure 52-3 on page 1282 during this discussion.

In this example, one of the processors initiates either an IO read or a memory-mapped IO read from a register within the IEEE 1394 FireWire controller on the PCI bus. The Root Complex acts as the Response Agent for the transaction and, because it will take awhile to obtain the requested read data, it issues a Deferred response to the read transaction:

1. The read transaction request is issued in BCLK cycle 1. Acting as the Response Agent, the Root Complex memorizes the address, the transaction type, and the Request Agent’s Agent ID and Transaction ID (see Table 49-10 on page 1221 and Figure 49-10 on page 1223).
53 Pentium® 4 FSB
IO Transactions

The Previous Chapter
This chapter provided a detailed description of the Deferred Transaction mechanism. It included:

- The Problem.
- Example Read From a PCI Express Device.
- The Read Receives the Deferred Response.
- The Root Complex Performs the Read.
- The Root Complex Issues a Deferred Reply Transaction.
- Example Write To a PCI Express Device.
- The Write Receives the Defer Response.
- The Root Complex Delivers the Write Data to the Target.
- The Root Complex Issues a Deferred Reply Transaction.

This Chapter
This chapter describes the characteristics of FSB IO transactions. It includes:

- The IO Address Range.
- The Data Transfer Length.
- Behavior Permitted by the Spec.

The Next Chapter
This chapter provides a detailed description of FSB Central Agent transactions. It includes:

- Point-to-Point vs. Broadcast.
- The Interrupt Acknowledge Transaction.
- The Special Transaction.
- The BTM Transaction Is Used for Program Debug.
Introduction

Refer to Figure 53-1 on page 1296. The processor performs an IO Read or IO Write transaction on the FSB due to the execution of an IO instruction (IN, INS, OUT, or OUTS).

There is nothing exotic about IO transactions. Like any other transaction type, an IO transaction consists of a Request, Snoop, Response and Data Phase. The following is a summary of general IO transaction characteristics:

- Since the processors never cache information from IO space, there will never be a hit on a cache line (the caches aren’t even checked).
- The only appropriate snoop results are a miss (HIT# and HITM# both deasserted), or snoop stall (both asserted) followed by a miss.
- DEFER# may be asserted by the Response Agent if it intends to issue a retry or a deferred response in the Response Phase.
- In the Response Phase, the only response that may not be issued is the implicit writeback response (because there will never be a hit on a modified IO cache line).

Figure 53-1: The Execution of an IO Instruction Results in an IO Transaction

The IO Address Range

The IO address range supported by the Pentium® 4 processor is from 000000000h through 000010002h (the overall range is 64KB+3 in size). This is backward-compatible with previous x86 processors. Consider the following:
Chapter 53: Pentium® 4 FSB IO Transactions

- A 2-byte IO access starting at IO address FFFFh. In this case, the 2-bytes of data straddles the 64KB address boundary. Since these two bytes reside in different qwords, the processor would perform this as two separate single-qword transactions.
- A 4-byte IO access starting at IO address FFFFh, FFFEh, or FFFDh. As before, the target dword straddles the 64KB address boundary, and the processor would perform this as two separate single-qword transactions.

In both cases, when accessing above the 64KB boundary, the processor would be asserting A[16]#.

The Data Transfer Length

Behavior Permitted by the Spec

When an IO read or write transaction is initiated, the data transfer length is output by the Request Agent in request packet B (see Table 49-8 on page 1219). The spec permits IO data transfer lengths of:

- A qword or less. Any combination of byte enables are valid, including none.
- Two full qwords. All byte enables must be asserted in request packet B.
- Four full qwords. All byte enables must be asserted in request packet B.
- Eight full qwords. All byte enables must be asserted in request packet B.

On a 0-byte read, the response must be the no data response (unless DEFER# is asserted by the Response Agent, indicating that it intends to retry or defer the transaction).

On a 0-byte write, the Response Agent must assert TRDY#, but the Request Agent must not assert DBSY# or DRDY# in response. Note that the author doesn’t know why an agent would initiate a 0-byte IO transaction. IA32 processors are incapable of doing this.

How the Pentium® 4 Processor Operates

The Pentium® 4 processor only performs IO read and write transactions due to the execution of IO read (IN or INS) or write (OUT or OUTS) instructions. The programmer may only specify the AL, AX, or EAX register as the target or source register for the read or write. This restricts the transfers to:
The Unabridged Pentium® 4

- a single byte.
- two contiguous bytes.
- four contiguous bytes.

This means that, at most, the transfer length will always be less than a qword and, at a maximum, four contiguous byte enables will be asserted. If the accessed data crosses a dword address boundary, the processor will behave as follows:

- If the transaction is an IO read and the access crosses the dword boundary within a qword (see Figure 53-2 on page 1298), one access is performed with the appropriate byte enables asserted.
- If the transaction is an IO read and the access crosses a qword boundary (see Figure 53-3 on page 1299), two separate single-qword accesses are performed with the appropriate byte enables asserted.
- If the transaction is an IO write and the access crosses the dword boundary within a qword (see Figure 53-4 on page 1299), two accesses are performed with the appropriate byte enables asserted.
- If the transaction is an IO write and the access crosses a qword boundary (see Figure 53-5 on page 1300), two separate single-qword accesses are performed with the appropriate byte enables asserted.

Figure 53-2: An IO Read that Crosses a Dword Address Boundary

If the IO read access crosses a dword address boundary within a qword, one access is performed with the appropriate Byte Enables asserted (in packet B of the Request Phase).
Chapter 53: Pentium® 4 FSB IO Transactions

Figure 53-3: An IO Read that Crosses a Qword Address Boundary

If the IO read access crosses a qword address boundary, two separate single qword IO read accesses are performed with the appropriate Byte Enables asserted (in packet B of the Request Phase) for each of the accesses.

Figure 53-4: An IO Write that Crosses a Dword Address Boundary

If the IO write access crosses a dword address boundary within a qword, two separate, single-qword IO write accesses are performed with the appropriate Byte Enables asserted (in packet B of the Request Phase).
The Previous Chapter
This chapter described the characteristics of FSB IO transactions. It included:

- The IO Address Range.
- The Data Transfer Length.
- Behavior Permitted by the Spec.

This Chapter
This chapter provides a detailed description of FSB Central Agent transactions. It includes:

- Point-to-Point vs. Broadcast.
- The Interrupt Acknowledge Transaction.
- The Special Transaction.
- The BTM Transaction Is Used for Program Debug.

The Next Chapter
This chapter provides a detailed description of FSB signal that were not described in earlier chapters.
Point-to-Point vs. Broadcast

Most transactions are point-to-point transactions—the Request Agent addresses a specific area of memory or IO space for a read or a write and the addressed target acts as the transaction’s Response Agent.

Some transactions generated by the processor don’t target any specific memory or IO device, however. Rather, the processor is performing one of the following operations:

- The **Interrupt Acknowledge** transaction to request the interrupt vector from the interrupt controller. In this case, the Root Complex would act as the Response Agent (because the interrupt controller typically resides within or beneath the Root Complex).
- The **Special transaction** to broadcast a message. No specific device is targeted by the transaction, but someone has to act as the Response Agent. It is typically the Root Complex.
- The **Branch Trace Message (BTM) transaction** to inform a debug tool that, when executed, a branch was taken. Once again, no specific device is addressed and yet someone has to act as the Response Agent. It is typically the Root Complex.

Intel® refers to these as *central agent transactions* because one, central device (the chipset) typically acts as the default Response Agent for these transaction types.

The Interrupt Acknowledge Transaction

**Background**

An IA32-based system incorporates an interrupt controller that receives interrupt requests from IO devices and passes them on to the processor (or to the processor cluster). The interrupt controller will either consist of a pair of cascaded 8259A’s in a single processor system (see “Before the Advent of the APIC” on page 1498), or an IO APIC module in a multiprocessor system.

Refer to Figure 54-1 on page 1304. In earlier chipsets, the interrupt controller was incorporated in the South Bridge. It is found in the ICH (the IO Control Hub) in the chipsets that are prevalent as of this writing. This is a strategically convenient place for it because the interrupt requests from PCI and legacy ISA targets (typically residing on the LPC—Low-Pin Count—bus) can easily be connected to it.
Chapter 54: Pentium® 4 FSB Central Agent Transactions

Assuming that the system uses the 8259A interrupt controllers, the interrupt controller asserts its INTR (Interrupt Request) output when it detects any interrupt requests from device adapters (see Figure 54-1 on page 1304). The INTR signal line is connected to the processor’s INTR input pin (also referred to as the LINT0 pin). In response to its assertion, the processor takes the following actions:

1. Assuming that recognition of external interrupts is enabled (in other words, the programmer has not executed the CLI instruction), the processor will recognize the request when it completes the execution of the current instruction.
2. The processor suspends execution of the interrupted program.
3. The processor generates an Interrupt Acknowledge transaction on its FSB to read the interrupt vector (of the highest priority request) from the interrupt controller.
4. The North Bridge or MCH (Memory Control Hub) passes the request for the interrupt vector to the South Bridge or ICH. In a North Bridge/South Bridge configuration, the North Bridge generates a PCI (or PCI-X) Interrupt Acknowledge to request the vector from the interrupt controller embedded within the South Bridge.
5. The South Bridge or ICH passes the vector back to the North Bridge or MCH.
6. The North Bridge or MCH passes the vector to the processor.
7. The processor uses the 8-bit vector as an index into the Interrupt Table in memory and reads the new CS:EIP value from the selected entry.
8. The processor pushes the contents of its CS, EIP and EFlags registers into stack memory (to mark its place in the interrupted program).
9. The processor then disables recognition of additional external interrupts (i.e., it clears the EFlags[IF] bit).
10. Using the new CS:EIP value, the processor jumps to the target interrupt service routine and executes it.
The Transaction Details

Earlier, pre-Pentium® Pro IA32 processors generated two, back-to-back Interrupt Acknowledge transactions when an interrupt was delivered on the INTR pin:

- One to command the interrupt controller to prioritize its pending requests.
- The second to request the interrupt vector for the most important one.

Starting with the P6 processor family, however, the processor only generates one Interrupt Acknowledge transaction. This transaction has the following characteristics:

- In Packet A, the request type issued on REQ[4:0]# is 01000b (this is the logical, not electrical, value). For more information, refer to Table 49-5 on page 1216.
- Although the content of the address bus in packet A is “don’t care,” it must be stable and is factored into the address parity on AP[1:0]#.
In Packet B, REQ[4:0]# is \(00\times00b\), where \(x\) is “don’t care.”
In Packet B, with the exception of A[15:8]# (the Byte Enables) and A[4]# (DEN#, Defer Enable), the content of the address bus is “don’t care.”
In Packet B, DEN# is asserted, granting the Response Agent permission to Defer or Retry the transaction if it so chooses.
In Packet B, only BE[0]# is asserted, indicating that it’s a single-byte read to obtain the interrupt vector over data path 0 (D[7:0]#).

**The Root Complex is the Response Agent**

In Figure 54-2 on page 1305, the Root Complex acts as the Response Agent if the interrupt controller resides within or beneath the Root Complex. Since it may take some time to obtain the vector, the Root Complex may choose to issue the Deferred response to the processor. The Root Complex forwards the vector request to the device containing the interrupt controller for fulfillment. When the Root Complex receives a reply packet containing the 8-bit vector, it initiates a Deferred Reply transaction on the FSB to deliver the vector to the processor.

![Figure 54-2: An Example PCI Express System](image-url)
The Previous Chapter
This chapter provides a detailed description of FSB signal that were not described in earlier chapters.

This Chapter
This chapter provides a detailed description of the software enhancements implemented in the Pentium® 4 processor. This includes:

- Miscellaneous New Instructions.
- Enhanced CPUID Instruction.
- The SSE2 Instruction Set.
- The SSE3 Instruction Set.
- Local APIC Enhancements.
- The Thermal Monitoring Facilities.
- FPU Enhancement.
- The MSRs.
- The Machine Check Architecture.
- Last Branch, Interrupt, and Exception Recording.
- The Debug Store (DS) Mechanism.
- New Exceptions.
- The Performance Monitoring Facility.

The Next Chapter
This chapter describes the characteristics of the Xeon processor based on the Pentium® 4 technology. It includes no new software enhancements.
The Unabridged Pentium® 4

The Foundation

Refer to Table 56-1 on page 1322. From a software perspective, the Pentium® 4 processor is the sum of all of the IA32 software architectural features that have been introduced since the 386 processor.

Table 56-1: The Elements of the Software Architecture

<table>
<thead>
<tr>
<th>Element</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>The instruction set</td>
<td>The IA32 instruction set has grown over the years. A description of the instructions (in order of introduction) can be found in:</td>
</tr>
<tr>
<td></td>
<td>• The 386 instruction set can be found in “Instruction Set Evolution” on page 115 on the CD.</td>
</tr>
<tr>
<td></td>
<td>• The 486 instruction set additions and/or changes are described in “Instruction Set Changes” on page 456.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® instruction set additions and/or changes are described in “Instruction Set Changes” on page 517.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® Pro instruction set additions and/or changes are described in “Instruction Set Changes” on page 626.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® II instruction set additions and/or changes are described in “Instruction Set Changes” on page 707.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® III’s SSE instruction set is described in “The Streaming SIMD Extensions (SSE)” on page 758.</td>
</tr>
<tr>
<td></td>
<td>• The 130nm Pentium® 4 instruction set additions (other than SSE2) are described in “Miscellaneous New Instructions” on page 1325.</td>
</tr>
<tr>
<td></td>
<td>• The 130nm Pentium® 4’s SSE2 instruction set is described in “The SSE2 Instruction Set” on page 1332.</td>
</tr>
<tr>
<td></td>
<td>• The 90nm Pentium® 4’s SSE3 instruction set is described in “The SSE3 Instruction Set” on page 1337.</td>
</tr>
<tr>
<td>Real Mode</td>
<td>A complete description of Real Mode can be found in “386 Real Mode Operation” on page 39.</td>
</tr>
</tbody>
</table>
### Chapter 56: Pentium® 4 Software Enhancements

<table>
<thead>
<tr>
<th>Element</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected Mode</td>
<td>A complete baseline description of Protected Mode can be found in the following chapters:</td>
</tr>
<tr>
<td></td>
<td>• “Protected Mode Introduction” on page 103.</td>
</tr>
<tr>
<td></td>
<td>• “Intro to Segmentation in Protected Mode” on page 109.</td>
</tr>
<tr>
<td></td>
<td>• “Code Segments” on page 133.</td>
</tr>
<tr>
<td></td>
<td>• “Data and Stack Segments” on page 157.</td>
</tr>
<tr>
<td></td>
<td>• “Creating a Task” on page 171.</td>
</tr>
<tr>
<td></td>
<td>• “Mechanics of a Task Switch” on page 191.</td>
</tr>
<tr>
<td></td>
<td>• “386 Demand Mode Paging” on page 209.</td>
</tr>
<tr>
<td></td>
<td>• “The Flat Model” on page 247.</td>
</tr>
<tr>
<td></td>
<td>• “Interrupts and Exceptions” on page 251.</td>
</tr>
<tr>
<td></td>
<td>• “Virtual 8086 Mode” on page 329.</td>
</tr>
<tr>
<td>Paging</td>
<td>A baseline description of 386 Paging can be found in “386 Demand Mode Paging” on page 209. Paging was incrementally improved over the years and the descriptions of the improvements (in order of introduction) can be found in:</td>
</tr>
<tr>
<td></td>
<td>• The 486 improvements are described in “Paging-Related Changes” on page 449 and “Invalidate TLB Entry (INVLPG)” on page 458.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® improvements are described in “4MB Pages” on page 501.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® Pro improvements are described in “Paging Enhancements” on page 554.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® II Xeon’s improvements are described in “PSE-36 Mode” on page 731.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® III Xeon’s improvements are described in “PAT Feature (Page Attribute Table)” on page 797.</td>
</tr>
<tr>
<td>VM86 Mode</td>
<td>A baseline description of the 386 processor’s Virtual 8086 Mode can be found in “Virtual 8086 Mode” on page 329.</td>
</tr>
<tr>
<td></td>
<td>• The Pentium® processor improved upon VM86 Mode and those improvements are described in “VM86 Extensions” on page 490.</td>
</tr>
<tr>
<td>MMX</td>
<td>The MMX instruction and register sets were introduced in the P55C version of the Pentium® processor and are described in “MMX Capability” on page 519.</td>
</tr>
</tbody>
</table>
The SSE instruction and register sets were introduced in the Pentium® III processor and a complete description can be found in “The Streaming SIMD Extensions (SSE)” on page 758.

The SSE2 instruction set was introduced in the 130nm Pentium® 4 processor and a complete description can be found in “The SSE2 Instruction Set” on page 1332.

The SSE3 instruction set was introduced in the 90nm Pentium® 4 processor and a complete description can be found in “The SSE3 Instruction Set” on page 1337.

Various debug-related features have been introduced over the years. The following sections provide a description of each of these features:

- Single-Step Mode. See the description of EFlags[TF] in Table 5-3 on page 49.
- See the description of EFlags[RF] in Table 5-3 on page 49.
- See “The Debug Registers” on page 375.
- See “Debug Trap Bit (T)” on page 181.
- See “The Resume Flag Prevents Multiple Debug Exceptions” on page 291.
- See “Debug Exception (1)” on page 293.
- See “Breakpoint Exception (3)” on page 295.
- See “Alignment Check Exception (17)” on page 321.
- See “Alignment Checking Feature” on page 448.
- See “Test Access Port (TAP)” on page 481.
- See “Debug Extension” on page 497.
- See “DebugCtl MSR” on page 621.
- See the description of bit 0 in Table 36-5 on page 871.
- See “BNR# Can Be Used by a Debug Tool” on page 1191.
- See “The BTM Transaction Is Used for Program Debug” on page 1309.
- See the description of the BPM[3:0]# outputs in Table 55-1 on page 1314.

<table>
<thead>
<tr>
<th>Element</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>The SSE instruction and register sets</td>
<td>The SSE instruction and register sets were introduced in the Pentium® III processor and a complete description can be found in “The Streaming SIMD Extensions (SSE)” on page 758.</td>
</tr>
<tr>
<td>The SSE2 instruction set</td>
<td>The SSE2 instruction and register sets were introduced in the 130nm Pentium® 4 processor and a complete description can be found in “The SSE2 Instruction Set” on page 1332.</td>
</tr>
<tr>
<td>The SSE3 instruction set</td>
<td>The SSE3 instruction set was introduced in the 90nm Pentium® 4 processor and a complete description can be found in “The SSE3 Instruction Set” on page 1337.</td>
</tr>
</tbody>
</table>
Chapter 56: Pentium® 4 Software Enhancements

Table 56-1: The Elements of the Software Architecture (Continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exceptions</td>
<td>Refer to “Detailed Description of the Software Exceptions” on page 292. Also see:</td>
</tr>
<tr>
<td></td>
<td>• “Software-Generated Exceptions” on page 260.</td>
</tr>
<tr>
<td></td>
<td>• “Interrupt/Exception Priority” on page 266.</td>
</tr>
<tr>
<td></td>
<td>• “Real Mode Interrupt/Exception Handling” on page 270.</td>
</tr>
<tr>
<td></td>
<td>• “Protected Mode Interrupt/Exception Handling” on page 272.</td>
</tr>
<tr>
<td></td>
<td>• “Interrupt/Exception Handling in VM86 Mode” on page 287.</td>
</tr>
<tr>
<td></td>
<td>• “Exception Error Codes” on page 288.</td>
</tr>
<tr>
<td>The Time Stamp Counter</td>
<td>See the “Time Stamp Counter” on page 498.</td>
</tr>
<tr>
<td>The Local APIC</td>
<td>See “The Local and IO APICs” on page 1497.</td>
</tr>
<tr>
<td>SM Mode</td>
<td>See “System Management Mode (SMM)” on page 1463.</td>
</tr>
<tr>
<td>The MTRRs</td>
<td>See “MTRRs Added” on page 572.</td>
</tr>
<tr>
<td>The x87 FPU</td>
<td>See “FPU Added On-Die” on page 432.</td>
</tr>
<tr>
<td>The MCA</td>
<td>See “MCA Enhanced” on page 588 and “The Machine Check Architecture” on page 1363.</td>
</tr>
</tbody>
</table>

Miscellaneous New Instructions

General

The 130nm Pentium® 4 processor added 144 new instructions to the IA32 instruction repertoire. This is referred to as the SSE2 instruction set. Of these, the author has chosen to discuss the following instructions separately in this section and the remainder are covered in “The SSE2 Instruction Set” on page 1332.
The Previous Chapter
This chapter provided a detailed description of the software enhancements implemented in the Pentium® 4 processor. This included:

- Miscellaneous New Instructions.
- Enhanced CPUID Instruction.
- The SSE2 Instruction Set.
- The SSE3 Instruction Set.
- Local APIC Enhancements.
- The Thermal Monitoring Facilities.
- FPU Enhancement.
- The MSR.
- The Machine Check Architecture.
- Last Branch, Interrupt, and Exception Recording.
- The Debug Store (DS) Mechanism.
- New Exceptions.
- The Performance Monitoring Facility.

This Chapter
This chapter describes the characteristics of the Xeon processor based on the Pentium® 4 technology. This Xeon includes no new software enhancements.

The Next Chapter
This chapter describes the hardware and software characteristics of the Pentium® M processor as well as an overview of the Centrino chipset. It includes:

- The Pentium® M and Centrino.
- Characteristics Overview.
- The FSB Characteristics.
The Unabridged Pentium® 4

- Enhanced Power Management Characteristics.
- Three Different Packaging Models.
- Improved Thermal Monitor Mode.
- Enhanced Branch Prediction.
- μop Fusion.
- Advanced Stack Management.
- Hardware-Based Data Prefetcher.
- The L2 Cache.
- The Data Cache and Hyper-Threading.
- The Next Pentium® M.

General

The currently available Xeon processors are all based on the Pentium® 4 processor core. Each Xeon also implements the SMBus (see “SMBus (System Management Bus)” on page 723). The Xeon is 100% soft-compatible with the Pentium® 4 processor.

The Pentium® 4 Xeon DP

The Dual-Processor version of the Xeon supports one or two processors on the FSB. While earlier models did not have an on-die L3 Cache, some of the later models do. The cache sizes are processor design-specific.

The Pentium® 4 Xeon MP

The multiprocessor version of the Xeon processor supports up to four processors on the FSB and has an on-die L3 Cache. The cache sizes are processor design-specific.
Part 11
Pentium® M

The Previous Part

The previous part provided a detailed description of the hardware design and software enhancements encompassed in the Pentium® 4 processor family. It consists of the following chapters:

- “Pentium® 4 Road Map” on page 813.
- “Pentium® 4 System Overview” on page 823.
- “Pentium® 4 Processor Overview” on page 835.
- “Pentium® 4 PowerOn Configuration” on page 855.
- “Pentium® 4 Processor Startup” on page 875.
- “Pentium® 4 Core Description” on page 897.
- “Hyper-Threading” on page 965.
- “The Pentium® 4 Caches” on page 1009.
- “Pentium® 4 Handling of Loads and Stores” on page 1061.
- “The Pentium® 4 Prescott” on page 1091.
- “Pentium® 4 FSB Electrical Characteristics” on page 1115.
- “Intro to the Pentium® 4 FSB” on page 1137.
- “Pentium® 4 CPU Arbitration” on page 1149.
- “Pentium® 4 Priority Agent Arbitration” on page 1165.
- “Pentium® 4 Locked Transaction Series” on page 1177.
- “Pentium® 4 FSB Blocking” on page 1189.
- “Pentium® 4 FSB Request Phase” on page 1201.
- “Pentium® 4 FSB Snoop Phase” on page 1225.
- “Pentium® 4 FSB Response and Data Phases” on page 1241.
- “Pentium® 4 FSB Transaction Deferral” on page 1277.
- “Pentium® 4 FSB IO Transactions” on page 1295.
- “Pentium® 4 FSB Central Agent Transactions” on page 1301.
- “Pentium® 4 FSB Miscellaneous Signals” on page 1313.
- “Pentium® 4 Software Enhancements” on page 1321.
- “Pentium® 4 Xeon Features” on page 1421.
The Unabridged Pentium® 4

This Part

This part describes the hardware and software characteristics of the Pentium® M processor and consists of the following chapter:

- “Pentium® M Processor” on page 1425.

The Next Part

The next part provides a detailed description of processor identification, System Management Mode, and the IO and Local APICs. It consists of the following chapters:

- “CPU Identification” on page 1443.
- “System Management Mode (SMM)” on page 1463.
- “The Local and IO APICs” on page 1497.
This chapter described the characteristics of the Xeon processor based on the Pentium® 4 technology. This Xeon includes no new software enhancements.

This chapter describes the hardware and software characteristics of the Pentium® M processor as well as an overview of the Centrino chipset. It includes:

- The Pentium® M and Centrino.
- Characteristics Overview.
- The FSB Characteristics.
- Enhanced Power Management Characteristics.
- Three Different Packaging Models.
- Improved Thermal Monitor Mode.
- Enhanced Branch Prediction.
- \texttt{\textmu oper} Fusion.
- Advanced Stack Management.
- Hardware-Based Data Prefetcher.
- The L2 Cache.
- The Data Cache and Hyper-Threading.
- The Next Pentium® M.

This chapter provides a detailed description of the CPUID instruction. It includes:

- Prior to the Advent of the CPUID Instruction.
- Determining if the CPUID instruction Is Supported.
- Determining Basic Request Types Supported.
- Determining Extended Request Types Supported.
The Unabridged Pentium® 4

- The Basic Request Types.
- Request Type 1.
- Request Type 2.
- Request Type 3.
- Request Type 4.
- Request Type 5.
- The Extended Request Types.
- Enhanced Processor Signature.

Background

The Pentium® M processor (not to be confused with the Pentium® 4 M) was code named Banias and was introduced on 03/12/03. It is the first Intel® IA32 processor to be designed from the ground up as a mobile (i.e., laptop) processor. Reducing power conservation was targeted in many areas of the processor design. It is not a member of the Pentium® 4 processor family. Intel® has never confirmed it, but it is based on the Pentium® III processor core rather than the Pentium® 4 core. Almost certainly, the next version of the Pentium® M will probably be designed around the Pentium® 4 core.

From a software perspective, the Pentium® M processor is 100% compatible with the 130nm Pentium® 4 processor. The 90nm Pentium® 4 instruction set is a superset of that found in the Pentium® M and the 130nm Pentium® 4.

The Pentium® M and Centrino

The Pentium® M processor was introduced at the same time that Intel® introduced the Centrino chipset. Currently, this chipset is comprised of:

- The Pentium® M processor.
- The 855 MCH. This component connects the processor FSB to the ICH4 (IO Control Hub-4), to the graphics adapter (although there is a version with an integrated graphics adapter), and to system memory.
- The PRO/Wireless network connection. This is Intel®’s wireless bridge chip.

When a system vendor integrates all of these components into a laptop design, they are entitled to use the Centrino name and logo. However, if any of the components are not used, the vendor cannot use the Centrino name and logo. For example, a number of laptop designs do not use the Intel® wireless chip, but do use the 855MCH and the Pentium® M processor.
Chapter 58: Pentium® M Processor

Characteristics Overview

The following is a list of the Pentium® M’s characteristics (in no particular order):

- It’s a variant on the Pentium® III core.
- Speeds of 1.2GHz, 1.4GHz, 1.5GHz, 1.6GHz, 1.7GHz (as of 6/12/04).
- Enhanced power management characteristics:
  - Deeper Sleep state added.
  - Enhanced SpeedStep technology.
  - FSB power utilization enhancements.
  - The processor automatically shuts down units that are not in use.
- 32KB 8-way L1 Code Cache (caches IA32 instructions, not µops). Latency = 3 clock cycles. Cache line size = 64 bytes.
- ITLB has 128 entries.
- 32KB 8-way WB L1 Data Cache. 64 byte line size.
- 1MB L2 ATC, 64 bytes per line.
- Enhanced branch prediction logic.
- Enhanced hardware-based data prefetch logic.
- 400MHz FSB (100MHz BCLK).
- 0.13 micron process.
- µop Fusion feature.
- Advanced Stack Management.
- Power-aware cache design.
- Includes SSE2.
- 32-bit address bus.

The FSB Characteristics

Uses the Pentium® 4 FSB Protocol

The FSB uses the same protocol as the Pentium® 4 processor family. The BCLK speed is 100MHz (rather than 200MHz as on the Pentium® 4). The address bus width is 32 bits consisting of A[31:3]# (rather than 36 bits as on the Pentium® 4’s FSB).
Pentium® M-Specific Signals

The signals in Table 58-1 on page 1428 are specific to the Pentium® M processor (i.e., they are not found on the Pentium® 4 family processors).

<table>
<thead>
<tr>
<th>Signal</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSI#</td>
<td>Output</td>
<td><strong>Power Status Indicator.</strong> Asserted when the processor is in the Deep Sleep or Deeper Sleep power management state. Asserted upon Deep Sleep entry and deasserted upon exit. PSI# can be provided as an input to the voltage regulator on the system board. When the processor asserts PSI#, the voltage regulator can use it to improve its light load efficiency (resulting in platform power savings). PSI# can also be used to simplify the design of the voltage regulator (it removes the need for the integrated 100µs timer required to mask the PWRGOOD signal during Deeper Sleep transitions). It also reduces the PWRGOOD monitoring requirements while the processor is in the Deeper Sleep state.</td>
</tr>
<tr>
<td>DPSLP#</td>
<td>Input</td>
<td><strong>Deep Sleep.</strong> When asserted to the processor by the chipset, this signal causes the processor to transition from the Sleep state to the Deep Sleep state (resulting in greater power savings). The chipset deasserts DPSLP# to return the processor to the Sleep state.</td>
</tr>
<tr>
<td>DPWR#</td>
<td>Input</td>
<td><strong>Data Bus Power.</strong> When asserted to the processor by the chipset, the processor’s data bus input buffers are deactivated to conserve power. The MCH deasserts DPWR# when data bus activity is detected, thereby re-enabling the processor’s data bus input receivers.</td>
</tr>
</tbody>
</table>
FSB Power Utilization Enhancements

The processor design implements the following FSB power-related changes:

- The FSB uses lower LVS (Low Voltage Swing) levels than earlier FSB versions. Vref is 2/3 of Vcc and Vcc is quite low.
- The processor incorporates on-die termination resistors for the FSB AGTL+ signals. Whenever any agent drives a signal low, the processor automatically disables its on-die termination resistor to save on power.
- DPWR# (Data Bus Power) input. This signal is described in Table 58-1 on page 1428.
- BPRI# input. When the processor doesn’t need the bus (its BR0# output is not asserted) and no Priority Agent needs the bus (the processor’s BPRI# input is deasserted), the processor disables its address bus inputs and its control inputs to conserve power. They are automatically re-enabled when the processor or a Priority Agent needs the bus (i.e., the processor detects BPRI# asserted by the chipset).
- The address bus width is 32-bits (rather than 36 bits) wide because laptops typically do not need to address more than 4GB of memory. A side-benefit, however, is that it takes less power to drive a narrower address bus.

Enhanced Power Management Characteristics

Background

For background on the power conservation states available in earlier IA32 processors (including the Pentium® 4), refer to “Pentium® II Power Management Features” on page 683.

Entry to the Deep Sleep State

On the Pentium® 4 and earlier processors, the Deep Sleep state is entered (from the Sleep state) if the chipset causes the system board clock generator to turn off the BCLK to the processor (see Figure 58-1 on page 1431).

Refer to Figure 58-2 on page 1432. The chipset can transition the Pentium® M processor to the Deeper Sleep state (from the Sleep state) by asserting the DPSLP# signal to the processor. Deasserting the DPSLP# signal causes a transition back to the Sleep state.
The Previous Chapter
This chapter described the hardware and software characteristics of the Pentium® M processor as well as an overview of the Centrino chipset. It included:

- The Pentium® M and Centrino.
- Characteristics Overview.
- The FSB Characteristics.
- Enhanced Power Management Characteristics.
- Three Different Packaging Models.
- Improved Thermal Monitor Mode.
- Enhanced Branch Prediction.
- µop Fusion.
- Advanced Stack Management.
- Hardware-Based Data Prefetcher.
- The L2 Cache.
- The Data Cache and Hyper-Threading.
- The Next Pentium® M.

This Chapter
This chapter provides a detailed description of the CPUID instruction. It includes:

- Prior to the Advent of the CPUID Instruction.
- Determining if the CPUID instruction Is Supported.
- Determining Basic Request Types Supported.
- Determining Extended Request Types Supported.
- The Basic Request Types.
- Request Type 1.
- Request Type 2.
- Request Type 3.
The Unabridged Pentium® 4

- Request Type 4.
- Request Type 5.
- The Extended Request Types.
- Enhanced Processor Signature.

The Next Chapter

This chapter provides a detailed description of System Management Mode (SMM). It includes:

- What Falls Under the Heading of System Management?
- The Genesis of SMM.
- SMM Has Its Own Private Memory Space.
- The Basic Elements of SMM.
- How the Processor Knows the SM Memory Start Address.
- The Organization of SM RAM.
- Entering SMM.
- Exiting SMM.
- The Auto Halt Restart Feature.
- The IO Instruction Restart Feature.
- Caching from SM Memory.
- Setting Up the SMI Handler in SM Memory.
- Relocating the SM RAM Base Address.
- SMM in an MP System.

Prior to the Advent of the CPUID Instruction

This chapter only covers the identification of IA32 processors that support the CPUID instruction. This instruction was introduced in the Pentium® processor and then migrated backwards into the later versions of the 486 processor. If the reader wants to know how to identify the processor type on an earlier processor, refer to the following Intel® document:

Application Note 486, February 2004Intel; Processor Identification and the CPUID Instruction; Document Number: 241618-025.

Determining if the CPUID instruction Is Supported

Before executing the CPUID instruction, the programmer must first ascertain if the processor implements it. This is accomplished by attempting to write a one into the EFlags[ID] bit (see Figure 59-1 on page 1445). If the bit can be changed to a one, then the processor supports the CPUID instruction.
Chapter 59: CPU Identification

It would seem that the attempted execution of the CPUID instruction on a processor that does not support it would result in an invalid opcode exception. However, Intel® specifically says (in AP Note AP-485) “Do not depend on the absence of an invalid opcode trap on the CPUID opcode to detect the CPUID instruction.” This implies that at least one of the earlier (pre-Pentium®) processors that doesn’t support the CPUID instruction does not generate an invalid opcode exception when an attempt is made to execute the CPUID instruction.

Figure 59-1: The ID Bit Is in the EFlags Register

Note: All bits shown with a 1 or a 0 are Intel reserved. They must always be set to the values previously read from them.
General

The CPUID instruction was first introduced in the Pentium® and migrated backwards into the later models of the 486.

Prior to the advent of the Pentium® 4 processor, only basic information about the processor could be requested. The Pentium® 4 processor added the ability to requested extended information.

Determining the Request Types Supported

Determining Basic Request Types Supported

The programmer can determine the types of basic information requests supported by preloading the EAX register with zero and then executing the CPUID instruction. The processor returns the following information:

- The value returned in the EAX register represents the highest basic information request type supported.
- The EBX:ECX:EDX registers contain the character string “GenuineIntel”.

Determining Extended Request Types Supported

The programmer can determine the types of extended information request types supported by preloading the EAX register with 80000000h and then executing the CPUID instruction. The value returned in the EAX register represents the highest extended information request type supported.

The Basic Request Types

Request Type 1

General

Request type 1 was introduced in the Pentium® processor and is supported by all subsequent IA32 processors (as well as the later 486 models). When the input value in EAX = 1 (i.e., request type 1), the processor returns the following items of information:
Prior to the advent of the Pentium® 4 processor, the information shown in Figure 59-2 on page 1448 is returned in EAX. With the advent of the Pentium® 4 processor, the information shown in Figure 59-3 on page 1448 is returned in EAX. Table 59-2 on page 1449 defines the processor type field values.

Prior to the advent of the Pentium® 4 processor, the capability bit mask shown in Figure 59-4 on page 1449 was returned in the EDX register. With the advent of the Pentium® 4 processor, the information shown in Figure 59-5 on page 1450 is returned in EDX and the information shown in Table 59-3 on page 1450 is returned in ECX.

With the advent of the Pentium® III processor, a request type 1 also returns additional information in EBX (see Figure 59-6 on page 1452).

— “The Brand Index” on page 1447 describes the Brand Index value.
— The APIC ID field is described in “Processor Enumeration” on page 975 and “The Local APIC ID” on page 864.
— The Logical Processors field is described in “Processor Enumeration” on page 975.
— The Cache Line Size field is described in “The Cache Line Flush Instruction” on page 1326.

The Brand Index

With the advent of the Pentium® III processor, the information shown in Figure 59-6 on page 1452 is returned in the EBX register by a CPUID request type 1.

The processor Brand Index is returned in EBX[7:0]. This number provides an entry into a memory-based brand string table that contains brand strings for IA32 processors.

The Brand ID Table is placed in memory by system software (e.g., the BIOS) and it is accessible by both OS kernel and user-level code. In the table (see Table 59-1 on page 1448), each brand index value is associated with an ASCII brand ID string that identifies the Intel® family and model number of a processor (e.g., “Intel® Pentium® III processor”).

The first table entry (index 0) is reserved, allowing for backward compatibility with processors that do not support the brand ID feature. Table 59-1 shows the brand indices that currently have processor brand ID strings associated with them.

The brand string is architecturally defined to be 48 bytes in length, with the first 47 bytes containing ASCII characters and the 48th byte defined to be null (0). The string may be right justified (with leading spaces) for implementation simplicity.
The Previous Chapter
This chapter provided a detailed description of the CPUID instruction. It included:

- Prior to the Advent of the CPUID Instruction.
- Determining if the CPUID instruction Is Supported.
- Determining Basic Request Types Supported.
- Determining Extended Request Types Supported.
- The Basic Request Types.
- Request Type 1.
- Request Type 2.
- Request Type 3.
- Request Type 4.
- Request Type 5.
- The Extended Request Types.
- Enhanced Processor Signature.

This Chapter
This chapter provides a detailed description of System Management Mode (SMM). It includes:

- What Falls Under the Heading of System Management?
- The Genesis of SMM.
- SMM Has Its Own Private Memory Space.
- The Basic Elements of SMM.
- How the Processor Knows the SM Memory Start Address.
- The Organization of SM RAM.
The Unabridged Pentium® 4

- Entering SMM.
- Exiting SMM.
- The Auto Halt Restart Feature.
- The IO Instruction Restart Feature.
- Caching from SM Memory.
- Setting Up the SMI Handler in SM Memory.
- Relocating the SM RAM Base Address.
- SMM in an MP System.

The Next Chapter

This chapter provides a complete description of the Local and IO APICs. It includes:

- Message Transfer Mechanism Prior to the Pentium® 4.
- Message Transfer Mechanism Starting with the Pentium® 4.
- A Short History of the APIC.
- Detecting the Presence and Version of the Local APIC.
- Enabling/Disabling the Local APIC.
- Local Cluster and APIC ID Assignment.
- Local Interrupt Sources.
- Remote Interrupt Sources.
- Introduction to Interrupt Priority.
- An Intro to Edge-Triggered Interrupts.
- An Intro to Level-Sensitive Interrupts.
- The Local APIC Register Set.
- Locally Generated Interrupts.
- Task and Processor Priority.
- Interrupt Messages.
- The IO APIC.
- Message Signaled Interrupts (MSI).
- The FSB Message Format.
- The APIC Bus Message Format.
- The Spurious Interrupt Vector.
- The Agents in an Interrupt Message Transaction.
- BSP Selection Process.
- The APIC, the MPS and ACPI.
Chapter 60: System Management Mode (SMM)

What Falls Under the Heading of System Management?

The types of operations that typically fall under the heading of System Management are power management and management of the system’s thermal environment (e.g., temperature monitoring in the platform’s various thermal zones and fan control). It should be stressed, however, that system management is not necessarily limited to these specific areas.

The following are some example situations that would require action by the SM handler program:

- A laptop chipset implements a timer that tracks how long it’s been since the hard drive was last accessed. If this timer should elapse, the chipset generates an SMI (System Management Interrupt) to the processor to invoke the SM handler program. In the handler, software checks a chipset-specific status register to determine the cause of the SMI (in this case, a prolonged cessation of accesses to the hard drive). In response, the SM handler issues a command to the hard disk controller to spin down the spindle motor (to save on energy consumption).

- A laptop chipset implements a timer that tracks how long it’s been since the keyboard and/or mouse was used. If this timer should elapse, the chipset generates an SMI to the processor to invoke the SM handler program. In the handler, software checks a chipset-specific status register to determine the cause of the SMI (in this case, a prolonged cessation of user interaction). In response, the SM handler issues a command to the display controller to dim or turn off the display’s backlighting (to save on energy consumption).

- In a server platform, the chipset or system board logic detects that a thermal sensor in a specific zone of the platform is experiencing a rise in temperature. It generates an SMI to the processor to invoke the SM handler program. In the handler, software checks a chipset-specific status register to determine the cause of the SMI (in this case, a potential overheat condition). In response, the SM handler issues a command to the system board’s fan control logic to turn on an exhaust fan in that zone.

The Genesis of SMM

Intel® first implemented SMM in the 386SL processor and has not changed very much since then. While it was not present in the earlier 486 models, it was implemented in all of the later models of the 486 and in all subsequent IA32 processors. In all IA32 processors, SMM is entered by generating an SMI (System
Management Interrupt) to the processor. Prior to the P54C version of the Pentium® processor, the chipset could only deliver the interrupt to the processor by asserting the processor’s SMI# input pin. Starting with the P54C (which was the first IA32 processor to incorporate the Local APIC module) and up to and including the Pentium® III processor, the chipset could also deliver the interrupt to the processor by sending an SMI IPI (SMI Inter Processor Interrupt) message to the processor over the 3-wire APIC bus (see “The Local and IO APICs” on page 1497 for more information). With the advent of the Pentium® 4 processor, the 3-wire APIC bus was eliminated and IPIs (including the SMI IPI) are sent to and from a processor by performing a special memory write transaction on the FSB.

With the advent of the P54C processor, SMM was enhanced to include the IO Instruction Restart feature (described in this chapter).

The base address of the area of memory assigned to System Management Mode (SMM) has a default value of 30000h assigned. While it could be reprogrammed on the earlier IA32 processors, the newly assigned address had to be aligned on an address that was evenly divisible by 32K. Starting with the Pentium® Pro, this constraint was eliminated.

SMM Has Its Own Private Memory Space

Prior to the generation of an SMI to the processor, the chipset directs all memory accesses generated by the processor to system RAM memory:

- When interrupted by an SMI, the processor signals to the chipset that all subsequent memory accesses generated by the processor are to be directed to a special, separate area of memory referred to as SM RAM.
- Upon concluding the execution of the SM handler program, the processor signals to the chipset that all subsequent memory accesses generated by the processor are to be directed to system RAM memory rather than SM RAM.

The platform vendor’s implementation of SM RAM can be up to 4GB in size.

The Basic Elements of SMM

The following is a list of the basic elements associated with SMM:

- The processor’s SMI# input.
- The APIC SMI IPI message.
The chipset/system board logic responsible for monitoring conditions within the platform that might required an invocation of the SM handler program.

The chipset’s ability to assert SMI# to the processor to invoke the SMI program.

The chipset’s ability to send an SMI IPI message to the processor to invoke the SMI program.

The Resume (RSM) instruction.

The SM RAM area.

The 512-byte processor context state save/restore area (i.e., data structure) in memory.

The SMI Acknowledge message was added to the message repertoire of the Special transaction.

The processor’s SMMEM# output (also referred to as the EFX4# output).

The chipset’s ability to discern when the processor is addressing regular RAM memory versus when it is addressing SM RAM memory. It does this by monitoring for the processor’s issuance of the SMI Acknowledge message and whether or not the processor is asserting the SMMEM# signal during a processor-initiated memory transaction.

A Very Simple Example Scenario

Assume that the platform logic (i.e., the chipset or the system board logic) detects a condition that requires management by the SM handler program (see “What Falls Under the Heading of System Management?” on page 1465 for some examples). In response, an SMI is generated to the processor. The following sequence of events occurs (this description assumes that the processor is a Pentium® Pro or a subsequent IA32 processor):

1. The processor recognizes the SMI on the next instruction boundary and suspends execution of the currently executing program.
2. The processor generates a Special transaction on its FSB and outputs the SMI Acknowledge message on its Byte Enable outputs to inform the chipset that until the processor generates another SMI Acknowledge message, all memory accesses generated by the processor are to be directed to SM memory rather than to regular RAM memory.
3. The processor then generates a series of memory write transactions on the FSB to store a snapshot of the processor’s registers in the 512-byte State Save Area of SM memory. This is done so the processor can, at the conclusion of the execution of the SM handler program, resume execution of the interrupted Real Mode or Protected Mode program.
4. The processor then begins to execute the SM handler program.
The Previous Chapter
This chapter provided a detailed description of System Management Mode (SMM). It included:

- What Falls Under the Heading of System Management?
- The Genesis of SMM.
- SMM Has Its Own Private Memory Space.
- The Basic Elements of SMM.
- How the Processor Knows the SM Memory Start Address.
- The Organization of SM RAM.
- Entering SMM.
- Exiting SMM.
- The Auto Halt Restart Feature.
- The IO Instruction Restart Feature.
- Caching from SM Memory.
- Setting Up the SMI Handler in SM Memory.
- Relocating the SM RAM Base Address.
- SMM in an MP System.

This Chapter
This chapter provides a complete description of the Local and IO APICs. It includes:

- Message Transfer Mechanism Prior to the Pentium® 4.
- Message Transfer Mechanism Starting with the Pentium® 4.
- A Short History of the APIC.
- Detecting the Presence and Version of the Local APIC.
- Enabling/Disabling the Local APIC.
- Local Cluster and APIC ID Assignment.
- Local Interrupt Sources.
Remote Interrupt Sources.
Introduction to Interrupt Priority.
An Intro to Edge-Triggered Interrupts.
An Intro to Level-Sensitive Interrupts.
The Local APIC Register Set.
Locally Generated Interrupts.
Task and Processor Priority.
Interrupt Messages.
The IO APIC.
Message Signaled Interrupts (MSI).
The FSB Message Format.
The APIC Bus Message Format.
The Spurious Interrupt Vector.
The Agents in an Interrupt Message Transaction.
BSP Selection Process.
The APIC, the MPS and ACPI.

Before the Advent of the APIC

Most IA32-based systems incorporate an Interrupt Controller that receives interrupt requests from IO devices and passes them to the processor (or, in a multiprocessor system, to one or more of the processors). The Interrupt Controller typically consists of one of the following:

- In a single processor PC-AT compatible machine, a pair of cascaded 8259A PICs (Programmable Interrupt Controllers). See Figure 61-1 on page 1500.
- In a multiprocessor system, an IO APIC module. See Figure 61-5 on page 1506.

Refer to Figure 61-2 on page 1501. In older chipsets, the Interrupt Controller was incorporated in the PCI-to-ISA Bridge (commonly referred to as the South Bridge), and in the ICH (IO Control Hub) in later chipsets. This was a strategically convenient place for it because the interrupt requests from PCI and ISA devices could easily be connected to it.

Assuming that the system is a single processor, PC-AT compatible machine (Figure 61-1 on page 1500), the master 8259A asserts its INTR (Interrupt Request) output when it detects any interrupt requests from device adapters. This is connected to the INTR pin (also referred to as the LINT0 pin) on the processor. In response to its assertion, the processor takes the following actions:
Chapter 61: The Local and IO APICs

1. Assuming that recognition of external interrupts is enabled (in other words, the programmer has not executed a CLI instruction), the processor will recognize the request when it completes the execution of the current instruction.

2. The processor temporarily ceases execution of the interrupted program.

3. The processor generates an Interrupt Acknowledge transaction to obtain the interrupt vector associated with the highest priority request from the Interrupt Controller.

4. The North Bridge passes the transaction to the PCI bus to make it visible to the chip that contains the Interrupt Controller (i.e., the South Bridge in the example system).

5. The Interrupt Controller supplies the 8-bit interrupt vector associated with the highest priority request to the North Bridge.

6. The North Bridge supplies the interrupt vector to the processor.

7. The processor uses the 8-bit vector as an index into the IDT in memory and reads the CS:EIP value from the selected entry. This CS:EIP value points to the entry point of the interrupt handler within the associated device’s driver.

8. The processor pushes the contents of its CS, EIP and EFlags registers into stack memory (to mark its place in the interrupted program).

9. The processor then automatically disables recognition of additional external hardware interrupts (i.e., it clears EFlags[IF] to 0).

10. Using the new CS:EIP value, the processor starts fetching the instructions that comprise the interrupt handler and executes it.

A detailed description of the dual 8259A PICs can be found in chapter 18 of the MindShare book entitled *ISA System Architecture, Third Edition.*
Figure 61-1: Legacy PC-AT Compatible Interrupt Controllers
MP Systems Need a Better Interrupt Distribution Mechanism

Introduction

As just described, the legacy interrupt delivery mechanism interrupts the processor by asserting the processor’s INTR input signal. The processor recognizes the interrupt on the next instruction boundary and must then perform an Interrupt Acknowledge transaction on its FSB to obtain the interrupt vector from the interrupt controller. This method is inefficient in the following ways:

- Refer to Figure 61-3 on page 1502. Using the INTR signal to deliver interrupts to the processors in a multiprocessor (MP) system is a poor approach. All of the interrupts would be delivered to the processor that is connected to the output of the master 8259A PIC and that processor would have the burden of servicing all hardware interrupts. In an MP system, any proces-
# Acronyms

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/D</td>
<td>Analog-to-Digital converter.</td>
</tr>
<tr>
<td>AC</td>
<td>Alignment Check.</td>
</tr>
<tr>
<td>AC '97 Link</td>
<td>Audio Codec (AC) '97 Link.</td>
</tr>
<tr>
<td>ACPI</td>
<td>Advanced Configuration and Power Interface.</td>
</tr>
<tr>
<td>AF</td>
<td>Auxiliary Carry bit in the EFlags register.</td>
</tr>
<tr>
<td>AGP</td>
<td>Accelerated Graphics Port.</td>
</tr>
<tr>
<td>AGTL+</td>
<td>Assisted Gunning Transceiver Logic Plus.</td>
</tr>
<tr>
<td>ALU</td>
<td>Arithmetic Logic Unit.</td>
</tr>
<tr>
<td>AM</td>
<td>The Alignment Mask bit in CR0.</td>
</tr>
<tr>
<td>AOS</td>
<td>Array of Structures.</td>
</tr>
<tr>
<td>AP</td>
<td>Application Processor (as opposed to Boot Strap Processor).</td>
</tr>
<tr>
<td>APIC</td>
<td>Advanced Programmable Interrupt Controller.</td>
</tr>
<tr>
<td>APR</td>
<td>Arbitration Priority Register.</td>
</tr>
<tr>
<td>ASZ</td>
<td>Address Size field in a FSB transaction.</td>
</tr>
<tr>
<td>ATC</td>
<td>Advanced Transfer Cache (the L2 Cache).</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ATTR[7:0]</td>
<td>The Attribute signals indicate the type of memory (UC, WC, WP, WT or WB) being addressed in a FSB transaction.</td>
</tr>
</tbody>
</table>
| B       | • The Busy bit in a TSS descriptor.  
         | • The Big bit in a Stack segment descriptor.                                |
| BBL     | Back Side Bus Logic that connects the L2 Cache to the processor core.         |
| BCLK    | FSB Bus Clock.                                                              |
| BE[7:0] | The processor’s Byte Enable outputs:  
         | • Indicates the bytes being addressed in a memory, IO, or BTM transaction.  
         | • Indicates the message type in a Special transaction.                      |
| BGA     | Ball Grid Array package.                                                    |
| BIOS ROM| Binary Input Output System Read-Only Memory.                                |
| BIOS Update | This refers to the Microcode Update feature implemented in the P6 and Pentium® 4 processor families. |
| BIPI    | Bootstrap Inter Processor Interrupt message (only applies to the P6 processor family). |
| BIST    | Built-In Self-Test.                                                        |
| BOS     | Bottom of Stack.                                                           |
| BPU     | Branch Prediction Unit.                                                    |
| BSB     | The Back Side Bus that connects the L2 Cache to the processor core.          |
| BSP     | Boot Strap Processor.                                                      |
| BSQ     | Bus Sequence Queue (another name for the processor’s FSB Interface Unit).    |
| BSU     | Bus Sequence Unit (another name for the processor’s FSB Interface Unit).    |
## Acronyms

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTB</td>
<td>Branch Target Buffer. This is the dynamic branch predictor that maintains branch history.</td>
</tr>
<tr>
<td>BTM</td>
<td>Branch Trace Message transaction.</td>
</tr>
<tr>
<td>BTS</td>
<td>Branch Trace Store feature.</td>
</tr>
<tr>
<td>Byte</td>
<td>8-bits.</td>
</tr>
<tr>
<td>C</td>
<td>The Conforming bit in a code segment descriptor.</td>
</tr>
<tr>
<td>C/D</td>
<td>The Code or Data bit in a non-system segment descriptor.</td>
</tr>
<tr>
<td>CCCR</td>
<td>Counter Configuration Control Register.</td>
</tr>
<tr>
<td>CD</td>
<td>The Cache Disable bit in CR0.</td>
</tr>
<tr>
<td>CESR</td>
<td>Counter Event Select Register.</td>
</tr>
<tr>
<td>CF</td>
<td>The Carry Flag bit in the EFlags register.</td>
</tr>
<tr>
<td>CID</td>
<td>Context ID.</td>
</tr>
<tr>
<td>CISC</td>
<td>Complex Instruction Set Computer.</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metallic Oxide.</td>
</tr>
<tr>
<td>CMP</td>
<td>Chip Multiprocessing.</td>
</tr>
<tr>
<td>CPI</td>
<td>Clocks per Instruction.</td>
</tr>
<tr>
<td>CPL</td>
<td>Current Privilege Level.</td>
</tr>
<tr>
<td>CR</td>
<td>Control Register.</td>
</tr>
<tr>
<td>CR0</td>
<td>Control Register 0.</td>
</tr>
<tr>
<td>CR2</td>
<td>Control Register 1.</td>
</tr>
<tr>
<td>CR3</td>
<td>Control Register 3.</td>
</tr>
<tr>
<td>CR4</td>
<td>Control Register 4.</td>
</tr>
</tbody>
</table>
### The Unabridged Pentium® 4

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRU</td>
<td>Cache References Unit.</td>
</tr>
<tr>
<td>CS</td>
<td>Code Segment register.</td>
</tr>
<tr>
<td>CWR</td>
<td>x87 FPU’s Control Word Register.</td>
</tr>
<tr>
<td>D/B</td>
<td>Default or Big bit in a non-system segment descriptor.</td>
</tr>
<tr>
<td>DAC</td>
<td>Data cache Access Control unit.</td>
</tr>
<tr>
<td>DAT</td>
<td>The IO APIC’s Data register.</td>
</tr>
<tr>
<td>DAZ</td>
<td>The Denormals Are Zeros bit in the MXCSR.</td>
</tr>
<tr>
<td>DDR</td>
<td>Double Data Rate memory.</td>
</tr>
<tr>
<td>DE</td>
<td>• The Debug Extensions bit in CR4.</td>
</tr>
<tr>
<td></td>
<td>• The Denormal operand error bit in the x87 FPU’s Status register.</td>
</tr>
<tr>
<td></td>
<td>• The Denormal operand error bit in the MXCSR.</td>
</tr>
<tr>
<td>DEP</td>
<td>Double Extended Precision 80-bit FP number.</td>
</tr>
<tr>
<td>DF</td>
<td>The Direction Flag bit in the EFlags register.</td>
</tr>
<tr>
<td>DFR</td>
<td>The Destination Format Register.</td>
</tr>
<tr>
<td>DIBA</td>
<td>Dual Independent Bus Architecture.</td>
</tr>
<tr>
<td>DID</td>
<td>Deferred ID.</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access.</td>
</tr>
<tr>
<td>DNA</td>
<td>The Device Not Available exception.</td>
</tr>
<tr>
<td>Double Qword</td>
<td>16 bytes starting on an address divisible by 16.</td>
</tr>
<tr>
<td>DP</td>
<td>A 64-bit Double Precision FP number.</td>
</tr>
<tr>
<td>DPL</td>
<td>Data Prefetch Logic (refers to the hardware-based prefetcher that prefetches data into the processor’s top-level cache.</td>
</tr>
<tr>
<td>DR6</td>
<td>The Debug Status register.</td>
</tr>
</tbody>
</table>
## Acronyms

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR7</td>
<td>The Debug Control register.</td>
</tr>
<tr>
<td>DR[3:0]</td>
<td>The four Debug breakpoint address registers.</td>
</tr>
<tr>
<td>DR[7:0]</td>
<td>The Debug register set.</td>
</tr>
<tr>
<td>DS</td>
<td>The debug store feature.</td>
</tr>
<tr>
<td></td>
<td>The Data Segment register.</td>
</tr>
<tr>
<td>DSE</td>
<td>The Dedicated Stack Engine.</td>
</tr>
<tr>
<td>DTLB</td>
<td>The Data Translation Lookaside Buffer.</td>
</tr>
<tr>
<td>Dword</td>
<td>A 32-bit data object.</td>
</tr>
<tr>
<td>EBC</td>
<td>External Bus Control (refers to the FSB control logic).</td>
</tr>
<tr>
<td>EBL</td>
<td>External Bus Logic (refers to the FSB Interface Unit).</td>
</tr>
<tr>
<td>EBP</td>
<td>Extended Base Pointer register.</td>
</tr>
<tr>
<td>ECC</td>
<td>Error Code Correcting memory.</td>
</tr>
<tr>
<td>EDI</td>
<td>Extended Destination Index register.</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Electrically Erasable Programmable Read-Only Memory.</td>
</tr>
<tr>
<td>EEROM</td>
<td>Electrically Eraseable Read-Only Memory.</td>
</tr>
<tr>
<td>EFlags</td>
<td>Extended Flags register.</td>
</tr>
<tr>
<td>EIP</td>
<td>Extended Instruction Pointer register.</td>
</tr>
<tr>
<td>EIPV</td>
<td>Error Instruction Pointer Valid bit.</td>
</tr>
<tr>
<td>EM</td>
<td>The FL Emulation bit in CR0.</td>
</tr>
<tr>
<td>EMSB</td>
<td>Enhanced Mode Scaleable Bus (i.e., the FSB).</td>
</tr>
<tr>
<td>EOI</td>
<td>End-of-Interrupt.</td>
</tr>
<tr>
<td>EOIR</td>
<td>End-of-Interrupt Register.</td>
</tr>
<tr>
<td>ES</td>
<td>The E Data Segment register.</td>
</tr>
</tbody>
</table>
Symbols
µop Decode events 1412
µop Fusion 1437
µops 10, 843
“u” and the “v” pipelines 465

Numerics
16-bit Code Optimization, Pentium II 672
286 255
286 DOS Extender Programs 485
286 DOS Extenders on Post-286 Processors 487
2MB Page 559, 561
2-wide SSE instructions 754
386 Power-Up State 66
386SL 1465
3D Rasterization 790
486 412
486 FS B 415
486 Instruction Set Changes 456
486DX 412
486DX2 (WB) 412
486DX2 (WT) 412
486DX4 412
486SX 412
486SX2 412
487SX 412
4MB Pages 244, 501, 735
4-wide SSE instructions 754
8088 255
8259A interrupt controller 17, 255, 351, 1303, 1498
855MCH 1426

A
A bit 224
A[15:8]# 1209, 1220
A[23:16]# 1209, 1220
A[31:24]# 1209, 1220
A[35:32]# 1202, 1208, 1213
A[35:32]# 1219
A[7:3]# 1209, 1220
A20 Mask 419
A20M# 89, 419, 1314
Aborts 260
AC ‘97 Link 17
Accelerated Graphics Port 16
Accessed Bit 224, 243
ACPI 833, 1597
ACPI Table 833, 1516
ACPI table 1596
Active Thread field 1399
Address and Data Strobes 1119
Address bit 20 Mask 1314
address bus width, Pentium 4 1429
Address Size 1217
Address Size Override Prefix 339
Address Strobe 1203
Address Strobe signals 1120
ADD RV 595
AD S# 1213
ADSTB[1:0]# 1120, 1203
Advanced Configuration and Power Interface 1516
Advanced Stack Management 1439
Advanced Transfer Cache 746
Agent ID 1141, 1221
Agent ID assignment, Pentium 4 1150
Agent Type 1221
AGP 16
AGTL+ 723, 1116, 1127, 1429
AGTL+ Sample Point 1131
AGTL+ Setup and Hold Specs 1134
Alarm output of the real-time clock 258
Alignment Check 269
Alignment Check Exception 321, 448, 460
Alignment Checking 448
Alignment Checking, SSE 792
All Excluding Self 1559
All Including Self 1559
Alternate (Fast) Hot Reset 486
analyzer 1309
AP[1:0]# 1202
APIC 839, 1308, 1497
Index

APIC bus 1338, 1503
APIC Enhancements, Pentium Pro 569
APIC Global Enable/Disable bit 1510, 1592
APIC ID 1338, 1504
APIC ID assignment 1513
APIC ID, changing the Local 1515
APIC register set, changing the base address of the Local 569
APIC timer 1504
APIC, enable/disable the Local 569
APIC, history 1507
APIC, Local 548
APIC_BASE MSR 569, 613, 696, 1348, 1507
APIC’s Register Set, Local 510
Application Processors 833, 1596
APR 1528
AP’s 833, 1596
Arbitration Algorithm, Symmetric 1152
Arbitration Event 1163
Arbitration Priority Register 1528
Arbitration, Priority Agent 1165
Arbitration, symmetric 1149
Array Bounds Check Exception 297
Assist 1410
Assisted GTL+ 1116, 1130
ASZ 1216, 1217
ATC 746
At-Retirement Counting 1383, 1395
ATTR[7:0]# 587, 1209, 1220
Attribute field 1220
Attribute signals 1209
Audio Codec (AC) ‘97 Link 17
Auto Halt Restart Feature 1484
Auto HALT Restart Field, SMM 1473
AutoHalt Power Down State 686
Available Bit 131

B

B 441
Back Side Bus 669, 839
Backside Bus (BSB) Interface Unit 547
Backside Bus Logic 696
Backside Bus Logic (BBL) Registers 702
Balias 820, 1426
Base address, Local APIC 1524
base address, SM memory 1468
BBL 696
BBL Registers 702
BBL.CR_ADDR[63:0] 703
BBL.CR_BUSY 705
BBL.CR_CTL 704
BBL.CR_CTL3 706
BBL.CR_D0[63:0] 702
BBL.CR_D1[63:0] 702
BBL.CR_D2[63:0] 702
BBL.CR_D3[63:0] 698, 703
BBL.CR.DECC[63:0] 703
BBL.CR_TRIG 705
BCLK 693
BCLK frequency 1117
BCLK Is a Differential Signal 1117
BCLK[1:0] 1117
BCLK0 1118
BD bit 380, 381
BE[7:0]# 1209, 1220
Big Real Mode 90
BINIT# 605, 1135, 1314
BIOS 24, 35
BIOS Data Area 486
BIOS ROM 17
BIOS, Microcode Image Management 640
BIOS_SIGN 615, 698
Block Next Request 1190
Blocking transactions 1189
Blue Screen Compositing 527
BNR# 1135, 1190, 1434
BNR# and a Debug Tool 1191
BNR# Behavior at Powerup 1197
BNR# Behavior During Runtime 1199
Bogus 1409
boot ROM 14
Boot Strap Processor 832, 1595
boot.ini file 566

1620
INDEX

BOOTSELECT 1315
BOUND instruction 262
Bounds Exception 297
BP 621
BPM 621
BPM and BP Pin Usage, P6 622
BPM[3:0] # 1315
BPRI# 1166, 1429
BPU 1351
BPU counter group 1403
BR[3:0] # 679
BR[3:1] # 1159
BR0# 1156, 1180
Branch Hints 1332
Branch Prediction Unit 1351
Branch Prediction, Pentium M enhanced 1436
Branch Predictor, Indirect 1436
Branch Recording Registers 702
Branch Recording registers, P6 620
Branch Recording Registers, Pentium 4 1361
Branch Target Buffer (BTB) 467
Branch Trace Message Transaction 547, 1210, 1302, 1366
Branch Trace Messaging, enable P6 623
Branch Trace Store (BTS) buffer 1367
Branch Trace Store (BTS) facility 1366
Branch, Exception, Interrupt Recording Facility, enable P6 623
Branches, mispredicted 787
Brand Index 1447
Brand Index Request 793
Brand String 1459
Breakpoint 262, 268, 275
Breakpoint Exception 295
breakpoint instruction 376
BREQ0# 1159
BS bit 382
BSB 669, 746, 839
BSB and the L2 Cache, Pentium II 678
BSB Interface Unit 547
BSEL# 679
BSEL[1:0] 1117, 1315
BSEN# 482
BSP 832, 1595
BSP bit 1507
BSP Selection Process, Pentium 4 1595
BSP Selection, P6 832
BSP Selection, Pentium 4 832
BSP, detecting the 570
BSU 1353
BTSWAP 457, 459
BT bit 382
BTB 467
BTM 1210, 1302, 1366
BTM Capability, enabling 1310
BTM transaction 1309
BTS absolute maximum 1367
BTS buffer 1367
BTS buffer base 1367
BTS feature 1366
BTS Feature, enabling the 1369
BTS Index 1367
BTS interrupt threshold 1368
BTS Record Format 1370
Burst Transaction 472
Bus Parking 1156
Bus Sequence Unit 1353
BUSCHK# 504
Busy 186, 1153
Busy bit 202, 204
Busy bit, TSS 186
Busy/Idle Indicator 1153
Byte Enables 42, 469, 1220, 1306
Byte Swap (BSWAP) instruction 457
C
C/D bit 135
Cache and TLB Information 1452
Cache Architecture 399
Cache Data, PIROM 726
Cache Directory Entry 388
Cache Error Protection, Pentium II 671
Index

Cache Line 387
Cache Line Flush Instruction 1326
Cache Line, locking a 1184
Cache Miss 387
Cache Real Estate Management 406
Cache References Unit 1354
Cache Snooping 393
Cache, Eight-Way Set Associative 404
Cache, Four-Way Set Associative 402
Cache, Fully-Associative 399
Cache, Non-Blocking 410
Cache, Split 409, 476
Cache, Two-Way Set Associative 400
Cache, Unified 408
Cache, Write Back 391
Cache, Write-Through 388
Cache-related errors 590
Cache-Related Instructions, SSE 773
Caching Overview 385
Caching Rules, paging 244
Call Gate 143
Call Gate Example 145
Call Gate Privilege Check 151
Cartridge 658, 662
Cascade, Performance Counter 1401
Cascading, extended performance counter 1406
Castout 407
Castout of a Modified Line 407
Castout of an E or S Line 408
Catastrophic Shutdown Detector 1341
CC[3:0] 441
CCCR 1372, 1375, 1397
CCCR[Compare] 1398
CCCR[Complement] 1398
CCCR[Edge] 1398
CCCR[ESCR Select] 1398
CCCR[Threshold] 1398
CD 454
Celeron 660
Celeron M Dothan 820
Celeron Northwood 817
Celeron Prescott 817
Celeron Willamette 817
Celeron, Pentium 4 817
Celeron, Pentium III 742
Celeron, the introduction of the 679
Central Agent Transactions, Pentium 4 1301
Centrino 1426, 1435
CESR 506, 515, 612
Chroma-Key 527
Classes, Interrupt and Exception 302
Clear Interrupt Enable 253, 345
CLI Handling in VM86 extended mode 492
CLI instruction 23, 253, 345
CLI/STI Instruction Handling in VM86 Mode, efficient 492
clock generator 1117
Clock Modulation, software-controlled 1344
Clocks, counting 1417
Clockticks 1417
Clockticks, nonhalted 1417, 1418
Clockticks, nonsleep 1418, 1419
Cluster 831, 1279
Cluster Bridge 1166
Cluster ID 1504
Cluster ID assignment 1513
Cluster Model 1563
Cluster Number 509
CMOS RAM 486
CMOV 626, 627
CMP 787
CMPXCHG 456, 457
CMPXCHG8B 517
Code Segment 73, 133
Code Segment Descriptor 134
Code segment limit violation 268
Code Segment, Conforming 141
Code Segment, Non-Conforming 141
Code Segment, Real Mode 73
Code Segment, types of 127
<table>
<thead>
<tr>
<th>Command Line Interpreter 23</th>
<th>CPUID Extended L2 Cache Features Function 1460</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND.COM 24</td>
<td>CPUID Extended Request Types 1459</td>
</tr>
<tr>
<td>Common Clock Signals 1118</td>
<td>CPUID Instruction 457</td>
</tr>
<tr>
<td>COMP[1:0] 1315</td>
<td>CPUID Instruction, detecting support for the 1444</td>
</tr>
<tr>
<td>Compare 1399</td>
<td>CPUID Request Types Supported 1446</td>
</tr>
<tr>
<td>Compare and Exchange (CMPXCHG) Instruction 456</td>
<td>CPUID, The Pentium 4 enhancements to 1332</td>
</tr>
<tr>
<td>Compare bit 1409</td>
<td>CR[NE] 318</td>
</tr>
<tr>
<td>Complement 1399</td>
<td>CR0 Cache Control Bits 454</td>
</tr>
<tr>
<td>Complement bit 1409</td>
<td>CR0[AM] 323, 448</td>
</tr>
<tr>
<td>Compute Sum of Absolute Differences 790</td>
<td>CR0[EM] 793</td>
</tr>
<tr>
<td>Condition Code 441</td>
<td>CR0[ET] 434</td>
</tr>
<tr>
<td>Conditional Branches, eliminating 527</td>
<td>CR0[MP] 793</td>
</tr>
<tr>
<td>Conditional Move 626</td>
<td>CR0[NE] 434, 445</td>
</tr>
<tr>
<td>Conforming bit 135</td>
<td>CR0[PG] 219</td>
</tr>
<tr>
<td>Conforming Code Segment 141</td>
<td>CR0[WP] 449</td>
</tr>
<tr>
<td>Coppermine 746</td>
<td>CR2 316</td>
</tr>
<tr>
<td>Coprocessor segment overrun abort 256, 264, 304</td>
<td>CR3 220, 451, 557</td>
</tr>
<tr>
<td>Copy-on-Write Strategy 451</td>
<td>CR3 Field, TSS 182</td>
</tr>
<tr>
<td>Core 838</td>
<td>CR3[PCD] 449</td>
</tr>
<tr>
<td>Core and Bus Frequencies, Pentium II 676 core, Pentium II 670</td>
<td>CR3[PWT] 449</td>
</tr>
<tr>
<td>Counter Configuration Control Registers 1372, 1397</td>
<td>CR4 455</td>
</tr>
<tr>
<td>Counter Event Select Register 506</td>
<td>CR4[MCE] 504, 589, 1207, 1269, 1270, 1273, 1274</td>
</tr>
<tr>
<td>Counter Mask 608</td>
<td>CR4[OSFXSR] 770, 793</td>
</tr>
<tr>
<td>Covington 661</td>
<td>CR4[OSXMMEXCPT] 327, 771, 793</td>
</tr>
<tr>
<td>CPI 1417</td>
<td>CR4[PAE] 732</td>
</tr>
<tr>
<td>CPI, nominal 1418</td>
<td>CR4[PCE] 628, 1406</td>
</tr>
<tr>
<td>CPI, non-halted 1418</td>
<td>CR4[PGE] 568</td>
</tr>
<tr>
<td>CPL 139</td>
<td>CR4[PSE] 732</td>
</tr>
<tr>
<td>CPL, definition of 140</td>
<td>CR4[PVI] 497</td>
</tr>
<tr>
<td>CPU Arbitration 1149</td>
<td>CR4[TSD] 498, 517</td>
</tr>
<tr>
<td>CPU Identification 1443</td>
<td>CR4[VME] 490</td>
</tr>
<tr>
<td>CPUID Basic Request Types 1446</td>
<td>CRU 1354</td>
</tr>
<tr>
<td>CPUID Extended Address Sizes Function 1460</td>
<td>CS 73</td>
</tr>
<tr>
<td>CPUID extended feature bits 1459</td>
<td>Current Count Register 1534</td>
</tr>
<tr>
<td>cycles per instruction 1417</td>
<td>Current Privilege Level 139</td>
</tr>
</tbody>
</table>
Index

D
D bit 224
D[63:0]# 1123, 1247
DAC 1354
DAT 1574
Data Bus 1247
Data Bus Busy 1247
Data Bus Inversion 1123, 1125, 1247
Data Bus Parity, Pentium 4 1247, 1270
Data Bus Power 1428, 1429
Data Bus Reset 1315
Data cache Access Control unit 1354
Data Copy Operations, optimized 752
Data Phase 1241
Data Phase Signal Group 1246
Data Phase Wait States 1266
Data Phase(s) 1146
Data Ready 1247
Data Register 1574
Data Segment 73
Data Segment Privilege Check 159
Data Segment, Real Mode 81
Data Segment, types of 126
Data Segments 158
Data Segments, Real Mode 84
Data Strobe Signals 1123, 1247
Data Transfer Length 1217
DAZ 768
DAZ Mode 768, 1336
DBI[3:0]# 1247
DBINST# 481
DBR# 1315
DBSY# 1247
DBSY# Deassertion, relaxed 1264
DDR 17, 824
DE 440
Debug breakpoint 275
Debug Exception 262, 293, 376
Debug Extension 497
denormal numbers 762, 763
denormalized operand, FPU 319
denormals Are Zeros Mode 768, 1336
DEP 761
deschutes 540, 541, 660
descriptor privilege level 130, 139
descriptor tables 114
destination field 1558, 1582
destination format register 1531, 1562
destination ID 1587
destination mode 1504, 1514, 1560, 1584, 1588
destination mode, logical 1505
destination mode, physical 1504
destination shorthand 1559
device drivers 14
Debug Store (DS) Mechanism 1366, 1414
Debug Store Save Area 1366
debug tool 380
Debug Tool and BNR# 1191
DEBUGCTL 614, 1310, 1366
DEBUGCTL, P6 621
DEBUGCTL[BTF] 625
DEBUGCTL[BTS] 1369
DEBUGCTLMSR 697
Debugging features 1324
dedicated stack engine 1439
deep sleep state 693, 756, 1428, 1429
deeper sleep state 1428, 1430
default/big bit 123
defer 1282
defer enable 1222
defer# 1229
defered ID 1209, 1220, 1284
Deferred Reply Transaction 547, 1210, 1281, 1282, 1283, 1291
defered response 1141, 1245, 1281, 1282, 1288
defurred transaction 841, 1277
delivery mode 1541, 1561, 1589
delivery status 1543, 1560, 1583, 1588
den# 1222
denormal numbers 762, 763
denormalized operand, FPU 319
denormals are zeros mode 768, 1336
dep 761
descuates 540, 541, 660
descriptor privilege level 130, 139
descriptor tables 114
destination field 1558, 1582
destination format register 1531, 1562
destination ID 1587
destination mode 1504, 1514, 1560, 1584, 1588
destination mode, logical 1505
destination mode, physical 1504
destination shorthand 1559
device drivers 14
Device Not Available exception 263, 268, 299
DFR 1531, 1562
DFR's Model field 1562
DIBA 658, 839
DID 1220
DID[3:0]# 1221
DID[6:4]# 1221
DID[7:0]# 1209
DID# 1221
Dirty bit 224, 243
Divide Configuration Register 1534
Divide-by-Zero exception 261, 292
Divide-by-zero, FPU 319
Dixon 661
DM 438
DMA channel 30
DMA controller 24
DNA exception 268, 299
DOS Application 330
DOS environment 211
DOS Extender Programs 485, 487
DOS Memory 335
DOSSHELL 24
Dothan 820, 1440
Double Extended Precision format 443, 761
double fault exception 256, 263, 301
Double-Data Rate RAM 17
double-pumped 1120
DP FP Number Representation 1334
DP[3:0]# 1247, 1270, 1273, 1274
DPL 130, 139
DPL, definition of 140
DPSLP# 1428
DPWR# 1428, 1429
DR7 1474
DR7 Bits Fields 378
DRDY# 1247
DS 73
DS (Debug Store) Save Area 1366
DS Buffer Management Area Format 1367
DS feature detection 1367
DS Feature is disabled sometimes 1415
DS Feature, setting up the 1367
DS Mechanism 1366, 1414
DSE 1439
DSTBN[3:0]# 1123, 1247
DSTBP[3:0]# 1123, 1247
Dual Independent Bus Architecture 658, 839
DVD player 789
Dword Count 144

E
E 609
EBC_Soft_Poweron 1207, 1269
EBL_CR_POWERON MSR 594, 614, 697
ECC 589, 670, 671, 746
Edge 1400
Edge and event count filtering 1409
Edge Detect bit 609
Edge-Triggered Interrupts 1522, 1569, 1583
EEPROM Select Address pins 729
EEPROM, Scratch 724
EFlags field, TSS 182
EFlags[AC] 323, 448
EFlags[ID] 1444
EFlags[IF] 254, 492, 1474
EFlags[OF] 269
EFlags[RF] 376
EFlags[TF] 625, 1474
EFlags[VIF] 493
EFlags[VM] 330
EIP field, TSS 182
EIPV bit 593
EMI 680
EMMS 532
Empty MMX state 532
EMSB 1138
EN 608
Enable bit 608
ENABLE_PEBs_MY_THR 1416

1625
Index

ENABLE_PEBS_OTH_THR 1416
End-of-Interrupt 255, 1523
End-of-Interrupt (EOI) register 1537
Enhanced Mode Scaleable Bus 1138
EOI 1523, 1537
EOI command 255
EOI Register 1530, 1575, 1578
EOIR 1575
Error Codes, exception 288
Error Instruction Pointer Valid bit 593
Error Logging Notes 605
Error Phase 841, 1142
Error Status Register 1533
Error Summary Status 441
ES 73, 441
ESCR 1351, 1352, 1353, 1354, 1355, 1372, 1375, 1382
ESCR Select 1399
ESCR[Event Mask] 1398
ESCR[Event Select] 1398
ESCR[OS] 1398
ESCR[USR] 1398
ESMA 731
ESP field, TSS 182
ESP0 153, 183
ESP1 183
ESP2 183
ESR 1533
Ethernet 16
Event Classes 1385, 1395, 1407
Event counting, Compare bit and 1409
Event counting, Complement bit and 1409
Event counting, Edge bit and 1409
Event Counting, halting 1407
Event counting, Non-Retirement 1407
Event counting, setting up 1408
Event counting, Threshold value and 1409
Event Filtering 1408
Event Mask 1384, 1385, 1395, 1407
event queue 29
Event Select 609, 1384
Event Select Control Register 1351, 1352, 1353, 1354, 1355, 1372, 1382
events, At-retirement 1373
events, non-retirement 1373
EVNTSEL0 615, 698
EVNTSEL1 615, 698
Exception 13d 536
Exception 14d 536
Exception 17 448
Exception 18d 536
exception 19 792
Exception 3 376
Exception 9 459
Exception Error Codes 288
Exception Occurs in VM86 Mode 359
Exception/Interrupt Priority 266
Exceptions 251, 260
Exceptions, 386 99
Exceptions, Pentium 4 1325
Exchange and Add (XADD) instruction 456
Exclusive state 391
EXF[4:0]# 1209, 1220
EXF0# 1222
EXF1# 1222
EXF2# 1222
EXF3# 1222
EXF4# 1222, 1478
Exponent 763
Extended Cascade Enable 1401
Extended Function bit 4 1478
Extended function signals 1209
Extended Functions 1222
Extended Functions field 1220
Extended Memory, accessing in Real Mode 87, 419
Extended Request field 1219
Extended Server Memory Architecture 731
External Bus (FSB) Control Frequency ID Register 1349
External Bus (FSB) Control Hard Power On register 1349
External Bus (FSB) Control Soft Power On register 1349
External Task Priority registers 1566
ExtINT interrupt delivery mode 1542

false register dependency 852
Fast Hot Reset 486
Fast Return from System Call 707
Fast String Enable 621
Fast System Call 708
Fast System Call Entry Point 707
Fast System Enter/Exit Registers 707, 1361
Faults 260
FCMOV 626, 627
FCOMI 626, 628
FCOMIP 628
FCW Register 438
Feature Data 727
Fence Instructions 1326
FERR# 1316
Filtering event counting 1408
Fixed interrupt delivery mode 1541, 1561
Fixed Interrupts 1522
FLAME counter group 1403
Flat Cluster Model 1563
Flat Model 247, 1563
Flat Real mode 90
Floppy interface 258
Flush 267, 1222, 1307
Flush Acknowledge 1222, 1308
FLUSH# 686, 689, 1308
FLUSH# and SMI# 1493
flushing 1212
Flush-to-Zero Mode 765, 768, 1336
Focus Processor Check bit 1592
Fopcode Compatibility Mode 1346
Force Overflow 1400
Foster DP 817
Foster MP 817
FP Compare and Set EFlags 626
FP Conditional Move 626
FP Data Operand Format 443
FP Data Registers 436, 442, 848
FP error exception 265
FP Exception Error Status Bits, MXCSR 767
FP Exception Mask Bits, MXCSR 767
FP Rounding Control, MXCSR 768
FPU 259, 432
FPU Busy 441
FPU Data Pointer Register 443
FPU exception 269, 318
FPU Instruction Pointer Register 443
FPU Register Set 437
FRC Mode 483
Free State 1195
Frequency Mode 756
Front Side Bus 11
Front Side Bus (FSB) interface 839
FRSTOR 712
FS 73
FSAVE 712
FSB 11, 839
FSB Agents 1138
FSB Arbitration Scheme, Pentium II 676
FSB Arbitration, Xeon 723
FSB Electrical Characteristics, Pentium 4 1115
FSB Enhancements, Pentium 4 841
FSB Interface Unit, P6 546
FSB Power Utilization Enhancements, Pentium M 1429
FSB Protocol, Pentium II 675
FSB signal groups 1143
FSB speed 1116
FSB, intro to the Pentium 4 1137
FSW Register 440
FTW Register 442
FTZ 765, 768, 1336
FTZ mode 766
FUCOMI 628
FUCOMIP 628
Fused Load and Operate 1438
Index

Fused Store 1438
Fusion, µop 1437
FWH 17
FXRSTOR 714, 715, 769
FXSAVE 714, 715, 769
FXSR 714

G
G0 378
G1 378
G2 379
G3 379
Gallatin 815
Gallatin 4M 818
Gallatin+MP 818
GD 380
GDT 112, 115, 206
GDT register 112
GDTR 112
GE 380
General Detect condition 262, 380
General Protection exception 264, 269, 311
General Register Fields, TSS 181
Geyserville 755
GFX 824
GHI# 756
Gilo 821
Global Descriptor Table 112, 115
Global pages 244, 567
Global Pages, detecting support for 568
GP exception 258, 264, 311, 536
Granularity Bit 122
graphics adapter 824
GS 73
GTL Reference input 1316
GTLREF 1131, 1316
Gunning Transceiver Logic 1116

H
Halt 1222, 1307
Halt Grant Snoop state 689
Halt Message 687
Halt/Grant Snoop State 691
Hard drive interface interrupt 259
Hard Failure response 1245, 1261
Hard Reset 1318
Hardware Interrupt Occurs in VM86 Mode 351
HDTV digital television 790
HFM 756, 1433
Hidden-Markov Model 788
Hierarchical Cluster Model 1564
High Frequency Mode 756
High Memory Area 88
Highest Frequency Mode 1433
High-Temperature Interrupt Enable 1343
HIT# and HITM# 394, 1135, 1229
HMA 88
HMA and the VMM 337
HMM 788
Hot Reset 486
Hot Reset “intercept” 486
Hot Reset and 286 DOS Extender Programs 485
Hot Reset command 486
HTT and PEBS 1416
HTT and Thermal Monitoring 1345
Hub Interface 16
Hyper-Threading 965

I
IA Instructions 843
IA32 instructions 10
IA32 processors 10
IA32 Register Set 846
IA32 Specification 9
IA32_APIBASE 1348, 1510
IA32_BIOS_SIGN 1350
IA32_BIOS_SIGN_ID 638
IA32_BIOS_UPDT_TRIG 1350
IA32_BIOS_UPDT_TRIG 615, 638, 698
IA32_CR_MISC_ENABLES[BOOT_NT4] 1457
IA32_CR_PAT 799, 1350
Index

IA32_CR_PAT and MTRR 807
IA32_CR_PAT compatibility with earlier systems 809
IA32_CR_PAT, changing the contents of 806
IA32_CTL 1359
IA32_DEBUGCTL 1349
IA32_DS_AREA 1350, 1372
IA32_MCG_CAP 1359, 1363
IA32_MCG_CTL 1359
IA32_MCG_EAX 1359
IA32_MCG_EBP 1359
IA32_MCG_EBX 1359
IA32_MCG_ECX 1359
IA32_MCG_EDI 1359
IA32_MCG_EDX 1359
IA32_MCG_EFLAGS 1359
IA32_MCG_EIP 1359
IA32_MCG_ESI 1359
IA32_MCG_ESP 1359
IA32_MCG_MISC 1360
IA32_MCG_STATUS 1359
IA32_MISC_ENABLE 1316, 1342, 1346, 1349, 1367, 1372
IA32_MISC_ENABLE[PEBS_UNAVAILABLE] 1415
IA32_MTRR_FIX16K_80000 1358
IA32_MTRR_FIX16K_A0000 1358
IA32_MTRR_FIX4K_C0000 1358
IA32_MTRR_FIX4K_C8000 1358
IA32_MTRR_FIX4K_D0000 1358
IA32_MTRR_FIX4K_D8000 1358
IA32_MTRR_FIX4K_E0000 1358
IA32_MTRR_FIX4K_E8000 1358
IA32_MTRR_FIX4K_F0000 1358
IA32_MTRR_FIX4K_F8000 1358
IA32_MTRR_FIX64K_00000 1358
IA32_MTRR_PHYSBASE0 1356
IA32_MTRR_PHYSBASE1 1356
IA32_MTRR_PHYSBASE2 1356
IA32_MTRR_PHYSBASE3 1356
IA32_MTRR_PHYSBASE4 1356
IA32_MTRR_PHYSBASE5 1357
IA32_MTRR_PHYSBASE6 1357
IA32_MTRR_PHYSBASE7 1357
IA32_MTRR_PHYSMASK0 1356
IA32_MTRR_PHYSMASK1 1356
IA32_MTRR_PHYSMASK2 1356
IA32_MTRR_PHYSMASK3 1356
IA32_MTRR_PHYSMASK4 1357
IA32_MTRR_PHYSMASK5 1357
IA32_MTRR_PHYSMASK6 1357
IA32_MTRR_PHYSMASK7 1357
IA32_MTRR_DEF_TYPE 1358
IA32_MISC_ENABLE 1355, 1370, 1413, 1416
IA32_P5_MC_ADDR 1347
IA32_P5_MC_TYPE 1347
IA32_PEBS_ENABLE 1355, 1370, 1413, 1416
IA32_PLATFORM_ID 636, 696, 1347
IA32_STATUS 1359
IA32_SYSENTER_CS 1361
IA32_SYSENTER_ESP 1361
IA32_SYSENTER_ESP 1361
IA32_THERM_CONTROL 1362
IA32_THERM_INTERRUPT 1362
IA32_THERM_INTERRUPT 1362
IA32_THERM_INTERRUPT 1343
IA32_THERM_STATUS 1342, 1362
IA32_THERM_STATUS 1342, 1362
IA32_THERM_STATUS 1342, 1362
IBM PC 255
IBM PC-AT 255
IC 439
ICE 262
ICH 16, 1269, 1273, 1302, 1498
ICH4 1426
ICR 1534
ID 1575
ID Register 1581
IDE 16, 18
Identification Register 1575
Idle 1153, 1583
Idle Response 1244
IDT 114
Index

IDTR 272, 274
IE 440
IEEE 1149.1 481
IEEE FP Primer 761
IEEE Standard 754 761
IERR# 1316
IGNNE# 1316
Ignore Numeric Error 1316
II field 601
illegal opcode 268
Illegal Register Address error bit, Local APIC 1507
IM 438
Implicit Writeback Response 1245
IMVP 1435
in-circuit emulator 262
IND 1574
Index Register 1574
Inexact result (precision), FPU 320
Infinity Control 439
INIT 267
INIT interrupt delivery mode 1542, 1561
INIT Level De-assert Delivery Mode 1560, 1561
INIT# 485, 1415, 1474
Initial Count Register 1534
In-Order Queue 1190
In-Service Register 1532
Instruction Pipeline, 90nm (Prescott) 846
Instruction Queue 1353
Instruction Restart 265
Instruction Set Architecture 9
instruction set, Pentium 4 1322
Instructions, miscellaneous new
  Pentium 4 1325
INT 608
INT instruction 35
INT Instruction handling in VM86
  extended mode 495
INT nn 347
INT nn Executed in VM86 Mode 358
INT3 breakpoint instruction 262, 268
In-Target Probe 1315
In-Target Probe tool clock 1316
integer data registers 848
Intel Mobile Voltage Positioning 1435
Interleaved Memory Architecture 473
Internal Error 1316
Internet Streaming SIMD Extensions 749
Inter-Processor Interrupt Messages 1503
Interrupt Acknowledge transaction 254, 547, 1210, 1302
Interrupt and Exception Classes 302
Interrupt Class 1520
Interrupt Command Register 1534
interrupt controller, slave 256
Interrupt Delivery Order 1584
Interrupt Delivery, edge-triggered 1569
interrupt delivery, legacy 1498, 1502
Interrupt Delivery, level-sensitive 1571
interrupt delivery, memory
  sync’d on 1590
Interrupt Descriptor Table 114
Interrupt Descriptor Table Register 272
Interrupt Enable bit 608
Interrupt Gate 273, 275, 276, 333, 1536
Interrupt Handlers, linked list of 1578
Interrupt Input Pin Polarity 1583
Interrupt Instructions 266
Interrupt Message Address Format 1587
Interrupt Message Data Format 1588
Interrupt message sent by Local APIC 1556
Interrupt Message Transaction and the MCH 1593
Interrupt Messages 1503, 1555
Interrupt On Overflow enable bit 1400
Interrupt On Overflow enable bit for logical processor 0 1400
Interrupt On Overflow enable bit for logical processor 1 1401
Interrupt on Performance Counter Overflow 1402
Interrupt Priority 1517
Interrupt Redirection bit map 180, 495
Interrupt Request Buffering 1537
Interrupt Request Register 1533
Interrupt Return 255, 347
interrupt service routine 251
Interrupt Servicing, maskable 254
Interrupt Sources 1516
Interrupt Sources, local 1517
Interrupt Sources, remote 1517
interrupt vector 254
Interrupt, User-defined 1518
Interrupt/Exception Generation and Handling in VM86 Mode 348
Interrupt/Exception Handling, Protected Mode 272
Interrupt/Exception Handling, Real Mode 92, 270
Interrupt/Exception Priority 266
Interrupts 251
Interrupts, Edge-Triggered 1522
Interrupts, hardware 252
Interrupts, level-sensitive 1523
Interrupts, Local 1539
Interrupts, maskable 253
INTO instruction 262, 269
INTR signal 252, 1303, 1501, 1503
INV 608
invalid opcode 263
Invalid OpCode Exception 298
Invalid operation, FPU 319
Invalid state 391
Invalid TSS exception 264, 305
Invalidate Cache (INVD) instruction 456
Invalidate Page Table Entry instruction 236
Invalidate TLB Entry (INVLPG) instruction 456
INVD 456, 457, 1307
Invert bit 608
INVLPG 236, 456, 458
IO Accesses in VM86 Mode 340
IO address 0092h 486
IO Address Range 1296
IO APIC 17, 509, 1302, 1497, 1567
IO APIC Register Set 1573
IO APIC Register Set Base Address 1573
IO breakpoints 497
IO Control Hub 16, 1302
IO Control Hub-4 1456
IO Data Transfer Length 1297
IO Instruction Restart Feature 1466, 1485
IO Instruction Restart field 1473, 1485
IO Permission bit map 177, 178
IO Permission Check in Protected Mode 177
IO Permission Check in VM86 Mode 178
IO port 0064h 486
IO port 70h 259
IO Port Access Protection 175
IO Port Addressing 70
IO Privilege Level 176
IO Protection 105
IO Protection in Real Mode 175
IO Read and Write 12
IO Read transaction 1211, 1297
IO Transactions 1295
IO Write transaction 1211, 1297
IOPL 176, 253
IOPL-Sensitive Instructions 345
IOPL-sensitive instructions 176
IOQ 658, 744, 840, 1148, 1190
IQ 1353
IQ counter group 1404
IRET 255, 347
IRETD 376
IRQ 1522
IRQ Assignment, PC-compatible 256
IRQ Lines, non-shareable 1578
IRQ Lines, shareable 1578
IRQ Pin Assertion Register 1574, 1577
IRQ0 255
IRQ7 255
IRQPA 1574
IRR 1523, 1533, 1535
Index

ISA 9
ISR 251, 1532, 1535
ISSE 749
ITP 380, 1315
ITP_CLK[1:0] 1316
ITPCLKOUT[1:0] 1316

J
Jayhawk 819

K
Katmai 540, 542
Katmai New Instructions 748
kernel 14
Keyboard 256
Keyboard/Mouse interface 486
Kill 392
Klamath 540
KNI 748

L
L0 378
L1 378
L1 Caches, Pentium II 671
L1 Caches, Pentium III 745
L1 Code Cache, P6 548
L1 Data Cache, P6 548
L1 Data Cache, Pentium 4 839
L1 Data Cache, Pentium M 1440
L2 379
L2 Cache, Pentium 4 839
L2 Cache, Pentium II 669
L2 Cache, Pentium III 745
L2 Cache, Pentium M 1440
L3 379
L3 Cache, Pentium 4 839
LAN Controller Interface 16
Land Grid Array 665
LAR 92
Last Branch Record Stack 1366
Last Branch Recording 623

Last Branch, Interrupt, and Exception Recording 1365
Last Exception Recording Registers 1362
LASTBRANCHFROMIP 620, 623, 702
LASTBRANCHTOIP 620, 623, 702
LASTEXCEPTIONFROMIP 623
LASTEXCEPTIONTOIP 623
LASTINTFROMIP 620, 702
LASTINTTOIP 620, 702
LBR 623
LBR Stack 1366
LCI 16
LDMXCSR 769
LDR 1531, 1562
LDT 112, 116, 206
LDT register 112
LDT Selector 181
LDT Structure 121
LDTR 112, 117
LE 380
Least-Recently Used 755
LEN 1216, 1217
LEN0, 1, 2, 3 378
Level-sensitive 1583
Level-Sensitive Interrupt Delivery 1571
Level-Sensitive Interrupts 1523
LFM 756, 1433
LGA 665
LIDT instruction 270, 272
Linear Address 213
Linear Memory Space 214
Link field, TSS 184
Linkage Modification 203
Linked Tasks 201
LINT0 1503, 1540
LINT0 LVT Register 1535
LINT1 1476, 1504, 1543
LINT1 LVT Register 1535
LL field 600
LLDT 92
Load and Operate, Fused 1438
Load Fence Instruction 1326
Index

Local 1308
Local APIC 479, 507, 839, 1497
Local APIC base address 1524
Local APIC Enhancements,
  Pentium 4 1338
Local APIC ID Register 1526
Local APIC ID, reading the 1516
Local APIC Initiates an Interrupt Message Transaction 1594
Local APIC Register Set 1524
Local APIC Timer 1544
Local APIC Version Register 1526
Local APIC, characteristics when disabled 1512
Local APIC, detecting the 1509
Local APIC, determining the version of the 1509
Local APIC, enable or disable the 1507, 1510
Local APIC, temporarily disabling the 1511
Local APIC’s Error Interrupt 1549
Local APICs, maximum number of 1515
Local Descriptor Table 112, 116
Local Interrupt 0 1540
Local Interrupt 1 1543
Local Interrupt pin 0 1503
Local Interrupt pin 1 1504
Local Interrupts 1503
Local Vector Table 569, 1338, 1507, 1539
Local Vector Table Entries 1534
lock 1180
LOCK prefix 1182
LOCK# 1168, 1180
LOCK# and split locked access 620
Locked RMW 1182
Locked Transaction Series 1177
Locked Transaction Series duration 1183
Locking a cache line 1184
Logical destination mode 1560, 1562, 1584
Logical Destination Register 1531, 1562
logical processor 0 1383
logical processor 1 1383
logical processors 1150
Loop Detector 1436
Low Voltage Swing 1429
Lowest Frequency Mode 1433
Lowest Priority Delivery Mode 509, 1561, 1565, 1587
Lowest Priority Delivery, chipset assisted 1565
Low-Pin Count Bus 17, 1302
Low-Temperature Interrupt Enable 1343
low-voltage swing 1116
LPC Bus 17, 1302
LRU 755
LSL 92
LTR 92, 188
LVS 1116, 1429
LVT 569, 1343, 1507, 1540
LVT entries 1534
LVT Error Register 1535
LVT Timer Register 1534

M
Machine Check Address register 513
Machine Check Architecture 504, 1207, 1269, 1270, 1273, 1274
Machine Check Architecture,
  Pentium 4 1363
Machine Check Error 1269, 1273, 1317
Machine Check exception 267, 324, 504, 536, 588, 589, 1207, 1269, 1270, 1273, 1274, 1415
Machine Check exception, detecting support for the 589
Machine Check In Progress bit 593
Machine Check register set 504
Machine Check Save State registers 1359
Machine Check Type register 513
mantissa 761, 763
Mask bit 1541, 1583
Maskable Interrupt Servicing 254
Masked Move Operation 788

1633
Index

Masking User-Defined Interrupts 1522
MASKMOVDQU 1330
MASKMOVQ 782, 788
Master Abort 1283, 1287, 1291
MAX/MIN 788
MC Error Code 598
MC Exception 605
MC Extended State MSRs 1364
MC Global Control register 591
MC Global Count and Present register 591
MC Global Status register 591
MC0_ADDR 618, 701
MC0_CTL 594, 618, 701
MC0_MISC 612, 618, 701
MC0_STATUS 618, 701
MC1_ADDR 618, 701
MC1_CTL 618, 701
MC1_MISC 612, 618, 701
MC1_STATUS 618, 701
MC2_ADDR 619, 701
MC2_CTL 619, 701
MC2_MISC 612, 619, 701
MC2_STATUS 619, 701
MC3_ADDR 619, 702
MC3_CTL 619, 702
MC3_MISC 612, 619, 702
MC3_STATUS 619, 702
MC4_ADDR 619, 702
MC4_CTL 619, 702
MC4_MISC 619, 702
MC4_STATUS 619, 702
MCA 504
MCA "Other Information" field 595
MCA Address Register Valid Bit 597
MCA Bank 0 error logging registers 618, 1360
MCA Bank 1 error logging registers 618, 1360
MCA Bank 2 error logging registers 1360
MCA Bank 3 error logging registers 619, 1361
MCA Bank 4 registers 619
MCA Bank Control Register 594
MCA Bank Status Register 595
MCA Elements 588
MCA Enhancements, Pentium 4 1363
MCA Error Code 595, 598
MCA Error Codes, compound 599
MCA Error Codes, simple 598
MCA Error Enabled Bit 596
MCA error logging bank 0 701
MCA error logging bank 1 701
MCA error logging bank 2 701
MCA error logging bank 3 702
MCA error logging bank 4 702
MCA error logging register banks 612
MCA Error Valid Bit 596
MCA FSB Error Interpretation 602
MCA Global Control Register 593
MCA Global Count and Present Register 592
MCA Global Registers 591
MCA Miscellaneous Register Valid Bit 596
MCA Model Specific Error Code 598
MCA Overflow Bit 596
MCA Processor Context Corrupt Bit 597
MCA Register Bank composition 594
MCA Register Set initialization 606
MCA register set, detecting support for the 589
MCA Registers 618, 701, 1359
MCA Uncorrectable Error Bit 596
MCA, Pentium Pro enhanced 588
MCAR 504
MCERR# 1135, 1269, 1317
MCG_CAP 591, 618, 701
MCG_CTL 591, 612, 618, 701
MCG_CTL_P 592, 1363
MCG_EXT_CNT 1363
MCG_EXT_P 1363
MCG_STATUS 591, 618, 701
MCH 16, 1205, 1273, 1303
MCH, 855 1426
MCIP bit 593
1634
Index

MCTR 504
MDA 1562
memory address, physical 214
Memory Addressing, Real Mode 71
Memory Code Read transaction 1212
Memory Control Hub 16, 1205, 1273, 1303
Memory Data Read transaction 12, 1212
Memory Data Write 13
Memory Fence Instruction 1326
Memory Instruction Read 13
Memory Line Writeback transaction 547
Memory Order Buffer 1354
Memory Protection 104
Memory Read and Invalidate transaction 547, 1185, 1211
memory semaphore 1179
memory streaming 754
Memory Type and Range Registers 572, 1356
Memory Type Determination 803
Memory Type, Paging definition of 587
Memory Types 581
Memory Write 547
Memory Write (may be retried) transaction 1212
Memory Write (may not be retried) transaction 1212
Memory-Mapped IO in VM86 Mode 343
Mendocino 661
Merom 821
MESI 391
MESI cache 407
Message 1210, 1222, 1306
Message Destination Address 1562
Message Signaled Interrupts 1584
Message Types 1306
Message, interrupt 1555
Messages, NMI, SMI and INIT 1504
Microcode binary image 633
microcode ROM 590
Microcode Signature MSR 636
Microcode Store ROM 1351, 1352
Microcode Store ROM counter group 1403
Microcode Update Checksum 635
Microcode Update Control Function Call 648
Microcode Update Date 634
Microcode Update Feature 631, 1347
Microcode Update Header 634
Microcode Update Header Loader Revision field 635
Microcode Update Header Processor field 635
Microcode Update Header Update Revision field 634
Microcode Update Header Version 634
Microcode Update Image 633
Microcode Update Image Management 640
Microcode Update Image Management BIOS 640
Microcode Update Image/Processor Match 636
Microcode Update in a Multiprocessor System 639
Microcode Update Loader 637
Microcode Update MSRs 615, 1350
Microcode Update Presence Detect Function Call 641
Microcode Update Read Microcode Update Data Function Call 650
Microcode Update Signature 636
Microcode Update Write Microcode Update Data Function Call 643
Microcode Update, Effect of RESET# or INIT# on 653
Min/Max Determination 788
MINPS/PMIN 788
Misaligned Transfers 43
MISCV 595
Mispredicted Branches 787
MMX 519, 572, 626, 670
MMX Capability, detecting 529
MMX Instruction Set Syntax 533
Index

MMX Programming Environment 532
MOB 1354
Mobile Pentium 4 820
Model field 1562
Model Specific Error Code 595
Model, cluster 1563
Model, flat 1563
Modified state 392
Monitor 1458
Motion Compensation 789
Motion-Estimation 790
MOVAPS 792
Move Unaligned Packed SP FP 792
MOVMSKPS 788
MOVNTDQ 1327
MOVNTI 1563
MOVNTQ 781
MOVUPS 792
MP spec 833, 1597
MP system 1501
MP-capable OS 1597
MPEG-2 Motion Compensation 789
MPS 833, 1597
MRI 1211
MS 1351, 1352, 1353, 1354, 1355
MS counter group 1403
MS DOS 23
MSI 1567, 1584
MSI Address register 1585
MSI Control register 1585
MSI Data register 1585
MSI message, direct-delivery of an 1586
MSR 1351
MSR_ALF_ESCR[1:0] 1355
MSR_BPU_ESCR[3:0] 1351
MSR_BPU_COUNTER[3:0] 1351
MSR_BPU_COUNTER0 1403
MSR_BPU_COUNTER1 1403
MSR_BPU_COUNTER2 1403
MSR_BPU_COUNTER3 1403
MSR_BPU_ESCR[1:0] 1351
MSR_BSU_ESCR[1:0] 1353
MSR_BSU_ESCR[1:0] 1353
MSR_CRU_ESCR[5:0] 1354
MSR_CRU_ESCR[5:0] 1354
MSR_DAC_ESCR[1:0] 1354
MSR_DAC_ESCR[1:0] 1354
MSR_EBC_HARD_POWERON 1349
MSR_EBC_HARD_POWERON 1349
MSR_EBC_SOFT_POWERON 1349
MSR_EBC_SOFT_POWERON 1349
MSR_FIRM_ESCR[1:0] 1353
MSR_FIRM_ESCR[1:0] 1353
MSR_FLAME_COUNTER[3:0] 1352
MSR_FLAME_COUNTER[3:0] 1352
MSR_FLAME_COUNTER[3:0] 1403
MSR_FLAME_COUNTER[3:0] 1403
MSR_FLAME_COUNTER[3:0] 1352
MSR_FLAME_COUNTER[3:0] 1352
MSR_FSB_ESCR[1:0] 1353
MSR_FSB_ESCR[1:0] 1353
MSR_IQ_CCCR[5:0] 1353
MSR_IQ_CCCR[5:0] 1353
MSR_IQ_CCCR4 1416
MSR_IQ_CCCR4 1416
MSR_IQ_CCCR5 1416
MSR_IQ_CCCR5 1416
MSR_IQ_COUNTER[5:0] 1353, 1404
MSR_IQ_COUNTER[5:0] 1353, 1404
MSR_IQ_ESCR[1:0] 1353
MSR_IQ_ESCR[1:0] 1353
MSR_IS_ESCR[1:0] 1354
MSR_IS_ESCR[1:0] 1354
MSR_ITLB_ESCR[1:0] 1354
MSR_ITLB_ESCR[1:0] 1354
MSR_IQ_ESCR[1:0] 1355
MSR_IQ_ESCR[1:0] 1355
MSR_LASTBRANCH_[3:0] 1361
MSR_LASTBRANCH_[3:0] 1361
MSR_LASTBRANCH_0 1366
MSR_LASTBRANCH_0 1366
MSR_LASTBRANCH_3 1366
MSR_LASTBRANCH_3 1366
MSR_LASTBRANCH_TOS 1361, 1366
MSR_LASTBRANCH_TOS 1361, 1366
MSR_LER_FROM_LIP 1362, 1366
MSR_LER_FROM_LIP 1362, 1366
MSR_LER_TO_LIP 1362, 1366
MSR_LER_TO_LIP 1362, 1366
MSR_MOB_ESCR[1:0] 1354
MSR_MOB_ESCR[1:0] 1354
MSR_MS_CCCR[3:0] 1352
MSR_MS_CCCR[3:0] 1352
MSR_MS_COUNTER[3:0] 1351
MSR_MS_COUNTER[3:0] 1351
MSR_MS_COUNTER0 1403
MSR_MS_COUNTER0 1403
MSR_MS_COUNTER1 1403
MSR_MS_COUNTER1 1403
MSR_MS_COUNTER2 1403
MSR_MS_COUNTER2 1403
MSR_MS_COUNTER3 1403
MSR_MS_COUNTER3 1403
MSR_MS_ESCR[1:0] 1352
MSR_MS_ESCR[1:0] 1352
MSR_PEBBS_MATRIX_VERT 1355, 1413
MSR_PEBBS_MATRIX_VERT 1355, 1413
MSR_PMH_ESCR[1:0] 1354
MSR_PMH_ESCR[1:0] 1354
MSR_RAT_ESCR[1:0] 1355
MSR_RAT_ESCR[1:0] 1355
MSR_SAAT_ESCR[1:0] 1355
MSR_SAAT_ESCR[1:0] 1355
MSR_SSAU_ESCR0 1355
MSR_SSAU_ESCR0 1355
MSR_TBPU_ESCR[1:0] 1355
MSR_TBPU_ESCR[1:0] 1355
MSR_TC_ESCR[1:0] 1355
MSR_TC_ESCR[1:0] 1355

1636
Index

MSR_TC_PRECISE_EVENT 1355
MSR_U2L_ESCR[1:0] 1354
MSRs, Pentium 512
MSRs, Pentium 4 1347
MSRs, Pentium II 696
MSRs, Pentium III 696
MSRs, Pentium Pro 612
MTRR Default Type register 1358
MTRRCAP 575, 616, 699
MTRRDEFTYPE 576, 617, 700
MTRRfix16K_80000 617, 700
MTRRfix16K_A0000 617, 700
MTRRfix4K_C0000 617, 700
MTRRfix4K_C8000 617, 700
MTRRfix4K_D0000 617, 700
MTRRfix4K_D8000 617, 700
MTRRfix4K_E0000 617, 700
MTRRfix4K_E8000 617, 700
MTRRfix4K_F0000 617, 700
MTRRfix4K_F8000 617, 700
MTRRfix64K_00000 617, 700
MTRRphysBase0 616, 699
MTRRphysBase1 616, 699
MTRRphysBase2 616, 699
MTRRphysBase3 616, 699
MTRRphysBase4 616, 699
MTRRphysBase5 616, 699
MTRRphysBase6 617, 700
MTRRphysBase7 617, 700
MTRRphysBasen 581
MTRRphysMask0 616, 699
MTRRphysMask1 616, 699
MTRRphysMask2 616, 699
MTRRphysMask3 616, 699
MTRRphysMask4 616, 699
MTRRphysMask5 616, 699
MTRRphysMask6 617, 700
MTRRphysMask7 617, 700
MTRRPhysMaskn 581
MTRRs 572, 580, 616, 699, 1356
MTRRs after Reset 576
MTRRs, detecting support for the Fixed-Range 575
MTRRs, detecting the number of pairs of Variable-Range 575
MTRRs, Fixed-Range 577, 1358
MTRRs, Variable-Range 580, 1356
MTRRs, detecting support for the 574
Multiprocessing Table 833, 1516, 1596, 1597
multiprocessor (MP) system 1501
MXCSR 766, 1336
MXCSR Mask Field 716
MXCSR[DAZ] 1336
MXCSR[FTZ] 765
MXCSR_MASK 714

N
NaNs 762
Nehalem 816
Newton-Raphson 789
NMI 252, 259, 262, 268, 295, 1269, 1273, 1314, 1474, 1476, 1504
NMI delivery mode 1542, 1561
NMI Recognition in the SM Handler 1477
No Data Response 1245
No Shorthand 1559
Nocona DP 819
Non-Bogus 1409
Non-Conforming Code Segment 141
Non-Maskable Interrupt 252, 259, 268, 1269, 1273
Non-precise event-based sampling 1374, 1407, 1414
Non-Retirement Counting 1385, 1407, 1408
Non-Retirement Event Counting 1407
non-temporal stores 777, 1327
NOP 1222, 1306
Normal Data Response 1246
normal numbers 762
Normal State 686, 1318
North Bridge 1303
### Index

Northwood 815  
Northwood B with HyperThreading 815  
Northwood with 800MHz FSB 815  
N-R iteration 789  
NT bit 193  
Numeric overflow, FPU 320  
Numeric underflow, FPU 320  
NW 454  

#### O

ODTEN 1127, 1317  
OE 440  
OM 438  
On-Demand Clock Modulation Duty Cycle 1344  
On-Demand Clock Modulation Enable 1344  
On-Die Termination Enable 1127, 1317  
One Shot Mode 1545  
OpCode register 1345  
Open-Drain signals 1135  
Operating System bit 609  
OPTIMIZED/COMPAT# 1317  
OS 609  
OS bit 1382  
OS kernel 14  
OS loader 14  
OSFXSR 770  
OSXMMEXCPT 771  
Overflow 1401  
Overflow Exception 296  
OVF_PMI_T0 1400  
OVF_PMI_T1 1401  

#### P

P bit 224  
P5_MC_ADDR 513, 612, 696  
P5_MC_TYPE 513, 612, 696  
P54C 464, 479, 509, 1466  
P55C 464  
P6 Processor Family 539  
PA-1 735  
PA-2 735  
Package Data 726  
Packaging, Pentium M 1435  
Packet A 1120, 1209, 1212  
Packet B 1120, 1209, 1219  
PAE-36 Mode 554, 731  
PAE-36 Mode, detecting support for 555  
PAE-36 Mode, enabling 556  
PAE-36 Mode, Linux support 567  
PAE-36 Mode, paging 245  
PAE-36 Mode, Windows support 566  
Page Access Permission 237  
Page Attribute Table 797, 1350  
Page Caching 453  
Page Directory 217, 220  
Page Directory Caching 451  
Page Directory Entry 220, 449  
Page Fault 225, 230, 239, 258, 265, 268, 269, 314, 536, 558  
Page Fault Causes 239  
Page Fault Error Code 241  
Page Privilege Check 237  
Page Read/Write Check 238  
Page Size bit 501  
Page Size Extension 450  
Page Table 217  
Page Table base address 224  
Page Table Caching 452  
Page Table Entry 221, 450  
Pages, 4MB 501  
Paging 209  
Paging Evolution 244  
Paging Unit 40, 214  
Paging Write Protect Feature 450  
Paging, enable 219  
Paging, Pentium 4 1323  
Parity port one 258  
Parallel port two 258  
Parity Checking, Pentium 4 Request Phase 1205  
parity error, PCI 1283, 1288, 1291  
Parity Reversal Register 513
Parity when transferring less than 32 bytes, Pentium 4 1275
Parity, Pentium 4 Data Bus 1247, 1270
Parity, Pentium 4 Request Phase 1202, 1204
Parity, Pentium 4 Response Phase 1243, 1268
Parking, Priority Agent 1175
Part Number Data 727
PAT feature 245, 797
PAT Support, detecting 797
PAUSE 1331
PAVG 789
PAVGW 789
PBE# 1316
PC 255, 438, 609
PC-AT compatible machine 1498
PCC 595
PCD 449, 450
PCI 17
PCI Express 824
PCI interrupts 1584
PDE 217, 220, 449
PDE Format 225
PDPT 557
PE 440
PEBS 1372, 1374, 1411, 1414
PEBS absolute maximum 1368
PEBS and Hyper-Threading 1416
PEBS buffer 1367
PEBS buffer base 1368
PEBS Capability, detecting 1415
PEBS counter reset value 1368
PEBS Feature, enabling the 1370
PEBS index 1368
PEBS Interrupt Handler 1415
PEBS interrupt threshold 1368
PEBS Record Format 1370
PEBS, enabling 1415
PEN# 504
Pending Break Event 1316
Pentium 820
Pentium 4 814
Pentium 4 CPU Arbitration 1149
Pentium 4 Extreme Edition (Gallatin) 815
Pentium 4 FSB Blocking 1189
Pentium 4 FSB Electrical Characteristics 1115
Pentium 4 FSB enhancements 841
Pentium 4 FSB Request Phase 1201
Pentium 4 FSB Response and Data Phases 1241
Pentium 4 FSB Snoop Phase 1225
Pentium 4 FSB Transaction Deferral 1277
Pentium 4 FSB, intro to the 1137
Pentium 4 Locked Transaction Series 1177
Pentium 4 M 1426
Pentium 4 Prescott 837
Pentium 4 Priority Agent Arbitration 1165
Pentium 4 Processor Basic Organization 838
Pentium 4 Processor Family 836
Pentium 4 Processor Overview 835
Pentium 4 Road Map 813
Pentium 4 Software Enhancements 1321
Pentium 4 System Overview 823
Pentium 4, mobile 820
Pentium Address/Data Bus Structure 469
Pentium Data Cache 478
Pentium Flavors 464
Pentium Hardware Overview 463
Pentium II 660
Pentium II Hardware Overview 657
Pentium II Roadmap 660
Pentium II Software Enhancements 695
Pentium III Hardware Overview 741
Pentium III Roadmap 744
Pentium III Software Enhancements 757
Pentium Instruction Set Changes 517
Pentium M Enhanced Power Management Characteristics 1429
Pentium M FSB Characteristics 1427
Pentium M Precise Event-Based Sampling 1429

1639
Index

Pentium M Processor 1425
Pentium M, the next 1440
Pentium M-Specific Signals 1428
Pentium Pro Software Enhancements 553
Pentium Software Enhancements 489
PERFCTR0 506, 515, 608, 615, 698
PERFCTR1 506, 515, 608, 615, 698
PerfEvtSel0 608
PerfEvtSel1 608
Performance Counter Groups 1403
Performance Counter Cascading 1401
Performance Counter Enable 1399
Performance Counter Overflow
interrupt 569, 1504, 1547
Performance Counter Overflow LVT
entry 1507
Performance Counter/CCCR/ESCR
Relationship 1376
Performance Counters, accessing the 1406
Performance Counters, P6 606
Performance Monitor Control and Event
Select register 515
Performance Monitor Counter 0 515
Performance Monitor Counter 1 515
Performance Monitor Sampling
Methods 1374
Performance Monitor, At-Retirement
Event Counting 1409
Performance Monitoring Available
bit 1372
Performance Monitoring Counters LVT
Register 1535
Performance Monitoring Facility 1371
Performance Monitoring Interrupt on
Overflow, P6 611
Performance Monitoring MSRs,
Pentium 4 1351
Performance Monitoring MSRs,
Pentium Pro 615
Performance Monitoring, Pentium 505
Periodic Mode 1545
Phase-Locked Loop 1117
Phases 1142
Physical Delivery Mode 509, 1560
Physical Destination Mode 1562, 1584
physical memory address 229
Physical Page 222
physical processors 1150
PIC 17, 254, 351
Pin Control bit 609
Pipelining transactions 1143
PIROM 17, 724
PLL 692, 1117, 1434
PM 438
PM[5:4]# 1315
PMI 1408
PMI bit 1400
PMIN/PMAX 788
PMOVMSKB 788
PMULHUW 790
PMULHW 790
Pop 78
Pop Flags Instruction 347
POST 14
Posted Write Buffer 387
Potomac MP 819
Power Good 1317
Power Management Modes 683
Power Status Indicator 1428
Power-On Self-Test 14
PP field 601
PPR 1529, 1552
PRDY# 1315
Precise Event-Based Sampling 1372, 1374, 1414
Precise Event-Based Sampling
Unavailable bit 1372
Precision Control 438
Preemption 1158
preemptive multitasking 28
PREFETCH 773, 1212
Prefetcher, enhanced hardware-based
data 1440
Prefetcher, hardware-based data 747
Index

prefix, LOCK 1182
PREQ# 1315
Prescott, Pentium 4 816, 837
Prestonia B 818
Prestonia with 1MB L3 818
Prestonia with Hyper-Threading 818
Priority Agent 1429
Priority Agent Arbitration 1165
Priority Agent Parking 1175
Priority Agents 1166
Priority of User-Defined Interrupts 1519
Priority Request Agents 1141
Priority, Exceptions and Interrupts 266
Priority, Interrupt 1517
Priority, Processor 1551
Priority, Task 1551
Privilege Check, Page 237
Privilege Checking 139
Privilege Level 0 - 2 Stack Definition Fields, TSS 183
privilege levels 33, 106
PRO/Wireless network connection 1426
Probe Ready 1315
Probe Request 1315
Procedure, definition of 140
processor clock 1117
processor core 838
Processor Core Data 725
Processor Core, P6 548
Processor Electronic Signature 727
Processor Hot 1317
Processor Information ROM 17, 724
Processor Priority 1551
Processor Priority Register 1529, 1552
processor signature 1459
Processor Type 1449
PROCHOT# 1317
Programmable Interrupt Controllers 1498
Protected Mode Virtual Interrupts 497
PS 501
PS/2 compatibility port 486
PSADBW 790
PSE 450, 455
PSE-36 Mode 731
PSE-36 Mode, detecting support for 732
PSE-36 Mode, enabling 732
PSE-36 Mode, paging 245
PSE-36 Mode, Windows support for 736
PSI# 1428
PTE 218, 221, 450
PTE Format 230
Push 77
Push Flags Instruction 347
PVI 455
PWB 387
PWRGOOD 1317, 1319, 1428
FWT 449, 450
Q
QDF Number 725
QNaN 762, 763
Quiet NaN 762
R
R/S# 482
R/W 224
R/W Field In DR7 380
R/W0, 1, 2, 3 378
RAID 18
RAM disk driver 736
RC 439
RCP 789
RDMSR 513, 518
RDPMC 626, 628, 1406
RDTSC 498, 499, 517, 1419
Read For Ownership 392, 1185, 1211
read parity error, PCI 1283, 1288
Read Performance Counter 626, 1406
Read With Intent To Modify 392, 1185, 1211
Read/Write Check, Page 238
Real Big mode 90
Real-Time Clock 259
Reciprocal 788, 789

1641
Index

Reciprocal Square Root 788, 789
Reciprocal/Reciprocal Square Root
    Unit 752
REDIR_TBL 1576
Redirection Hint bit 1587
Redirection Table register set 1567, 1576, 1581
Redirection Table, interrupt 1567
Redundant Array of Inexpensive
    Drives 18
Reference input 1316
Reference Voltage 1131
register set, Local APIC 1524
Relaxed DBSY# Deassertion 1264
Remote IRR 1543, 1583
Remote Register Read 1508
Replay 1067, 1410
REQ[4:0]# 1203, 1208, 1213, 1219
Request Agent 1138, 1141
Request Bus 1203
Request Phase 1144, 1150, 1201
Request Phase Parity Checking,
    Pentium 4 1205
Request Phase Parity, Pentium 4 1204
Request Phase signal group 1208
Request Type 1 1446
Request Type 1 Enhanced 1450
Request Type 2 1452
Request Type 3 793, 1457
Request Type 4 1457
Request Type 5 1458
Requestor Privilege Level 139
Reservation Station 1438
Reset Code Byte 486
RESET# 1318, 1415
Response Agent 1138
Response Bus 1243
Response Phase 1145, 1241, 1242
Response Phase End Point 1243
Response Phase Parity, Pentium 4 1243, 1268
Response Phase Signal Group 1243
Response Phase Start Point 1243
Response Types 1244
Restart Instruction Pointer Valid bit 593
Resume 1468, 1476, 1483
Resume Flag 291
Resume from System Management Mode
    (RSM) instruction 456
Retire 1409
Retry Response 840, 1140, 1244, 1280
Retry, PCI 1287
Return Instruction Pointer 1374, 1408
RFO 392, 1185, 1211
RIP 1374, 1408
RIPV bit 593
RMW 1180
RMW, Locked 1182
Roadmap, Pentium 4 813
Roadmap, Pentium III 744
ROB 847
ROB_CR_BKUPTMPDR6 614, 621, 697
ROM, microcode 590
Root Complex 824, 1166, 1205, 1273, 1280, 1305
Rotating ID 1152
Rounding Control 439
Rounding Control, MXCSR 768
RPL 139
RPL, definition of 141
RRRRR field 600
RS[2:0]# 1243, 1268, 1270
RSM 456, 458, 1468, 1476, 1483
RSP# 1243, 1268, 1270
RSQRT 789
RT 1567, 1581
RTC chip 258, 259
Rtt 1127
RWTM 392, 1185, 1211
S
SAD 790
saturation math 526
Scalable Bus Speed 1349
Scalar Operations, SSE 772
scaler 28
SCI 756
Scratch EEPROM 724
SEC 662
Segment 125
Segment Base Address 123
Segment Descriptor 110, 114
Segment Descriptor Format 121
Segment Not Present exception 257, 264, 269, 308
Segment Present Bit 129
Segment Register Fields, TSS 181
Segment Registers in VM86 Mode 339
Segment Selector 114
Segment Size 123
Segment Type 125
Segment Unit 40
Segment Wraparound 338, 421
Segmentation, eliminating 248
Segments, Real Mode Limitations 110
Self 1559
Self-Snoop 1238
semaphore 1179
Send Pending 1583
Serial Number 1457
Serial port one 257
Serial port two 257
Serializing Instructions 499, 1183
Set Interrupt Enable 253, 346
SF 440
SFENCE 784
Shared Resource 1178
Shared state 391
Shuffle/Logical Unit 751
Shutdown 303, 1222, 1307
SIDT 273
Signaling NaN 762
signature 1459
Signature, Enhanced Processor 1460
Signature, Microcode Update 636
significand 761
SIMD 524
SIMD Floating-Point Exception 326
SIMD FP capability 758
SIMD FP Error Priorities 328
SIMD FP exception 269, 770, 792
SIMD SP FP 751
Single-Edge Cartridge 662
Single-Step exception 262, 268
Single-Step on Branch, Exception, or Interrupt, P6 625
SIO 17
SIPI 833, 1561
SKTOCC# 1318
SLDT 92
Sleep State 692, 1318
Slot 1 722
SLOT0C# 680
SLP# 680, 689, 692, 1318
SM base address 1468, 1495
SM Memory and caching 1487
SM RAM above the 1st MB 1496
SM RAM access mapping 1488
SM RAM organization 1468
SM_ALERT# 729, 1318
SM_CLK 729, 1318
SM_DAT 729, 1318
SM_TS_A[1:0] 730, 1318
SM_VCC 730, 1318
SM_WP 730, 1318
SMB_PRT 728, 1318
SMBase Field 1473
SMBus 17, 723
SMBus Alert 729
SMBus Clock 729
SMBus Data 729
SMBus Present 728
SMBus Signals 728
SMBus Write Protect 730
SMI 267
SMI Acknowledge 1222, 1309, 1468, 1478, 1483
Index

SMI Delivery Mode 1542, 1561
SMI Inter Processor Interrupt 1466
SMI interrupt message 1508
SMI IPI 1466
SMI occurs within the NMI Handler 1477
SMI# 756, 1309, 1465, 1473
SMI# and FLUSH# 1493
SMM 460, 1415, 1463
SM and NMI Handling 1476
SMM and Power Down 1479
SM and Real Mode Address
   Formation 1475
SM and Single Stepping 1475
SM and the HLT Instruction 1485
SM and the IDT 1475
SMM Auto Halt Restart Feature 1484
SMM base address 1466
SM Context Save 1478
SM Enhancement, Pentium Pro 572
SM in an MP System 1496
SM Revision ID 1473, 1482
SM, entering 1473
SM, Exceptions and Software
   Interrupts 1474
SM, exiting 1483
SMMEM# 1222, 1309, 1478
SMP 507, 1151
SMRAM State Save Area 1471
SNaN 762, 763
Snoop Agents 1138, 1226
snoop hit on a clean line 1139
snoop hit on a modified line 1139
snoop miss 1139
Snoop Phase 1145, 1225
Snoop Phase and non-memory
   transactions 1239
Snoop Phase duration 1229
Snoop Phase Has Two Purposes 1228
Snoop Result 691, 830, 1232
Snoop Stall 1234
snoop transaction 828
snoopers 1138
Snooping 826
Snooping and the WB Cache 397
Snooping and the WT Cache 396
socket 8 661
Socket Occupied 1318
Soft Reset (INIT#) 485
Software Interrupt Instruction 347
Source-synchronous strobes 1203
SP FP Format 762
SP FP Numeric Format 761
Special Transaction 547, 1210, 1302, 1306
Speculative Execution 853
SpeedStep technology 755
SpeedStep, Enhanced 1433
SPLCK# 1222
Split Lock 1222
Spurious Interrupt Vector 1591
Spurious Interrupt Vector Register 1532,
   1592
SS 73
SS0 153, 183
SS1 183
SS2 183
SSD 790
SSE 748, 758
SSE Alignment Checking 792
SSE Capability, detecting 749
SSE Control/Status register 766
SSE Data Types 760
SSE Elements 759
SSE Execution Units 750
SSE Instruction Set 791
SSE instruction set, enabling 770
SSE instructions, 2-wide 754
SSE instructions, 4-wide 754
SSE Register Set 760
SSE Scalar Operations 772
SSE Setup 793
SSE SIMD (Packed) Operations 772
SSE SIMD FP Exception, enabling the 771
SSE Support, detecting 758
SSE2 Instruction Set 1332
Index

SSE2 instruction set, enabling the 771
SSE3 Instruction Set 1337
S-spec 725
Stack exception 264, 309
Stack Management, Advanced 1439
Stack Segment 73
Stack Segment Privilege Check 170
Stack Segment, types of 126
Stack Switch, Automatic 153
Stack Usage, Processor 80
Stack, Expand-Down 164
Stack, Expand-Up 162
Stall, Snoop 1234
Stalled State 1193
Stalled/Throttled/Free Indicator 1192
standard voltage 1435
Start-Up Delivery Mode 1561
Startup Inter Processor Interrupt 833
Start-up IPI 1561
STI Handling in VM86 extended mode 495
STI instruction 253, 346
STMXCSR 769
Stop Clock 267, 1308, 1318
Stop Clock and the Thermal Monitor 1340
Stop Grant Acknowledge message 689, 1222, 1308
Stop Grant State 688, 1318
Store Buffer 1212
Store IDT instruction 273
Store, Fused 1438
Stores, Non-Temporal 1327
Stores, streaming 754, 1327
Store-to-Load Forwarding 1070
STPCLK# 267, 684, 687, 688, 1308, 1316, 1318, 1340, 1433
STR 92, 188
Streaming Buffer Disable 620
Streaming SIMD Extensions 758
Streaming stores 754, 776, 1327
streaming, memory 754
Strobe Setup and Hold Specs 1126
Strobes 1119, 1203
Sum-of-absolute-differences 790
Sum-of-square-differences 790
Super IO chip 17
SVR 1532
Symmetric Agent Arbitration 1151
Symmetric Multiprocessing 507, 1151
Symmetric Request Agents 1141
Sync 1222, 1308
synchronizing 1183
SYSENTER 707, 710
SYSENTER_CS_MSR 707
SYSENTER_EIP_MSR 707
SYSENTER_ESP_MSR 707
SYSEXIT 707, 711
SYSR/S# 482
System bit 130
System Control Interrupt 756
System Control Port A 486
System Enter/Exit Registers 707, 1361
System Management Bus 17, 723
System Management Interrupt 1309, 1465
System Management Memory 1222
System Management Mode 460, 1463
System Segments, types of 130
System timer 256
T
T bit 181, 267
T field 601
T0OS 1383, 1398
T0USR 1383, 1398
T1OS 1383, 1398
T1USR 1383, 1398
Table Indicator 112
Tag Enable bit 1384
Tag Value 1383
Tagging 1409
Tagging Mechanisms, performance monitor 1411
Tagging of µops 1384
Tagging, execution 1411, 1412

1645
Tagging, Front-End 1411
Tagging, Multi 1411
Tagging, Replay 1413
tags, no 1411
TAP 481, 1315
Target Abort 1283, 1287, 1291
Target Ready 1243
Task Creation 171
Task Gate 173, 192, 193, 194, 195, 273, 333
Task Priority 1551
Task Priority Register 509, 1527, 1552
Task Register 187
Task State Segment 27, 31
Task Switch 106, 172
Task Switch as a Result of a Far Call 197
Task Switch as the Result of a Far Jump 197
Task Switch details 196
Task Switch due to a BOUND/INT/INTO/INT3 198
Task Switch due to an Interrupt or Exception 196
Task Switch due to Execution of an IRET 198
Task Switch, events that trigger a 192
Task, definition of 139
Tasks, linked 201
task-switch breakpoint 262
TBPU 1355
TC 843, 1355
TCK 481
TC, TDI, TDO, TMS, TRST# 1318
TDI 481
TDO 481
Tejas 816
Test Access Port 481, 1315
Test Clock 481
Test Control Register 620
Test Data In 481
Test Data Out 481
Test Mode Select 481
Test Register 1 513
Test Register 10 514
Test Register 11 514
Test Register 12 514, 515
Test Register 2 514
Test Register 3 514
Test Register 4 514
Test Register 5 514
Test Register 6 514
Test Register 7 514
Test Register 8 514
Test Registers 456, 512
Test Reset 481
TEST_CTL 614, 620, 697
TESTHI[x:0] 1319
thermal diode 1341
Thermal Diode Anode and Cathode 1319
Thermal Interrupt Control 1362
Thermal Monitor 2 1435
Thermal Monitor and Interrupts 1343
Thermal Monitor Control 1362
Thermal Monitor Enable 1342
Thermal Monitor Feature Detection 1340
Thermal Monitor Interrupt 1343
Thermal Monitor related Registers 1362
Thermal Monitor Status 1362
Thermal Monitoring and HTT 1345
Thermal Monitoring Facilities, Pentium 4 1340
Thermal Monitoring, Automatic 1342
Thermal Ref. Data 727
Thermal Reference Byte 727
thermal sensing device 724
Thermal Sensor interrupt 1338, 1504, 1508, 1548
Thermal Sensor LVT Register 1534
Thermal Sensor Select Address pins 730
Thermal Status bit 1342
Thermal Status Log bit 1342
Thermal Trip 1319
THERMDA, THERMDC 1319
THERMTRIP# 1319
Threshold 1399
Index

Threshold value 1409
Throttled State 1194
Ti bit 112
Time Stamp Counter 498, 612, 1308, 1418, 1419
Time Stamp Disable 517
Timer, APIC 1504, 1544
Timer, System 256
Timeslice 172
Timeslice Timer 172
timeslicing 28
TLB 234, 567
TLB Information 1452
TLB Maintenance 235
TLB-related errors 590
TMR 1533, 1535
TMS 481
Toggle Mode Dword Transfer Order, 486 416
Toggle Mode Transfer Order, Pentium/ P6/Pentium 4/Pentium M 473
Tonga 660
TOF 436, 441
Top of Stack 441, 1366
TOS 1366
TPR 509, 1527, 1552
TR 187
Trace Cache 839, 843, 1355
Trace Cache events 1412
Tracking transactions 1147
Transaction Deferral 1141, 1277
Transaction ID 1141, 1221
Transaction Phases 1142
Transaction Pipelining 1143
Transaction Tracking 1147
Transaction Types 1210, 1216
Translation Lookaside Buffer 234
Trap bit 181, 376
Trap bit, TSS 267
Trap Gate 273, 275, 281, 333, 1536
Traps 260
TRDY# 1243
Trigger Mode 1541, 1559, 1583, 1588
Trigger Mode Register 1533
TRST# 481
TSC 498, 515, 612, 696, 1308, 1418, 1419
TSC Wraparound 499
TSC, writing to the 499
TSS 27, 31, 172, 331
TSS Busy bit 186
TSS CR3 field 182
TSS descriptor 172, 185, 192, 194
TSS EFlags field 182
TSS EIP field 182
TSS ESP field 182
TSS Mapping 207
TSS place within a Page 208
TSS Privilege Level 0 - 2 Stack Definition Fields 183
TSS Segment Register Fields 181
TSS Structure 173
TSS Trap bit 181
TSS, invalid 269
TT field 600
Type of mis-predicted branches by BPU 1355

U

U/S bit 224
UC- 801, 803
UC memory 582, 1220
UE 440
UM 438
Uncacheable (UC) Memory 582
Unified code/data cache 476
Unified L2 Cache, P6 548
Unit Mask 609
Unix Copy-on-Write Strategy 451
Unpacked Data 524
UnReal mode 90
UP 726
Index

Update Header data structure 633  
USB 17, 18  
User bit 609  
User/Supervisor bit 224  
User-Defined Interrupt 1518  
User-Defined Interrupt Eligibility Test 1553  
User-Defined Interrupt priority 1519  
User-Defined Interrupts, masking 1522  
USR 609  
USR bit 1382

VM86 Mode, Attempted Execution of a POPF Instruction 369  
VM86 Mode, Attempted Execution of a PUSHF Instruction 369  
VM86 Mode, Attempted Execution of the INT nn Instruction 369  
VM86 Mode, Attempted Execution of the STI Instruction 368  
VM86 Mode, instructions usable in 373  
VM86 Mode, Pentium 4 1323  
VM86 Mode, register accessible in 372  
VM86 Mode, using a task as a handler 370  
VM86 Task privilege 339  
VME 455  
VME feature 490  
VMM 331  
Voltage ID 726, 1126, 1319  
Voltage Identification, Pentium II 680  
voltage regulator 1428  
voltage, low 1435  
voltage, standard 1435  
voltage, ultra low 1435  
VRCHGNG# 756  
Vref 1429

Wait States, Data Phase 1266  
wait-for-IPI 833  
Ways 400  
WB memory 584, 1221  
WBINVD 456, 458, 1308  
WC memory 582, 1220  
WCB 1212  
WCBs, Pentium III 754  
Willamette 814  
WP 1221  
WP bit 450  
Write Back and Invalidate instruction 456, 1308  
Write Combining memory type, detecting support for the 575  
write parity error, PCI 1291

V

VAL 595  
Vector 254, 1210  
Vector 15 318  
Vector Assignment 255  
Vector field 1543, 1562  
VER 1576  
VERR 92  
Version Register 1576, 1581  
VERW 92  
VGA vertical retrace interrupt 258  
VID 726, 1434  
VID[4:0] 680, 1126  
VID[5:0] 1126, 1319  
Virtual 8086 Mode 106, 329  
Virtual Interrupts 497  
Virtual Machine Monitor 331  
Virtual Memory Paging 105  
VM86 Extensions 490  
VM86 Mode 329  
VM86 Mode and Display Frame Buffer Updates 344  
VM86 Mode Evolution 374  
VM86 Mode Interrupt/Exception Generation and Handling 348  
VM86 Mode, Access to a Forbidden IO Port 364  
VM86 Mode, Attempted Execution of a CLI Instruction 365

W
Write Protect feature, paging 244, 450
Write Receives the Defer Response 1288
Write-Back (WB) Memory 584
Writeback Buffers 755
Write-Combining (WC) Memory 582
Write-Protect (WP) Memory 584
Write-Through (WT) Memory 583
WRMSR 513, 518
WT memory 583, 584, 1221

X
XADD 456, 457
xAPIC 1338, 1508, 1515
Xeon 660
Xeon Cartridge 722
Xeon DP, Pentium 4 1422
Xeon MP, Pentium 4 1422
Xeon, Pentium II 719
Xeon, Pentium III 742, 795
XMM[7:0] 766
XTP (External Task Priority) registers 1566

Z
ZE 440
ZM 438