



AMD-K6™-III E+

Embedded Processor

Data Sheet

| |
|---|
| Publication # 23543 Rev: A Amendment/ 0 Issue Date: September 2000 |
|---|

© 2000 Advanced Micro Devices, Inc. All rights reserved.

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

Trademarks

AMD, the AMD logo, K6, 3DNow!, and combinations thereof, AMD PowerNow!, E86, and Super7 are trademarks, FusionE86 is a service mark, and AMD-K6 and RISC86 are registered trademarks of Advanced Micro Devices, Inc.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

NetWare is a registered trademark of Novell, Inc.

MMX is a trademark of Intel Corporation.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

The TAP State Diagram is reprinted from IEEE Std 1149.1-1990 "IEEE Standard Test Access Port and Boundary-Scan Architecture," Copyright © 1990 by the Institute of Electrical and Electronics Engineers, Inc. The IEEE disclaims any responsibility or liability resulting from the placement and use in the described manner. Information is reprinted with the permission of the IEEE.

IF YOU HAVE QUESTIONS, WE'RE HERE TO HELP YOU.

The AMD customer service network includes U.S. offices, international offices, and a customer training center. Expert technical assistance is available from the AMD worldwide staff of field application engineers and factory support staff to answer E86™ family hardware and software development questions.

Frequently accessed numbers are listed below. Additional contact information is listed on the back of this manual. AMD's WWW site lists the latest phone numbers.

Technical Support

Answers to technical questions are available online, through e-mail, and by telephone.

Go to AMD's home page at www.amd.com and follow the Support link for the latest AMD technical support phone numbers, software, and Frequently Asked Questions.

For technical support questions on all E86 embedded products, send e-mail to epd.support@amd.com (in the US and Canada) or euro.tech@amd.com (in Europe and the UK).

You can also call the AMD Corporate Applications Hotline at:

(800) 222-9323 Toll-free for U.S. and Canada

44-(0) 1276-803-299 U.K. and Europe hotline

WWW Support

For specific information on E86 products, access the AMD home page at www.amd.com and follow the **Embedded Processors** link. These pages provide information on upcoming product releases, overviews of existing products, information on product support and tools, and a list of technical documentation. Support tools include online benchmarking tools and CodeKit software—tested source code example applications. Many of the technical documents are available online in PDF form.

Questions, requests, and input concerning AMD's WWW pages can be sent via e-mail to web.feedback@amd.com.

Documentation and Literature Support

Data books, user's manuals, data sheets, application notes, and product CDs are free with a simple phone call. Internationally, contact your local AMD sales office for product literature.

To order literature:

Web: www.amd.com/support/literature.html

U.S. and Canada: (800) 222-9323

Third-Party Support

AMD FusionE86SM program partners provide an array of products designed to meet critical time-to-market needs. Products and solutions available include chipsets, emulators, hardware and software debuggers, board-level products, and software development tools, among others. The WWW site and the *E86™ Family Products Development Tools CD, order #21058*, describe these solutions. In addition, mature development tools and applications for the x86 platform are widely available in the general marketplace.

Contents

| | |
|--|-------------|
| Revision History | xvii |
| About this Data Sheet | xix |
| 1 AMD-K6™-III+ Embedded Processor | 1 |
| 1.1 AMD-K6™-III+ Embedded Processor Features | 3 |
| 1.2 Process Technology | 7 |
| 1.3 Super7™ Platform | 8 |
| 2 Internal Architecture | 11 |
| 2.1 Microarchitecture Overview | 11 |
| 2.2 Cache, Instruction Prefetch, and Predecode Bits | 16 |
| 2.3 Instruction Fetch and Decode | 17 |
| 2.4 Centralized Scheduler | 21 |
| 2.5 Execution Units | 22 |
| 2.6 Branch-Prediction Logic | 25 |
| 3 Software Environment | 27 |
| 3.1 Registers | 27 |
| 3.2 Model-Specific Registers (MSR) | 44 |
| 3.3 Memory Management Registers | 54 |
| 3.4 Paging | 56 |
| 3.5 Descriptors and Gates | 59 |
| 3.6 Exceptions and Interrupts | 62 |
| 3.7 Instructions Supported by the AMD-K6™-III+ Processor | 63 |
| 4 Logic Symbol Diagram | 91 |
| 5 Signal Descriptions | 93 |
| 5.1 Signal Terminology | 93 |
| 5.2 A20M# (Address Bit 20 Mask) | 94 |
| 5.3 A[31:3] (Address Bus) | 95 |
| 5.4 ADS# (Address Strobe) | 96 |
| 5.5 ADSC# (Address Strobe Copy) | 96 |
| 5.6 AHOLD (Address Hold) | 97 |
| 5.7 AP (Address Parity) | 98 |
| 5.8 APCHK# (Address Parity Check) | 99 |
| 5.9 BE[7:0]# (Byte Enables) | 100 |
| 5.10 BF[2:0] (Bus Frequency) | 101 |
| 5.11 BOFF# (Backoff) | 102 |
| 5.12 BRDY# (Burst Ready) | 103 |
| 5.13 BRDYC# (Burst Ready Copy) | 104 |
| 5.14 BREQ (Bus Request) | 104 |
| 5.15 CACHE# (Cacheable Access) | 105 |
| 5.16 CLK (Clock) | 105 |
| 5.17 D/C# (Data/Code) | 106 |
| 5.18 D[63:0] (Data Bus) | 107 |

| | | |
|----------|---|------------|
| 5.19 | DP[7:0] (Data Parity) | 108 |
| 5.20 | EADS# (External Address Strobe) | 109 |
| 5.21 | EWBE# (External Write Buffer Empty) | 110 |
| 5.22 | FERR# (Floating-Point Error) | 111 |
| 5.23 | FLUSH# (Cache Flush) | 112 |
| 5.24 | HIT# (Inquire Cycle Hit) | 113 |
| 5.25 | HITM# (Inquire Cycle Hit To Modified Line) | 113 |
| 5.26 | HLDA (Hold Acknowledge) | 114 |
| 5.27 | HOLD (Bus Hold Request) | 115 |
| 5.28 | IGNNE# (Ignore Numeric Exception) | 116 |
| 5.29 | INIT (Initialization) | 117 |
| 5.30 | INTR (Maskable Interrupt) | 118 |
| 5.31 | INV (Invalidation Request) | 118 |
| 5.32 | KEN# (Cache Enable) | 119 |
| 5.33 | LOCK# (Bus Lock) | 120 |
| 5.34 | M/IO# (Memory or I/O) | 121 |
| 5.35 | NA# (Next Address) | 122 |
| 5.36 | NMI (Non-Maskable Interrupt) | 123 |
| 5.37 | PCD (Page Cache Disable) | 124 |
| 5.38 | PCHK# (Parity Check) | 125 |
| 5.39 | PWT (Page Writethrough) | 126 |
| 5.40 | RESET (Reset) | 127 |
| 5.41 | RSVD (Reserved) | 128 |
| 5.42 | SCYC (Split Cycle) | 129 |
| 5.43 | SMI# (System Management Interrupt) | 130 |
| 5.44 | SMIACT# (System Management Interrupt Active) | 131 |
| 5.45 | STPCLK# (Stop Clock) | 132 |
| 5.46 | TCK (Test Clock) | 133 |
| 5.47 | TDI (Test Data Input) | 133 |
| 5.48 | TDO (Test Data Output) | 133 |
| 5.49 | TMS (Test Mode Select) | 134 |
| 5.50 | TRST# (Test Reset) | 134 |
| 5.51 | VCC2DET (VCC2 Detect) | 135 |
| 5.52 | VCC2H/L# (VCC2 High/Low) | 136 |
| 5.53 | VID[4:0] (Voltage Identification) | 137 |
| 5.54 | W/R# (Write/Read) | 138 |
| 5.55 | WB/WT# (Writeback or Writethrough) | 139 |
| 5.56 | Pin Tables by Type | 140 |
| 5.57 | Bus Cycle Definitions | 142 |
| 6 | AMD PowerNow!™ Technology | 143 |
| 6.1 | Enhanced Power Management Features | 143 |
| 6.2 | Dynamic Core Frequency and Core Voltage Control | 150 |
| 7 | Bus Cycles | 153 |
| 7.1 | Timing Diagrams | 153 |
| 7.2 | Bus States | 155 |
| 7.3 | Memory Reads and Writes | 158 |
| 7.4 | I/O Read and Write | 166 |

| | | |
|-----------|--|------------|
| 7.5 | Inquire and Bus Arbitration Cycles | 168 |
| 7.6 | Special Bus Cycles | 190 |
| 8 | Power-on Configuration and Initialization | 199 |
| 8.1 | Signals Sampled During the Falling Transition of RESET | 199 |
| 8.2 | RESET Requirements | 200 |
| 8.3 | State of Processor After RESET | 200 |
| 8.4 | State of Processor After INIT | 203 |
| 9 | Cache Organization | 205 |
| 9.1 | MESI States in the L1 Data Cache and L2 Cache | 207 |
| 9.2 | Predecode Bits | 208 |
| 9.3 | Cache Operation | 208 |
| 9.4 | Cache Disabling and Flushing | 211 |
| 9.5 | L2 Cache Testing | 213 |
| 9.6 | Cache-Line Fills | 213 |
| 9.7 | Cache-Line Replacements | 214 |
| 9.8 | Write Allocate | 215 |
| 9.9 | Prefetching | 220 |
| 9.10 | Cache States | 221 |
| 9.11 | Cache Coherency | 222 |
| 9.12 | Writethrough and Writeback Coherency States | 227 |
| 9.13 | A20M# Masking of Cache Accesses | 227 |
| 10 | Write Merge Buffer | 229 |
| 10.1 | EWBE# Control | 229 |
| 10.2 | Memory Type Range Registers | 231 |
| 10.3 | Memory-Range Restrictions | 233 |
| 10.4 | Examples | 235 |
| 11 | Floating-Point and Multimedia Execution Units | 237 |
| 11.1 | Floating-Point Execution Unit | 237 |
| 11.2 | Multimedia and 3DNow!™ Execution Units | 239 |
| 11.3 | Floating-Point and MMX™/3DNow!™ Instruction Compatibility | 240 |
| 12 | System Management Mode (SMM) | 241 |
| 12.1 | SMM Operating Mode and Default Register Values | 241 |
| 12.2 | SMM State-Save Area | 243 |
| 12.3 | SMM Revision Identifier | 245 |
| 12.4 | SMM Base Address | 246 |
| 12.5 | Halt Restart Slot | 246 |
| 12.6 | I/O Trap Doubleword | 247 |
| 12.7 | I/O Trap Restart Slot | 248 |
| 12.8 | Exceptions, Interrupts, and Debug in SMM | 250 |
| 13 | Test and Debug | 251 |
| 13.1 | Built-In Self-Test (BIST) | 251 |
| 13.2 | Three-State Test Mode | 252 |
| 13.3 | Boundary-Scan Test Access Port (TAP) | 253 |

| | | |
|-----------|---|------------|
| 13.4 | Cache Inhibit | 263 |
| 13.5 | L2 Cache and Tag Array Testing | 264 |
| 13.6 | Debug | 268 |
| 14 | Clock Control | 277 |
| 14.1 | Clock Control States | 277 |
| 14.2 | Halt State | 280 |
| 14.3 | Stop Grant State | 280 |
| 14.4 | Stop Grant Inquire State | 282 |
| 14.5 | EPM Stop Grant State | 283 |
| 14.6 | Stop Clock State | 285 |
| 15 | Electrical Data | 287 |
| 15.1 | Operating Ranges | 288 |
| 15.2 | Absolute Ratings | 289 |
| 15.3 | DC Characteristics | 289 |
| 15.4 | Power Dissipation | 291 |
| 15.5 | Power and Grounding | 293 |
| 16 | Signal Switching Characteristics | 297 |
| 16.1 | CLK Switching Characteristics | 298 |
| 16.2 | Clock Switching Characteristics for 100-MHz Bus Operation | 298 |
| 16.3 | Clock Switching Characteristics for 66-MHz Bus Operation | 299 |
| 16.4 | Valid Delay, Float, Setup, and Hold Timings | 300 |
| 16.5 | Output Delay Timings for 100-MHz Bus Operation | 300 |
| 16.6 | Input Setup and Hold Timings for 100-MHz Bus Operation | 302 |
| 16.7 | Output Delay Timings for 66-MHz Bus Operation | 304 |
| 16.8 | Input Setup and Hold Timings for 66-MHz Bus Operation | 306 |
| 16.9 | RESET and Test Signal Timing | 308 |
| 16.10 | Timing Diagrams | 311 |
| 17 | Thermal Design | 315 |
| 17.1 | Package Thermal Specifications | 315 |
| 17.2 | Measuring Case Temperature | 319 |
| 17.3 | Layout and Airflow Considerations | 319 |
| 18 | Pin Designations | 323 |
| 18.1 | Pins Designations for CPGA Package | 324 |
| 18.2 | Pins Designations for OBGA Package | 328 |
| 19 | Package Specifications | 333 |
| 19.1 | 321-Pin Staggered CPGA Package Specification | 333 |
| 19.2 | 349-Ball OBGA Package Specification | 334 |
| 20 | Ordering Information | 335 |
| | Index | 337 |

List of Figures

| | | |
|------------|--|----|
| Figure 1. | AMD-K6™-III+ Processor Block Diagram | 13 |
| Figure 2. | Cache Sector Organization | 16 |
| Figure 3. | The Instruction Buffer | 18 |
| Figure 4. | AMD-K6™-III+ Processor Decode Logic | 19 |
| Figure 5. | AMD-K6™-III+ Processor Scheduler | 22 |
| Figure 6. | Register X and Y Pipeline Functional Units | 24 |
| Figure 7. | EAX Register with 16-Bit and 8-Bit Name Components | 28 |
| Figure 8. | Integer Data Registers | 29 |
| Figure 9. | Segment Register | 30 |
| Figure 10. | Segment Usage | 31 |
| Figure 11. | Floating-Point Register | 32 |
| Figure 12. | FPU Status Word Register | 32 |
| Figure 13. | FPU Control Word Register | 33 |
| Figure 14. | FPU Tag Word Register | 33 |
| Figure 15. | Packed Decimal Data Register | 34 |
| Figure 16. | Precision Real Data Registers | 34 |
| Figure 17. | MMX™/3DNow!™ Registers | 35 |
| Figure 18. | MMX™ Technology Data Types | 36 |
| Figure 19. | 3DNow!™ Technology Data Types | 37 |
| Figure 20. | EFLAGS Registers | 38 |
| Figure 21. | Control Register 4 (CR4) | 39 |
| Figure 22. | Control Register 3 (CR3) | 39 |
| Figure 23. | Control Register 2 (CR2) | 39 |
| Figure 24. | Control Register 1 (CR1) | 40 |
| Figure 25. | Control Register 0 (CR0) | 40 |
| Figure 26. | Debug Register DR7 | 41 |
| Figure 27. | Debug Register DR6 | 42 |
| Figure 28. | Debug Registers DR5 and DR4 | 42 |
| Figure 29. | Debug Registers DR3, DR2, DR1, and DR0 | 43 |
| Figure 30. | Machine-Check Address Register (MCAR) | 45 |
| Figure 31. | Machine-Check Type Register (MCTR) | 45 |
| Figure 32. | Test Register 12 (TR12) | 46 |
| Figure 33. | Time Stamp Counter (TSC) | 46 |
| Figure 34. | Extended Feature Enable Register (EFER) | 47 |
| Figure 35. | SYSCALL/SYSRET Target Address Register (STAR) | 48 |
| Figure 36. | Write Handling Control Register (WHCR) | 48 |

| | | |
|------------|---|-----|
| Figure 37. | UC/WC Cacheability Control Register (UWCCR) | 49 |
| Figure 38. | Processor State Observability Register (PSOR) | 49 |
| Figure 39. | Page Flush/Invalidate Register (PFIR) | 50 |
| Figure 40. | L2 Tag or Data Location for AMD-K6™-III+ Processor—EDX | 51 |
| Figure 41. | L2 Data —EAX | 51 |
| Figure 42. | L2 Tag Information for AMD-K6™-III+ Processor—EAX | 52 |
| Figure 43. | Enhanced Power Management Register (EPMR) | 53 |
| Figure 44. | Memory Management Registers | 54 |
| Figure 45. | Task State Segment (TSS) | 55 |
| Figure 46. | 4-Kbyte Paging Mechanism | 56 |
| Figure 47. | 4-Mbyte Paging Mechanism | 57 |
| Figure 48. | Page Directory Entry 4-Kbyte Page Table (PDE) | 58 |
| Figure 49. | Page Directory Entry 4-Mbyte Page Table (PDE) | 58 |
| Figure 50. | Page Table Entry (PTE) | 59 |
| Figure 51. | Application Segment Descriptor | 60 |
| Figure 52. | System Segment Descriptor | 61 |
| Figure 53. | Gate Descriptor | 62 |
| Figure 54. | Enhanced Power Management Register (EPMR) | 144 |
| Figure 55. | EPM 16-Byte I/O Block | 146 |
| Figure 56. | Bus Divisor and Voltage ID Control (BVC) Field | 147 |
| Figure 57. | Processor State Observability Register (PSOR)—Low- Power Versions of the Processor | 148 |
| Figure 58. | Waveform Definitions | 154 |
| Figure 59. | Bus State Machine Diagram | 155 |
| Figure 60. | Non-Pipelined Single-Transfer Memory Read/Write and Write Delayed by EWBE# | 159 |
| Figure 61. | Misaligned Single-Transfer Memory Read and Write | 161 |
| Figure 62. | Burst Reads and Pipelined Burst Reads | 163 |
| Figure 63. | Burst Writeback due to Cache-Line Replacement | 165 |
| Figure 64. | Basic I/O Read and Write | 166 |
| Figure 65. | Misaligned I/O Transfer | 167 |
| Figure 66. | Basic HOLD/HLDA Operation | 169 |
| Figure 67. | HOLD-Initiated Inquire Hit to Shared or Exclusive Line | 171 |
| Figure 68. | HOLD-Initiated Inquire Hit to Modified Line | 173 |
| Figure 69. | AHOLD-Initiated Inquire Miss | 175 |
| Figure 70. | AHOLD-Initiated Inquire Hit to Shared or Exclusive Line | 177 |
| Figure 71. | AHOLD-Initiated Inquire Hit to Modified Line | 179 |
| Figure 72. | AHOLD Restriction | 181 |

| | | |
|-------------|---|-----|
| Figure 73. | BOFF# Timing. | 183 |
| Figure 74. | Basic Locked Operation. | 185 |
| Figure 75. | Locked Operation with BOFF# Intervention. | 187 |
| Figure 76. | Interrupt Acknowledge Operation. | 189 |
| Figure 77. | Basic Special Bus Cycle (Halt Cycle) | 191 |
| Figure 78. | Shutdown Cycle | 192 |
| Figure 79. | Stop Grant and Stop Clock Modes, Part 1 | 194 |
| Figure 80. | Stop Grant and Stop Clock Modes, Part 2 | 195 |
| Figure 81. | INIT-Initiated Transition from Protected Mode to Real Mode | 197 |
| Figure 82. | L1 and L2 Cache Organization for the AMD-K6™-III+ Processor | 206 |
| Figure 83. | L1 Cache Sector Organization. | 207 |
| Figure 84. | Write Handling Control Register (WHCR) | 217 |
| Figure 85. | Write Allocate Logic Mechanisms and Conditions | 218 |
| Figure 86. | Page Flush/Invalidate Register (PFIR) | 224 |
| Figure 87. | UC/WC Cacheability Control Register (UWCCR) | 232 |
| Figure 88. | External Logic for Supporting Floating-Point Exceptions. | 239 |
| Figure 89. | SMM Memory | 242 |
| Figure 90. | TAP State Diagram | 261 |
| Figure 91. | L2 Cache Organization for AMD-K6™-III+ Processor | 265 |
| Figure 92. | L2 Cache Sector and Line Organization | 265 |
| Figure 93. | L2 Tag or Data Location for the AMD-K6™-III+ Processor—EDX | 266 |
| Figure 94. | L2 Data - EAX. | 267 |
| Figure 95. | L2 Tag Information for the AMD-K6™-III+ Processor—EAX | 268 |
| Figure 96. | LRU Byte. | 268 |
| Figure 97. | Debug Register DR7 | 270 |
| Figure 98. | Debug Register DR6 | 271 |
| Figure 99. | Debug Registers DR5 and DR4. | 271 |
| Figure 100. | Debug Registers DR3, DR2, DR1, and DR0. | 272 |
| Figure 101. | Clock Control State Transitions for Standard-Power Versions of the AMD-K6™-III+ Processor. | 278 |
| Figure 102. | Clock Control State Transitions for Low-Power Versions of the AMD-K6™-III+ Processor. | 279 |
| Figure 103. | Suggested Component Placement for CPGA Package | 294 |
| Figure 104. | CLK Waveform | 299 |
| Figure 105. | Key to Timing Diagrams | 311 |

| | |
|--|-----|
| Figure 106. Output Valid Delay Timing | 312 |
| Figure 107. Maximum Float Delay Timing | 312 |
| Figure 108. Input Setup and Hold Timing | 312 |
| Figure 109. Reset and Configuration Timing | 313 |
| Figure 110. TCK Waveform | 314 |
| Figure 111. TRST# Timing | 314 |
| Figure 112. Test Signal Timing Diagram | 314 |
| Figure 113. Thermal Model (CPGA Package) | 317 |
| Figure 114. Power Consumption and Thermal Resistance (CPGA Package) | 317 |
| Figure 115. Processor Heat Dissipation Path | 318 |
| Figure 116. Measuring Case Temperature | 319 |
| Figure 117. Voltage Regulator Placement | 320 |
| Figure 118. Airflow for a Heatsink with Fan | 321 |
| Figure 119. Airflow Path in a Dual-Fan System | 321 |
| Figure 120. Airflow Path in an ATX Form-Factor System | 322 |
| Figure 121. CPGA Connection Diagram (Top-Side View) | 324 |
| Figure 122. CPGA Connection Diagram (Bottom-Side View) | 325 |
| Figure 123. OBGA Connection Diagram (Top-Side View) | 328 |
| Figure 124. OBGA Connection Diagram (Bottom-Side View) | 329 |
| Figure 125. 321-Pin Staggered CPGA Package Specification | 333 |
| Figure 126. 349-Ball OBGA Package Specification | 334 |

List of Tables

| | | |
|-----------|--|-----|
| Table 1. | Execution Latency and Throughput of Execution Units | 23 |
| Table 2. | General-Purpose Registers | 28 |
| Table 3. | General-Purpose Register Doubleword, Word, and Byte Names | 29 |
| Table 4. | Segment Registers | 30 |
| Table 5. | AMD-K6™-III+ Processor Model-Specific Registers | 44 |
| Table 6. | Extended Feature Enable Register (EFER) Definition | 47 |
| Table 7. | SYSCALL/SYSRET Target Address Register (STAR) Definition | 48 |
| Table 8. | Memory Management Registers | 54 |
| Table 9. | Application Segment Types | 60 |
| Table 10. | System Segment and Gate Types | 61 |
| Table 11. | Summary of Exceptions and Interrupts | 62 |
| Table 12. | Integer Instructions | 65 |
| Table 13. | Floating-Point Instructions | 82 |
| Table 14. | MMX™ Instructions | 86 |
| Table 15. | 3DNow!™ Instructions | 89 |
| Table 16. | 3DNow!™ Technology DSP Extensions | 90 |
| Table 17. | Processor-to-Bus Clock Ratios | 101 |
| Table 18. | Output Pin Float Conditions for VCC2 High/Low | 136 |
| Table 19. | Input Pin Types | 140 |
| Table 20. | Output Pin Float Conditions | 141 |
| Table 21. | Input/Output Pin Float Conditions | 141 |
| Table 22. | Test Pin Types | 141 |
| Table 23. | Bus Cycle Definition | 142 |
| Table 24. | Special Cycles | 142 |
| Table 25. | Enhanced Power Management Register (EPMR) Definition | 145 |
| Table 26. | EPM 16-Byte I/O Block Definition | 146 |
| Table 27. | Bus Divisor and Voltage ID Control (BVC) Definition | 147 |
| Table 28. | Processor-to-Bus Clock Ratios | 149 |
| Table 29. | Bus-Cycle Order During Misaligned Memory Transfers | 160 |
| Table 30. | A[4:3] Address-Generation Sequence During Bursts | 162 |
| Table 31. | Bus-Cycle Order During Misaligned I/O Transfers | 167 |
| Table 32. | Interrupt Acknowledge Operation Definition | 188 |
| Table 33. | Encodings for Special Bus Cycles | 190 |
| Table 34. | Output Signal State After RESET | 200 |
| Table 35. | Register State After RESET | 201 |
| Table 36. | PWT Signal Generation | 210 |
| Table 37. | PCD Signal Generation | 210 |
| Table 38. | CACHE# Signal Generation | 211 |
| Table 39. | L1 and L2 Cache States for Read and Write Accesses | 221 |

| | | |
|-----------|---|-----|
| Table 40. | Valid L1 and L2 Cache States and Effect of Inquire Cycles . | 225 |
| Table 41. | L1 and L2 Cache States for Snoops, Flushes, and Invalidation. | 226 |
| Table 42. | EWBEC Settings and Performance | 231 |
| Table 43. | WC/UC Memory Type | 233 |
| Table 44. | Valid Masks and Range Sizes for UWCCR Register | 234 |
| Table 45. | Initial State of Registers in SMM. | 243 |
| Table 46. | SMM State-Save Area Map | 243 |
| Table 47. | SMM Revision Identifier | 246 |
| Table 48. | I/O Trap Doubleword Configuration | 248 |
| Table 49. | I/O Trap Restart Slot | 249 |
| Table 50. | Boundary Scan Bit Definitions | 257 |
| Table 51. | Device Identification Register | 259 |
| Table 52. | Supported TAP Instructions | 259 |
| Table 53. | Tag versus Data Selector. | 266 |
| Table 54. | DR7 LEN and RW Definitions | 274 |
| Table 55. | Operating Ranges for Low-Power AMD-K6™-III+ Devices | 288 |
| Table 56. | Operating Ranges for Standard-Power AMD-K6™-III+ Devices | 288 |
| Table 57. | Absolute Ratings | 289 |
| Table 58. | DC Characteristics for the AMD-K6™-III+ Processor | 289 |
| Table 59. | Power Dissipation for Low-Power AMD-K6™-III+ Devices | 291 |
| Table 60. | Power Dissipation for Standard-Power AMD-K6™-III+ Devices | 292 |
| Table 61. | Supported Frequencies and Voltages for Low-Power AMD-K6™-III+ Processors Enabled with AMD PowerNow!™ Technology | 292 |
| Table 62. | CLK Switching Characteristics for 100-MHz Bus Operation . | 298 |
| Table 63. | CLK Switching Characteristics for 66-MHz Bus Operation . . | 299 |
| Table 64. | Output Delay Timings for 100-MHz Bus Operation | 300 |
| Table 65. | Input Setup and Hold Timings for 100-MHz Bus Operation . | 302 |
| Table 66. | Output Delay Timings for 66-MHz Bus Operation | 304 |
| Table 67. | Input Setup and Hold Timings for 66-MHz Bus Operation . . | 306 |
| Table 68. | RESET and Configuration Signals for 100-MHz Bus Operation | 308 |
| Table 69. | RESET and Configuration Signals for 66-MHz Bus Operation | 309 |
| Table 70. | TCK Waveform and TRST# Timing at 25 MHz | 310 |
| Table 71. | Test Signal Timing at 25 MHz. | 310 |
| Table 72. | Package Thermal Specification for Low-Power AMD-K6™-III+ Devices. | 316 |

| | | |
|-----------|--|-----|
| Table 73. | Package Thermal Specification for Standard-Power AMD-K6™-III+ Devices | 316 |
| Table 74. | Pin Differences Between the CPGA and OBGA Packages . . . | 323 |
| Table 75. | CPGA Pin Designations by Functional Grouping | 326 |
| Table 76. | CPGA Pin Designations for No Connect, Reserved, Power, and Ground Pins | 327 |
| Table 77. | OBGA Pin Designations by Functional Grouping | 330 |
| Table 78. | OBGA Pin Designations for No Connect, Reserved, Power, and Ground Pins | 331 |
| Table 79. | AMD-K6™-III+ Embedded Processor Valid Ordering Part Number Combinations | 336 |

Revision History

| Date | Rev | Description |
|----------------|-----|--|
| September 2000 | A | Initial published release |
| September 2000 | A | Second Printing: Revised trademarks. |
| September 2000 | A | Second Printing: Changed setting of NOL2 bit on page 148. |
| September 2000 | A | Second Printing: Changed bus speed from 60 MHz to 66 MHz in Note 4, Table 58 on page 289. |
| September 2000 | A | Second Printing: Revised headings in Table 59 on page 291, Table 60 on page 292, and Table 61 on page 292. Changed Note 2 in Table 59 on page 291 and Table 60 on page 292 to apply to 400-MHz parts only. |

About this Data Sheet

The *AMD-K6™-III+ Embedded Processor Data Sheet* is the complete specification of the *AMD-K6™-III+ embedded processor*.

Overview

This data sheet is organized into the following sections:

Chapter 1, “AMD-K6™-III+ Embedded Processor” on page 1, provides a list of the AMD-K6-III+ processor’s distinguishing characteristics, a description of the key features, and a discussion about the Super7™ platform initiative.

Chapter 2, “Internal Architecture” on page 11, describes the functional elements of the advanced design techniques, known as the RISC86® microarchitecture, implemented by the AMD-K6-III+ processor.

Chapter 3, “Software Environment” on page 27, provides a general overview of the AMD-K6-2E processor’s x86 software environment and briefly describes the data types, registers, operating modes, interrupts, and instructions supported by the AMD-K6-III+ processor’s architecture and design implementation.

Chapter 4, “Logic Symbol Diagram” on page 91, contains the AMD-K6-III+ processor logic symbol diagram.

Chapter 5, “Signal Descriptions” on page 93, lists the signals and their descriptions alphabetically and by function.

Chapter 6, “AMD PowerNow!™ Technology” on page 143, describes the enhanced power management features available on the low-power versions of the AMD-K6-III+ processor.

Chapter 7, “Bus Cycles” on page 153, describes and illustrates the timing and relationship of bus signals during various types of bus cycles.

Chapter 8, “Power-on Configuration and Initialization” on page 199, describes how the system logic resets the AMD-K6-III+ processor using the RESET signal.

Chapter 9, “Cache Organization” on page 205, describes the basic architecture and resources of the AMD-K6-III+ processor’s internal caches.

Chapter 10, “Write Merge Buffer” on page 229, describes the 8-byte write merge buffer and how merging multiple write cycles into a single write cycle ultimately increases overall system performance.

Chapter 11, “Floating-Point and Multimedia Execution Units” on page 237, describes the AMD-K6-III+ processor’s IEEE 754-compatible and 854-compatible floating point execution unit, the multimedia and 3DNow!™ technology execution units, and the floating-point and MMX™/3DNow! technology instruction compatibility.

Chapter 12, “System Management Mode (SMM)” on page 241, describes SMM, the state-save area, entry into and exit from SMM, exceptions and interrupts in SMM, memory allocation and addressing in SMM, and the SMI# and SMIACT# signals.

Chapter 13, “Test and Debug” on page 251, describes the various test and debug modes that enable the functional and manufacturing testing of systems and boards that use the AMD-K6-III+ processor and that allow designers to debug the instruction execution of software components.

Chapter 14, “Clock Control” on page 277, describes the five modes of clock control supported by the AMD-K6-III+ processor.

Chapter 15, “Electrical Data” on page 287, includes operating ranges, absolute ratings, DC characteristics, power dissipation data, power and grounding information, and decoupling recommendations.

Chapter 16, “Signal Switching Characteristics” on page 297, provides tables listing valid delay, float, setup, and hold timing specifications for the AMD-K6-III+ processor signals.

Chapter 17, “Thermal Design” on page 315, lists the package thermal specifications, discusses how to measure case temperature, and provides layout and airflow considerations.

Chapter 18, “Pin Designations” on page 323, provides top- and bottom-view connection diagrams for each package type and lists the AMD-K6-III+ processor’s pin designations by functional grouping.

Chapter 19, “Package Specifications” on page 333, provides diagrams showing the specifications for the 321-pin CPGA package and the 349-ball OBGA package.

Chapter 20, “Ordering Information” on page 335, provides the ordering part number (OPN) and valid OPN combinations.

1 AMD-K6™-III+ Embedded Processor

The following are key features of the AMD-K6™-III+ processor:

- Member of the AMD-K6™E family of 32-bit embedded processors
 - ◆ Brings the power, performance, and value of the AMD-K6 family to the embedded market
 - ◆ Enables improved time-to-market by leveraging existing hardware and software infrastructure and field-proven development tools
 - ◆ Offers a wide software- and platform-compatible growth path with product longevity to help preserve development investments
- Functionally-compatible embedded version of the AMD-K6-III+ processor with internal 256-Kbyte L2 cache
 - ◆ Provides the highest Super7™ platform performance with reduced total system cost
 - ◆ Microsoft® Windows® compatible processor
 - ◆ x86 binary software compatible
 - ◆ Supports real-time operating systems such as pSOS, QNX, RTX, and VxWorks
- Advanced 6-issue RISC86® superscalar microarchitecture
 - ◆ Ten parallel specialized execution units
 - ◆ Multiple sophisticated x86-to-RISC86 instruction decoders
 - ◆ Advanced two-level branch prediction
 - ◆ Speculative and out-of-order execution
 - ◆ Register renaming and data forwarding
 - ◆ Up to six RISC86 instructions per clock
- Innovative
 - ◆ 320-Kbyte total internal cache
 - Internal split, two-way set associative, 64-Kbyte L1 Cache
 - 32-Kbyte instruction cache with additional 20-Kbytes of predecode cache
 - 32-Kbyte writeback dual-ported data cache
 - MESI protocol support
 - Internal full-speed, four-way set associative, 256-Kbyte, L2 Cache
 - ◆ Multiport internal cache design enabling simultaneous 64-bit reads/writes of L1 and L2 caches
- Super7 platform is Socket 7-compatible
 - ◆ Leverages high-speed 100-MHz processor bus

- ◆ 2x Accelerated Graphic Port (AGP) support
- ◆ Takes advantage of existing system support, logic integration, and designs for superior value
- ◆ Provides an easy upgrade path for embedded applications and a bridge to legacy applications
- AMD PowerNow!™ technology dynamically manages power and performance
 - ◆ Monitors application requirements for performance or power utilization
 - ◆ Supports continuously varying operating frequency and voltage, delivering performance on demand while dissipating the lowest amount of power possible
- 3DNow!™ technology for better multimedia and audio performance
 - ◆ x86 instruction set extension accelerates 3D graphics and other single-precision floating-point compute-intensive operations
 - ◆ Offers fast frame rates on high-resolution graphics applications, superior modeling of real-world environments and physics, life-like images and graphics, and big-screen sound and video
 - ◆ Additional 3DNow! technology DSP instructions enhance communications applications
 - ◆ Separate multiplier and ALU for superscalar instruction execution
- High-performance IEEE 754-compatible and 854-compatible floating-point unit
- High-performance industry-standard MMX™ instructions
 - ◆ Dual-integer ALU for superscalar execution
- Industry-standard System Management Mode (SMM)
- IEEE 1149.1 boundary scan
- 321-Pin Ceramic Pin Grid Array (CPGA) or 349-Ball Organic Ball Grid Array (OBGA) package
- Low-voltage 0.18-micron process technology
 - ◆ Split-plane power with support for full 3.3 V I/O
 - ◆ Lower core voltages enable low-power operation
- Operating frequencies
 - ◆ Standard-power and standard-temperature devices: 400, 450, 500, and 550 MHz
 - ◆ Low-power and extended-temperature devices: 400, 450, and 500 MHz

1.1 AMD-K6™-III+ Embedded Processor Features

The innovative AMD-K6-III+ processor brings industry-leading performance to embedded systems. Its Super7™ platform-compatible, 321-pin ceramic pin grid array (CPGA) or 349-ball organic ball grid array (OBGA) package enables embedded system designers to reduce time-to-market by leveraging today's cost-effective, industry-standard infrastructure.

Manufactured using AMD's 0.18 micron low-power process, the AMD-K6-III+ processor incorporates the innovative and efficient RISC86® microarchitecture, a 320-Kbyte total internal cache, a fast 100 MHz frontside bus, and a powerful IEEE 754-compatible and 854-compatible floating-point execution unit. The AMD-K6-III+ processor also incorporates a superscalar MMX™ unit and AMD's innovative 3DNow! technology for high-performance multimedia and 3D graphics operation.

The AMD-K6-III+ processor is a functionally compatible embedded version of the AMD-K6-III+ processor. The cache design provides a large 64-Kbyte L1 cache and a 256-Kbyte L2 cache operating at full processor speed on a backside bus. The size and speed of the cache memory subsystem gives the AMD-K6-III+ processor a significant performance edge over competing Socket 7 solutions. The low-power versions of the AMD-K6-III+ processor also support AMD's enhanced power management features, called AMD PowerNow! technology.

The AMD-K6-III+ processor is part of the AMD-K6E family of embedded processors. Within this family:

- The AMD-K6-2E processor provides the best value and performance for cost-sensitive embedded applications.
- The AMD-K6-2E+ with its 128-Kbyte internal L2 cache offers higher performance balanced with cost.
- The AMD-K6-III+ with its 256-Kbyte internal L2 cache offers the highest performance available for Super7 and Socket 7 platforms.

All AMD-K6E family processors in the CPGA package share the same footprint and support the Socket 7-compatible Super7 platform. The AMD-K6E family provides embedded designers with an assured growth plan and supply stability, along with product longevity. All AMD-K6E family processors are x86-binary compatible, allowing preservation of the initial software investment.

The AMD-K6-III+ embedded processor is particularly well-suited for use in applications where high performance is required. It is designed to offer compelling, yet affordable, power and performance for high-end embedded applications, such as information appliances, set-top boxes, embedded PCs, point-of-sale terminals, public and private communications infrastructure, and industrial control.

The AMD-K6-III+ embedded processor is available in two versions.

- The low-power version operates at the lowest core voltage in order to offer the lowest available power and extended temperature ratings. Enhanced power management features are provided via AMD PowerNow! technology in the low-power versions of the processor.
- The standard-power version has a 2.0-V core voltage and offers standard power and temperature specifications similar to desktop PC processors.

Innovative Cache Design for Faster Data Access

Recognizing the benefits of a large and fast cache design in feeding performance-hungry applications, AMD developed an innovative cache architecture that enhances the performance available for embedded applications based on the Super7 platform.

AMD's cache design innovations include:

- An internal 256-Kbyte L2 write-back cache operating at the full speed of the processor and complementing the 64-Kbyte L1 cache, which is standard in all AMD-K6 family processors.
- A multiport internal cache design, enabling simultaneous 64-bit reads and writes to both the L1 cache and the L2 cache.
- A 4-way set associative backside L2 cache design enabling optimal data management and external frontside data bus bandwidth efficiency.

The processor's multiport internal cache design enables both the 64-Kbyte L1 cache and the 256-Kbyte L2 cache to perform simultaneous 64-bit read and write operations in a clock cycle. This multiport capability allows data to be processed faster and more efficiently than non-multiported designs. In addition, the processor core can access both L1 and L2 caches simultaneously, which further enhances overall CPU throughput.

The cache design is exceptionally fast, with the backside 256-Kbyte L2 cache operating at full processor speed.

For example, the internal L2 cache of an AMD-K6-III+/450 processor operates at 450 MHz and provides nine times the peak bandwidth of an external L2 cache operating at 100 MHz. The maximum peak bandwidth of an external L2 cache operating at 100 MHz is 800 Mbytes/s, while an internal L2 cache operating at 450 MHz delivers a maximum peak bandwidth of 3,600 Mbytes/s per port. Because the internal L2 cache of the AMD-K6-III+ processor is dual-ported for simultaneous reads and writes, the total peak bandwidth is doubled to 7,200 Mbytes/s, resulting in a maximum peak bandwidth nine times as large as a 100-MHz cache implementation.

3DNow!™ Technology

The AMD-K6-III+ processor supports AMD's 3DNow! technology, an extension to the x86 instruction set that includes 21 new instructions to accelerate 3D graphics and other single-precision floating-point compute intensive operations.

3DNow! technology was defined and implemented in collaboration with Microsoft, application developers, and graphics vendors, and has received an enthusiastic reception. It is compatible with today's existing x86 software and requires no operating system support, thereby enabling a broad class of applications to benefit from 3DNow! technology.

The worldwide installed base of 3DNow! technology-enhanced PCs has grown to more than 25 million desktop and notebook systems, revolutionizing the 3D experience with up to four times the peak floating-point performance of previous sixth generation solutions. Support for 3DNow! technology exists today in leading industry-standard APIs, including Microsoft® DirectX and SGI's OpenGL APIs. Additionally, numerous hardware and software products have been optimized for 3DNow! technology. AMD is now bringing this advanced capability to embedded systems.

3DNow! technology enables fast frame rates on high-resolution 3D-rendered scenes, realistic physical modeling of real-world environments, sharp and detailed 3D imaging, smooth video playback, and theater-quality audio.

In addition, the AMD-K6-III+ processor adds support for five new digital signal processing (DSP) instructions, developed to enhance the performance of communications applications, including soft xDSL modems, MP3 recording, and Dolby Digital and Surround Sound processing.

AMD PowerNow!™ Technology for Enhanced Power Management

AMD has added a number of new power management features to the low-power versions of the AMD-K6-III+ processor. Collectively, these hardware and software features are called AMD PowerNow!™ technology.

AMD PowerNow! technology allows the AMD-K6-III+ processor to run at different frequencies and voltages, depending on the application's need for maximum performance or the most efficient power utilization.

AMD PowerNow! technology includes AMD's unique "automatic mode" feature, which allows the system to monitor application usage and to continuously vary the operating frequency and voltage, delivering performance on demand while dissipating the lowest amount of power possible.

- When application demands require the processor to run at maximum performance, the AMD PowerNow! technology steps up the performance to meet the demand.
- As platform demand for performance subsides, AMD PowerNow! technology can dynamically drop into a lower power state.

AMD PowerNow! technology enables embedded products to dynamically manage power and performance.

System Management Mode and Other Power Management Features

The AMD-K6-III+ processor includes the complete industry-standard system management mode (SMM), which is critical to system resource and power management.

The AMD-K6-III+ processor also features the industry-standard Stop-Clock (STPCLK#) control circuitry and the Halt instruction, both required for implementing the ACPI power management specification.

Microarchitecture

The AMD-K6-III+ processor's 6-issue RISC86 microarchitecture is a decoupled decode/execution superscalar design that implements state-of-the-art design techniques to achieve leading-edge performance.

Advanced design techniques implemented in the AMD-K6-III+ processor include multiple x86 instruction decode, single-clock internal RISC operations, ten execution units that support superscalar operation, out-of-order execution, data forwarding, speculative execution, and register renaming.

In addition, the processor supports advanced branch prediction logic by implementing an 8192-entry branch history table, a branch target cache, and a return address stack, which combine to deliver better than a 95% prediction rate. These design techniques enable the AMD-K6-III+ processor to issue, execute, and retire multiple x86 instructions per clock, resulting in excellent scaleable performance.

Industry-Standard x86 Architecture

The AMD-K6-III+ processor is x86 binary code compatible. AMD's extensive experience through six generations of x86 processors has been carefully integrated into the processor to enable compatibility with Windows® 98, Windows 95, Windows 3.x, Windows NT, DOS, Linux, OS/2, Unix, Solaris, NetWare®, and other leading x86 operating systems and applications. The AMD-K6-III+ processor is also compatible with leading real-time operating systems (RTOS) commonly used in embedded applications, such as pSOS, QNX, RTX, and VxWorks. Additionally, the AMD FusionE86SM third-party tool support program offers extensive development support for AMD-K6-III+ processor designs.

The AMD-K6-III+ processor is Super7 and Socket 7-compatible. The Super7 platform is an extension to the popular and robust Socket 7 platform. See "Super7™ Platform" on page 8 for more information.

AMD is the world's second-leading supplier of PC processors compatible with the Windows operating system, having shipped more than 120 million x86 microprocessors, including more than 60 million Windows-compatible processors. The AMD-K6-III+ processor for embedded applications is the latest member in this long line of processors. With its combination of state-of-the-art features, industry-leading performance, high-performance 3DNow! technology and multimedia engines, x86 compatibility, and low-cost infrastructure, the AMD-K6-III+ processor is the superior choice for high-performance embedded systems.

1.2 Process Technology

The AMD-K6-III+ processor is implemented using an AMD-developed, state-of-the-art low power 0.18-micron process technology. This process technology features a split-plane design that enables the AMD-K6-III+ processor to deliver excellent performance solutions while utilizing a lower processor core voltage, which results in lower power consumption, while the I/O portion operates at the industry-standard 3.3-V level.

1.3 Super7™ Platform

The Super7 platform is an extension to the popular Socket 7 platform. AMD and its industry partners have invested in the future of Socket 7 with the Super7 platform initiative. The goal of the initiative is to maintain the competitive vitality of the Socket 7 infrastructure through a series of enhancements, including the development of an industry-standard 100-MHz processor bus protocol.

In addition to the 100-MHz processor bus protocol, the Super7 initiative includes the introduction of chipsets that support the AGP specification, and support for a backside L2 cache. Currently, over 40 motherboard vendors and all major BIOS and chipset vendors offer Super7-based products.

All AMD-K6 embedded processors in CPGA packages remain pin compatible with existing Socket 7 solutions; however, for maximum system performance, the AMD-K6-III+ processor works optimally in Super7 designs that incorporate advanced features such as support for the 100-MHz frontside bus and AGP graphics.

100-MHz Processor Bus

The AMD-K6-III+ processor supports a 100-MHz, 800 Mbyte/second frontside bus to provide a high-speed interface to Super7 platform-based chipsets. The 100-MHz interface speeds up access to main memory by 50 percent over the 66-MHz Socket 7 interface—resulting in a significant 10 percent increase in overall system performance.

Accelerated Graphics Port Support

Accelerated Graphics Port (AGP) support improves the performance of video graphics systems that have small amounts of video memory on the graphics card. The industry-standard AGP specification enables a 133-MHz graphics interface and will scale to even higher levels of performance.

Support For Backside L2 Cache

The Super7 platform has the ‘headroom’ to support higher-performance AMD-K6 processors like the AMD-K6-III+ processor, which features a full-speed, internal backside 256-Kbyte L2 cache designed to enable new levels of performance to leading-edge embedded systems.

Super7™ Platform Advantages

The Super7 platform:

- Delivers performance and features competitive with alternate platforms at the same clock speed, and at a significantly lower cost
- Takes advantage of existing system designs for superior value
- Enables OEMs and resellers to take advantage of mature, high-volume infrastructure supported by multiple BIOS, chipset, graphics, and motherboard suppliers
- Reduces inventory and design costs with one motherboard for a wide range of products
- Builds on a huge installed base of more than 100 million motherboards
- Provides an easy upgrade path for embedded applications, as well as a bridge to legacy applications

By taking advantage of the low-cost, mature Socket 7 infrastructure, the Super7 platform will continue to provide superior value and leading-edge performance for embedded systems.

2 Internal Architecture

The AMD-K6-III+ processor implements advanced design techniques known as the RISC86 microarchitecture. The RISC86 microarchitecture is a decoupled decode/execution design approach that yields superior sixth-generation performance for x86-based software. This chapter describes the techniques used and the functional elements of the RISC86 microarchitecture.

2.1 Microarchitecture Overview

When discussing processor design, it is important to understand the terms *architecture*, *microarchitecture*, and *design implementation*.

- *Architecture* refers to the instruction set and features of a processor that are visible to software programs running on the processor. The architecture determines what software the processor can run. The architecture of the AMD-K6-III+ processor is the industry-standard x86 instruction set.
- *Microarchitecture* refers to the design techniques used in the processor to reach the target cost, performance, and functionality goals. The AMD-K6 family of processors are based on a sophisticated RISC core known as the Enhanced RISC86 microarchitecture. The Enhanced RISC86 microarchitecture is an advanced, second-order decoupled decode/execution design approach that enables industry-leading performance for x86-based software.
- *Design implementation* refers to the actual logic and circuit designs from which the processor is created according to the microarchitecture specifications.

**Enhanced RISC86[®]
Microarchitecture**

The Enhanced RISC86 microarchitecture defines the characteristics of the AMD-K6 family of processors. The innovative RISC86 microarchitecture approach implements the x86 instruction set by internally translating x86 instructions into RISC86 operations. These RISC86 operations were specially designed to include direct support for the x86 instruction set while observing the RISC performance principles of fixed length encoding, regularized instruction fields, and a large register set.

The Enhanced RISC86 microarchitecture used in the AMD-K6-III+ processor enables higher processor core performance and promotes straightforward extensions, such as those added in the current AMD-K6-III+ processor and those planned for the future. Instead of directly executing complex x86 instructions, which have lengths of 1 to 15 bytes, the AMD-K6-III+ processor executes the simpler and easier fixed-length RISC86 operations, while maintaining the instruction coding efficiencies found in x86 programs.

The AMD-K6-III+ processor contains parallel decoders, a centralized RISC86 operation scheduler, and ten execution units that support superscalar operation—multiple decode, execution, and retirement—of x86 instructions. These elements are packed into an aggressive and highly efficient six-stage pipeline.

**AMD-K6™-III+
Processor Block
Diagram**

As shown in Figure 1 on page 13, the high-performance, out-of-order execution engine of the AMD-K6-III+ processor is mated to a split, level-one, 64-Kbyte, writeback cache with 32 Kbytes of instruction cache and 32 Kbytes of data cache. Backing up the level-one (L1) cache is a large, unified, level-two (L2), 256-Kbyte, writeback cache. The L1 instruction cache feeds the decoders and, in turn, the decoders feed the scheduler. The ICU issues and retires RISC86 operations contained in the scheduler. The system bus interface is an industry-standard 64-bit Super7 and Socket 7 demultiplexed bus.

The AMD-K6-III+ processor combines the latest in processor microarchitecture to provide the highest x86 performance for today's computational systems. The AMD-K6-III+ processor offers true sixth-generation performance and x86 binary software compatibility.

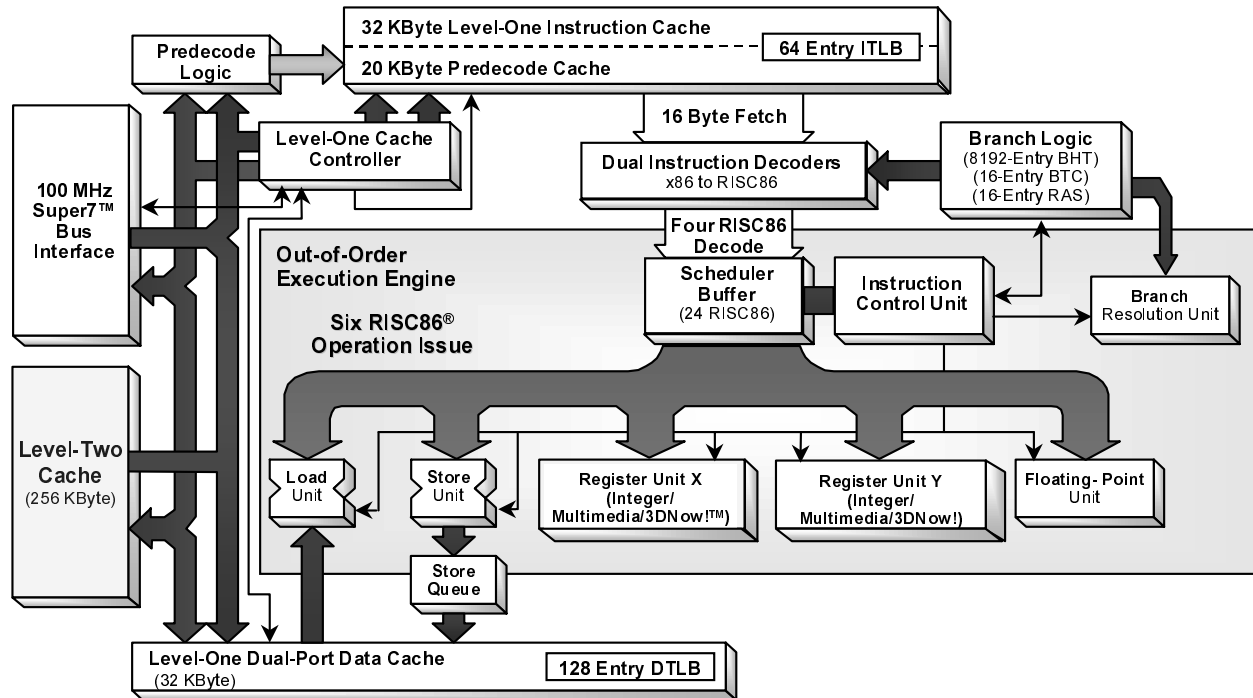


Figure 1. AMD-K6™-III+ Processor Block Diagram

Decoders

Decoding of the x86 instructions begins when the on-chip L1 instruction cache is filled. Predecode logic determines the length of an x86 instruction on a byte-by-byte basis. This predecode information is stored, along with the x86 instructions, in the L1 instruction cache, to be used later by the decoders. The decoders translate on-the-fly, with no additional latency, up to two x86 instructions per clock into RISC86 operations.

Note: In this chapter, “clock” refers to a processor clock.

The AMD-K6-III+ processor categorizes x86 instructions into three types of decodes—short, long, and vector. The decoders process either two short, one long, or one vector decode at a time.

The three types of decodes have the following characteristics:

- Short decodes—x86 instructions less than or equal to seven bytes in length

- Long decodes—x86 instructions less than or equal to 11 bytes in length
- Vector decodes—complex x86 instructions

Short and long decodes are processed completely within the decoders. Vector decodes are started by the decoders and then completed by fetched sequences from an on-chip ROM. After decoding, the RISC86 operations are delivered to the scheduler for dispatching to the executions units.

Scheduler/Instruction Control Unit

The centralized scheduler or buffer is managed by the Instruction Control Unit (ICU). The ICU buffers and manages up to 24 RISC86 operations at a time. This equals from 6 to 12 x86 instructions. This buffer size (24) is perfectly matched to the processor's six-stage RISC86 pipeline and four RISC86-operations decode rate.

The scheduler accepts as many as four RISC86 operations at a time from the decoders and retires up to four RISC86 operations per clock cycle. The ICU is capable of simultaneously issuing up to six RISC86 operations at a time to the execution units. This consists of the following types of operations:

- Memory load operation
- Memory store operation
- Complex integer, MMX or 3DNow! register operation
- Simple integer, MMX or 3DNow! register operation
- Floating-point register operation
- Branch condition evaluation

Registers

When managing the RISC86 operations, the ICU uses 69 physical registers contained within the RISC86 microarchitecture.

- Forty-eight of the physical registers are located in a general register file.
 - Twenty-four of these are rename registers.
 - The other twenty-four are committed or architectural registers, consisting of 16 scratch registers and 8 registers that correspond to the x86 general-purpose registers—EAX, EBX, ECX, EDX, EBP, ESP, ESI, and EDI.

- An analogous set of 21 registers is available specifically for MMX and 3DNow! operations.
 - Twelve of these are MMX/3DNow! rename registers.
- Nine are MMX/3DNow! committed or architectural registers, consisting of one scratch register and eight registers that correspond to the MMX registers (mm0–mm7, as shown in Figure 17 on page 35).

Branch Logic

The AMD-K6-III+ processor is designed with highly sophisticated dynamic branch logic consisting of the following:

- Branch history/prediction table
- Branch target cache
- Return address stack

The AMD-K6-III+ processor implements a two-level branch prediction scheme based on an 8192-entry branch history table. The branch history table stores prediction information that is used for predicting conditional branches. Because the branch history table does not store predicted target addresses, special address ALUs calculate target addresses on the fly during instruction decode.

The branch target cache augments predicted branch performance by avoiding a one clock cache-fetch penalty. This specialized target cache does this by supplying the first 16 bytes of target instructions to the decoders when branches are predicted. The return address stack is a unique device specifically designed for optimizing CALL and RETURN pairs. In summary, the AMD-K6-III+ processor uses dynamic branch logic to minimize delays due to the branch instructions that are common in x86 software.

3DNow!™ Technology

AMD has taken a lead role in improving the multimedia and 3D capabilities of the x86 processor family with the introduction of 3DNow! technology, which uses a packed, single-precision, floating-point data format and Single Instruction Multiple Data (SIMD) operations based on the MMX technology model.

2.2 Cache, Instruction Prefetch, and Predecode Bits

The writeback level-one (L1) cache on the AMD-K6-III+ processor is organized as a separate 32-Kbyte instruction cache and a 32-Kbyte data cache with two-way set associativity.

The level-two (L2) cache is 256 Kbytes, and is organized as a unified, four-way set-associative cache. The cache line size is 32 bytes, and lines are fetched from external memory using an efficient pipelined burst transaction.

As the L1 instruction cache is filled from the L2 cache or from external memory, each instruction byte is analyzed for instruction boundaries using predecoding logic. Predecoding annotates information (5 bits per byte) to each instruction byte that later enables the decoders to efficiently decode multiple instructions simultaneously.

Cache

The processor cache design takes advantage of a sectored organization (see Figure 2). Each sector consists of 64 bytes configured as two 32-byte cache lines. The two cache lines of a sector share a common tag but have separate pairs of MESI (Modified, Exclusive, Shared, Invalid) bits that track the state of each cache line.

| | | | | | | | | | | |
|-------------|--------------|---------|----------------|---------|----------------|-------|-------|--------|----------------|-----------|
| Tag Address | Cache Line 0 | Byte 31 | Predecode Bits | Byte 30 | Predecode Bits | | | Byte 0 | Predecode Bits | MESI Bits |
| | Cache Line 1 | Byte 31 | Predecode Bits | Byte 30 | Predecode Bits | | | Byte 0 | Predecode Bits | MESI Bits |

Figure 2. Cache Sector Organization

Two forms of cache misses and associated cache fills can take place—a tag-miss cache fill and a tag-hit cache fill.

- *Tag-Miss Cache Fill*—The L1 cache miss is due to a tag mismatch, in which case the required cache line is filled either from the L2 cache or from external memory, and the L1 cache line within the sector that was not required is marked as invalid.
- *Tag-Hit Cache Fill*—The address matches the tag, but the requested cache line is marked as invalid. The required L1 cache line is filled from the L2 cache or from external memory, and the L1 cache line within the sector that is not required remains in the same cache state.

Prefetching

The AMD-K6-III+ processor conditionally performs cache prefetching, which results in the filling of the required cache line first, and a prefetch of the second cache line making up the other half of the sector. From the perspective of the external bus, the two cache-line fills typically appear as two 32-byte burst read cycles occurring back-to-back or, if allowed, as pipelined cycles.

The 3DNow! technology includes an instruction called PREFETCH that allows a cache line to be prefetched into the L1 data cache and the L2 cache. The PREFETCH instruction format is defined in Table 15, “3DNow!™ Instructions,” on page 89. For more detailed information, see the *3DNow!™ Technology Manual*, order# 21928.

Predecode Bits

Decoding x86 instructions is particularly difficult because the instructions are variable-length and can be from 1 to 15 bytes long. Predecode logic supplies the five predecode bits that are associated with each instruction byte. The predecode bits indicate the number of bytes to the start of the next x86 instruction. The predecode bits are stored in an extended instruction cache alongside each x86 instruction byte as shown in Figure 2 on page 16. The predecode bits are passed with the instruction bytes to the decoders where they assist with parallel x86 instruction decoding.

2.3 Instruction Fetch and Decode

Instruction Fetch

The processor can fetch up to 16 bytes per clock out of the L1 instruction cache or branch target cache. The fetched information is placed into a 16-byte instruction buffer that feeds directly into the decoders (see Figure 3 on page 18). Fetching can occur along a single execution stream with up to seven outstanding branches taken.

The instruction fetch logic is capable of retrieving any 16 contiguous bytes of information within a 32-byte boundary. There is no additional penalty when the 16 bytes of instructions lie across a cache line boundary. The instruction bytes are loaded into the instruction buffer as they are consumed by the decoders.

Although instructions can be consumed with byte granularity, the instruction buffer is managed on a memory-aligned word

(two bytes) organization. Therefore, instructions are loaded and replaced with word granularity. When a control transfer occurs—such as a JMP instruction—the entire instruction buffer is flushed and reloaded with a new set of 16 instruction bytes.

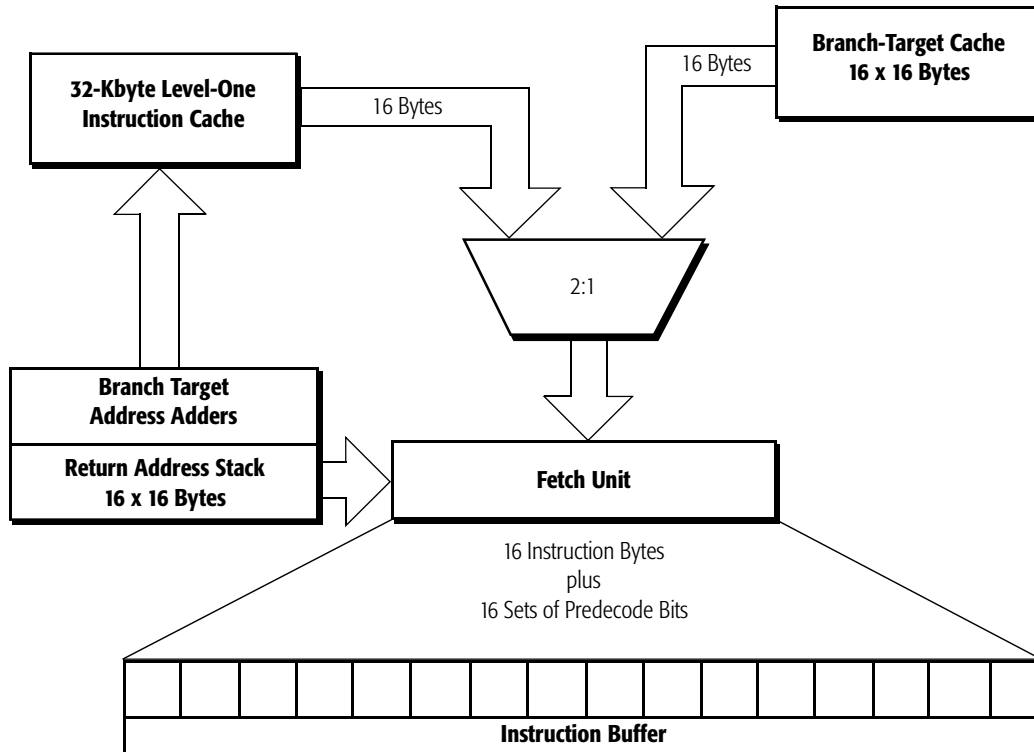


Figure 3. The Instruction Buffer

Instruction Decode

The AMD-K6-III+ processor decode logic is designed to decode multiple x86 instructions per clock (see Figure 4 on page 19). The decode logic accepts x86 instruction bytes and their predecode bits from the instruction buffer, locates the actual instruction boundaries, and generates RISC86 operations from these x86 instructions.

RISC86 operations are fixed-length internal instructions. Most RISC86 operations execute in a single clock. RISC86 operations are combined to perform every function of the x86 instruction set. Some x86 instructions are decoded into as few as zero RISC86 operations—for instance a NOP—or one RISC86

operation—a register-to-register add. More complex x86 instructions are decoded into several RISC86 operations.

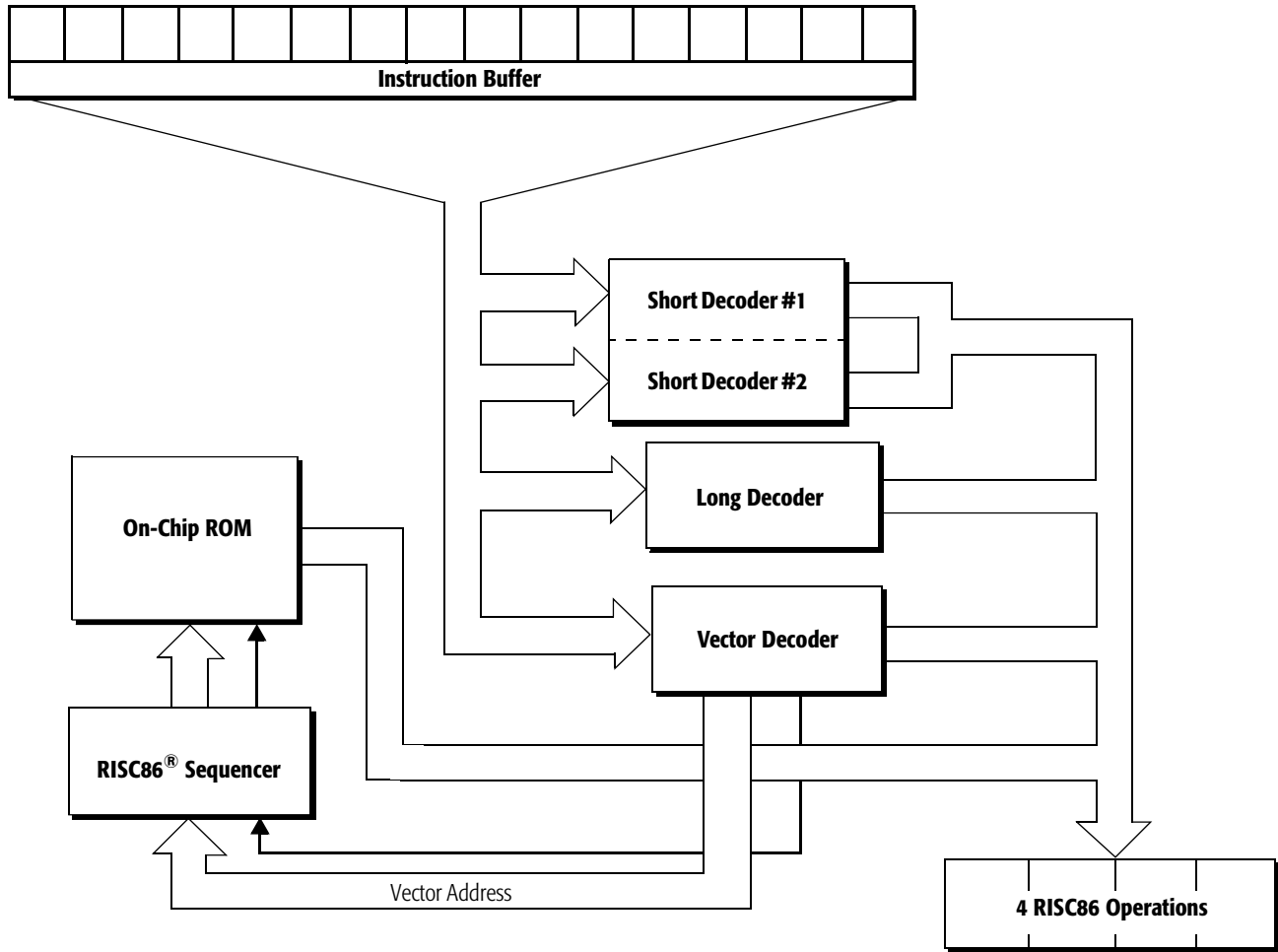


Figure 4. AMD-K6™-III+ Processor Decode Logic

The AMD-K6-III+ processor uses a combination of decoders to convert x86 instructions into RISC86 operations. The hardware consists of three sets of decoders—two parallel short decoders, one long decoder, and one vector decoder.

Parallel Short Decoders. The two parallel short decoders translate the most commonly-used x86 instructions (moves, shifts, branches, ALU, FPU) and the extensions to the x86 instruction set (including MMX and 3DNow! instructions) into zero, one, or two RISC86 operations each. The short decoders only operate on x86 instructions that are up to seven bytes long. In addition,

they are designed to decode up to two x86 instructions per clock.

Long Decoder. The commonly-used x86 instructions that are greater than seven bytes but not more than 11 bytes long and less-commonly-used x86 instructions that are up to seven bytes long are handled by the long decoder. The long decoder only performs one decode per clock and generates up to four RISC86 operations.

Vector Decoder. All other translations (complex instructions, serializing conditions, interrupts and exceptions, etc.) are handled by a combination of the vector decoder and RISC86 operation sequences fetched from an on-chip ROM. For complex operations, the vector decoder logic provides the first set of RISC86 operations and a vector (initial ROM address) to a sequence of further RISC86 operations. The same types of RISC86 operations are fetched from the ROM as those that are generated by the hardware decoders.

Note: Although all three sets of decoders are simultaneously fed a copy of the instruction buffer contents, only one of the three types of decoders is used during any one decode clock.

Grouped Operations. The decoders or the on-chip RISC86 ROM always generate a group of four RISC86 operations. For decodes that cannot fill the entire group with four RISC86 operations, RISC86 NOP operations are placed in the empty locations of the grouping. For example, a long-decoded x86 instruction that converts to only three RISC86 operations is padded with a single RISC86 NOP operation and then passed to the scheduler. Up to six groups or 24 RISC86 operations can be placed in the scheduler at a time.

Floating Point Instructions. All of the common, and a few of the uncommon, floating-point instructions (also known as ESC instructions) are hardware decoded as short decodes. This decode generates a RISC86 floating-point operation and, optionally, an associated floating-point load or store operation. Floating-point or ESC instruction decode is only allowed in the first short decoder, but non-ESC instructions can be decoded simultaneously by the second short decoder along with an ESC instruction decode in the first short decoder.

MMX™ and 3DNow!™ Instructions. All of the MMX and 3DNow! instructions, with the exception of the EMMS, FEMMS, and PREFETCH instructions, are hardware decoded as short decodes. The MMX instruction decode generates a RISC86 MMX operation and, optionally, an associated MMX load or store operation. A 3DNow! instruction decode generates a RISC86 3DNow! operation and, optionally, an associated load or store operation. MMX and 3DNow! instructions can be decoded in either or both of the short decoders.

2.4 Centralized Scheduler

The scheduler is the heart of the AMD-K6-III+ processor (see Figure 5 on page 22). It contains the logic necessary to manage out-of-order execution, data forwarding, register renaming, simultaneous issue and retirement of multiple RISC86 operations, and speculative execution.

The scheduler's buffer can hold up to 24 RISC86 operations. This equates to a maximum of 12 x86 instructions. The scheduler can issue RISC86 operations from any of the 24 locations in the buffer. When possible, the scheduler can simultaneously issue a RISC86 operation to any available execution unit (store, load, branch, register X integer/multimedia, register Y integer/multimedia, or floating-point). In total, the scheduler can issue up to six and retire up to four RISC86 operations per clock.

The main advantage of the scheduler and its operation buffer is the ability to examine an x86 instruction window equal to 12 x86 instructions at one time. This advantage is due to the fact that the scheduler operates on the RISC86 operations in parallel and allows the AMD-K6-III+ processor to perform dynamic on-the-fly instruction code scheduling for optimized execution. Although the scheduler can issue RISC86 operations for out-of-order execution, it always retires x86 instructions in order.

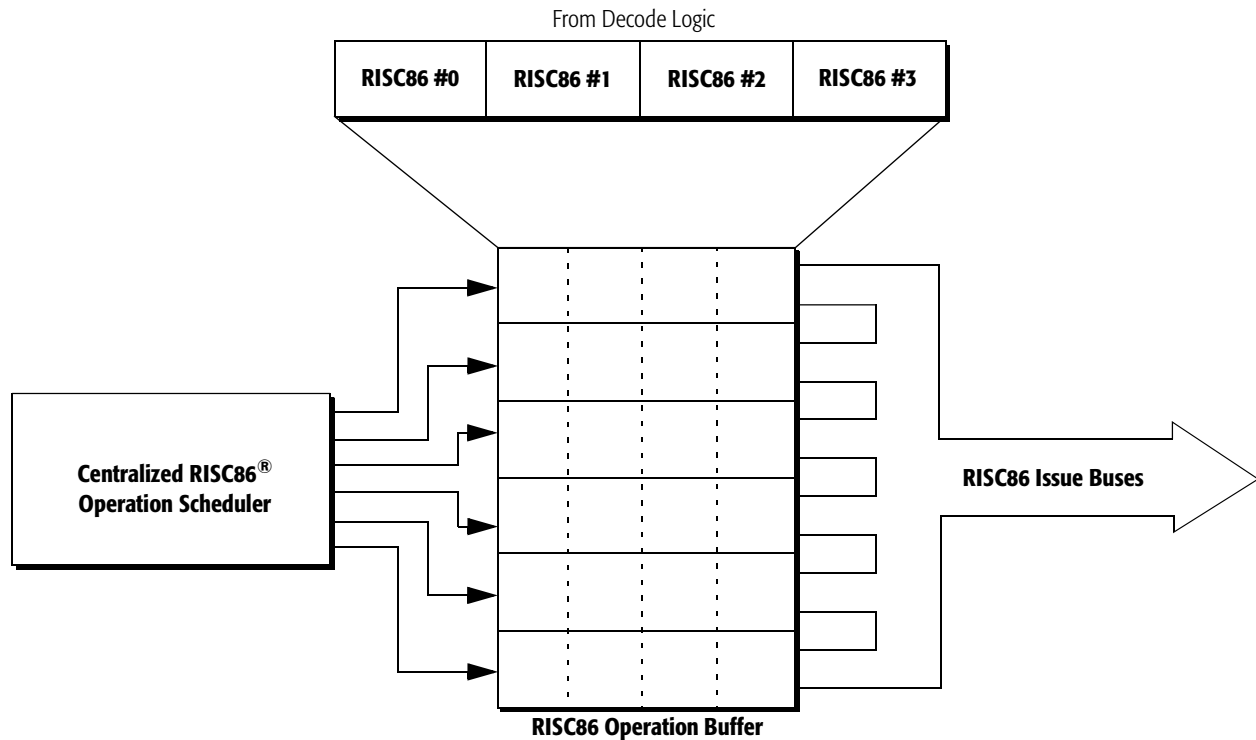


Figure 5. AMD-K6™-III+ Processor Scheduler

2.5 Execution Units

The AMD-K6-III+ processor contains ten parallel execution units—store, load, integer X ALU, integer Y ALU, MMX ALU (X), MMX ALU (Y), MMX/3DNow! multiplier, 3DNow! ALU, floating-point, and branch condition. Each unit is independent and capable of handling the RISC86 operations issued to it. Table 1 on page 23 details the execution units, functions performed within these units, operation latency, and operation throughput.

Note that the integer, MMX, and 3DNow! execution units share the register X and Y issue pipelines. See “Register X and Y Pipelines” on page 24.

The store and load execution units are two-stage pipelined designs.

- The store unit performs data writes and register calculation for LEA/PUSH instructions. Data memory and register

writes from stores are available after one clock. Store operations are held in a store queue prior to execution. From there, they execute in order.

- The load unit performs data memory reads. Data is available from the load unit after two clocks.

The Integer X execution unit can operate on all ALU operations, multiplies, divides (signed and unsigned), shifts, and rotates.

The Integer Y execution unit can operate on the basic word and doubleword ALU operations—ADD, AND, CMP, OR, SUB, XOR, zero-extend and sign-extend operands.

The branch condition unit is separate from the branch prediction logic (see “Branch-Prediction Logic” on page 25) in that it resolves conditional branches such as JCC and LOOP after the branch condition has been evaluated.

Table 1. Execution Latency and Throughput of Execution Units

| Functional Unit | Function | Latency | Throughput |
|---|--|---------|------------|
| Store | LEA/PUSH, Address (Pipelined) | 1 | 1 |
| | Memory Store (Pipelined) | 1 | 1 |
| Load | Memory Loads (Pipelined) | 2 | 1 |
| Integer X | Integer ALU | 1 | 1 |
| | Integer Multiply | 2–3 | 2–3 |
| | Integer Shift | 1 | 1 |
| Multimedia (processes MMX instructions) | MMX ALU | 1 | 1 |
| | MMX Shifts, Packs, Unpack | 1 | 1 |
| | MMX Multiply | 2 | 1 |
| Integer Y | Basic ALU (16-bit and 32-bit operands) | 1 | 1 |
| Branch | Resolves Branch Conditions | 1 | 1 |
| FPU | FADD, FSUB, FMUL | 2 | 2 |
| 3DNow! | 3DNow! ALU | 2 | 1 |
| | 3DNow! Multiply | 2 | 1 |
| | 3DNow! Convert | 2 | 1 |

Register X and Y Pipelines

The functional units that execute MMX and 3DNow! instructions share pipeline control with the Integer X and Integer Y units.

The register X and Y functional units are attached to the issue bus for the register X execution pipeline or the issue bus for the register Y execution pipeline or both.

Each register pipeline has dedicated resources that consist of an integer execution unit and an MMX ALU execution unit, therefore allowing superscalar operation on integer and MMX instructions.

In addition, both the X and Y issue buses are connected to the 3DNow! ALU, the MMX/3DNow! multiplier and MMX shifter, which allows the appropriate RISC86 operation to be issued through either bus. Figure 6 shows the details of the X and Y register pipelines.

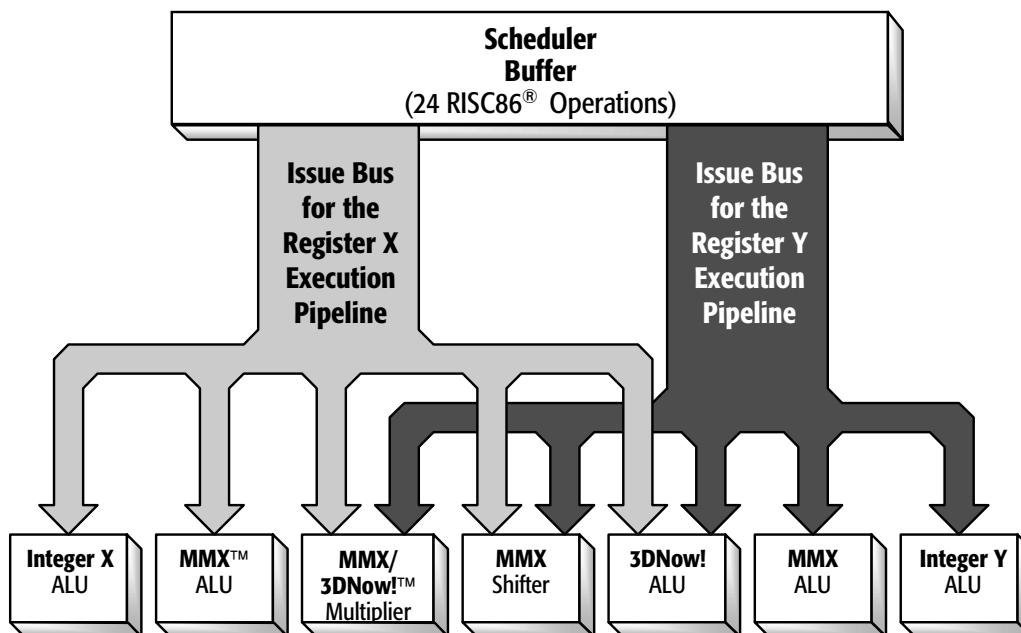


Figure 6. Register X and Y Pipeline Functional Units

2.6 Branch-Prediction Logic

Sophisticated branch logic that can minimize or hide the impact of changes in program flow is designed into the AMD-K6-III+ processor. Branches in x86 code fit into two categories:

- *Unconditional branches* always change program flow (that is, the branches are always taken)
- *Conditional branches* may or may not divert program flow (that is, the branches are taken or not-taken). When a conditional branch is not taken, the processor simply continues decoding and executing the next instructions in memory.

Typical applications have up to 10% of unconditional branches and another 10% to 20% conditional branches. The AMD-K6-III+ processor branch logic has been designed to handle this type of program behavior and to minimize its negative effects on instruction execution, such as stalls due to delayed instruction fetching and the draining of the processor pipeline. The branch logic contains an 8192-entry branch history table, a 16-entry by 16-byte branch target cache, a 16-entry return address stack, and a branch execution unit.

Branch History Table

The AMD-K6-III+ processor handles unconditional branches without any penalty by redirecting instruction fetching to the target address of the unconditional branch. However, conditional branches require the use of the dynamic branch-prediction mechanism built into the AMD-K6-III+ processor.

A two-level adaptive history algorithm is implemented in an 8192-entry branch history table. This table stores executed branch information, predicts individual branches, and predicts the behavior of groups of branches.

To accommodate the large branch history table, the AMD-K6-III+ processor does not store predicted target addresses. Instead, the branch target addresses are calculated on-the-fly using ALUs during the decode stage. The adders calculate all possible target addresses before the instructions are fully decoded and the processor chooses which addresses are valid.

Branch Target Cache

To avoid a one clock cache-fetch penalty when a branch is predicted taken, a built-in branch target cache supplies the first 16 bytes of instructions directly to the instruction buffer (assuming the target address hits this cache). (See Figure 3 on page 18.)

The branch target cache is organized as 16 entries of 16 bytes. In total, the branch prediction logic achieves branch prediction rates greater than 95%.

Return Address Stack

The return address stack is a special device designed to optimize CALL and RET pairs. Software is typically compiled with subroutines that are frequently called from various places in a program. This is usually done to save space.

Entry into the subroutine occurs with the execution of a CALL instruction. At that time, the processor pushes the address of the next instruction in memory following the CALL instruction onto the stack (allocated space in memory). When the processor encounters a RET instruction (within or at the end of the subroutine), the branch logic pops the address from the stack and begins fetching from that location. To avoid the latency of main memory accesses during CALL and RET operations, the return address stack caches the pushed addresses.

Branch Execution Unit

The branch execution unit enables efficient speculative execution. This unit gives the processor the ability to execute instructions beyond conditional branches before knowing whether the branch prediction was correct.

The AMD-K6-III+ processor does not permanently update the x86 registers or memory locations until all speculatively executed conditional branch instructions are resolved. When a prediction is incorrect, the processor backs out to the point of the mispredicted branch instruction and restores all registers. The AMD-K6-III+ processor can support up to seven outstanding branches.

3 Software Environment

This chapter provides a general overview of the AMD-K6-III+ processor's x86 software environment and briefly describes the data types, registers, operating modes, interrupts, and instructions supported by the AMD-K6-III+ processor architecture and design implementation.

The AMD-K6-III+ processor implements the same ten Model-Specific Registers (MSRs) as the AMD-K6-2 and AMD-K6-2E processors Model 8/[F:8], and the bits and fields within these ten MSRs are defined identically. The AMD-K6-III+ processor supports an additional MSR for cache control. The low-power versions of the AMD-K6-III+ processor support a twelfth MSR to control the AMD PowerNow! technology functions.

See “Model-Specific Registers (MSR)” on page 44 for the MSR definitions.

The model number for the AMD-K6-III+ processor is Model D/[3:0], where the actual stepping can be any value in the range [3:0].

3.1 Registers

The AMD-K6-III+ processor contains all the registers defined by the x86 architecture, including general-purpose, segment, floating-point, MMX/3DNow!, EFLAGS, control, task, debug, test, and descriptor/memory-management registers.

In addition, this chapter provides information on the AMD-K6-III+ processor MSRs.

Note: *Areas of the register designated as Reserved should not be modified by software.*

General-Purpose Registers

The eight 32-bit x86 general-purpose registers are used to hold integer data or memory pointers used by instructions. Table 2 contains a list of the general-purpose registers and the functions for which they are used.

Table 2. General-Purpose Registers

| Register | Function |
|----------|--|
| EAX | Commonly used as an accumulator |
| EBX | Commonly used as a pointer |
| ECX | Commonly used for counting in loop operations |
| EDX | Commonly used to hold I/O information and to pass parameters |
| EDI | Commonly used as a destination pointer by the ES segment |
| ESI | Commonly used as a source pointer by the DS segment |
| ESP | Used to point to the stack segment |
| EBP | Used to point to data within the stack segment |

In order to support byte and word operations, EAX, EBX, ECX, and EDX can also be used as 8-bit and 16-bit registers. The shorter registers are overlaid on the longer ones. For example, the name of the 16-bit version of EAX is AX (low 16 bits of EAX) and the 8-bit names for AX are AH (high order bits) and AL (low order bits). The same naming convention applies to EBX, ECX, and EDX.

EDI, ESI, ESP, and EBP can be used as smaller 16-bit registers called DI, SI, SP, and BP respectively, but these registers do not have 8-bit versions. Figure 7 shows the EAX register with its name components, and Table 3 on page 29 lists the doubleword (32-bit) general-purpose registers and their corresponding word (16-bit) and byte (8-bit) versions.

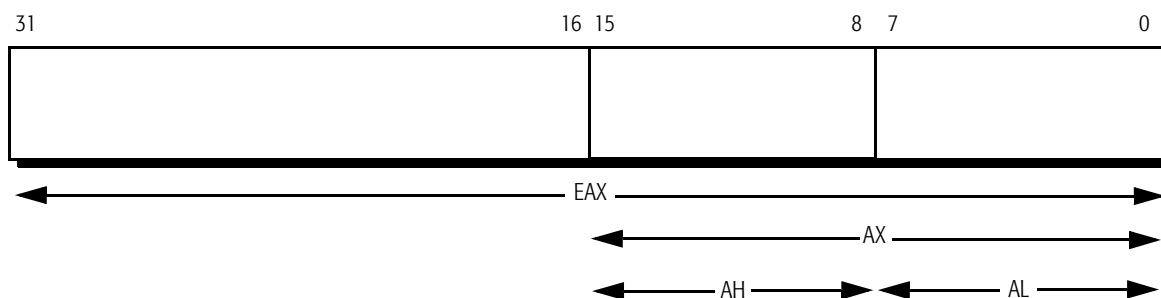


Figure 7. EAX Register with 16-Bit and 8-Bit Name Components

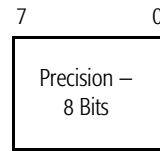
Table 3. General-Purpose Register Doubleword, Word, and Byte Names

| 32-Bit Name (Doubleword) | 16-Bit Name (Word) | 8-Bit Name (High-order Bits) | 8-Bit Name (Low-order Bits) |
|-----------------------------|-----------------------|---------------------------------|--------------------------------|
| EAX | AX | AH | AL |
| EBX | BX | BH | BL |
| ECX | CX | CH | CL |
| EDX | DX | DH | DL |
| EDI | DI | — | — |
| ESI | SI | — | — |
| ESP | SP | — | — |
| EBP | BP | — | — |

Integer Data Types

Four types of data are used in general-purpose registers—byte, word, doubleword, and quadword integers. Figure 8 shows the format of the integer data registers.

Byte Integer



Word Integer



Doubleword Integer



Quadword Integer



Figure 8. Integer Data Registers

Segment Registers

The six 16-bit segment registers are used as pointers to areas (segments) of memory. Table 4 lists the segment registers and their functions. Figure 9 shows the format for all six segment registers.

Table 4. Segment Registers

| Segment Register | Segment Register Function |
|------------------|--|
| CS | Code segment, where instructions are located |
| DS | Data segment, where data is located |
| ES | Data segment, where data is located |
| FS | Data segment, where data is located |
| GS | Data segment, where data is located |
| SS | Stack segment |

**Figure 9. Segment Register****Segment Usage**

The operating system determines the type of memory model that is implemented. The segment register usage is determined by the operating system's memory model. In a real mode memory model, the segment register points to the base address in memory.

In a protected mode memory model the segment register is called a selector and it selects a segment descriptor in a descriptor table. This descriptor contains a pointer to the base of the segment, the limit of the segment, and various protection attributes. For more information on descriptor formats, see "Descriptors and Gates" on page 59. Figure 10 on page 31 shows segment usage for Real mode and Protected mode memory models.

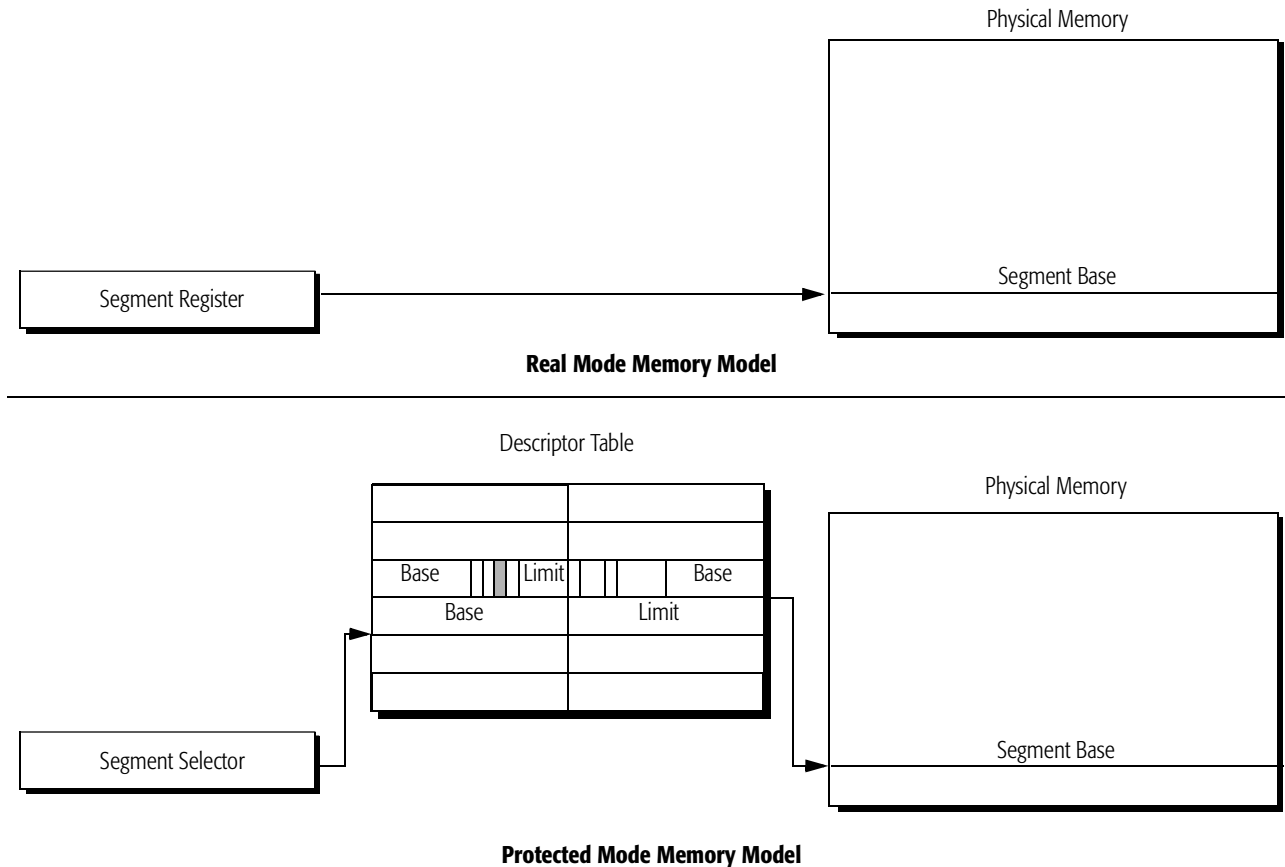


Figure 10. Segment Usage

Instruction Pointer

The instruction pointer (EIP or IP) is used in conjunction with the code segment register (CS). The instruction pointer is either a 32-bit register (EIP) or a 16-bit register (IP) that keeps track of where the next instruction resides within memory. This register cannot be directly manipulated, but can be altered by modifying return pointers when a JMP or CALL instruction is used.

Floating-Point Registers

The floating-point execution unit in the AMD-K6-III+ processor is designed to perform mathematical operations on non-integer numbers. This floating-point unit conforms to the IEEE 754 and 854 standards and uses several registers to meet these standards—eight numeric floating-point registers, a status word register, a control word register, and a tag word register.

The eight floating-point registers are physically 80 bits wide and labeled FPR0–FPR7. Figure 11 shows the format of the floating-point registers. See “Floating-Point Register Data Types” on page 34 for information on allowable floating-point data types.



Figure 11. Floating-Point Register

The 16-bit FPU status word register contains information about the state of the floating-point unit. Figure 12 shows the format of the FPU status word register.

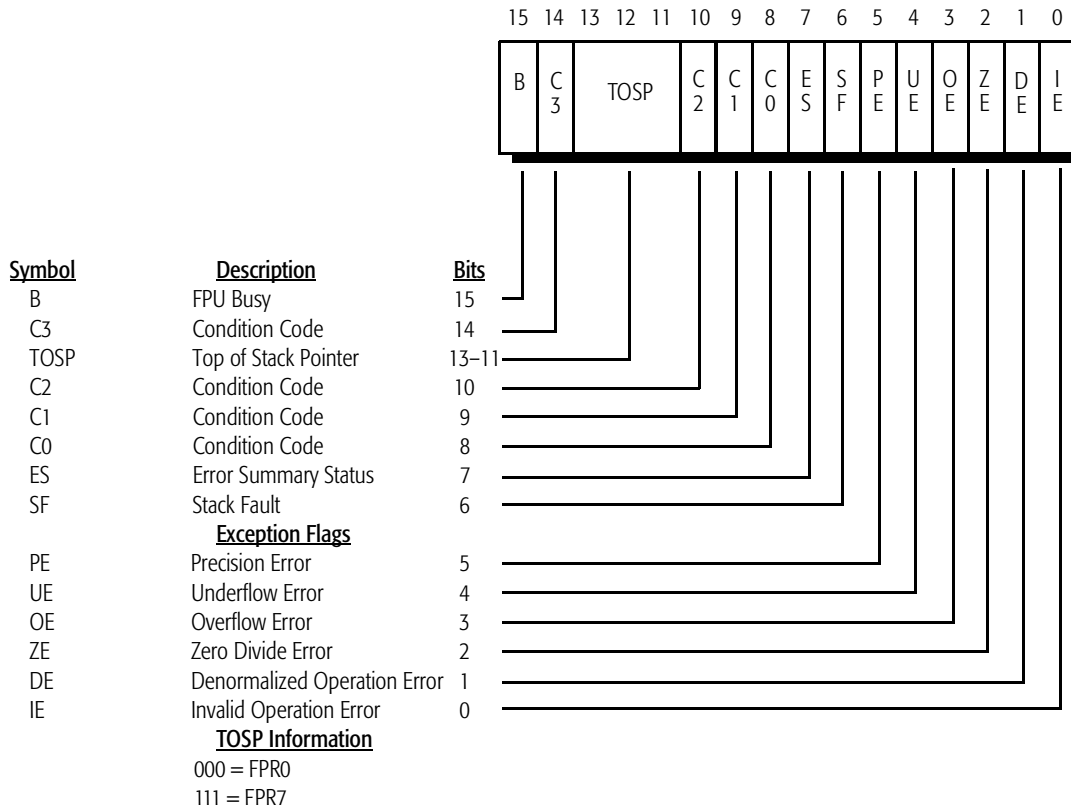


Figure 12. FPU Status Word Register

The FPU control word register allows a programmer to manage the FPU processing options. Figure 13 shows the format of the FPU control word register.

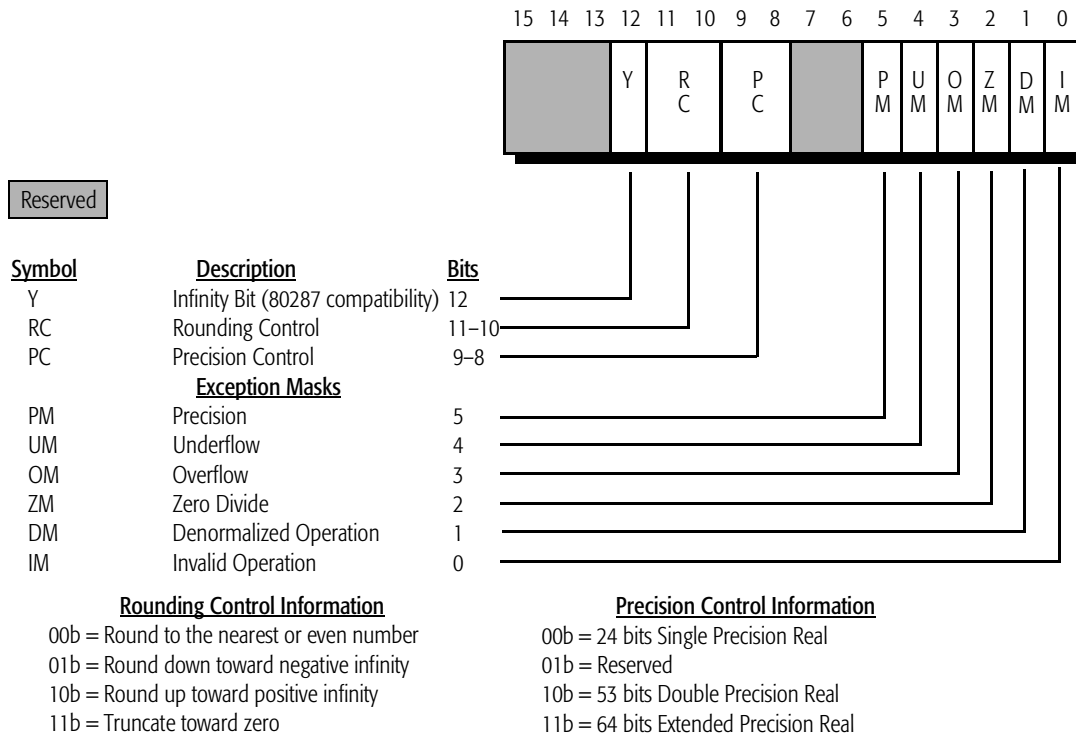


Figure 13. FPU Control Word Register

The FPU tag word register contains information about the registers in the register stack. Figure 14 shows the format of the FPU tag word register.

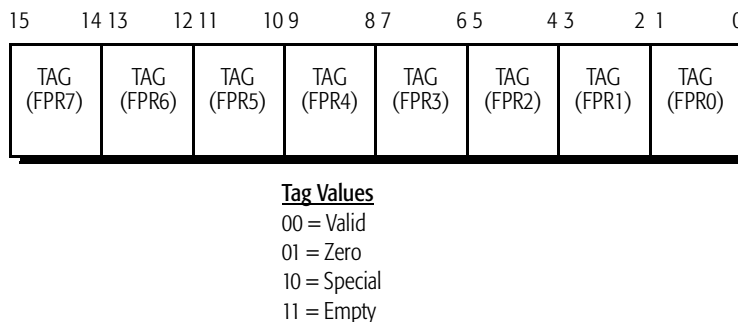


Figure 14. FPU Tag Word Register

Floating-Point Register Data Types

Floating-point registers use four different types of data—packed decimal, single-precision real, double-precision real, and extended-precision real. Figures 15 and 16 show the formats for these registers.

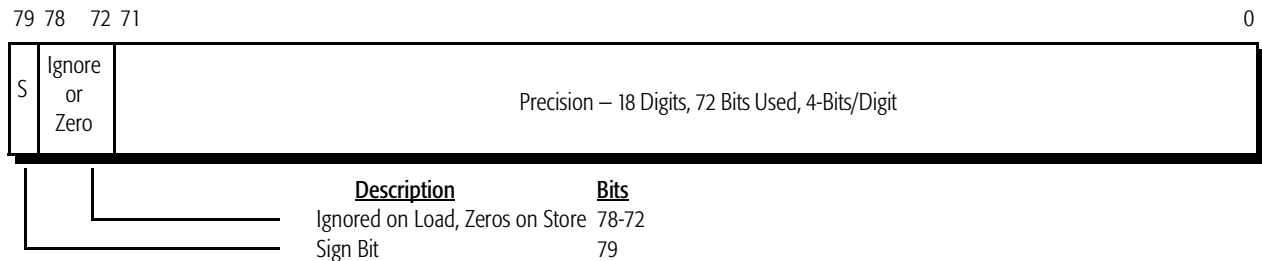


Figure 15. Packed Decimal Data Register

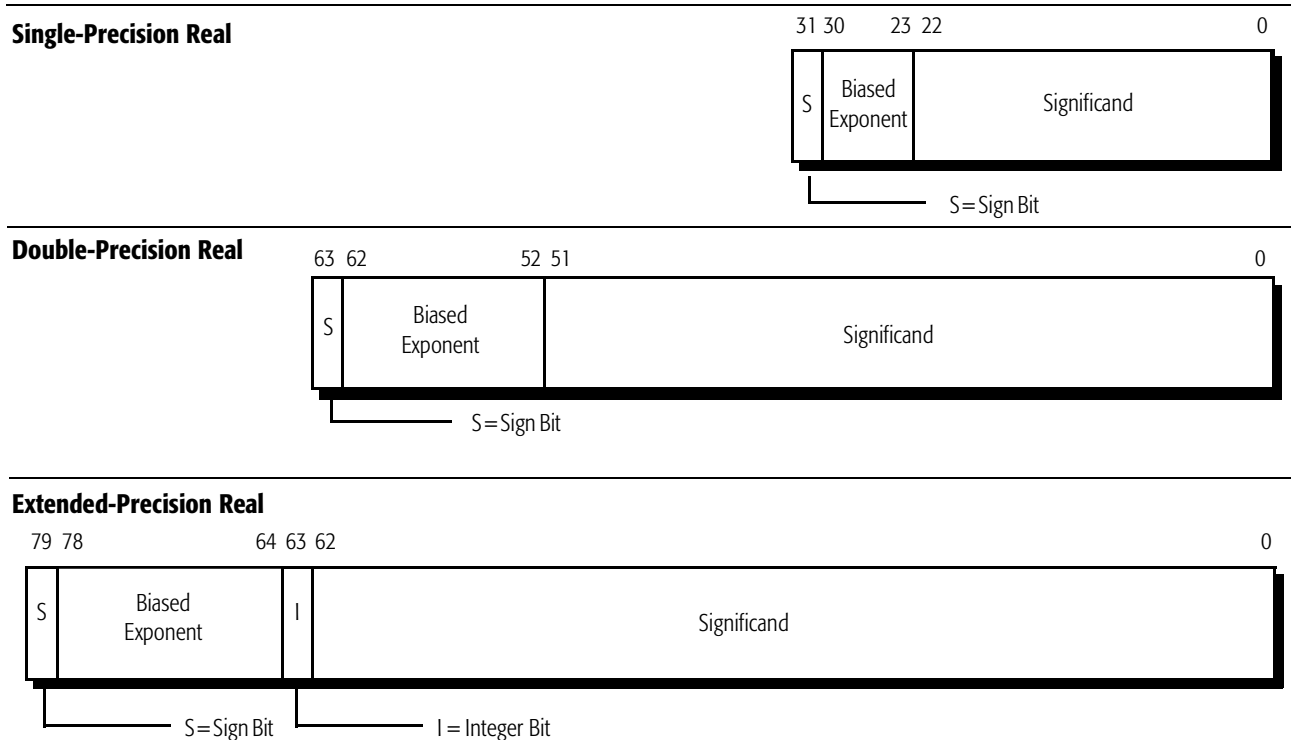


Figure 16. Precision Real Data Registers

**MMX™/3DNow!™
Registers**

The AMD-K6-III+ processor implements eight 64-bit MMX/3DNow! registers for use by multimedia software. These registers are mapped on the floating-point register stack. The MMX and 3DNow! instructions refer to these registers as mm0 to mm7. Figure 17 shows the format of these registers. For more information, see the *AMD-K6® Processor Multimedia Technology Manual*, order# 20726 and the *3DNow! Technology Manual*, order# 21928.

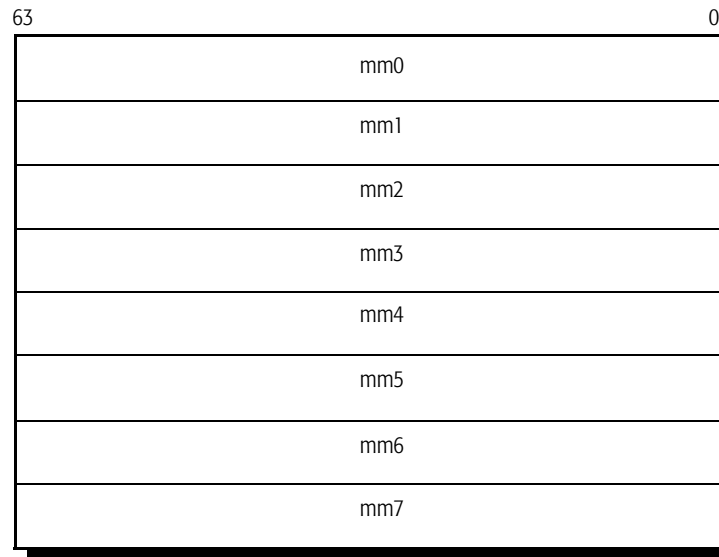
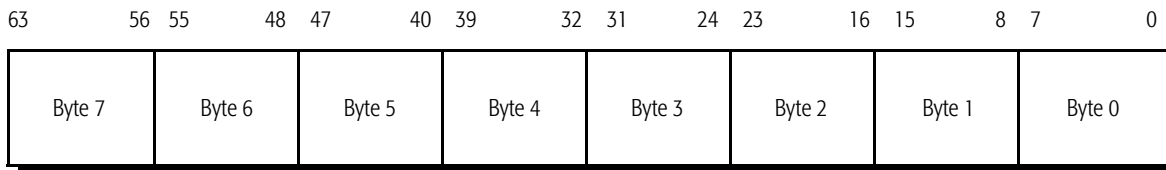


Figure 17. MMX™/3DNow!™ Registers

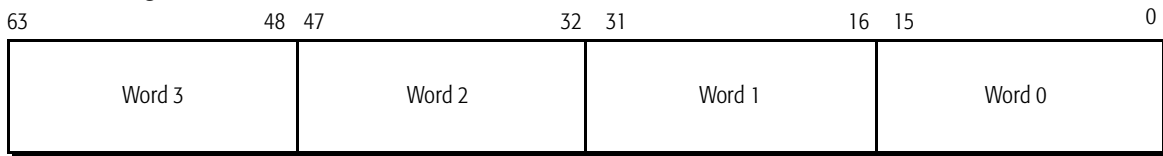
MMX™ Technology Data Types

For the MMX instructions, the MMX registers use three types of data—packed eight-byte integer, packed quadword integer, and packed dual doubleword integer. Figure 18 on page 36 shows the format of these data types.

Packed Bytes Integer



Packed Words Integer



Packed Doubleword Integer

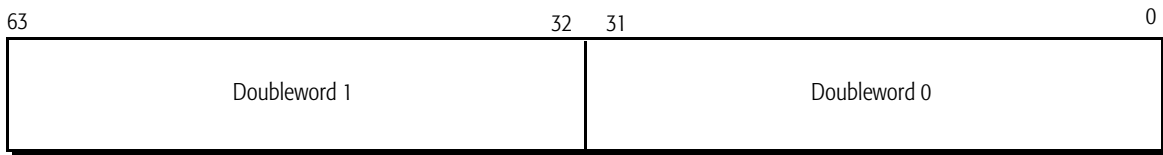


Figure 18. MMX™ Technology Data Types

3DNow!™ Technology Data Types

For 3DNow! instructions, the MMX/3DNow! registers use packed single-precision real data. Figure 19 shows the format of the 3DNow! data type.

Packed Single Precision Floating Point

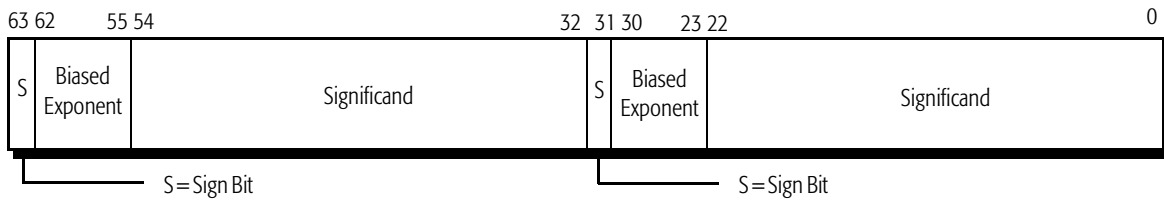


Figure 19. 3DNow!™ Technology Data Types

EFLAGS Register

The EFLAGS register provides for three different types of flags—system, control, and status. The system flags provide operating system controls, the control flag provides directional information for string operations, and the status flags provide information resulting from logical and arithmetic operations. Figure 20 shows the format of the EFLAGS register.

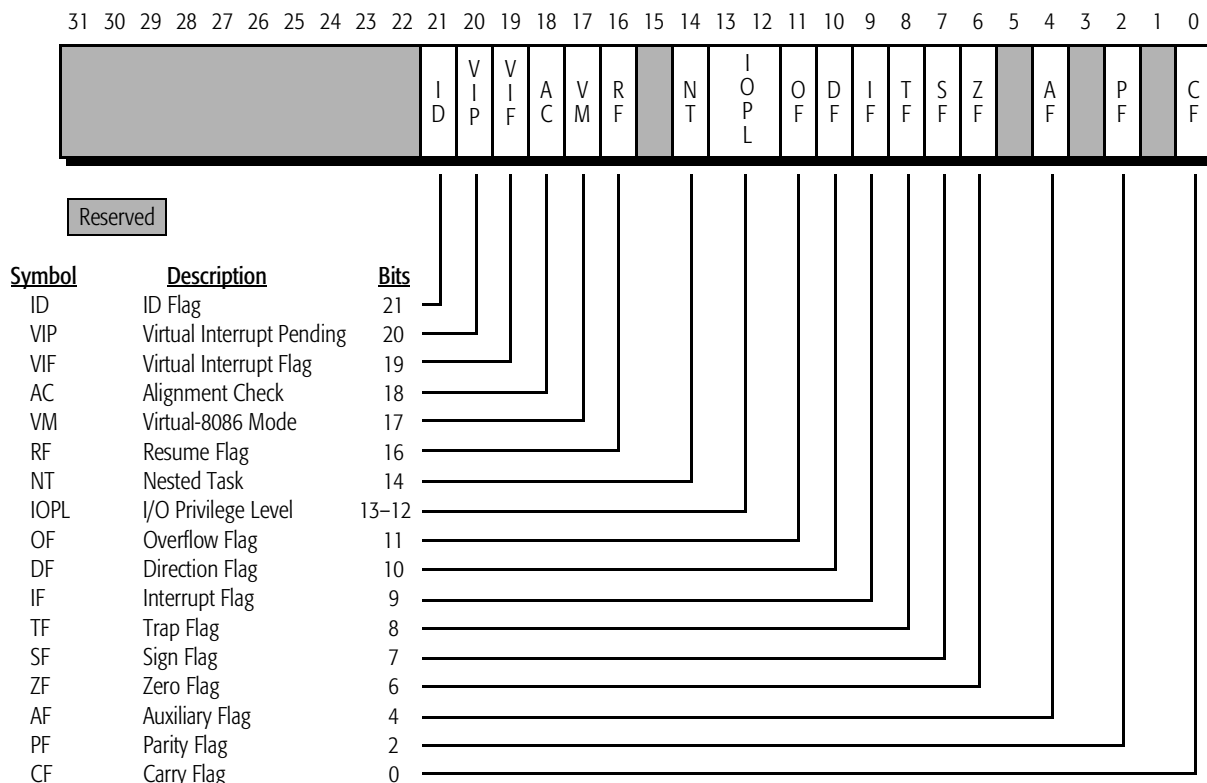


Figure 20. EFLAGS Registers

Control Registers

The five control registers contain system control bits and pointers. Figures 21 through 25 show the formats of the control registers.

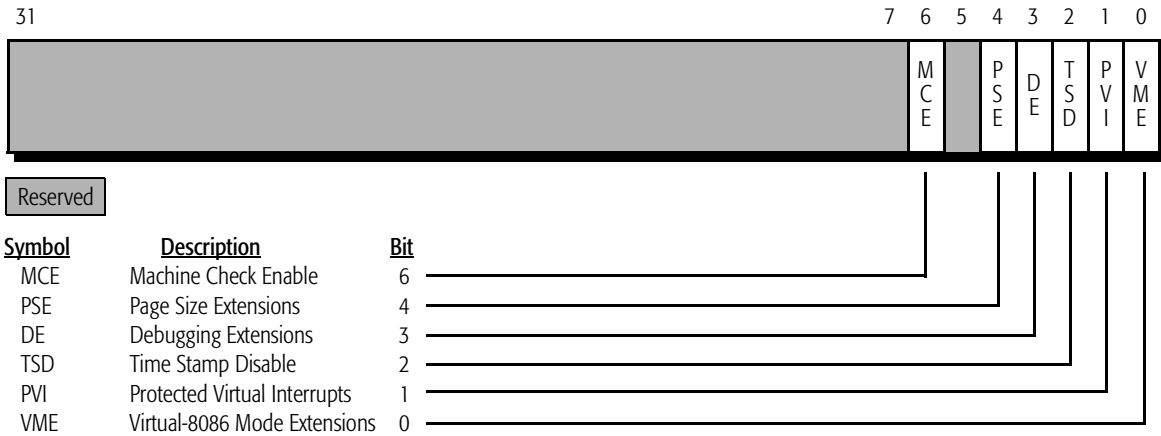


Figure 21. Control Register 4 (CR4)

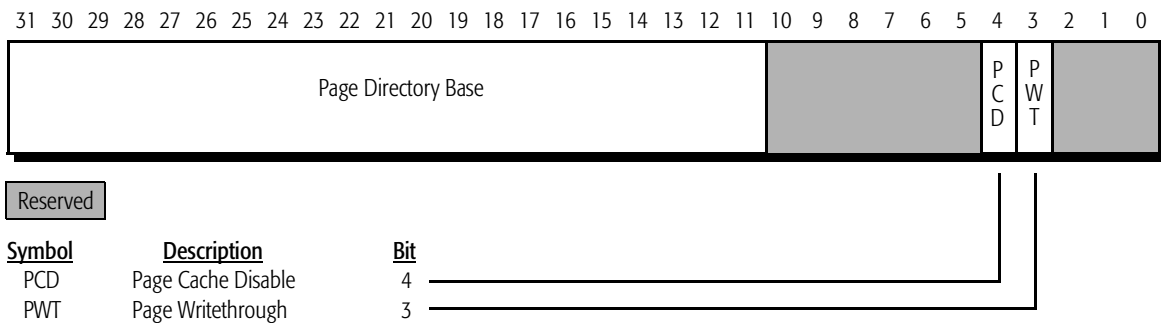


Figure 22. Control Register 3 (CR3)



Figure 23. Control Register 2 (CR2)



Figure 24. Control Register 1 (CR1)

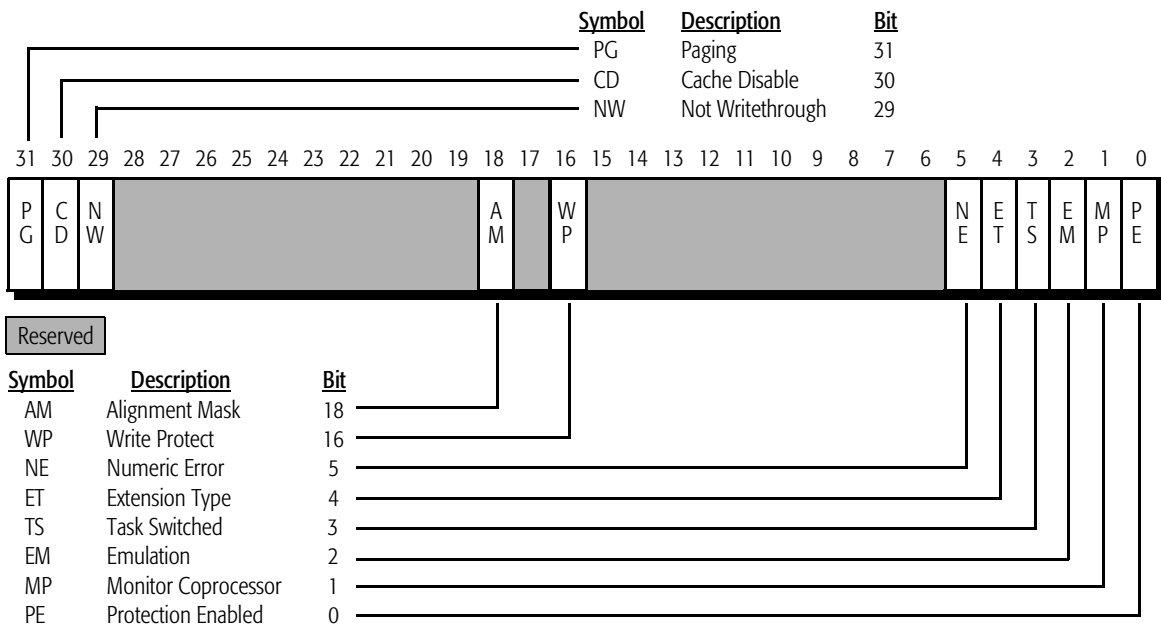


Figure 25. Control Register 0 (CR0)

Debug Registers

Figures 26 through 29 show the 32-bit debug registers supported by the processor. These registers are further described in “Debug” on page 268.

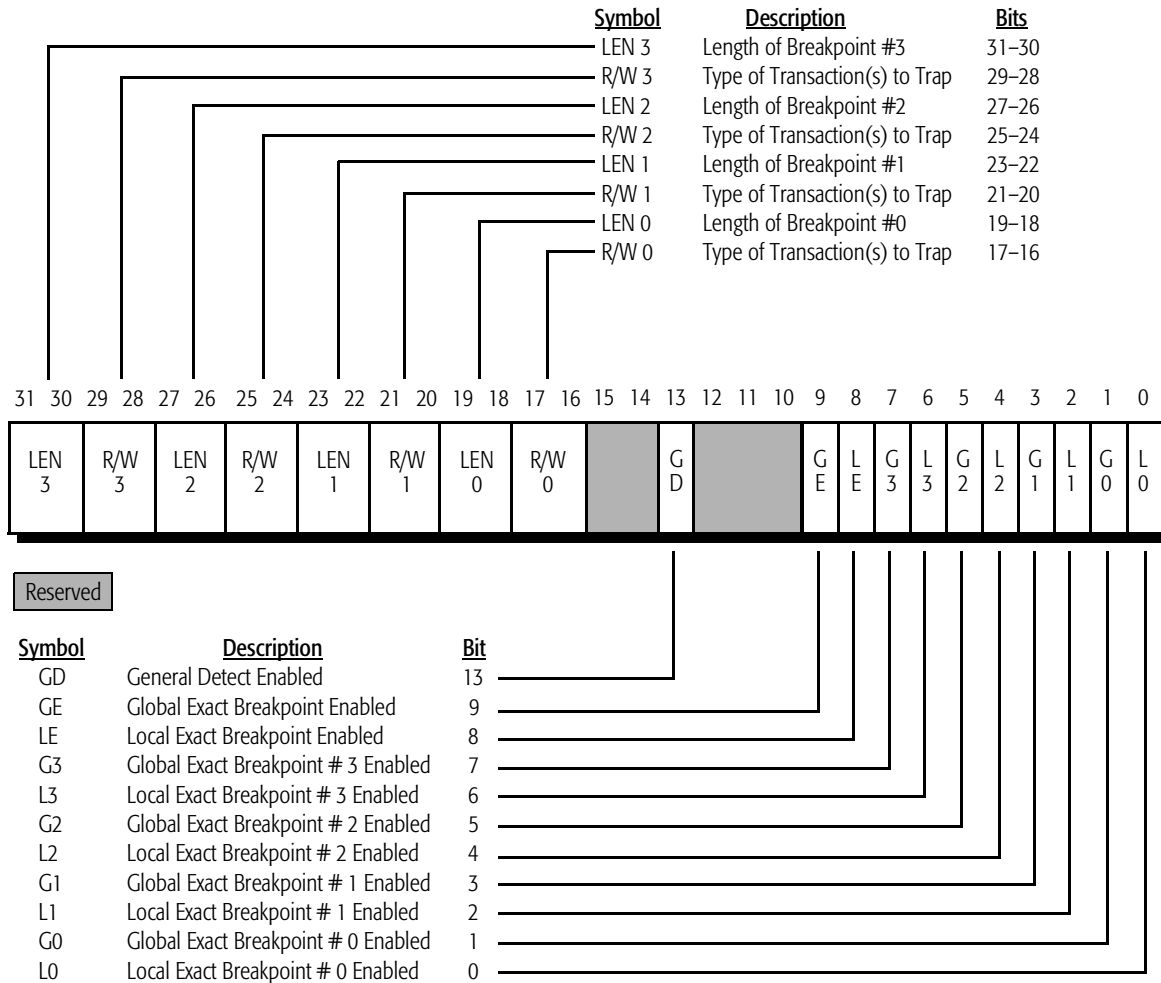


Figure 26. Debug Register DR7

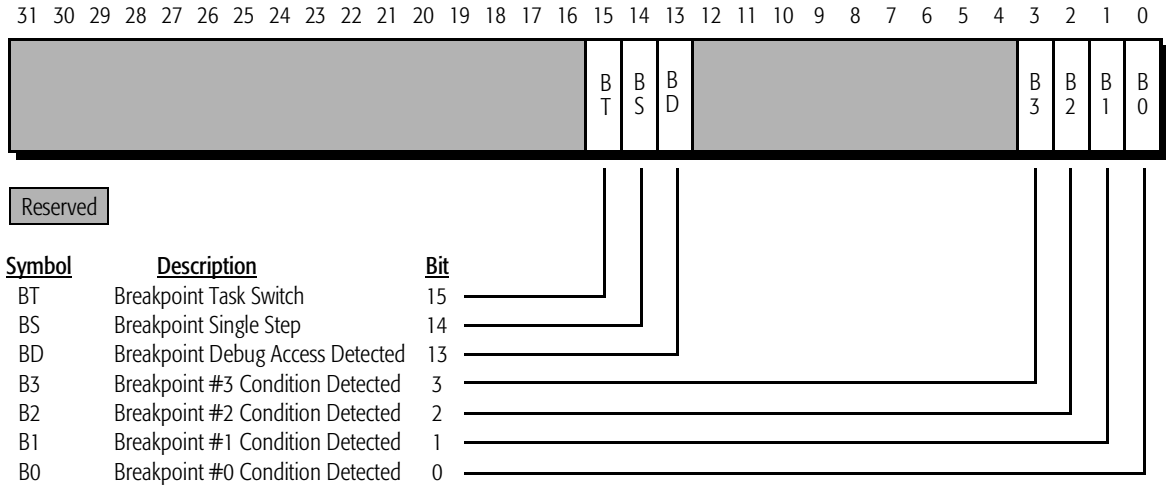
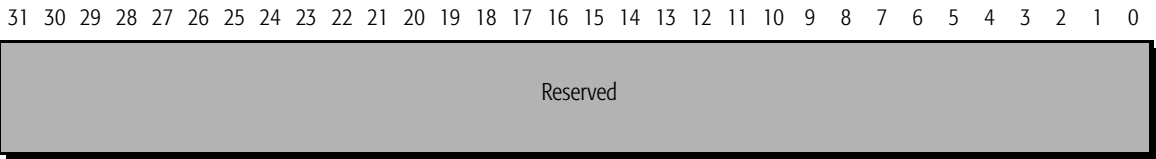


Figure 27. Debug Register DR6

DR5



DR4

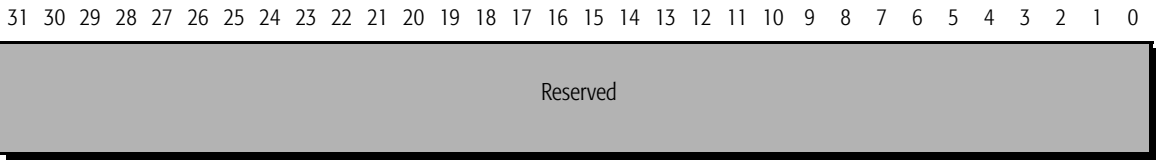
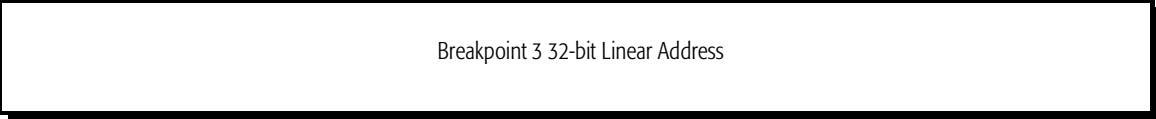


Figure 28. Debug Registers DR5 and DR4

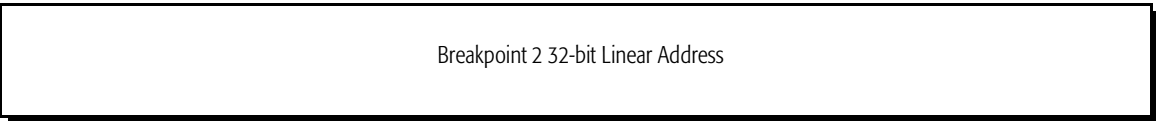
DR3

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



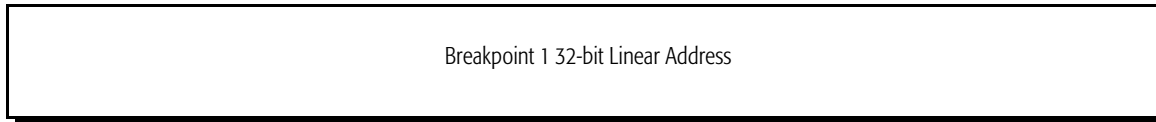
DR2

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



DR1

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



DR0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

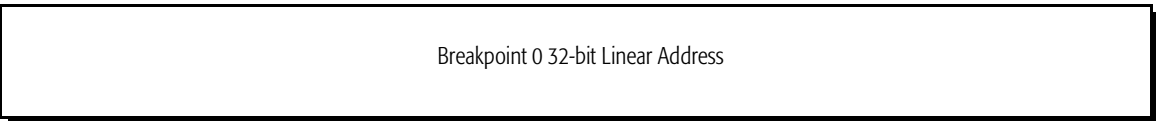


Figure 29. Debug Registers DR3, DR2, DR1, and DR0

3.2 Model-Specific Registers (MSR)

The AMD-K6-III+ processor provides eleven Model-Specific Registers (MSRs) in the standard-power versions and twelve MSRs in the low-power versions.

- The value in the ECX register selects the MSR to be addressed by the RDMSR and WRMSR instructions.
- The values in EAX and EDX are used as inputs and outputs by the RDMSR and WRMSR instructions.

Table 5 lists the MSRs and the corresponding value of the ECX register. Figures 30 through 43 starting on page 45 show the MSR formats.

Table 5. AMD-K6™-III+ Processor Model-Specific Registers

| Model-Specific Register | Value of ECX |
|--|--------------|
| Machine Check Address Register (MCAR) | 00h |
| Machine Check Type Register (MCTR) | 01h |
| Test Register 12 (TR12) | 0Eh |
| Time Stamp Counter (TSC) | 10h |
| Extended Feature Enable Register (EFER) | C000_0080h |
| SYSCALL/SYSRET Target Address Register (STAR) | C000_0081h |
| Write Handling Control Register (WHCR) | C000_0082h |
| UC/WC Cacheability Control Register (UWCCR) | C000_0085h |
| Processor State Observability Register (PSOR) | C000_0087h |
| Page Flush/Invalidate Register (PFIR) | C000_0088h |
| Level-2 Cache Array Register (L2AAR) | C000_0089h |
| Enhanced Power Management Register (EPMR) ¹ | C000_0086h |

Notes:

1. The EPMR register is supported in the low-power versions only of the AMD-K6-III+ processor.

For more information about the MSRs, see the *Embedded AMD-K6™ Processors BIOS Design Guide Application Note*, order# 23913.

For more information about the RDMSR and WRMSR instructions, see the *AMD K86™ Family BIOS and Software Tools Development Guide*, order# 21062.

**Machine Check
Address Register
(MCAR) and Machine
Check Type Register
(MCTR)**

The AMD-K6-III+ processor does not support the generation of a machine check exception. However, the processor does provide a 64-bit machine check address register (MCAR), a 64-bit machine check type register (MCTR), and a machine check enable (MCE) bit in CR4.

Because the processor does not support machine check exceptions, the contents of the MCAR and MCTR are only affected by the WRMSR instruction and by RESET being sampled asserted (where all bits in each register are reset to 0).

The formats for the machine-check address register and the machine-check type register are shown in Figure 30 and Figure 31, respectively. The MCAR register is MSR 00h, and the MCTR register is MSR 01h.

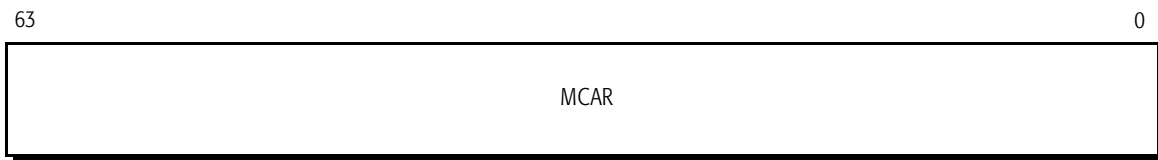


Figure 30. Machine-Check Address Register (MCAR)

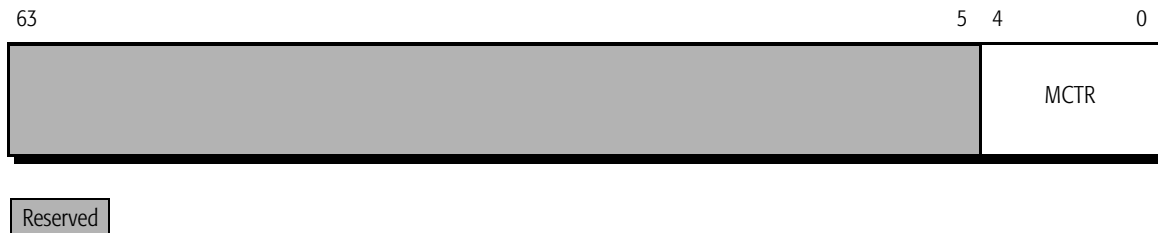


Figure 31. Machine-Check Type Register (MCTR)

Test Register 12 (TR12)

Test register 12 provides a method for disabling the L1 caches. Figure 32 shows the format of TR12. The TR12 register is MSR 0Eh.

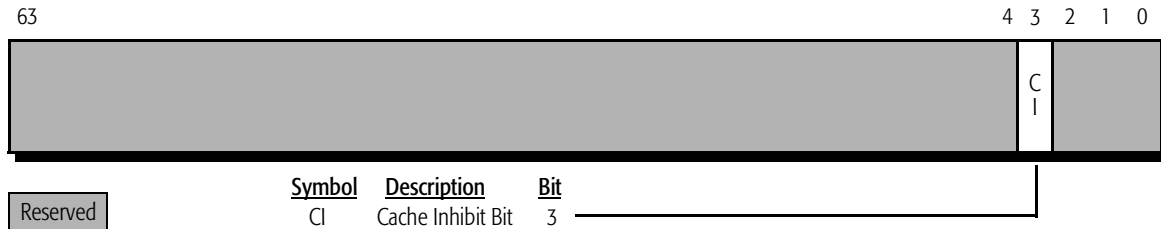


Figure 32. Test Register 12 (TR12)

Time Stamp Counter

With each processor clock cycle, the processor increments the 64-bit time stamp counter (TSC) MSR. Figure 33 shows the format of the TSC. The TSC register is MSR 10h.

The counter can be written or read using the WRMSR or RDMSR instructions when the ECX register contains the value 10h and CPL = 0. The counter can also be read using the RDTSC instruction, but the procedure must be executing at privilege level 0 for the RDTSC instruction to execute. This condition is reflected by the status of the Time Stamp Disable (TSD) bit in CR4.

With either of these instructions, the EDX and EAX registers hold the upper and lower dwords of the 64-bit value to be written to or read from the TSC, as follows:

- EDX—Upper 32 bits of TSC
- EAX—Lower 32 bits of TSC

The TSC can be loaded with any arbitrary value. This feature is compatible with the Pentium processor.

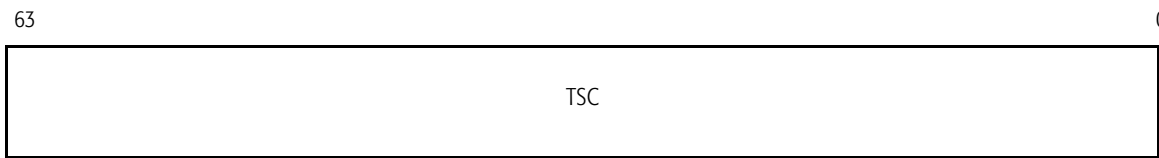


Figure 33. Time Stamp Counter (TSC)

Extended Feature Enable Register (EFER)

The Extended Feature Enable Register (EFER) contains the control bits that enable the extended features of the processor. Figure 34 shows the format of the EFER register, and Table 6 defines the function of each bit of the EFER register. The EFER register is MSR C000_0080h.

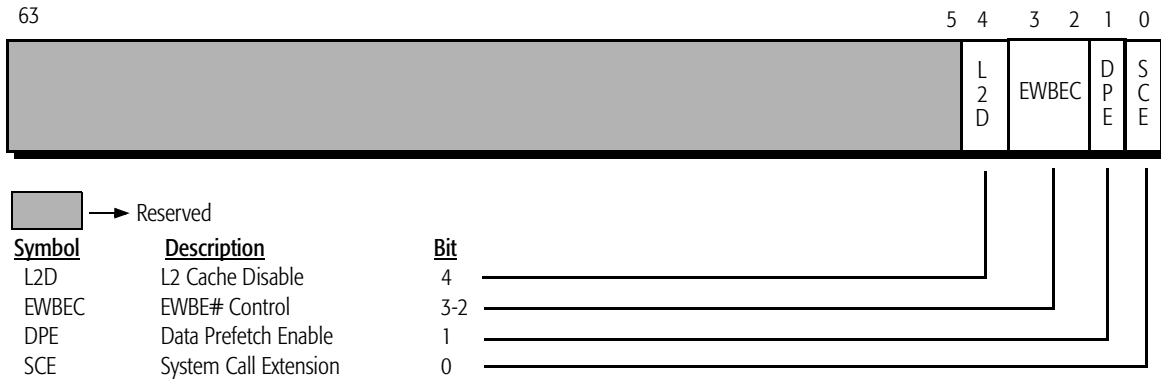


Figure 34. Extended Feature Enable Register (EFER)

Table 6. Extended Feature Enable Register (EFER) Definition

| Bit | Description | R/W | Function |
|------|-----------------------------|-----|--|
| 63–5 | Reserved | R | Writing a 1 to any reserved bit causes a general protection fault to occur. All reserved bits are always read as 0. |
| 4 | L2D | R/W | If L2D is set to 1, the L2 cache is completely disabled. This bit is provided for debug and testing purposes. For normal operation and maximum performance, this bit must be set to 0 (this is the default setting following reset). |
| 3–2 | EWBE Control (EWBEC) | R/W | This 2-bit field controls the behavior of the processor with respect to the ordering of write cycles and the EWBE# signal. EFER[3] and EFER[2] are Global EWBE Disable (GEWBED) and Speculative EWBE Disable (SEWBED), respectively. |
| 1 | Data Prefetch Enable (DPE) | R/W | DPE must be set to 1 to enable data prefetching (this is the default setting following reset). If enabled, cache misses initiated by a memory read within a 32-byte line are conditionally followed by cache-line fetches of the other line in the 64-byte sector. |
| 0 | System Call Extension (SCE) | R/W | SCE must be set to 1 to enable the usage of the SYSCALL and SYSRET instructions. |

For more information about the EWBE# Control, see “EWBE# Control” on page 229.

SYSCALL/SYSRET Target Address Register (STAR)

The SYSCALL/SYSRET target address register (STAR) contains the target EIP address used by the SYSCALL instruction and the 16-bit code and stack segment selector bases used by the SYSCALL and SYSRET instructions. Figure 35 shows the format of the STAR register, and Table 7 defines the function of each bit of the STAR register. For more information, see the *SYSCALL and SYSRET Instruction Specification Application Note*, order# 21086. The STAR register is MSR C000_0081h.

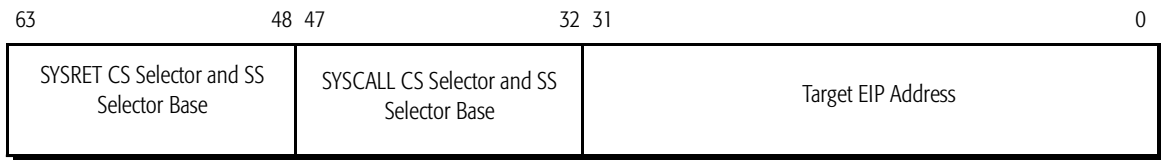


Figure 35. SYSCALL/SYSRET Target Address Register (STAR)

Table 7. SYSCALL/SYSRET Target Address Register (STAR) Definition

| Bit | Description | R/W |
|-------|---------------------------------|-----|
| 63–48 | SYSRET CS and SS Selector Base | R/W |
| 47–32 | SYSCALL CS and SS Selector Base | R/W |
| 31–0 | Target EIP Address | R/W |

Write Handling Control Register (WHCR)

The Write Handling Control Register (WHCR) is a MSR that contains two fields—the Write Allocate Enable Limit (WAE LIM) field, and the Write Allocate Enable 15-to-16-Mbyte (WAE15M) bit (see Figure 36). For more information, see “Write Allocate” on page 215. The WHCR register is MSR C000_0082h.

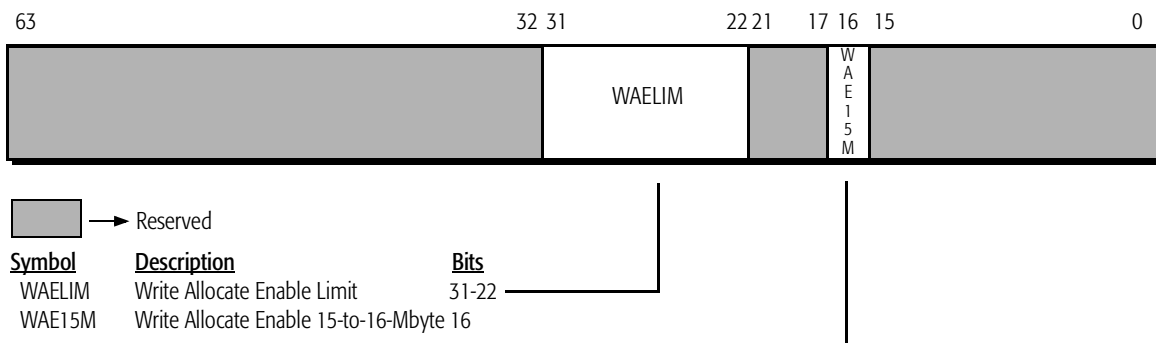


Figure 36. Write Handling Control Register (WHCR)

UC/WC Cacheability Control Register (UWCCR)

The AMD-K6-III+ processor provides two variable-range Memory Type Range Registers (MTRRs)—MTRR0 and MTRR1—that each specify a range of memory. Each range can be defined as uncacheable (UC) or write-combining (WC) memory. For more information, see “Memory Type Range Registers” on page 231. The UWCCR register is MSR C000_0085h.

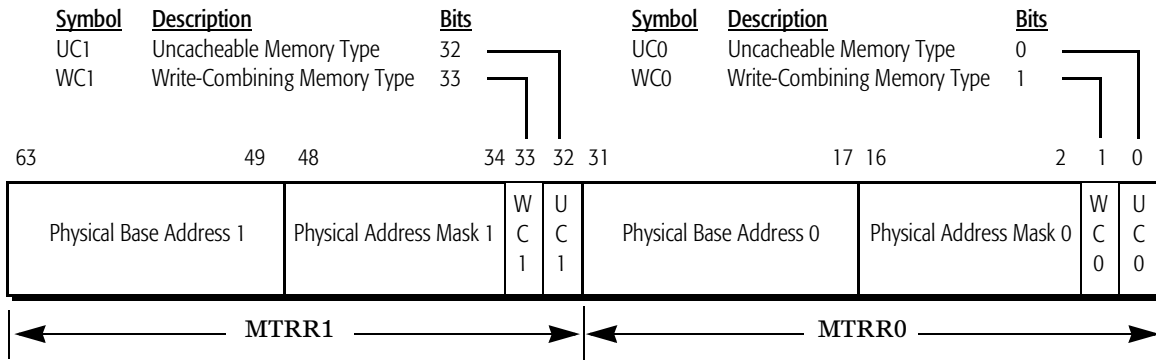


Figure 37. UC/WC Cacheability Control Register (UWCCR)

Processor State Observability Register (PSOR)

The AMD-K6-III+ processor provides the Processor State Observability Register (PSOR). The PSOR is defined as shown in Figure 38 for all standard-power versions of the AMD-K6-III+ processor. For a description of the PSOR register supported by the low-power versions of the processor, see page 148.

The PSOR register is MSR C000_0087h.

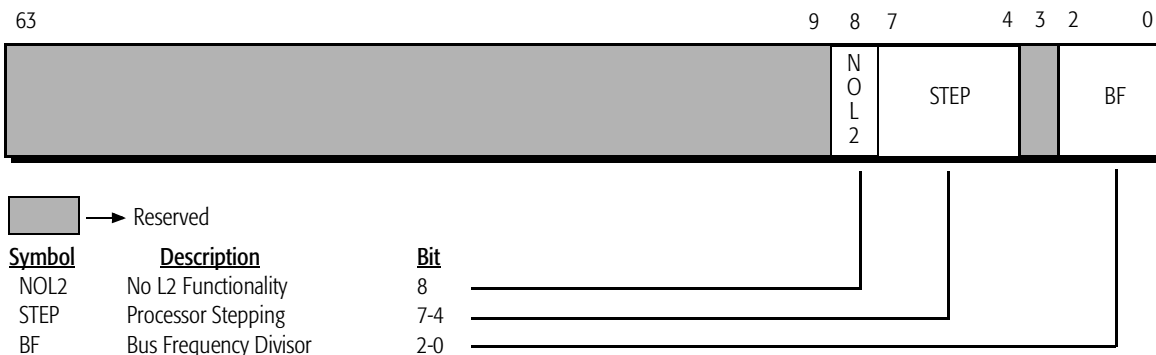


Figure 38. Processor State Observability Register (PSOR)

Page Flush/Invalidate Register (PFIR)

The AMD-K6-III+ processor contains the Page Flush/Invalidate Register (PFIR) (see Figure 39) that allows cache invalidation and optional flushing of a specific 4-Kbyte page from the linear address space. For more detailed information on PFIR, see “Page Flush/Invalidate Register (PFIR)” on page 223. The PFIR register is MSR C000_0088h.

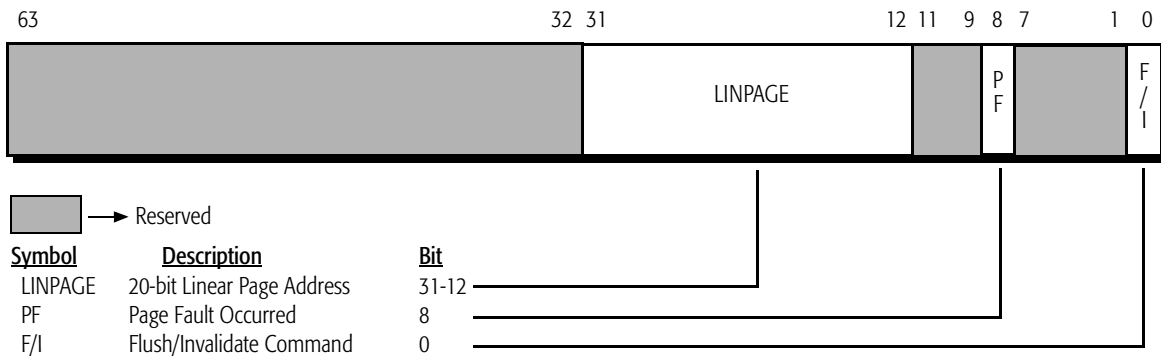


Figure 39. Page Flush/Invalidate Register (PFIR)

Level-2 Cache Array Access Register (L2AAR)

The AMD-K6-III+ processor provides the L2AAR register that allows for direct access to the L2 cache and L2 tag arrays. The L2AAR register is MSR C000_0089h.

The operation that is performed on the L2 cache is a function of the instruction executed—RDMSR or WRMSR—and the contents of the EDX register. The EDX register specifies the location of the access, and whether the access is to the L2 cache data or tags (refer to Figure 40).

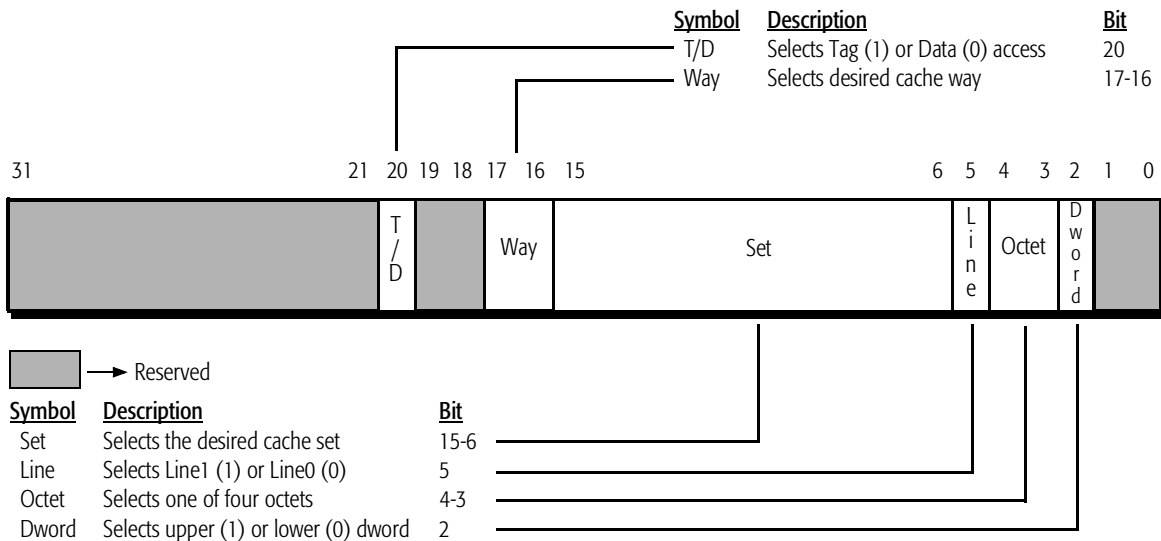


Figure 40. L2 Tag or Data Location for AMD-K6™-III+ Processor-EDX

If the L2 cache data is read (as opposed to reading the tag information), the result (doubleword) is placed in EAX in the format as illustrated in Figure 41. Similarly, if the L2 cache data is written, the write data is taken from EAX.



Figure 41. L2 Data -EAX

If the L2 tag is read (as opposed to reading the cache data), the result is placed in EAX in the format as illustrated in Figure 42 on page 52. Similarly, if the L2 tag is written, the write data is taken from EAX.

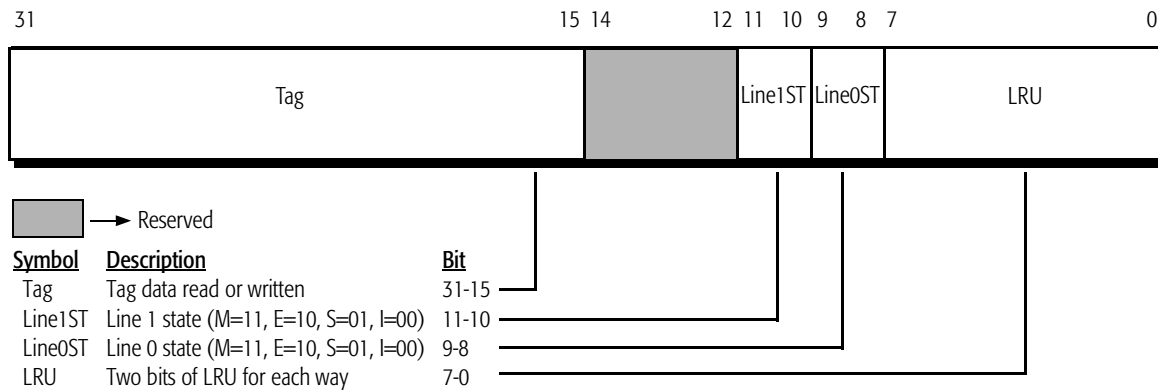


Figure 42. L2 Tag Information for AMD-K6™-III+ Processor-EAX

For more detailed information, refer to “L2 Cache and Tag Array Testing” on page 264.

Enhanced Power Management Register (EPMR)

The AMD-K6-III+ processor is designed with enhanced power management features, called AMD PowerNow! technology, which include dynamic bus divisor control and dynamic core voltage control. The EPMR register (see Figure 43) defines the base address for a 16-byte block of I/O address space. Enabling the EPMR allows software to access the EPM 16-byte I/O block, which contains bits for enabling, controlling, and monitoring the AMD PowerNow! technology features. The EPMR is MSR C000_0086h.

See “AMD PowerNow!™ Technology” on page 143 for more information about the definition and use of this register. Additional information can be found in the *Embedded AMD-K6™ Processors BIOS Design Guide Application Note*, order# 23913.

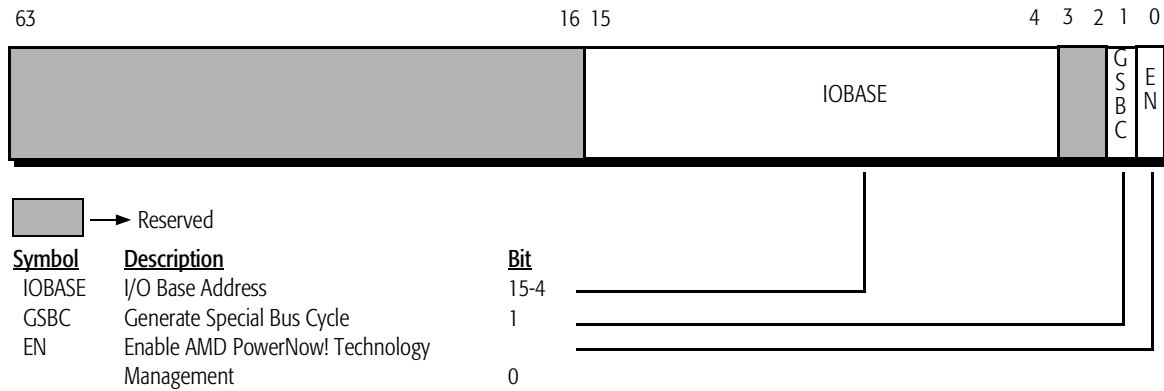


Figure 43. Enhanced Power Management Register (EPMR)

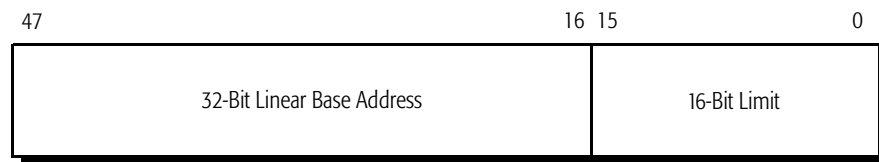
3.3 Memory Management Registers

The AMD-K6-III+ processor controls segmented memory management with the registers listed in Table 8. Figure 44 shows the formats of these registers.

Table 8. Memory Management Registers

| Register Name | Function |
|-------------------------------------|--|
| Global Descriptor Table Register | Contains a pointer to the base of the global descriptor table |
| Interrupt Descriptor Table Register | Contains a pointer to the base of the interrupt descriptor table |
| Local Descriptor Table Register | Contains a pointer to the local descriptor table of the current task |
| Task Register | Contains a pointer to the task state segment of the current task |

Global and Interrupt Descriptor Table Registers



Local Descriptor Table Register and Task Register

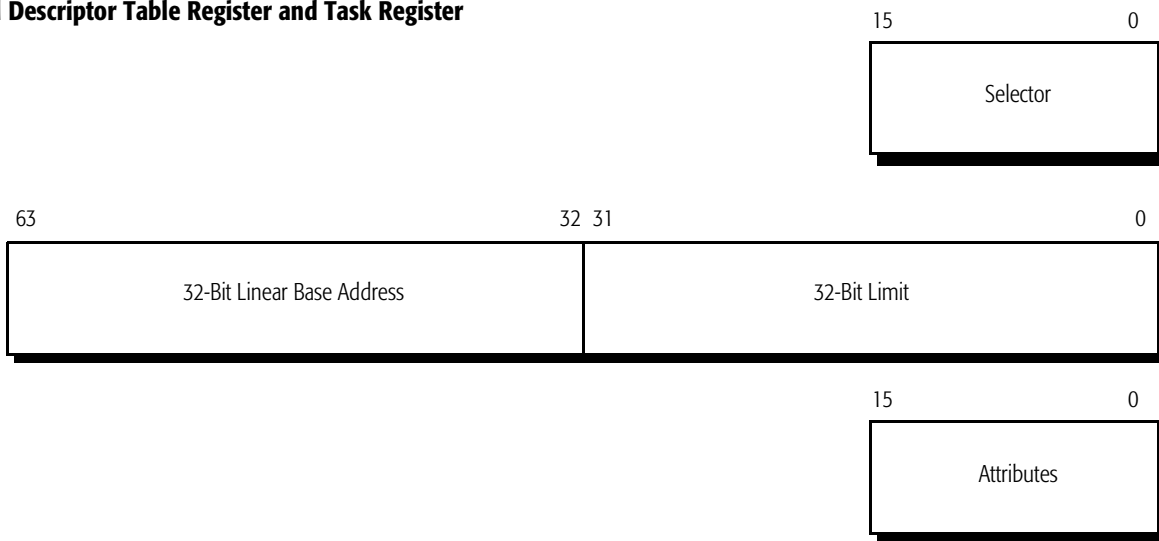


Figure 44. Memory Management Registers

Task State Segment Figure 45 shows the format of the task state segment (TSS).

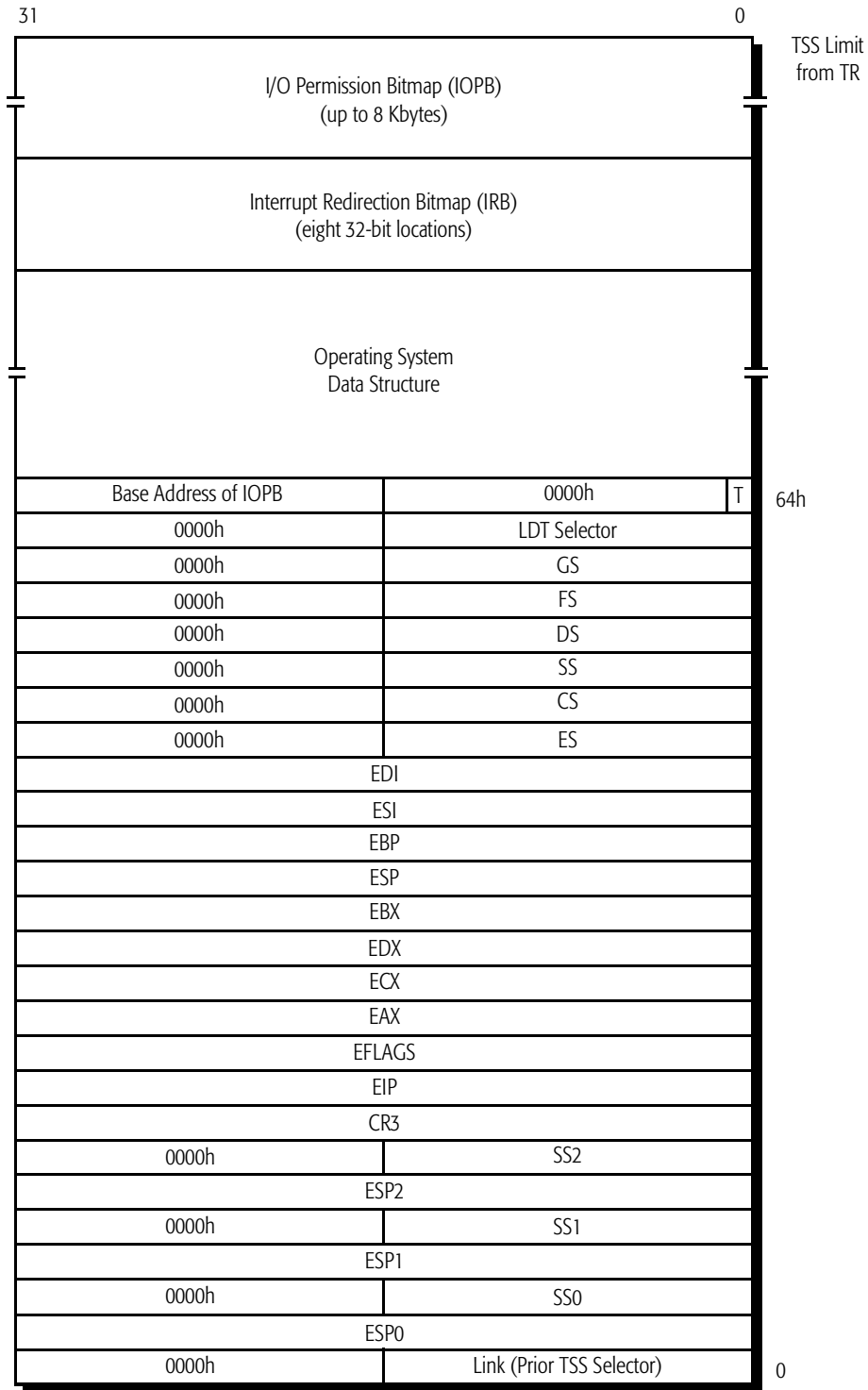


Figure 45. Task State Segment (TSS)

3.4 Paging

The AMD-K6-III+ processor can physically address up to four Gbytes of memory. This memory can be segmented into pages. The size of these pages is determined by the operating system design and the values set up in the page directory entries (PDE) and page table entries (PTE).

The processor can access both 4-Kbyte pages and 4-Mbyte pages, and the page sizes can be intermixed within a page directory. When the page size extension (PSE) bit in CR4 is set, the processor translates linear addresses using either the 4-Kbyte translation lookaside buffer (TLB) or the 4-Mbyte TLB, depending on the state of the page size (PS) bit in the page directory entry. Figures 46 and Figure 47 on page 57 show how 4-Kbyte and 4-Mbyte page translations work.

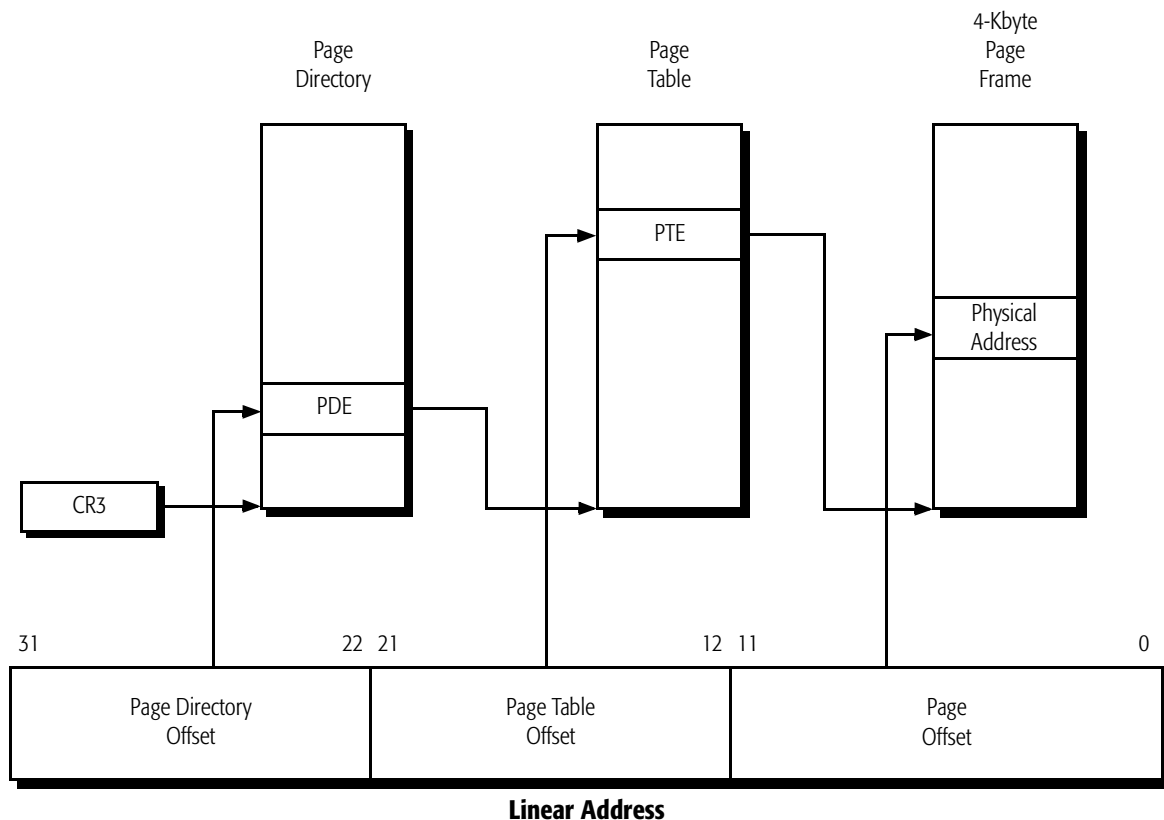


Figure 46. 4-Kbyte Paging Mechanism

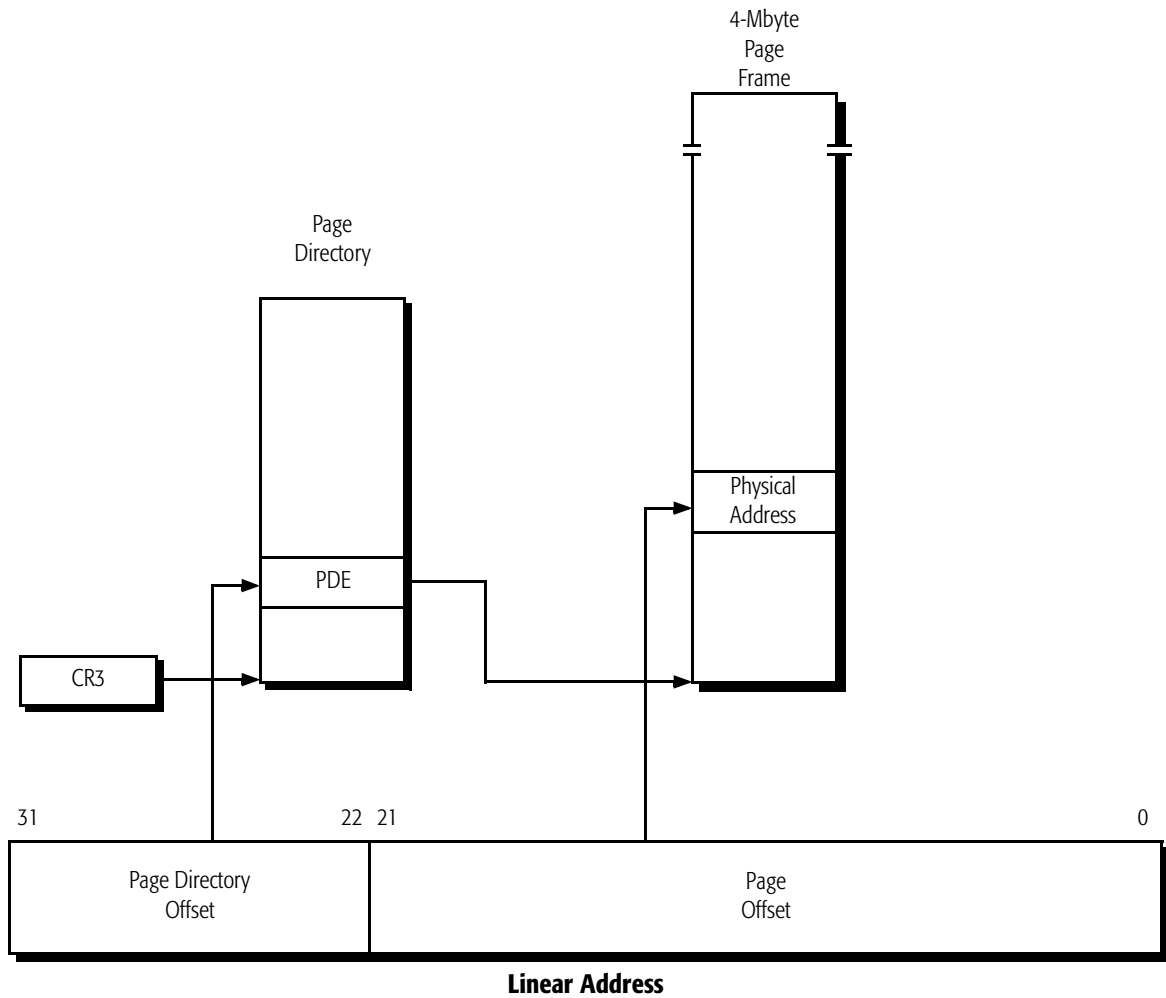


Figure 47. 4-Mbyte Paging Mechanism

Figures 48 through 50 starting on page 58 show the formats of the PDE and PTE. These entries contain information regarding the location of pages and their status.

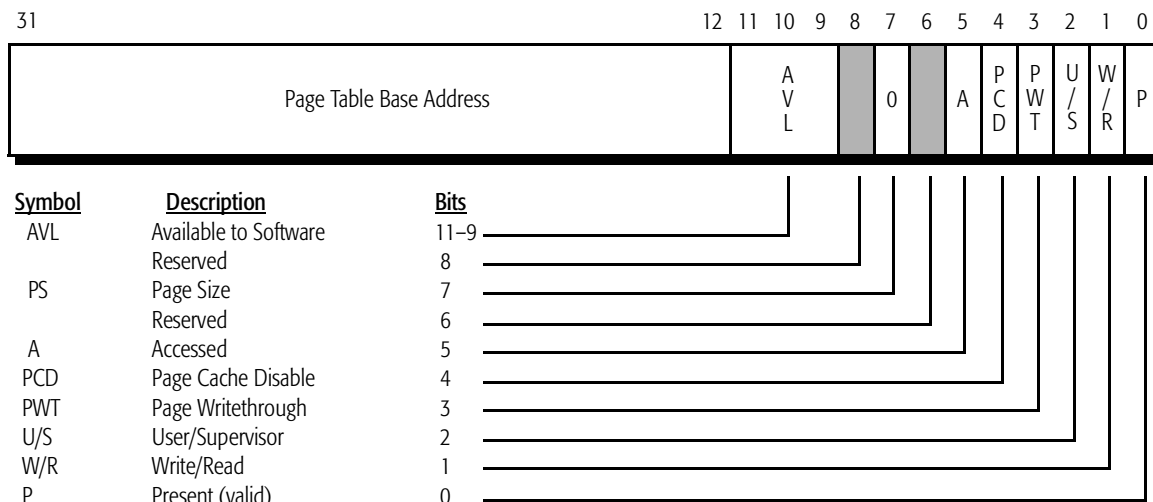


Figure 48. Page Directory Entry 4-Kbyte Page Table (PDE)

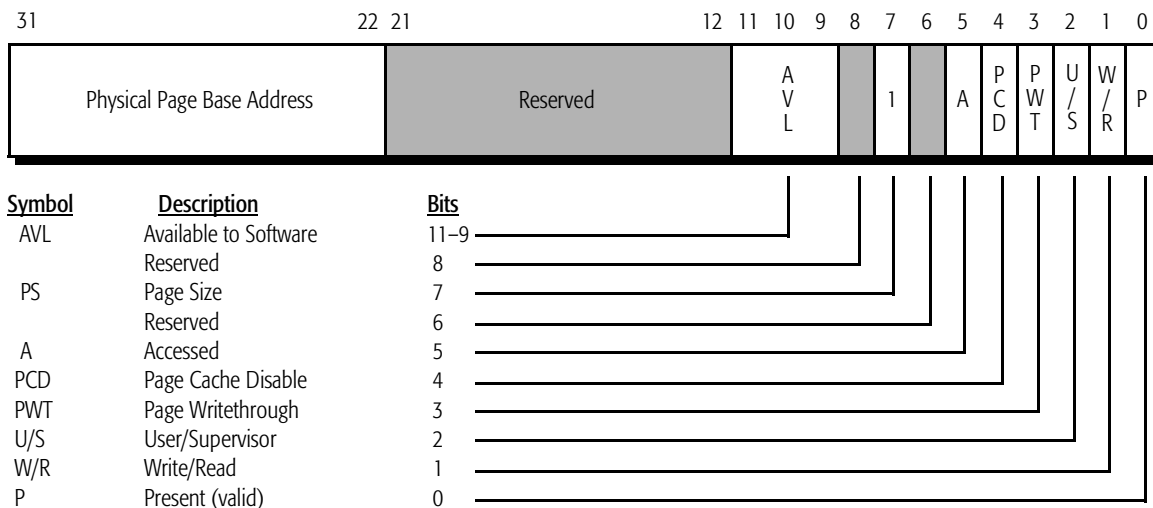


Figure 49. Page Directory Entry 4-Mbyte Page Table (PDE)

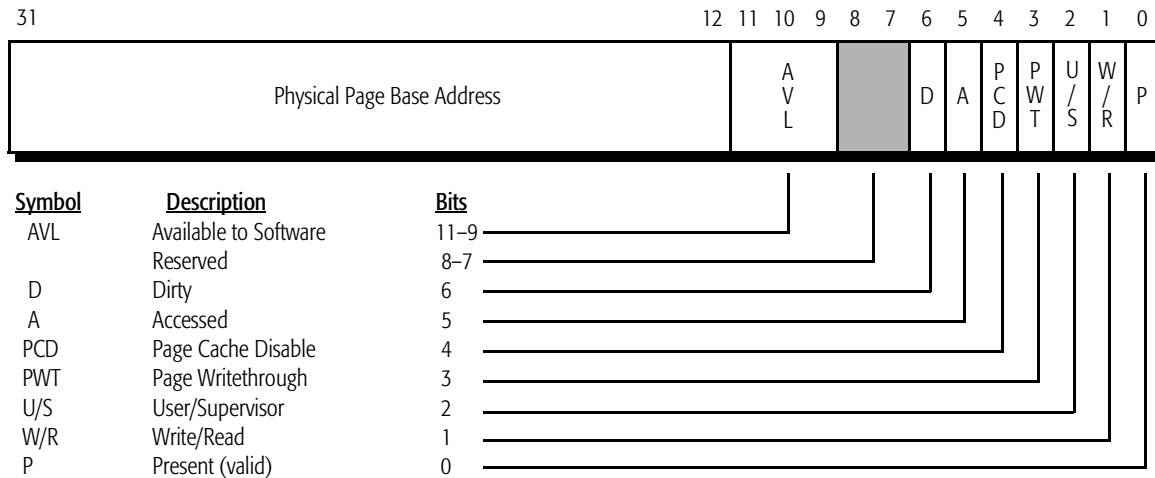


Figure 50. Page Table Entry (PTE)

3.5 Descriptors and Gates

There are various types of structures and registers in the x86 architecture that define, protect, and isolate code segments, data segments, task state segments, and gates. These structures are called descriptors.

- The *application segment descriptor* is used to point to either a data or code segment. Figure 51 on page 60 shows the application segment descriptor format. Table 9 on page 60 contains information describing the memory segment type to which the descriptor points.
- The *system segment descriptor* is used to point to a task state segment, a call gate, or a local descriptor table. Figure 52 on page 61 shows the system segment descriptor format. Table 10 on page 61 contains information describing the type of segment or gate to which the descriptor points.
- The AMD-K6-III+ processor uses *gates* to transfer control between executable segments with different privilege levels. Figure 53 on page 62 shows the format of the gate descriptor types. Table 10 on page 61 contains information describing the type of segment or gate to which the descriptor points.

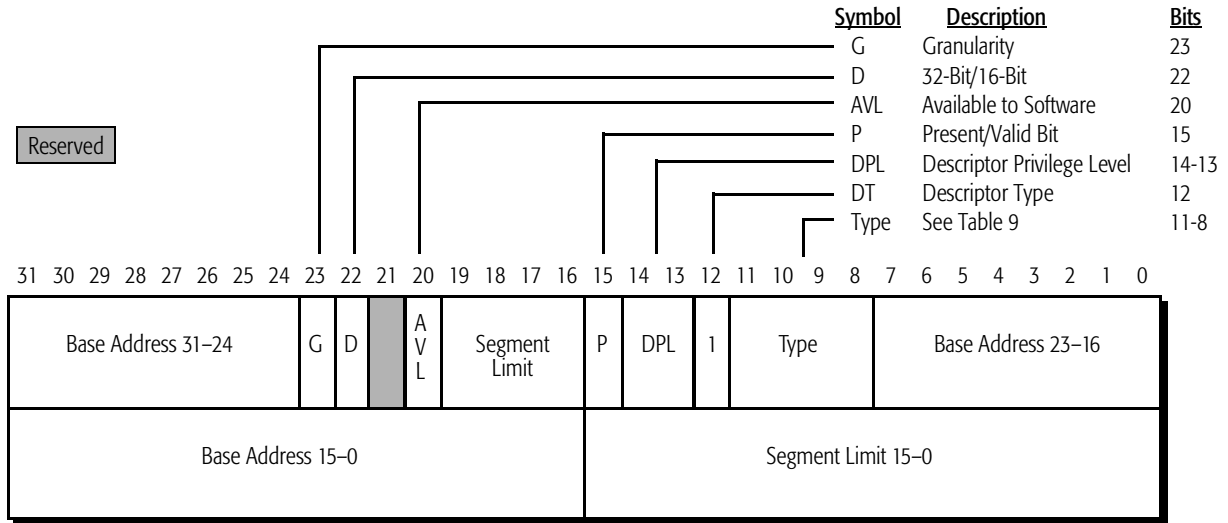


Figure 51. Application Segment Descriptor

Table 9. Application Segment Types

| Type | Data/Code | Description |
|------|-----------|--|
| 0 | Data | Read-Only |
| 1 | | Read-Only—Accessed |
| 2 | | Read/Write |
| 3 | | Read/Write—Accessed |
| 4 | | Read-Only—Expand-down |
| 5 | | Read-Only—Expand-down, Accessed |
| 6 | | Read/Write—Expand-down |
| 7 | | Read/Write—Expand-down, Accessed |
| 8 | Code | Execute-Only |
| 9 | | Execute-Only—Accessed |
| A | | Execute/Read |
| B | | Execute/Read—Accessed |
| C | | Execute-Only—Conforming |
| D | | Execute-Only—Conforming, Accessed |
| E | | Execute/Read-Only—Conforming |
| F | | Execute/Read-Only—Conforming, Accessed |

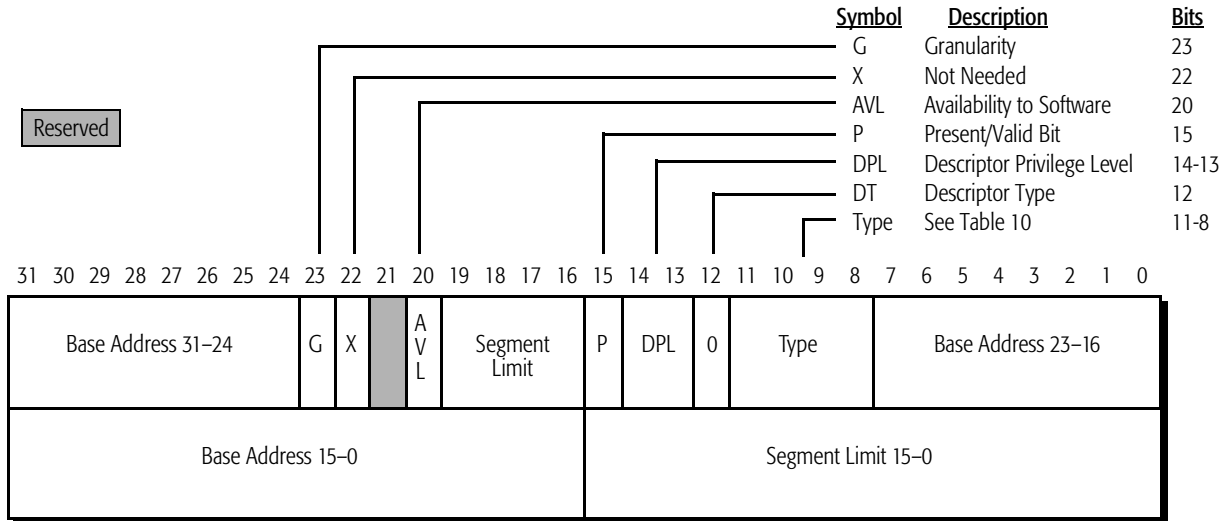


Figure 52. System Segment Descriptor

Table 10. System Segment and Gate Types

| Type | Description |
|------|-----------------------|
| 0 | Reserved |
| 1 | Available 16-bit TSS |
| 2 | LDT |
| 3 | Busy 16-bit TSS |
| 4 | 16-bit Call Gate |
| 5 | Task Gate |
| 6 | 16-bit Interrupt Gate |
| 7 | 16-bit Trap Gate |
| 8 | Reserved |
| 9 | Available 32-bit TSS |
| A | Reserved |
| B | Busy 32-bit TSS |
| C | 32-bit Call Gate |
| D | Reserved |
| E | 32-bit Interrupt Gate |
| F | 32-bit Trap Gate |

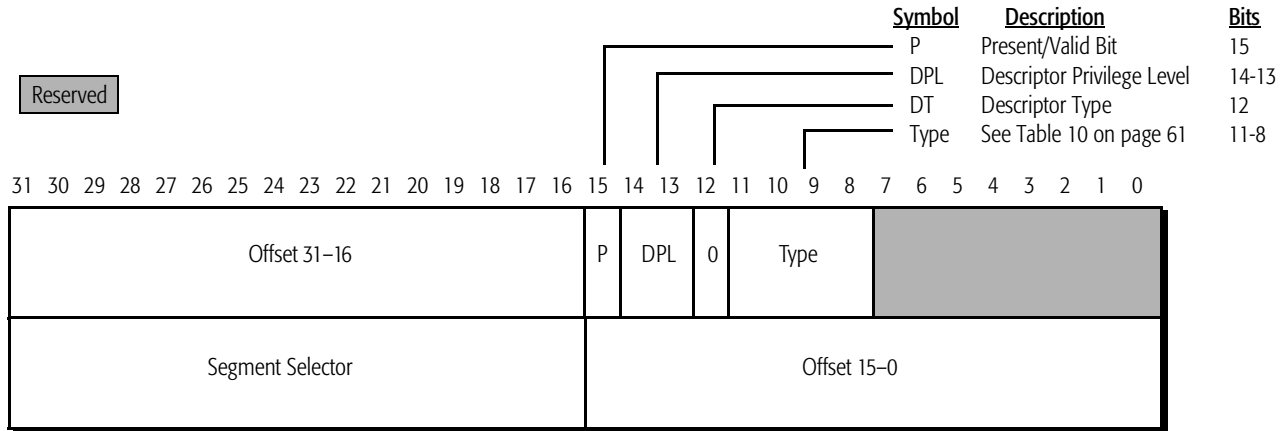


Figure 53. Gate Descriptor

3.6 Exceptions and Interrupts

Table 11 summarizes the exceptions and interrupts.

Table 11. Summary of Exceptions and Interrupts

| Interrupt Number | Interrupt Type | Cause |
|------------------|-------------------------|---|
| 0 | Divide by Zero Error | DIV, IDIV |
| 1 | Debug | Debug trap or fault |
| 2 | Non-Maskable Interrupt | NMI signal sampled asserted |
| 3 | Breakpoint | Int 3 |
| 4 | Overflow | INTO |
| 5 | Bounds Check | BOUND |
| 6 | Invalid Opcode | Invalid instruction |
| 7 | Device Not Available | ESC and WAIT |
| 8 | Double Fault | Fault occurs while handling a fault |
| 9 | Reserved - Interrupt 13 | — |
| 10 | Invalid TSS | Task switch to an invalid segment |
| 11 | Segment Not Present | Instruction loads a segment and present bit is 0 (invalid segment) |
| 12 | Stack Segment | Stack operation causes limit violation or present bit is 0 |
| 13 | General Protection | Segment related or miscellaneous invalid actions |
| 14 | Page Fault | Page protection violation or a reference to missing page |
| 16 | Floating-Point Error | Arithmetic error generated by floating-point instruction |
| 17 | Alignment Check | Data reference to an unaligned operand. (The AC flag and the AM bit of CR0 are set to 1.) |
| 0-255 | Software Interrupt | INT n |

3.7 Instructions Supported by the AMD-K6™-III+ Processor

This section documents all of the x86 instructions supported by the AMD-K6-III+ processor. Tables 12 through 16 starting on page 65 define the integer, floating-point, MMX, 3DNow! technology instructions, and 3DNow! technology digital signal processing (DSP) extensions for the AMD-K6-III+ processor, respectively. For details about the MMX instructions, 3DNow! technology instructions, and 3DNow! technology DSP extensions refer to the following manuals:

- MMX Instructions—AMD-K6® Processor Multimedia Technology Manual, order# 20726
- 3DNow! Technology Instructions—3DNow! Technology Manual, order# 21928
- 3DNow! Technology DSP Extensions—AMD Extensions to the 3DNow! and MMX Instruction Set Manual, order# 22466

Each table shows the instruction mnemonic, opcode, modR/M byte, decode type, and RISC86 operation(s) for each instruction.

Instruction Mnemonic and Operand Types

The first column in these tables indicates the instruction mnemonic and operand types with the following notations:

- *disp16/32*—16-bit or 32-bit displacement value
- *disp32/48*—doubleword or 48-bit displacement value
- *disp8*—8-bit displacement value
- *eXX*—register width depending on the operand size
- *imm16/32*—16-bit or 32-bit immediate value
- *imm8*—8-bit immediate value
- *mem16/32*—word or doubleword integer value in memory
- *mem32/48*—doubleword or 48-bit integer value in memory
- *mem32real*—32-bit floating-point value in memory
- *mem48*—48-bit integer value in memory
- *mem64*—64-bit integer value in memory
- *mem64real*—64-bit floating-point value in memory
- *mem8*—byte integer value in memory
- *mem80real*—80-bit floating-point value in memory
- *mmreg*—MMX/3DNow! register

- *mmreg1*—MMX/3DNow! register defined by bits 5, 4, and 3 of the modR/M byte
- *mmreg2*—MMX/3DNow! register defined by bits 2, 1, and 0 of the modR/M byte
- *mreg16/32*—word or doubleword integer register, or word or doubleword integer value in memory defined by the modR/M byte
- *mreg8*—byte integer register or byte integer value in memory defined by the modR/M byte
- *reg8*—byte integer register defined by instruction byte(s) or bits 5, 4, and 3 of the modR/M byte
- *reg16/32*—word or doubleword integer register defined by instruction byte(s) or bits 5, 4, and 3 of the modR/M byte

Opcode Bytes

The second and third columns list all applicable opcode bytes.

ModR/M Byte

The fourth column lists the modR/M byte when used by the instruction. The modR/M byte defines the instruction as a register or memory form. If modR/M bits 7 and 6 are documented as mm (memory form), mm can only be 10b, 01b or 00b.

Decode Type

The fifth column lists the type of instruction decode—short, long, and vector. The AMD-K6-III+ processor decode logic can process two short, one long, or one vector decode per clock.

RISC86® Operation

The sixth column lists the type of RISC86 operation(s) required for the instruction. The operation types and corresponding execution units are as follows:

- *alu*—either of the integer execution units
- *alux*—integer X execution unit only
- *branch*—branch condition unit
- *float*—floating-point execution unit
- *limm*—load immediate, instruction control unit
- *load, fload, mload*—load unit
- *meu*—multimedia execution units for MMX and 3DNow! instructions
- *store, fstore, mstore*—store unit

Table 12. Integer Instructions

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-----------------------------------|------------|-------------|-------------|-------------|-------------------|
| AAA | 37h | | | vector | |
| AAD | D5h | 0Ah | | vector | |
| AAM | D4h | 0Ah | | vector | |
| AAS | 3Fh | | | vector | |
| ADC mreg8, reg8 | 10h | | 11-xxx-xxx | vector | |
| ADC mem8, reg8 | 10h | | mm-xxx-xxx | vector | |
| ADC mreg16/32, reg16/32 | 11h | | 11-xxx-xxx | vector | |
| ADC mem16/32, reg16/32 | 11h | | mm-xxx-xxx | vector | |
| ADC reg8, mreg8 | 12h | | 11-xxx-xxx | vector | |
| ADC reg8, mem8 | 12h | | mm-xxx-xxx | vector | |
| ADC reg16/32, mreg16/32 | 13h | | 11-xxx-xxx | vector | |
| ADC reg16/32, mem16/32 | 13h | | mm-xxx-xxx | vector | |
| ADC AL, imm8 | 14h | | | vector | |
| ADC EAX, imm16/32 | 15h | | | vector | |
| ADC mreg8, imm8 | 80h | | 11-010-xxx | vector | |
| ADC mem8, imm8 | 80h | | mm-010-xxx | vector | |
| ADC mreg16/32, imm16/32 | 81h | | 11-010-xxx | vector | |
| ADC mem16/32, imm16/32 | 81h | | mm-010-xxx | vector | |
| ADC mreg16/32, imm8 (signed ext.) | 83h | | 11-010-xxx | vector | |
| ADC mem16/32, imm8 (signed ext.) | 83h | | mm-010-xxx | vector | |
| ADD mreg8, reg8 | 00h | | 11-xxx-xxx | short | alux |
| ADD mem8, reg8 | 00h | | mm-xxx-xxx | long | load, alux, store |
| ADD mreg16/32, reg16/32 | 01h | | 11-xxx-xxx | short | alu |
| ADD mem16/32, reg16/32 | 01h | | mm-xxx-xxx | long | load, alu, store |
| ADD reg8, mreg8 | 02h | | 11-xxx-xxx | short | alux |
| ADD reg8, mem8 | 02h | | mm-xxx-xxx | short | load, alux |
| ADD reg16/32, mreg16/32 | 03h | | 11-xxx-xxx | short | alu |
| ADD reg16/32, mem16/32 | 03h | | mm-xxx-xxx | short | load, alu |
| ADD AL, imm8 | 04h | | | short | alux |
| ADD EAX, imm16/32 | 05h | | | short | alu |
| ADD mreg8, imm8 | 80h | | 11-000-xxx | short | alux |
| ADD mem8, imm8 | 80h | | mm-000-xxx | long | load, alux, store |
| ADD mreg16/32, imm16/32 | 81h | | 11-000-xxx | short | alu |
| ADD mem16/32, imm16/32 | 81h | | mm-000-xxx | long | load, alu, store |
| ADD mreg16/32, imm8 (signed ext.) | 83h | | 11-000-xxx | short | alux |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-----------------------------------|------------|-------------|-------------|-------------|-------------------|
| ADD mem16/32, imm8 (signed ext.) | 83h | | mm-000-xxx | long | load, alux, store |
| AND mreg8, reg8 | 20h | | 11-xxx-xxx | short | alux |
| AND mem8, reg8 | 20h | | mm-xxx-xxx | long | load, alux, store |
| AND mreg16/32, reg16/32 | 21h | | 11-xxx-xxx | short | alu |
| AND mem16/32, reg16/32 | 21h | | mm-xxx-xxx | long | load, alu, store |
| AND reg8, mreg8 | 22h | | 11-xxx-xxx | short | alux |
| AND reg8, mem8 | 22h | | mm-xxx-xxx | short | load, alux |
| AND reg16/32, mreg16/32 | 23h | | 11-xxx-xxx | short | alu |
| AND reg16/32, mem16/32 | 23h | | mm-xxx-xxx | short | load, alu |
| AND AL, imm8 | 24h | | | short | alux |
| AND EAX, imm16/32 | 25h | | | short | alu |
| AND mreg8, imm8 | 80h | | 11-100-xxx | short | alux |
| AND mem8, imm8 | 80h | | mm-100-xxx | long | load, alux, store |
| AND mreg16/32, imm16/32 | 81h | | 11-100-xxx | short | alu |
| AND mem16/32, imm16/32 | 81h | | mm-100-xxx | long | load, alu, store |
| AND mreg16/32, imm8 (signed ext.) | 83h | | 11-100-xxx | short | alux |
| AND mem16/32, imm8 (signed ext.) | 83h | | mm-100-xxx | long | load, alux, store |
| ARPL mreg16, reg16 | 63h | | 11-xxx-xxx | vector | |
| ARPL mem16, reg16 | 63h | | mm-xxx-xxx | vector | |
| BOUND | 62h | | | vector | |
| BSF reg16/32, mreg16/32 | 0Fh | BCh | 11-xxx-xxx | vector | |
| BSF reg16/32, mem16/32 | 0Fh | BCh | mm-xxx-xxx | vector | |
| BSR reg16/32, mreg16/32 | 0Fh | BDh | 11-xxx-xxx | vector | |
| BSR reg16/32, mem16/32 | 0Fh | BDh | mm-xxx-xxx | vector | |
| BSWAP EAX | 0Fh | C8h | | long | alu |
| BSWAP ECX | 0Fh | C9h | | long | alu |
| BSWAP EDX | 0Fh | CAh | | long | alu |
| BSWAP EBX | 0Fh | CBh | | long | alu |
| BSWAP ESP | 0Fh | CCh | | long | alu |
| BSWAP EBP | 0Fh | CDh | | long | alu |
| BSWAP ESI | 0Fh | CEh | | long | alu |
| BSWAP EDI | 0Fh | CFh | | long | alu |
| BT mreg16/32, reg16/32 | 0Fh | A3h | 11-xxx-xxx | vector | |
| BT mem16/32, reg16/32 | 0Fh | A3h | mm-xxx-xxx | vector | |
| BT mreg16/32, imm8 | 0Fh | BAh | 11-100-xxx | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-----------------------------|------------|-------------|-------------|-------------|-------------------|
| BT mem16/32, imm8 | 0Fh | BAh | mm-100-xxx | vector | |
| BTC mreg16/32, reg16/32 | 0Fh | BBh | 11-xxx-xxx | vector | |
| BTC mem16/32, reg16/32 | 0Fh | BBh | mm-xxx-xxx | vector | |
| BTC mreg16/32, imm8 | 0Fh | BAh | 11-111-xxx | vector | |
| BTC mem16/32, imm8 | 0Fh | BAh | mm-111-xxx | vector | |
| BTR mreg16/32, reg16/32 | 0Fh | B3h | 11-xxx-xxx | vector | |
| BTR mem16/32, reg16/32 | 0Fh | B3h | mm-xxx-xxx | vector | |
| BTR mreg16/32, imm8 | 0Fh | BAh | 11-110-xxx | vector | |
| BTR mem16/32, imm8 | 0Fh | BAh | mm-110-xxx | vector | |
| BTS mreg16/32, reg16/32 | 0Fh | ABh | 11-xxx-xxx | vector | |
| BTS mem16/32, reg16/32 | 0Fh | ABh | mm-xxx-xxx | vector | |
| BTS mreg16/32, imm8 | 0Fh | BAh | 11-101-xxx | vector | |
| BTS mem16/32, imm8 | 0Fh | BAh | mm-101-xxx | vector | |
| CALL full pointer | 9Ah | | | vector | |
| CALL near imm16/32 | E8h | | | short | store |
| CALL mem16:16/32 | FFh | | 11-011-xxx | vector | |
| CALL near mreg32 (indirect) | FFh | | 11-010-xxx | vector | |
| CALL near mem32 (indirect) | FFh | | mm-010-xxx | vector | |
| CBW/CWDE EAX | 98h | | | vector | |
| CLC | F8h | | | vector | |
| CLD | FCh | | | vector | |
| CLI | FAh | | | vector | |
| CLTS | 0Fh | 06h | | vector | |
| CMC | F5h | | | vector | |
| CMP mreg8, reg8 | 38h | | 11-xxx-xxx | short | alux |
| CMP mem8, reg8 | 38h | | mm-xxx-xxx | short | load, alux |
| CMP mreg16/32, reg16/32 | 39h | | 11-xxx-xxx | short | alu |
| CMP mem16/32, reg16/32 | 39h | | mm-xxx-xxx | short | load, alu |
| CMP reg8, mreg8 | 3Ah | | 11-xxx-xxx | short | alux |
| CMP reg8, mem8 | 3Ah | | mm-xxx-xxx | short | load, alux |
| CMP reg16/32, mreg16/32 | 3Bh | | 11-xxx-xxx | short | alu |
| CMP reg16/32, mem16/32 | 3Bh | | mm-xxx-xxx | short | load, alu |
| CMP AL, imm8 | 3Ch | | | short | alux |
| CMP EAX, imm16/32 | 3Dh | | | short | alu |
| CMP mreg8, imm8 | 80h | | 11-111-xxx | short | alux |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-----------------------------------|------------|-------------|-------------|-------------|-------------------|
| CMP mem8, imm8 | 80h | | mm-111-xxx | short | load, alux |
| CMP mreg16/32, imm16/32 | 81h | | 11-111-xxx | short | alu |
| CMP mem16/32, imm16/32 | 81h | | mm-111-xxx | short | load, alu |
| CMP mreg16/32, imm8 (signed ext.) | 83h | | 11-111-xxx | long | load, alu |
| CMP mem16/32, imm8 (signed ext.) | 83h | | mm-111-xxx | long | load, alu |
| CMPSB mem8, mem8 | A6h | | | vector | |
| CMPSW mem16, mem32 | A7h | | | vector | |
| CMPSD mem32, mem32 | A7h | | | vector | |
| CMPXCHG mreg8, reg8 | 0Fh | B0h | 11-xxx-xxx | vector | |
| CMPXCHG mem8, reg8 | 0Fh | B0h | mm-xxx-xxx | vector | |
| CMPXCHG mreg16/32, reg16/32 | 0Fh | B1h | 11-xxx-xxx | vector | |
| CMPXCHG mem16/32, reg16/32 | 0Fh | B1h | mm-xxx-xxx | vector | |
| CMPXCHG8B EDX:EAX | 0Fh | C7h | 11-xxx-xxx | vector | |
| CMPXCHG8B mem64 | 0Fh | C7h | mm-xxx-xxx | vector | |
| CPUID | 0Fh | A2h | | vector | |
| CWD/CDQ EDX, EAX | 99h | | | vector | |
| DAA | 27h | | | vector | |
| DAS | 2Fh | | | vector | |
| DEC EAX | 48h | | | short | alu |
| DEC ECX | 49h | | | short | alu |
| DEC EDX | 4Ah | | | short | alu |
| DEC EBX | 4Bh | | | short | alu |
| DEC ESP | 4Ch | | | short | alu |
| DEC EBP | 4Dh | | | short | alu |
| DEC ESI | 4Eh | | | short | alu |
| DEC EDI | 4Fh | | | short | alu |
| DEC mreg8 | FEh | | 11-001-xxx | vector | |
| DEC mem8 | FEh | | mm-001-xxx | long | load, alux, store |
| DEC mreg16/32 | FFh | | 11-001-xxx | vector | |
| DEC mem16/32 | FFh | | mm-001-xxx | long | load, alu, store |
| DIV AL, mreg8 | F6h | | 11-110-xxx | vector | |
| DIV AL, mem8 | F6h | | mm-110-xxx | vector | |
| DIV EAX, mreg16/32 | F7h | | 11-110-xxx | vector | |
| DIV EAX, mem16/32 | F7h | | mm-110-xxx | vector | |
| IDIV mreg8 | F6h | | 11-111-xxx | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|---|------------|-------------|-------------|-------------|-------------------|
| IDIV mem8 | F6h | | mm-111-xxx | vector | |
| IDIV EAX, mreg16/32 | F7h | | 11-111-xxx | vector | |
| IDIV EAX, mem16/32 | F7h | | mm-111-xxx | vector | |
| IMUL reg16/32, imm16/32 | 69h | | 11-xxx-xxx | vector | |
| IMUL reg16/32, mreg16/32, imm16/32 | 69h | | 11-xxx-xxx | vector | |
| IMUL reg16/32, mem16/32, imm16/32 | 69h | | mm-xxx-xxx | vector | |
| IMUL reg16/32, imm8 (sign extended) | 6Bh | | 11-xxx-xxx | vector | |
| IMUL reg16/32, mreg16/32, imm8 (signed) | 6Bh | | 11-xxx-xxx | vector | |
| IMUL reg16/32, mem16/32, imm8 (signed) | 6Bh | | mm-xxx-xxx | vector | |
| IMUL AX, AL, mreg8 | F6h | | 11-101-xxx | vector | |
| IMUL AX, AL, mem8 | F6h | | mm-101-xxx | vector | |
| IMUL EDX:EAX, EAX, mreg16/32 | F7h | | 11-101-xxx | vector | |
| IMUL EDX:EAX, EAX, mem16/32 | F7h | | mm-101-xxx | vector | |
| IMUL reg16/32, mreg16/32 | 0Fh | AFh | 11-xxx-xxx | vector | |
| IMUL reg16/32, mem16/32 | 0Fh | AFh | mm-xxx-xxx | vector | |
| IN AL, imm8 | E4h | | | vector | |
| IN AX, imm8 | E5h | | | vector | |
| IN EAX, imm8 | E5h | | | vector | |
| IN AL, DX | ECh | | | vector | |
| IN AX, DX | EDh | | | vector | |
| IN EAX, DX | EDh | | | vector | |
| INC EAX | 40h | | | short | alu |
| INC ECX | 41h | | | short | alu |
| INC EDX | 42h | | | short | alu |
| INC EBX | 43h | | | short | alu |
| INC ESP | 44h | | | short | alu |
| INC EBP | 45h | | | short | alu |
| INC ESI | 46h | | | short | alu |
| INC EDI | 47h | | | short | alu |
| INC mreg8 | FEh | | 11-000-xxx | vector | |
| INC mem8 | FEh | | mm-000-xxx | long | load, alu, store |
| INC mreg16/32 | FFh | | 11-000-xxx | vector | |
| INC mem16/32 | FFh | | mm-000-xxx | long | load, alu, store |
| INVD | 0Fh | 08h | | vector | |
| INVLPG | 0Fh | 01h | mm-111-xxx | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-----------------------------|------------|-------------|-------------|-------------|-------------------|
| JO short disp8 | 70h | | | short | branch |
| JB/JNAE short disp8 | 71h | | | short | branch |
| JNO short disp8 | 71h | | | short | branch |
| JNB/JAE short disp8 | 73h | | | short | branch |
| JZ/JE short disp8 | 74h | | | short | branch |
| JNZ/JNE short disp8 | 75h | | | short | branch |
| JBE/JNA short disp8 | 76h | | | short | branch |
| JNBE/JA short disp8 | 77h | | | short | branch |
| JS short disp8 | 78h | | | short | branch |
| JNS short disp8 | 79h | | | short | branch |
| JP/JPE short disp8 | 7Ah | | | short | branch |
| JNP/JPO short disp8 | 7Bh | | | short | branch |
| JL/JNGE short disp8 | 7Ch | | | short | branch |
| JNL/JGE short disp8 | 7Dh | | | short | branch |
| JLE/JNG short disp8 | 7Eh | | | short | branch |
| JNLE/JG short disp8 | 7Fh | | | short | branch |
| JCXZ/JEC short disp8 | E3h | | | vector | |
| JO near disp16/32 | 0Fh | 80h | | short | branch |
| JNO near disp16/32 | 0Fh | 81h | | short | branch |
| JB/JNAE near disp16/32 | 0Fh | 82h | | short | branch |
| JNB/JAE near disp16/32 | 0Fh | 83h | | short | branch |
| JZ/JE near disp16/32 | 0Fh | 84h | | short | branch |
| JNZ/JNE near disp16/32 | 0Fh | 85h | | short | branch |
| JBE/JNA near disp16/32 | 0Fh | 86h | | short | branch |
| JNBE/JA near disp16/32 | 0Fh | 87h | | short | branch |
| JS near disp16/32 | 0Fh | 88h | | short | branch |
| JNS near disp16/32 | 0Fh | 89h | | short | branch |
| JP/JPE near disp16/32 | 0Fh | 8Ah | | short | branch |
| JNP/JPO near disp16/32 | 0Fh | 8Bh | | short | branch |
| JL/JNGE near disp16/32 | 0Fh | 8Ch | | short | branch |
| JNL/JGE near disp16/32 | 0Fh | 8Dh | | short | branch |
| JLE/JNG near disp16/32 | 0Fh | 8Eh | | short | branch |
| JNLE/JG near disp16/32 | 0Fh | 8Fh | | short | branch |
| JMP near disp16/32 (direct) | E9h | | | short | branch |
| JMP far disp32/48 (direct) | EAh | | | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-------------------------------|------------|-------------|-------------|-------------|-------------------|
| JMP disp8 (short) | EBh | | | short | branch |
| JMP far mreg32 (indirect) | EFh | | 11-101-xxx | vector | |
| JMP far mem32 (indirect) | EFh | | mm-101-xxx | vector | |
| JMP near mreg16/32 (indirect) | FFh | | 11-100-xxx | vector | |
| JMP near mem16/32 (indirect) | FFh | | mm-100-xxx | vector | |
| LAHF | 9Fh | | | vector | |
| LAR reg16/32, mreg16/32 | 0Fh | 02h | 11-xxx-xxx | vector | |
| LAR reg16/32, mem16/32 | 0Fh | 02h | mm-xxx-xxx | vector | |
| LDS reg16/32, mem32/48 | C5h | | mm-xxx-xxx | vector | |
| LEA reg16/32, mem16/32 | 8Dh | | mm-xxx-xxx | short | load, alu |
| LEAVE | C9h | | | long | load, alu, alu |
| LES reg16/32, mem32/48 | C4h | | mm-xxx-xxx | vector | |
| LFS reg16/32, mem32/48 | 0Fh | B4h | | vector | |
| LGDT mem48 | 0Fh | 01h | mm-010-xxx | vector | |
| LGS reg16/32, mem32/48 | 0Fh | B5h | | vector | |
| LIDT mem48 | 0Fh | 01h | mm-011-xxx | vector | |
| LLDT mreg16 | 0Fh | 00h | 11-010-xxx | vector | |
| LLDT mem16 | 0Fh | 00h | mm-010-xxx | vector | |
| LMSW mreg16 | 0Fh | 01h | 11-100-xxx | vector | |
| LMSW mem16 | 0Fh | 01h | mm-100-xxx | vector | |
| LODSB AL, mem8 | ACH | | | long | load, alu |
| LODSW AX, mem16 | ADh | | | long | load, alu |
| LODSD EAX, mem32 | ADh | | | long | load, alu |
| LOOP disp8 | E2h | | | short | alu, branch |
| LOOPE/LOOPZ disp8 | E1h | | | vector | |
| LOOPNE/LOOPNZ disp8 | E0h | | | vector | |
| LSL reg16/32, mreg16/32 | 0Fh | 03h | 11-xxx-xxx | vector | |
| LSL reg16/32, mem16/32 | 0Fh | 03h | mm-xxx-xxx | vector | |
| LSS reg16/32, mem32/48 | 0Fh | B2h | mm-xxx-xxx | vector | |
| LTR mreg16 | 0Fh | 00h | 11-011-xxx | vector | |
| LTR mem16 | 0Fh | 00h | mm-011-xxx | vector | |
| MOV mreg8, reg8 | 88h | | 11-xxx-xxx | short | alux |
| MOV mem8, reg8 | 88h | | mm-xxx-xxx | short | store |
| MOV mreg16/32, reg16/32 | 89h | | 11-xxx-xxx | short | alu |
| MOV mem16/32, reg16/32 | 89h | | mm-xxx-xxx | short | store |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-------------------------|------------|-------------|-------------|-------------|-------------------|
| MOV reg8, mreg8 | 8Ah | | 11-xxx-xxx | short | alux |
| MOV reg8, mem8 | 8Ah | | mm-xxx-xxx | short | load |
| MOV reg16/32, mreg16/32 | 8Bh | | 11-xxx-xxx | short | alu |
| MOV reg16/32, mem16/32 | 8Bh | | mm-xxx-xxx | short | load |
| MOV mreg16, segment reg | 8Ch | | 11-xxx-xxx | long | load |
| MOV mem16, segment reg | 8Ch | | mm-xxx-xxx | vector | |
| MOV segment reg, mreg16 | 8Eh | | 11-xxx-xxx | vector | |
| MOV segment reg, mem16 | 8Eh | | mm-xxx-xxx | vector | |
| MOV AL, mem8 | A0h | | | short | load |
| MOV EAX, mem16/32 | A1h | | | short | load |
| MOV mem8, AL | A2h | | | short | store |
| MOV mem16/32, EAX | A3h | | | short | store |
| MOV AL, imm8 | B0h | | | short | limm |
| MOV CL, imm8 | B1h | | | short | limm |
| MOV DL, imm8 | B2h | | | short | limm |
| MOV BL, imm8 | B3h | | | short | limm |
| MOV AH, imm8 | B4h | | | short | limm |
| MOV CH, imm8 | B5h | | | short | limm |
| MOV DH, imm8 | B6h | | | short | limm |
| MOV BH, imm8 | B7h | | | short | limm |
| MOV EAX, imm16/32 | B8h | | | short | limm |
| MOV ECX, imm16/32 | B9h | | | short | limm |
| MOV EDX, imm16/32 | BAh | | | short | limm |
| MOV EBX, imm16/32 | BBh | | | short | limm |
| MOV ESP, imm16/32 | BCh | | | short | limm |
| MOV EBP, imm16/32 | BDh | | | short | limm |
| MOV ESI, imm16/32 | BEh | | | short | limm |
| MOV EDI, imm16/32 | BFh | | | short | limm |
| MOV mreg8, imm8 | C6h | | 11-000-xxx | short | limm |
| MOV mem8, imm8 | C6h | | mm-000-xxx | long | store |
| MOV mreg16/32, imm16/32 | C7h | | 11-000-xxx | short | limm |
| MOV mem16/32, imm16/32 | C7h | | mm-000-xxx | long | store |
| MOV reg32, CR0 | 0Fh | 20h | 11-000-xxx | vector | |
| MOV reg32, CR2 | 0Fh | 20h | 11-010-xxx | vector | |
| MOV reg32, CR3 | 0Fh | 20h | 11-011-xxx | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|------------------------|------------|-------------|-------------|-------------|-------------------------|
| MOV reg32, CR4 | 0Fh | 20h | 11-100-xxx | vector | |
| MOV CR0, reg32 | 0Fh | 22h | 11-000-xxx | vector | |
| MOV CR2, reg32 | 0Fh | 22h | 11-010-xxx | vector | |
| MOV CR3, reg32 | 0Fh | 22h | 11-011-xxx | vector | |
| MOV CR4, reg32 | 0Fh | 22h | 11-100-xxx | vector | |
| MOVSB mem8, mem8 | A4h | | | long | load, store, alux, alux |
| MOVSD mem16, mem16 | A5h | | | long | load, store, alu, alu |
| MOVSW mem32, mem32 | A5h | | | long | load, store, alu, alu |
| MOVSX reg16/32, mreg8 | 0Fh | BEh | 11-xxx-xxx | short | alu |
| MOVSX reg16/32, mem8 | 0Fh | BEh | mm-xxx-xxx | short | load, alu |
| MOVSX reg32, mreg16 | 0Fh | BFh | 11-xxx-xxx | short | alu |
| MOVSX reg32, mem16 | 0Fh | BFh | mm-xxx-xxx | short | load, alu |
| MOVZX reg16/32, mreg8 | 0Fh | B6h | 11-xxx-xxx | short | alu |
| MOVZX reg16/32, mem8 | 0Fh | B6h | mm-xxx-xxx | short | load, alu |
| MOVZX reg32, mreg16 | 0Fh | B7h | 11-xxx-xxx | short | alu |
| MOVZX reg32, mem16 | 0Fh | B7h | mm-xxx-xxx | short | load, alu |
| MUL AL, mreg8 | F6h | | 11-100-xxx | vector | |
| MUL AL, mem8 | F6h | | mm-100-xxx | vector | |
| MUL EAX, mreg16/32 | F7h | | 11-100-xxx | vector | |
| MUL EAX, mem16/32 | F7h | | mm-100-xxx | vector | |
| NEG mreg8 | F6h | | 11-011-xxx | short | alux |
| NEG mem8 | F6h | | mm-011-xxx | vector | |
| NEG mreg16/32 | F7h | | 11-011-xxx | short | alu |
| NEG mem16/32 | F7h | | mm-011-xxx | vector | |
| NOP (XCHG EAX, EAX) | 90h | | | short | limm |
| NOT mreg8 | F6h | | 11-010-xxx | short | alux |
| NOT mem8 | F6h | | mm-010-xxx | vector | |
| NOT mreg16/32 | F7h | | 11-010-xxx | short | alu |
| NOT mem16/32 | F7h | | mm-010-xxx | vector | |
| OR mreg8, reg8 | 08h | | 11-xxx-xxx | short | alux |
| OR mem8, reg8 | 08h | | mm-xxx-xxx | long | load, alux, store |
| OR mreg16/32, reg16/32 | 09h | | 11-xxx-xxx | short | alu |
| OR mem16/32, reg16/32 | 09h | | mm-xxx-xxx | long | load, alu, store |
| OR reg8, mreg8 | 0Ah | | 11-xxx-xxx | short | alux |
| OR reg8, mem8 | 0Ah | | mm-xxx-xxx | short | load, alux |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|----------------------------------|------------|-------------|-------------|-------------|-------------------|
| OR reg16/32, mreg16/32 | 0Bh | | 11-xxx-xxx | short | alu |
| OR reg16/32, mem16/32 | 0Bh | | mm-xxx-xxx | short | load, alu |
| OR AL, imm8 | 0Ch | | | short | alux |
| OR EAX, imm16/32 | 0Dh | | | short | alu |
| OR mreg8, imm8 | 80h | | 11-001-xxx | short | alux |
| OR mem8, imm8 | 80h | | mm-001-xxx | long | load, alux, store |
| OR mreg16/32, imm16/32 | 81h | | 11-001-xxx | short | alu |
| OR mem16/32, imm16/32 | 81h | | mm-001-xxx | long | load, alu, store |
| OR mreg16/32, imm8 (signed ext.) | 83h | | 11-001-xxx | short | alux |
| OR mem16/32, imm8 (signed ext.) | 83h | | mm-001-xxx | long | load, alux, store |
| OUT imm8, AL | E6h | | | vector | |
| OUT imm8, AX | E7h | | | vector | |
| OUT imm8, EAX | E7h | | | vector | |
| OUT DX, AL | EEh | | | vector | |
| OUT DX, AX | EFh | | | vector | |
| OUT DX, EAX | EFh | | | vector | |
| POP ES | 07h | | | vector | |
| POP SS | 17h | | | vector | |
| POP DS | 1Fh | | | vector | |
| POP FS | 0Fh | A1h | | vector | |
| POP GS | 0Fh | A9h | | vector | |
| POP EAX | 58h | | | short | load, alu |
| POP ECX | 59h | | | short | load, alu |
| POP EDX | 5Ah | | | short | load, alu |
| POP EBX | 5Bh | | | short | load, alu |
| POP ESP | 5Ch | | | short | load, alu |
| POP EBP | 5Dh | | | short | load, alu |
| POP ESI | 5Eh | | | short | load, alu |
| POP EDI | 5Fh | | | short | load, alu |
| POP mreg 16/32 | 8Fh | | 11-000-xxx | short | load, alu |
| POP mem 16/32 | 8Fh | | mm-000-xxx | long | load, store, alu |
| POPA/POPAD | 61h | | | vector | |
| POPF/POPFD | 9Dh | | | vector | |
| PUSH ES | 06h | | | long | load, store |
| PUSH CS | 0Eh | | | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|----------------------|------------|-------------|-------------|-------------|-------------------|
| PUSH FS | 0Fh | A0h | | vector | |
| PUSH GS | 0Fh | A8h | | vector | |
| PUSH SS | 16h | | | vector | |
| PUSH DS | 1Eh | | | long | load, store |
| PUSH EAX | 50h | | | short | store |
| PUSH ECX | 51h | | | short | store |
| PUSH EDX | 52h | | | short | store |
| PUSH EBX | 53h | | | short | store |
| PUSH ESP | 54h | | | short | store |
| PUSH EBP | 55h | | | short | store |
| PUSH ESI | 56h | | | short | store |
| PUSH EDI | 57h | | | short | store |
| PUSH imm8 | 6Ah | | | long | store |
| PUSH imm16/32 | 68h | | | long | store |
| PUSH mreg16/32 | FFh | | 11-110-xxx | vector | |
| PUSH mem16/32 | FFh | | mm-110-xxx | long | load, store |
| PUSHA/PUSHAD | 60h | | | vector | |
| PUSHF/PUSHFD | 9Ch | | | vector | |
| RCL mreg8, imm8 | C0h | | 11-010-xxx | vector | |
| RCL mem8, imm8 | C0h | | mm-010-xxx | vector | |
| RCL mreg16/32, imm8 | C1h | | 11-010-xxx | vector | |
| RCL mem16/32, imm8 | C1h | | mm-010-xxx | vector | |
| RCL mreg8, 1 | D0h | | 11-010-xxx | vector | |
| RCL mem8, 1 | D0h | | mm-010-xxx | vector | |
| RCL mreg16/32, 1 | D1h | | 11-010-xxx | vector | |
| RCL mem16/32, 1 | D1h | | mm-010-xxx | vector | |
| RCL mreg8, CL | D2h | | 11-010-xxx | vector | |
| RCL mem8, CL | D2h | | mm-010-xxx | vector | |
| RCL mreg16/32, CL | D3h | | 11-010-xxx | vector | |
| RCL mem16/32, CL | D3h | | mm-010-xxx | vector | |
| RCR mreg8, imm8 | C0h | | 11-011-xxx | vector | |
| RCR mem8, imm8 | C0h | | mm-011-xxx | vector | |
| RCR mreg16/32, imm8 | C1h | | 11-011-xxx | vector | |
| RCR mem16/32, imm8 | C1h | | mm-011-xxx | vector | |
| RCR mreg8, 1 | D0h | | 11-011-xxx | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|----------------------|------------|-------------|-------------|-------------|-------------------|
| RCR mem8, 1 | D0h | | mm-011-xxx | vector | |
| RCR mreg16/32, 1 | D1h | | 11-011-xxx | vector | |
| RCR mem16/32, 1 | D1h | | mm-011-xxx | vector | |
| RCR mreg8, CL | D2h | | 11-011-xxx | vector | |
| RCR mem8, CL | D2h | | mm-011-xxx | vector | |
| RCR mreg16/32, CL | D3h | | 11-011-xxx | vector | |
| RCR mem16/32, CL | D3h | | mm-011-xxx | vector | |
| RDMSR | 0Fh | 32h | | vector | |
| RDTSC | 0Fh | 31h | | vector | |
| RET near imm16 | C2h | | | vector | |
| RET near | C3h | | | vector | |
| RET far imm16 | CAh | | | vector | |
| RET far | CBh | | | vector | |
| ROL mreg8, imm8 | C0h | | 11-000-xxx | vector | |
| ROL mem8, imm8 | C0h | | mm-000-xxx | vector | |
| ROL mreg16/32, imm8 | C1h | | 11-000-xxx | vector | |
| ROL mem16/32, imm8 | C1h | | mm-000-xxx | vector | |
| ROL mreg8, 1 | D0h | | 11-000-xxx | vector | |
| ROL mem8, 1 | D0h | | mm-000-xxx | vector | |
| ROL mreg16/32, 1 | D1h | | 11-000-xxx | vector | |
| ROL mem16/32, 1 | D1h | | mm-000-xxx | vector | |
| ROL mreg8, CL | D2h | | 11-000-xxx | vector | |
| ROL mem8, CL | D2h | | mm-000-xxx | vector | |
| ROL mreg16/32, CL | D3h | | 11-000-xxx | vector | |
| ROL mem16/32, CL | D3h | | mm-000-xxx | vector | |
| ROR mreg8, imm8 | C0h | | 11-001-xxx | vector | |
| ROR mem8, imm8 | C0h | | mm-001-xxx | vector | |
| ROR mreg16/32, imm8 | C1h | | 11-001-xxx | vector | |
| ROR mem16/32, imm8 | C1h | | mm-001-xxx | vector | |
| ROR mreg8, 1 | D0h | | 11-001-xxx | vector | |
| ROR mem8, 1 | D0h | | mm-001-xxx | vector | |
| ROR mreg16/32, 1 | D1h | | 11-001-xxx | vector | |
| ROR mem16/32, 1 | D1h | | mm-001-xxx | vector | |
| ROR mreg8, CL | D2h | | 11-001-xxx | vector | |
| ROR mem8, CL | D2h | | mm-001-xxx | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-----------------------------------|------------|-------------|-------------|-------------|-------------------|
| ROR mreg16/32, CL | D3h | | 11-001-xxx | vector | |
| ROR mem16/32, CL | D3h | | mm-001-xxx | vector | |
| RSM | 0Fh | AAh | | vector | |
| SAHF | 9Eh | | | vector | |
| SAR mreg8, imm8 | C0h | | 11-111-xxx | short | alux |
| SAR mem8, imm8 | C0h | | mm-111-xxx | vector | |
| SAR mreg16/32, imm8 | C1h | | 11-111-xxx | short | alu |
| SAR mem16/32, imm8 | C1h | | mm-111-xxx | vector | |
| SAR mreg8, 1 | D0h | | 11-111-xxx | short | alux |
| SAR mem8, 1 | D0h | | mm-111-xxx | vector | |
| SAR mreg16/32, 1 | D1h | | 11-111-xxx | short | alu |
| SAR mem16/32, 1 | D1h | | mm-111-xxx | vector | |
| SAR mreg8, CL | D2h | | 11-111-xxx | short | alux |
| SAR mem8, CL | D2h | | mm-111-xxx | vector | |
| SAR mreg16/32, CL | D3h | | 11-111-xxx | short | alu |
| SAR mem16/32, CL | D3h | | mm-111-xxx | vector | |
| SBB mreg8, reg8 | 18h | | 11-xxx-xxx | vector | |
| SBB mem8, reg8 | 18h | | mm-xxx-xxx | vector | |
| SBB mreg16/32, reg16/32 | 19h | | 11-xxx-xxx | vector | |
| SBB mem16/32, reg16/32 | 19h | | mm-xxx-xxx | vector | |
| SBB reg8, mreg8 | 1Ah | | 11-xxx-xxx | vector | |
| SBB reg8, mem8 | 1Ah | | mm-xxx-xxx | vector | |
| SBB reg16/32, mreg16/32 | 1Bh | | 11-xxx-xxx | vector | |
| SBB reg16/32, mem16/32 | 1Bh | | mm-xxx-xxx | vector | |
| SBB AL, imm8 | 1Ch | | | vector | |
| SBB EAX, imm16/32 | 1Dh | | | vector | |
| SBB mreg8, imm8 | 80h | | 11-011-xxx | vector | |
| SBB mem8, imm8 | 80h | | mm-011-xxx | vector | |
| SBB mreg16/32, imm16/32 | 81h | | 11-011-xxx | vector | |
| SBB mem16/32, imm16/32 | 81h | | mm-011-xxx | vector | |
| SBB mreg16/32, imm8 (signed ext.) | 83h | | 11-011-xxx | vector | |
| SBB mem16/32, imm8 (signed ext.) | 83h | | mm-011-xxx | vector | |
| SCASB AL, mem8 | A Eh | | | vector | |
| SCASW AX, mem16 | A Fh | | | vector | |
| SCASD EAX, mem32 | A Fh | | | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|----------------------|------------|-------------|-------------|-------------|-------------------|
| SETO mreg8 | 0Fh | 90h | 11-xxx-xxx | vector | |
| SETO mem8 | 0Fh | 90h | mm-xxx-xxx | vector | |
| SETNO mreg8 | 0Fh | 91h | 11-xxx-xxx | vector | |
| SETNO mem8 | 0Fh | 91h | mm-xxx-xxx | vector | |
| SETB/SETNAE mreg8 | 0Fh | 92h | 11-xxx-xxx | vector | |
| SETB/SETNAE mem8 | 0Fh | 92h | mm-xxx-xxx | vector | |
| SETNB/SETAE mreg8 | 0Fh | 93h | 11-xxx-xxx | vector | |
| SETNB/SETAE mem8 | 0Fh | 93h | mm-xxx-xxx | vector | |
| SETZ/SETE mreg8 | 0Fh | 94h | 11-xxx-xxx | vector | |
| SETZ/SETE mem8 | 0Fh | 94h | mm-xxx-xxx | vector | |
| SETNZ/SETNE mreg8 | 0Fh | 95h | 11-xxx-xxx | vector | |
| SETNZ/SETNE mem8 | 0Fh | 95h | mm-xxx-xxx | vector | |
| SETBE/SETNA mreg8 | 0Fh | 96h | 11-xxx-xxx | vector | |
| SETBE/SETNA mem8 | 0Fh | 96h | mm-xxx-xxx | vector | |
| SETNBE/SETA mreg8 | 0Fh | 97h | 11-xxx-xxx | vector | |
| SETNBE/SETA mem8 | 0Fh | 97h | mm-xxx-xxx | vector | |
| SETS mreg8 | 0Fh | 98h | 11-xxx-xxx | vector | |
| SETS mem8 | 0Fh | 98h | mm-xxx-xxx | vector | |
| SETNS mreg8 | 0Fh | 99h | 11-xxx-xxx | vector | |
| SETNS mem8 | 0Fh | 99h | mm-xxx-xxx | vector | |
| SETP/SETPE mreg8 | 0Fh | 9Ah | 11-xxx-xxx | vector | |
| SETP/SETPE mem8 | 0Fh | 9Ah | mm-xxx-xxx | vector | |
| SETNP/SETPO mreg8 | 0Fh | 9Bh | 11-xxx-xxx | vector | |
| SETNP/SETPO mem8 | 0Fh | 9Bh | mm-xxx-xxx | vector | |
| SETL/SETNGE mreg8 | 0Fh | 9Ch | 11-xxx-xxx | vector | |
| SETL/SETNGE mem8 | 0Fh | 9Ch | mm-xxx-xxx | vector | |
| SETNL/SETGE mreg8 | 0Fh | 9Dh | 11-xxx-xxx | vector | |
| SETNL/SETGE mem8 | 0Fh | 9Dh | mm-xxx-xxx | vector | |
| SETLE/SETNG mreg8 | 0Fh | 9Eh | 11-xxx-xxx | vector | |
| SETLE/SETNG mem8 | 0Fh | 9Eh | mm-xxx-xxx | vector | |
| SETNLE/SETG mreg8 | 0Fh | 9Fh | 11-xxx-xxx | vector | |
| SETNLE/SETG mem8 | 0Fh | 9Fh | mm-xxx-xxx | vector | |
| SGDT mem48 | 0Fh | 01h | mm-000-xxx | vector | |
| SIDT mem48 | 0Fh | 01h | mm-001-xxx | vector | |
| SHL/SAL mreg8, imm8 | C0h | | 11-100-xxx | short | alux |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|--------------------------------|------------|-------------|-------------|-------------|-------------------|
| SHL/SAL mem8, imm8 | C0h | | mm-100-xxx | vector | |
| SHL/SAL mreg16/32, imm8 | C1h | | 11-100-xxx | short | alu |
| SHL/SAL mem16/32, imm8 | C1h | | mm-100-xxx | vector | |
| SHL/SAL mreg8, 1 | D0h | | 11-100-xxx | short | alux |
| SHL/SAL mem8, 1 | D0h | | mm-100-xxx | vector | |
| SHL/SAL mreg16/32, 1 | D1h | | 11-100-xxx | short | alu |
| SHL/SAL mem16/32, 1 | D1h | | mm-100-xxx | vector | |
| SHL/SAL mreg8, CL | D2h | | 11-100-xxx | short | alux |
| SHL/SAL mem8, CL | D2h | | mm-100-xxx | vector | |
| SHL/SAL mreg16/32, CL | D3h | | 11-100-xxx | short | alu |
| SHL/SAL mem16/32, CL | D3h | | mm-100-xxx | vector | |
| SHR mreg8, imm8 | C0h | | 11-101-xxx | short | alux |
| SHR mem8, imm8 | C0h | | mm-101-xxx | vector | |
| SHR mreg16/32, imm8 | C1h | | 11-101-xxx | short | alu |
| SHR mem16/32, imm8 | C1h | | mm-101-xxx | vector | |
| SHR mreg8, 1 | D0h | | 11-101-xxx | short | alux |
| SHR mem8, 1 | D0h | | mm-101-xxx | vector | |
| SHR mreg16/32, 1 | D1h | | 11-101-xxx | short | alu |
| SHR mem16/32, 1 | D1h | | mm-101-xxx | vector | |
| SHR mreg8, CL | D2h | | 11-101-xxx | short | alux |
| SHR mem8, CL | D2h | | mm-101-xxx | vector | |
| SHR mreg16/32, CL | D3h | | 11-101-xxx | short | alu |
| SHR mem16/32, CL | D3h | | mm-101-xxx | vector | |
| SHLD mreg16/32, reg16/32, imm8 | 0Fh | A4h | 11-xxx-xxx | vector | |
| SHLD mem16/32, reg16/32, imm8 | 0Fh | A4h | mm-xxx-xxx | vector | |
| SHLD mreg16/32, reg16/32, CL | 0Fh | A5h | 11-xxx-xxx | vector | |
| SHLD mem16/32, reg16/32, CL | 0Fh | A5h | mm-xxx-xxx | vector | |
| SHRD mreg16/32, reg16/32, imm8 | 0Fh | ACH | 11-xxx-xxx | vector | |
| SHRD mem16/32, reg16/32, imm8 | 0Fh | ACH | mm-xxx-xxx | vector | |
| SHRD mreg16/32, reg16/32, CL | 0Fh | ADh | 11-xxx-xxx | vector | |
| SHRD mem16/32, reg16/32, CL | 0Fh | ADh | mm-xxx-xxx | vector | |
| SLDT mreg16 | 0Fh | 00h | 11-000-xxx | vector | |
| SLDT mem16 | 0Fh | 00h | mm-000-xxx | vector | |
| SMSW mreg16 | 0Fh | 01h | 11-100-xxx | vector | |
| SMSW mem16 | 0Fh | 01h | mm-100-xxx | vector | |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-----------------------------------|------------|-------------|-------------|-------------|-------------------|
| STC | F9h | | | vector | |
| STD | FDh | | | vector | |
| STI | FBh | | | vector | |
| STOSB mem8, AL | AAh | | | long | store, alux |
| STOSW mem16, AX | ABh | | | long | store, alux |
| STOSD mem32, EAX | ABh | | | long | store, alux |
| STR mreg16 | 0Fh | 00h | 11-001-xxx | vector | |
| STR mem16 | 0Fh | 00h | mm-001-xxx | vector | |
| SUB mreg8, reg8 | 28h | | 11-xxx-xxx | short | alux |
| SUB mem8, reg8 | 28h | | mm-xxx-xxx | long | load, alux, store |
| SUB mreg16/32, reg16/32 | 29h | | 11-xxx-xxx | short | alu |
| SUB mem16/32, reg16/32 | 29h | | mm-xxx-xxx | long | load, alu, store |
| SUB reg8, mreg8 | 2Ah | | 11-xxx-xxx | short | alux |
| SUB reg8, mem8 | 2Ah | | mm-xxx-xxx | short | load, alux |
| SUB reg16/32, mreg16/32 | 2Bh | | 11-xxx-xxx | short | alu |
| SUB reg16/32, mem16/32 | 2Bh | | mm-xxx-xxx | short | load, alu |
| SUB AL, imm8 | 2Ch | | | short | alux |
| SUB EAX, imm16/32 | 2Dh | | | short | alu |
| SUB mreg8, imm8 | 80h | | 11-101-xxx | short | alux |
| SUB mem8, imm8 | 80h | | mm-101-xxx | long | load, alux, store |
| SUB mreg16/32, imm16/32 | 81h | | 11-101-xxx | short | alu |
| SUB mem16/32, imm16/32 | 81h | | mm-101-xxx | long | load, alu, store |
| SUB mreg16/32, imm8 (signed ext.) | 83h | | 11-101-xxx | short | alux |
| SUB mem16/32, imm8 (signed ext.) | 83h | | mm-101-xxx | long | load, alux, store |
| SYSCALL | 0Fh | 05h | | vector | |
| SYSRET | 0Fh | 07h | | vector | |
| TEST mreg8, reg8 | 84h | | 11-xxx-xxx | short | alux |
| TEST mem8, reg8 | 84h | | mm-xxx-xxx | vector | |
| TEST mreg16/32, reg16/32 | 85h | | 11-xxx-xxx | short | alu |
| TEST mem16/32, reg16/32 | 85h | | mm-xxx-xxx | vector | |
| TEST AL, imm8 | A8h | | | long | alux |
| TEST EAX, imm16/32 | A9h | | | long | alu |
| TEST mreg8, imm8 | F6h | | 11-000-xxx | long | alux |
| TEST mem8, imm8 | F6h | | mm-000-xxx | long | load, alux |
| TEST mreg16/32, imm16/32 | F7h | | 11-000-xxx | long | alu |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|--------------------------|------------|-------------|-------------|-------------|-------------------|
| TEST mem16/32, imm16/32 | F7h | | mm-000-xxx | long | load, alu |
| VERR mreg16 | 0Fh | 00h | 11-100-xxx | vector | |
| VERR mem16 | 0Fh | 00h | mm-100-xxx | vector | |
| VERW mreg16 | 0Fh | 00h | 11-101-xxx | vector | |
| VERW mem16 | 0Fh | 00h | mm-101-xxx | vector | |
| WAIT | 9Bh | | | vector | |
| WBINVD | 0Fh | 09h | | vector | |
| WRMSR | 0Fh | 30h | | vector | |
| XADD mreg8, reg8 | 0Fh | C0h | 11-100-xxx | vector | |
| XADD mem8, reg8 | 0Fh | C0h | mm-100-xxx | vector | |
| XADD mreg16/32, reg16/32 | 0Fh | C1h | 11-101-xxx | vector | |
| XADD mem16/32, reg16/32 | 0Fh | C1h | mm-101-xxx | vector | |
| XCHG reg8, mreg8 | 86h | | 11-xxx-xxx | vector | |
| XCHG reg8, mem8 | 86h | | mm-xxx-xxx | vector | |
| XCHG reg16/32, mreg16/32 | 87h | | 11-xxx-xxx | vector | |
| XCHG reg16/32, mem16/32 | 87h | | mm-xxx-xxx | vector | |
| XCHG EAX, EAX | 90h | | | short | limm |
| XCHG EAX, ECX | 91h | | | long | alu, alu, alu |
| XCHG EAX, EDX | 92h | | | long | alu, alu, alu |
| XCHG EAX, EBX | 93h | | | long | alu, alu, alu |
| XCHG EAX, ESP | 94h | | | long | alu, alu, alu |
| XCHG EAX, EBP | 95h | | | long | alu, alu, alu |
| XCHG EAX, ESI | 96h | | | long | alu, alu, alu |
| XCHG EAX, EDI | 97h | | | long | alu, alu, alu |
| XLAT | D7h | | | vector | |
| XOR mreg8, reg8 | 30h | | 11-xxx-xxx | short | alux |
| XOR mem8, reg8 | 30h | | mm-xxx-xxx | long | load, alux, store |
| XOR mreg16/32, reg16/32 | 31h | | 11-xxx-xxx | short | alu |
| XOR mem16/32, reg16/32 | 31h | | mm-xxx-xxx | long | load, alu, store |
| XOR reg8, mreg8 | 32h | | 11-xxx-xxx | short | alux |
| XOR reg8, mem8 | 32h | | mm-xxx-xxx | short | load, alux |
| XOR reg16/32, mreg16/32 | 33h | | 11-xxx-xxx | short | alu |
| XOR reg16/32, mem16/32 | 33h | | mm-xxx-xxx | short | load, alu |
| XOR AL, imm8 | 34h | | | short | alux |
| XOR EAX, imm16/32 | 35h | | | short | alu |

Table 12. Integer Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-----------------------------------|------------|-------------|-------------|-------------|-------------------|
| XOR mreg8, imm8 | 80h | | 11-110-xxx | short | alux |
| XOR mem8, imm8 | 80h | | mm-110-xxx | long | load, alux, store |
| XOR mreg16/32, imm16/32 | 81h | | 11-110-xxx | short | alu |
| XOR mem16/32, imm16/32 | 81h | | mm-110-xxx | long | load, alu, store |
| XOR mreg16/32, imm8 (signed ext.) | 83h | | 11-110-xxx | short | alux |
| XOR mem16/32, imm8 (signed ext.) | 83h | | mm-110-xxx | long | load, alux, store |

Table 13. Floating-Point Instructions

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|---|------------|-------------|-------------|-------------|-------------------|
| F2XM1 | D9h | F0h | | short | float |
| FABS | D9h | F1h | | short | float |
| FADD ST(0), ST(i) ¹ | D8h | | 11-000-xxx | short | float |
| FADD ST(0), mem32real | D8h | | mm-000-xxx | short | float, float |
| FADD ST(i), ST(0) ¹ | DCh | | 11-000-xxx | short | float |
| FADD ST(0), mem64real | DCh | | mm-000-xxx | short | float, float |
| FADDP ST(i), ST(0) ¹ | DEh | | 11-000-xxx | short | float |
| FBLD | DFh | | mm-100-xxx | vector | |
| FBSTP | DFh | | mm-110-xxx | vector | |
| FCFS | D9h | E0h | | short | float |
| FCLEX | DBh | E2h | | vector | |
| FCOM ST(0), ST(i) ¹ | D8h | | 11-010-xxx | short | float |
| FCOM ST(0), mem32real | D8h | | mm-010-xxx | short | float, float |
| FCOM ST(0), mem64real | DCh | | mm-010-xxx | short | float, float |
| FCOMP ST(0), ST(i) ¹ | D8h | | 11-011-xxx | short | float |
| FCOMP ST(0), mem32real | D8h | | mm-011-xxx | short | float, float |
| FCOMP ST(0), mem64real | DCh | | mm-011-xxx | short | float, float |
| FCOMPP | DEh | D9h | 11-011-001 | short | float |
| FCOS | D9h | FFh | | short | float |
| FDECSTP | D9h | F6h | | short | float |
| FDIV ST(0), ST(i) (single precision) ¹ | D8h | | 11-110-xxx | short | float |
| FDIV ST(0), ST(i) (double precision) ¹ | D8h | | 11-110-xxx | short | float |
| FDIV ST(0), ST(i) (extended precision) ¹ | D8h | | 11-110-xxx | short | float |

Table 13. Floating-Point Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|---|------------|-------------|-------------|-------------|-------------------|
| FDIV ST(i), ST(0) (single precision) ¹ | DCh | | 11-111-xxx | short | float |
| FDIV ST(i), ST(0) (double precision) ¹ | DCh | | 11-111-xxx | short | float |
| FDIV ST(i), ST(0) (extended precision) ¹ | DCh | | 11-111-xxx | short | float |
| FDIV ST(0), mem32real | D8h | | mm-110-xxx | short | float, float |
| FDIV ST(0), mem64real | DCh | | mm-110-xxx | short | float, float |
| FDIVP ST(0), ST(i) ¹ | DEh | | 11-111-xxx | short | float |
| FDIVR ST(0), ST(i) ¹ | D8h | | 11-110-xxx | short | float |
| FDIVR ST(i), ST(0) ¹ | DCh | | 11-111-xxx | short | float |
| FDIVR ST(0), mem32real | D8h | | mm-111-xxx | short | float, float |
| FDIVR ST(0), mem64real | DCh | | mm-111-xxx | short | float, float |
| FDIVRP ST(i), ST(0) ¹ | DEh | | 11-110-xxx | short | float |
| FFREE ST(i) ¹ | DDh | | 11-000-xxx | short | float |
| FIADD ST(0), mem32int | DAh | | mm-000-xxx | short | float, float |
| FIADD ST(0), mem16int | DEh | | mm-000-xxx | short | float, float |
| FICOM ST(0), mem32int | DAh | | mm-010-xxx | short | float, float |
| FICOM ST(0), mem16int | DEh | | mm-010-xxx | short | float, float |
| FICOMP ST(0), mem32int | DAh | | mm-011-xxx | short | float, float |
| FICOMP ST(0), mem16int | DEh | | mm-011-xxx | short | float, float |
| FIDIV ST(0), mem32int | DAh | | mm-110-xxx | short | float, float |
| FIDIV ST(0), mem16int | DEh | | mm-110-xxx | short | float, float |
| FIDIVR ST(0), mem32int | DAh | | mm-111-xxx | short | float, float |
| FIDIVR ST(0), mem16int | DEh | | mm-111-xxx | short | float, float |
| FILD mem16int | DFh | | mm-000-xxx | short | float, float |
| FILD mem32int | DBh | | mm-000-xxx | short | float, float |
| FILD mem64int | DFh | | mm-101-xxx | short | float, float |
| FIMUL ST(0), mem32int | DAh | | mm-001-xxx | short | float, float |
| FIMUL ST(0), mem16int | DEh | | mm-001-xxx | short | float, float |
| FINCSTP | D9h | F7h | | short | |
| FINIT | DBh | E3h | | vector | |
| FIST mem16int | DFh | | mm-010-xxx | short | float, float |
| FIST mem32int | DBh | | mm-010-xxx | short | float, float |
| FISTP mem16int | DFh | | mm-011-xxx | short | float, float |
| FISTP mem32int | DBh | | mm-011-xxx | short | float, float |
| FISTP mem64int | DFh | | mm-111-xxx | short | float, float |

Table 13. Floating-Point Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|---------------------------------|------------|-------------|-------------|-------------|-------------------|
| FISUB ST(0), mem32int | DAh | | mm-100-xxx | short | float, float |
| FISUB ST(0), mem16int | DEh | | mm-100-xxx | short | float, float |
| FISUBR ST(0), mem32int | DAh | | mm-101-xxx | short | float, float |
| FISUBR ST(0), mem16int | DEh | | mm-101-xxx | short | float, float |
| FLD ST(i) ¹ | D9h | | 11-000-xxx | short | float, float |
| FLD mem32real | D9h | | mm-000-xxx | short | float, float |
| FLD mem64real | DDh | | mm-000-xxx | short | float, float |
| FLD mem80real | DBh | | mm-101-xxx | vector | |
| FLD1 | D9h | E8h | | short | float, float |
| FLDCW | D9h | | mm-101-xxx | vector | |
| FLDENV | D9h | | mm-100-xxx | short | float, float |
| FLDL2E | D9h | EAh | | short | float |
| FLDL2T | D9h | E9h | | short | float |
| FLDLG2 | D9h | ECh | | short | float |
| FLDLN2 | D9h | EDh | | short | float |
| FLDPI | D9h | EBh | | short | float |
| FLDZ | D9h | EEh | | short | float |
| FMUL ST(0), ST(i) ¹ | D8h | | 11-001-xxx | short | float |
| FMUL ST(i), ST(0) ¹ | DCh | | 11-001-xxx | short | float |
| FMUL ST(0), mem32real | D8h | | mm-001-xxx | short | float, float |
| FMUL ST(0), mem64real | DCh | | mm-001-xxx | short | float, float |
| FMULP ST(0), ST(i) ¹ | DEh | | 11-001-xxx | short | float |
| FNOP | D9h | D0h | | short | float |
| FPATAN | D9h | F3h | | short | float |
| FPREM | D9h | F8h | | short | float |
| FPREM1 | D9h | F5h | | short | float |
| FPTAN | D9h | F2h | | vector | |
| FRNDINT | D9h | FCh | | short | float |
| FRSTOR | DDh | | mm-100-xxx | vector | |
| FSAVE | DDh | | mm-110-xxx | vector | |
| FSCALE | D9h | FDh | | short | float |
| FSIN | D9h | FEh | | short | float |
| FSINCOS | D9h | FBh | | vector | |
| FSQRT (single precision) | D9h | FAh | | short | float |
| FSQRT (double precision) | D9h | FAh | | short | float |

Table 13. Floating-Point Instructions (continued)

| Instruction Mnemonic | First Byte | Second Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|----------------------------------|------------|-------------|-------------|-------------|-------------------|
| FSQRT (extended precision) | D9h | FAh | | short | float |
| FST mem32real | D9h | | mm-010-xxx | short | fstore |
| FST mem64real | DDh | | mm-010-xxx | short | fstore |
| FST ST(i) ¹ | DDh | | 11-010-xxx | short | fstore |
| FSTCW | D9h | | mm-111-xxx | vector | |
| FSTENV | D9h | | mm-110-xxx | vector | |
| FSTP mem32real | D9h | | mm-011-xxx | short | fstore |
| FSTP mem64real | DDh | | mm-011-xxx | short | fstore |
| FSTP mem80real | D9h | | mm-111-xxx | vector | |
| FSTP ST(i) ¹ | DDh | | 11-011-xxx | short | float |
| FSTSW AX | DFh | E0h | | vector | |
| FSTSW mem16 | DDh | | mm-111-xxx | vector | |
| FSUB ST(0), mem32real | D8h | | mm-100-xxx | short | float, float |
| FSUB ST(0), mem64real | DCh | | mm-100-xxx | short | float, float |
| FSUB ST(0), ST(i) ¹ | D8h | | 11-100-xxx | short | float |
| FSUB ST(i), ST(0) ¹ | DCh | | 11-101-xxx | short | float |
| FSUBP ST(0), ST(i) ¹ | DEh | | 11-101-xxx | short | float |
| FSUBR ST(0), mem32real | D8h | | mm-101-xxx | short | float, float |
| FSUBR ST(0), mem64real | DCh | | mm-101-xxx | short | float, float |
| FSUBR ST(0), ST(i) ¹ | D8h | | 11-100-xxx | short | float |
| FSUBR ST(i), ST(0) ¹ | DCh | | 11-101-xxx | short | float |
| FSUBRP ST(i), ST(0) ¹ | DEh | | 11-100-xxx | short | float |
| FTST | D9h | E4h | | short | float |
| FUCOM | DDh | | 11-100-xxx | short | float |
| FUCOMP | DDh | | 11-101-xxx | short | float |
| FUCOMPP | DAh | E9h | | short | float |
| FXAM | D9h | E5h | | short | float |
| FXCH | D9h | | 11-001-xxx | short | float |
| EXTRACT | D9h | F4h | | vector | |
| FYL2X | D9h | F1h | | short | float |
| FYL2XP1 | D9h | F9h | | short | float |
| FWAIT | 9Bh | | | vector | |

Notes:

1. The last three bits of the modR/M byte select the stack entry ST(i).

Table 14. MMX™ Instructions

| Instruction Mnemonic | Prefix Byte(s) | First Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|---------------------------------|----------------|------------|-------------|-------------|-------------------|
| EMMS | 0Fh | 77h | | vector | |
| MOVD mmreg, mreg32 ¹ | 0Fh | 6Eh | 11-xxx-xxx | short | meu |
| MOVD mmreg, mem32 | 0Fh | 6Eh | mm-xxx-xxx | short | mload |
| MOVD mreg32, mmreg ¹ | 0Fh | 7Eh | 11-xxx-xxx | short | mstore, load |
| MOVD mem32, mmreg | 0Fh | 7Eh | mm-xxx-xxx | short | mstore |
| MOVQ mmreg1, mmreg2 | 0Fh | 6Fh | 11-xxx-xxx | short | meu |
| MOVQ mmreg, mem64 | 0Fh | 6Fh | mm-xxx-xxx | short | mload |
| MOVQ mmreg2, mmreg1 | 0Fh | 7Fh | 11-xxx-xxx | short | meu |
| MOVQ mem64, mmreg | 0Fh | 7Fh | mm-xxx-xxx | short | mstore |
| PACKSSDW mmreg1, mmreg2 | 0Fh | 6Bh | 11-xxx-xxx | short | meu |
| PACKSSDW mmreg, mem64 | 0Fh | 6Bh | mm-xxx-xxx | short | mload, meu |
| PACKSSWB mmreg1, mmreg2 | 0Fh | 63h | 11-xxx-xxx | short | meu |
| PACKSSWB mmreg, mem64 | 0Fh | 63h | mm-xxx-xxx | short | mload, meu |
| PACKUSWB mmreg1, mmreg2 | 0Fh | 67h | 11-xxx-xxx | short | meu |
| PACKUSWB mmreg, mem64 | 0Fh | 67h | mm-xxx-xxx | short | mload, meu |
| PADDB mmreg1, mmreg2 | 0Fh | FCh | 11-xxx-xxx | short | meu |
| PADDB mmreg, mem64 | 0Fh | FCh | mm-xxx-xxx | short | mload, meu |
| PADD mmreg1, mmreg2 | 0Fh | FEh | 11-xxx-xxx | short | meu |
| PADD mmreg, mem64 | 0Fh | FEh | mm-xxx-xxx | short | mload, meu |
| PADDSB mmreg1, mmreg2 | 0Fh | ECh | 11-xxx-xxx | short | meu |
| PADDSB mmreg, mem64 | 0Fh | ECh | mm-xxx-xxx | short | mload, meu |
| PADDSW mmreg1, mmreg2 | 0Fh | EDh | 11-xxx-xxx | short | meu |
| PADDSW mmreg, mem64 | 0Fh | EDh | mm-xxx-xxx | short | mload, meu |
| PADDUSB mmreg1, mmreg2 | 0Fh | DCh | 11-xxx-xxx | short | meu |
| PADDUSB mmreg, mem64 | 0Fh | DCh | mm-xxx-xxx | short | mload, meu |
| PADDUSW mmreg1, mmreg2 | 0Fh | DDh | 11-xxx-xxx | short | meu |
| PADDUSW mmreg, mem64 | 0Fh | DDh | mm-xxx-xxx | short | mload, meu |
| PADDW mmreg1, mmreg2 | 0Fh | FDh | 11-xxx-xxx | short | meu |
| PADDW mmreg, mem64 | 0Fh | FDh | mm-xxx-xxx | short | mload, meu |
| PAND mmreg1, mmreg2 | 0Fh | DBh | 11-xxx-xxx | short | meu |
| PAND mmreg, mem64 | 0Fh | DBh | mm-xxx-xxx | short | mload, meu |
| PANDN mmreg1, mmreg2 | 0Fh | DFh | 11-xxx-xxx | short | meu |
| PANDN mmreg, mem64 | 0Fh | DFh | mm-xxx-xxx | short | mload, meu |
| PCMPEQB mmreg1, mmreg2 | 0Fh | 74h | 11-xxx-xxx | short | meu |

Table 14. MMX™ Instructions (continued)

| Instruction Mnemonic | Prefix Byte(s) | First Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|------------------------|----------------|------------|-------------|-------------|-------------------|
| PCMPEQB mmreg, mem64 | 0Fh | 74h | mm-xxx-xxx | short | mload, meu |
| PCMPEQD mmreg1, mmreg2 | 0Fh | 76h | 11-xxx-xxx | short | meu |
| PCMPEQD mmreg, mem64 | 0Fh | 76h | mm-xxx-xxx | short | mload, meu |
| PCMPEQW mmreg1, mmreg2 | 0Fh | 75h | 11-xxx-xxx | short | meu |
| PCMPEQW mmreg, mem64 | 0Fh | 75h | mm-xxx-xxx | short | mload, meu |
| PCMPGTB mmreg1, mmreg2 | 0Fh | 64h | 11-xxx-xxx | short | meu |
| PCMPGTB mmreg, mem64 | 0Fh | 64h | mm-xxx-xxx | short | mload, meu |
| PCMPGTD mmreg1, mmreg2 | 0Fh | 66h | 11-xxx-xxx | short | meu |
| PCMPGTD mmreg, mem64 | 0Fh | 66h | mm-xxx-xxx | short | mload, meu |
| PCMPGTW mmreg1, mmreg2 | 0Fh | 65h | 11-xxx-xxx | short | meu |
| PCMPGTW mmreg, mem64 | 0Fh | 65h | mm-xxx-xxx | short | mload, meu |
| PMADDWD mmreg1, mmreg2 | 0Fh | F5h | 11-xxx-xxx | short | meu |
| PMADDWD mmreg, mem64 | 0Fh | F5h | mm-xxx-xxx | short | mload, meu |
| PMULHW mmreg1, mmreg2 | 0Fh | E5h | 11-xxx-xxx | short | meu |
| PMULHW mmreg, mem64 | 0Fh | E5h | mm-xxx-xxx | short | mload, meu |
| PMULLW mmreg1, mmreg2 | 0Fh | D5h | 11-xxx-xxx | short | meu |
| PMULLW mmreg, mem64 | 0Fh | D5h | mm-xxx-xxx | short | mload, meu |
| POR mmreg1, mmreg2 | 0Fh | EBh | 11-xxx-xxx | short | meu |
| POR mmreg, mem64 | 0Fh | EBh | mm-xxx-xxx | short | mload, meu |
| PSLLD mmreg1, mmreg2 | 0Fh | F2h | 11-xxx-xxx | short | meu |
| PSLLD mmreg, mem64 | 0Fh | F2h | mm-xxx-xxx | short | mload, meu |
| PSLLD mmreg, imm8 | 0Fh | 72h | 11-110-xxx | short | meu |
| PSLLQ mmreg1, mmreg2 | 0Fh | F3h | 11-xxx-xxx | short | meu |
| PSLLQ mmreg, mem64 | 0Fh | F3h | mm-xxx-xxx | short | mload, meu |
| PSLLQ mmreg, imm8 | 0Fh | 73h | 11-110-xxx | short | meu |
| PSLLW mmreg1, mmreg2 | 0Fh | F1h | 11-xxx-xxx | short | meu |
| PSLLW mmreg, mem64 | 0Fh | F1h | mm-xxx-xxx | short | mload, meu |
| PSLLW mmreg, imm8 | 0Fh | 71h | 11-110-xxx | short | meu |
| PSRAD mmreg1, mmreg2 | 0Fh | E2h | 11-xxx-xxx | short | meu |
| PSRAD mmreg, mem64 | 0Fh | E2h | mm-xxx-xxx | short | mload, meu |
| PSRAD mmreg, imm8 | 0Fh | 72h | 11-100-xxx | short | meu |
| PSRAW mmreg1, mmreg2 | 0Fh | E1h | 11-xxx-xxx | short | meu |
| PSRAW mmreg, mem64 | 0Fh | E1h | mm-xxx-xxx | short | mload, meu |
| PSRAW mmreg, imm8 | 0Fh | 71h | 11-100-xxx | short | meu |
| PSRLD mmreg1, mmreg2 | 0Fh | D2h | 11-xxx-xxx | short | meu |

Table 14. MMX™ Instructions (continued)

| Instruction Mnemonic | Prefix Byte(s) | First Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|--------------------------|----------------|------------|-------------|-------------|-------------------|
| PSRLD mmreg, mem64 | 0Fh | D2h | mm-xxx-xxx | short | mload, meu |
| PSRLD mmreg, imm8 | 0Fh | 72h | 11-010-xxx | short | meu |
| PSRLQ mmreg1, mmreg2 | 0Fh | D3h | 11-xxx-xxx | short | meu |
| PSRLQ mmreg, mem64 | 0Fh | D3h | mm-xxx-xxx | short | mload, meu |
| PSRLQ mmreg, imm8 | 0Fh | 73h | 11-010-xxx | short | meu |
| PSRLW mmreg1, mmreg2 | 0Fh | D1h | 11-xxx-xxx | short | meu |
| PSRLW mmreg, mem64 | 0Fh | D1h | mm-xxx-xxx | short | mload, meu |
| PSRLW mmreg, imm8 | 0Fh | 71h | 11-010-xxx | short | meu |
| PSUBB mmreg1, mmreg2 | 0Fh | F8h | 11-xxx-xxx | short | meu |
| PSUBB mmreg, mem64 | 0Fh | F8h | mm-xxx-xxx | short | mload, meu |
| PSUBD mmreg1, mmreg2 | 0Fh | FAh | 11-xxx-xxx | short | meu |
| PSUBD mmreg, mem64 | 0Fh | FAh | mm-xxx-xxx | short | mload, meu |
| PSUBSB mmreg1, mmreg2 | 0Fh | E8h | 11-xxx-xxx | short | meu |
| PSUBSB mmreg, mem64 | 0Fh | E8h | mm-xxx-xxx | short | mload, meu |
| PSUBSW mmreg1, mmreg2 | 0Fh | E9h | 11-xxx-xxx | short | meu |
| PSUBSW mmreg, mem64 | 0Fh | E9h | mm-xxx-xxx | short | mload, meu |
| PSUBUSB mmreg1, mmreg2 | 0Fh | D8h | 11-xxx-xxx | short | meu |
| PSUBUSB mmreg, mem64 | 0Fh | D8h | mm-xxx-xxx | short | mload, meu |
| PSUBUSW mmreg1, mmreg2 | 0Fh | D9h | 11-xxx-xxx | short | meu |
| PSUBUSW mmreg, mem64 | 0Fh | D9h | mm-xxx-xxx | short | mload, meu |
| PSUBW mmreg1, mmreg2 | 0Fh | F9h | 11-xxx-xxx | short | meu |
| PSUBW mmreg, mem64 | 0Fh | F9h | mm-xxx-xxx | short | mload, meu |
| PUNPCKHBW mmreg1, mmreg2 | 0Fh | 68h | 11-xxx-xxx | short | meu |
| PUNPCKHBW mmreg, mem64 | 0Fh | 68h | mm-xxx-xxx | short | mload, meu |
| PUNPCKHDQ mmreg1, mmreg2 | 0Fh | 6Ah | 11-xxx-xxx | short | meu |
| PUNPCKHDQ mmreg, mem64 | 0Fh | 6Ah | mm-xxx-xxx | short | mload, meu |
| PUNPCKHWD mmreg1, mmreg2 | 0Fh | 69h | 11-xxx-xxx | short | meu |
| PUNPCKHWD mmreg, mem64 | 0Fh | 69h | mm-xxx-xxx | short | mload, meu |
| PUNPCKLBW mmreg1, mmreg2 | 0Fh | 60h | 11-xxx-xxx | short | meu |
| PUNPCKLBW mmreg, mem32 | 0Fh | 60h | mm-xxx-xxx | short | mload, meu |
| PUNPCKLDQ mmreg1, mmreg2 | 0Fh | 62h | 11-xxx-xxx | short | meu |
| PUNPCKLDQ mmreg, mem32 | 0Fh | 62h | mm-xxx-xxx | short | mload, meu |
| PUNPCKLWD mmreg1, mmreg2 | 0Fh | 61h | 11-xxx-xxx | short | meu |
| PUNPCKLWD mmreg, mem32 | 0Fh | 61h | mm-xxx-xxx | short | mload, meu |
| PXOR mmreg1, mmreg2 | 0Fh | EFh | 11-xxx-xxx | short | meu |

Table 14. MMX™ Instructions (continued)

| Instruction Mnemonic | Prefix Byte(s) | First Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|----------------------|----------------|------------|-------------|-------------|-------------------|
| PXOR mmreg, mem64 | 0Fh | EFh | mm-xxx-xxx | short | mload, meu |

Notes:

1. Bits 2, 1, and 0 of the modR/M byte select the integer register.

Table 15. 3DNow!™ Instructions

| Instruction Mnemonic | Prefix Byte(s) | Opcode Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-------------------------|----------------|-------------|-------------|-------------|-------------------|
| FEMMS | 0Fh | 0Eh | | vector | |
| PAVGUSB mmreg1, mmreg2 | 0Fh, 0Fh | BFh | 11-xxx-xxx | short | meu |
| PAVGUSB mmreg, mem64 | 0Fh, 0Fh | BFh | mm-xxx-xxx | short | mload, meu |
| PF2ID mmreg1, mmreg2 | 0Fh, 0Fh | 1Dh | 11-xxx-xxx | short | meu |
| PF2ID mmreg, mem64 | 0Fh, 0Fh | 1Dh | mm-xxx-xxx | short | mload, meu |
| PFACC mmreg1, mmreg2 | 0Fh, 0Fh | AEh | 11-xxx-xxx | short | meu |
| PFACC mmreg, mem64 | 0Fh, 0Fh | AEh | mm-xxx-xxx | short | mload, meu |
| PFADD mmreg1, mmreg2 | 0Fh, 0Fh | 9Eh | 11-xxx-xxx | short | meu |
| PFADD mmreg, mem64 | 0Fh, 0Fh | 9Eh | mm-xxx-xxx | short | mload, meu |
| PFCMPEQ mmreg1, mmreg2 | 0Fh, 0Fh | B0h | 11-xxx-xxx | short | meu |
| PFCMPEQ mmreg, mem64 | 0Fh, 0Fh | B0h | mm-xxx-xxx | short | mload, meu |
| PFCMPGE mmreg1, mmreg2 | 0Fh, 0Fh | 90h | 11-xxx-xxx | short | meu |
| PFCMPGE mmreg, mem64 | 0Fh, 0Fh | 90h | mm-xxx-xxx | short | mload, meu |
| PFCMPGT mmreg1, mmreg2 | 0Fh, 0Fh | A0h | 11-xxx-xxx | short | meu |
| PFCMPGT mmreg, mem64 | 0Fh, 0Fh | A0h | mm-xxx-xxx | short | mload, meu |
| PFMAX mmreg1, mmreg2 | 0Fh, 0Fh | A4h | 11-xxx-xxx | short | meu |
| PFMAX mmreg, mem64 | 0Fh, 0Fh | A4h | mm-xxx-xxx | short | mload, meu |
| PFMIN mmreg1, mmreg2 | 0Fh, 0Fh | 94h | 11-xxx-xxx | short | meu |
| PFMIN mmreg, mem64 | 0Fh, 0Fh | 94h | mm-xxx-xxx | short | mload, meu |
| PFMUL mmreg1, mmreg2 | 0Fh, 0Fh | B4h | 11-xxx-xxx | short | meu |
| PFMUL mmreg, mem64 | 0Fh, 0Fh | B4h | mm-xxx-xxx | short | mload, meu |
| PFRCP mmreg1, mmreg2 | 0Fh, 0Fh | 96h | 11-xxx-xxx | short | meu |
| PFRCP mmreg, mem64 | 0Fh, 0Fh | 96h | mm-xxx-xxx | short | mload, meu |
| PFRCPIT1 mmreg1, mmreg2 | 0Fh, 0Fh | A6h | 11-xxx-xxx | short | meu |
| PFRCPIT1 mmreg, mem64 | 0Fh, 0Fh | A6h | mm-xxx-xxx | short | mload, meu |
| PFRCPIT2 mmreg1, mmreg2 | 0Fh, 0Fh | B6h | 11-xxx-xxx | short | meu |
| PFRCPIT2 mmreg, mem64 | 0Fh, 0Fh | B6h | mm-xxx-xxx | short | mload, meu |
| PFRSQIT1 mmreg1, mmreg2 | 0Fh, 0Fh | A7h | 11-xxx-xxx | short | meu |

Table 15. 3DNow!™ Instructions (continued)

| Instruction Mnemonic | Prefix Byte(s) | Opcode Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|-------------------------------|----------------|-------------|-------------|-------------|-------------------|
| PFRSQIT1 mmreg, mem64 | 0Fh, 0Fh | A7h | mm-xxx-xxx | short | mload, meu |
| PFRSQRT mmreg1, mmreg2 | 0Fh, 0Fh | 97h | 11-xxx-xxx | short | meu |
| PFRSQRT mmreg, mem64 | 0Fh, 0Fh | 97h | mm-xxx-xxx | short | mload, meu |
| PFSUB mmreg1, mmreg2 | 0Fh, 0Fh | 9Ah | 11-xxx-xxx | short | meu |
| PFSUB mmreg, mem64 | 0Fh, 0Fh | 9Ah | mm-xxx-xxx | short | mload, meu |
| PFSUBR mmreg1, mmreg2 | 0Fh, 0Fh | AAh | 11-xxx-xxx | short | meu |
| PFSUBR mmreg, mem64 | 0Fh, 0Fh | AAh | mm-xxx-xxx | short | mload, meu |
| PI2FD mmreg1, mmreg2 | 0Fh, 0Fh | 0Dh | 11-xxx-xxx | short | meu |
| PI2FD mmreg, mem64 | 0Fh, 0Fh | 0Dh | mm-xxx-xxx | short | mload, meu |
| PMULHRW mmreg1, mmreg2 | 0Fh, 0Fh | B7h | 11-xxx-xxx | short | meu |
| PMULHRW mmreg1, mem64 | 0Fh, 0Fh | B7h | mm-xxx-xxx | short | mload, meu |
| PREFETCH mem8 ¹ | 0Fh | 0Dh | mm-000-xxx | vector | load |
| PREFETCHW mem8 ^{1,2} | 0Fh | 0Dh | mm-001-xxx | vector | load |

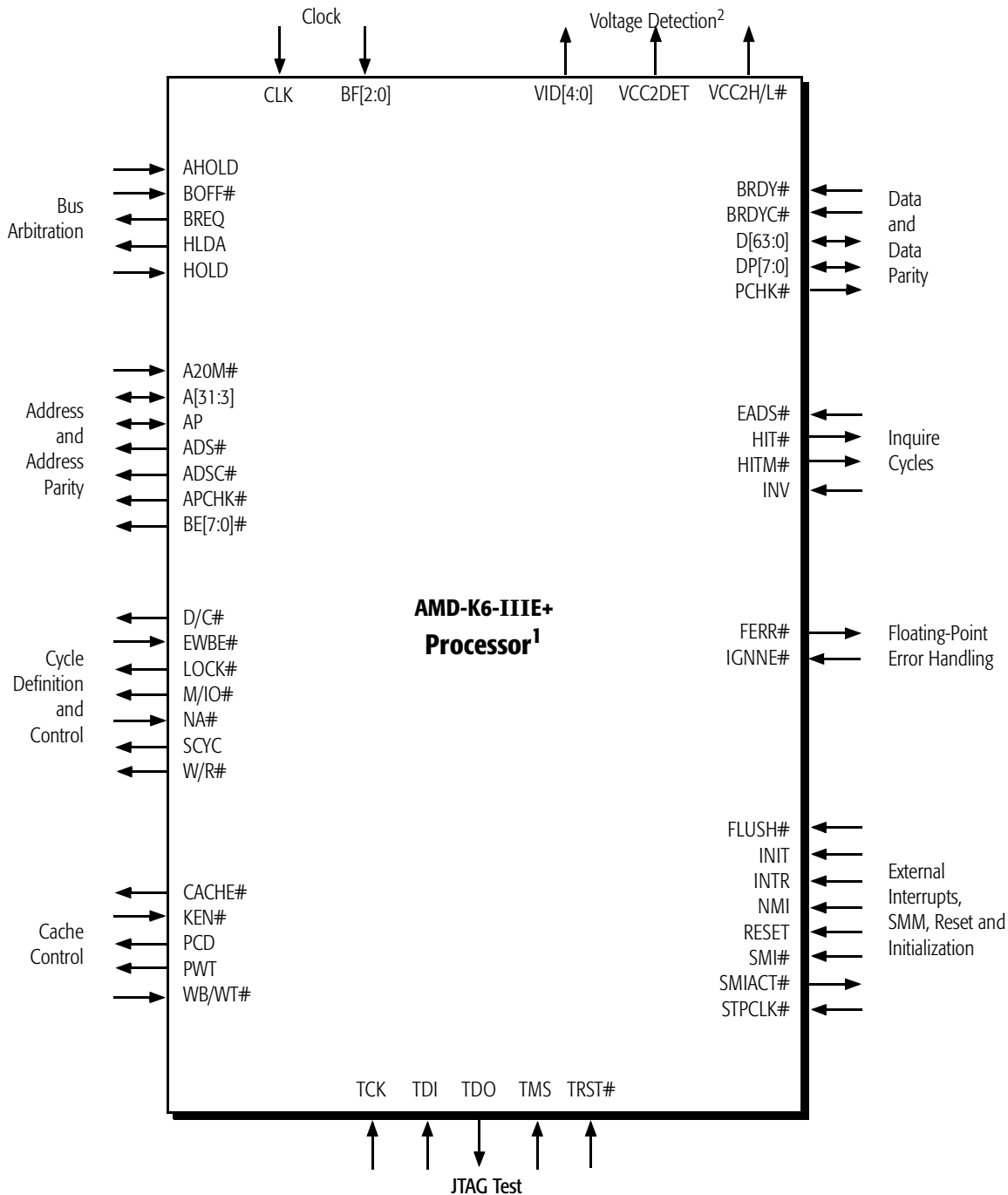
Notes:

1. For PREFETCH and PREFETCHW, the mem8 value refers to a byte address within the 32-byte line that will be prefetched.
2. PREFETCHW will be implemented in a future K86 processor. On the AMD-K6-III+ processor, this instruction performs in the same manner as the PREFETCH instruction.

Table 16. 3DNow!™ Technology DSP Extensions

| Instruction Mnemonic | Prefix Byte(s) | Opcode Byte | ModR/M Byte | Decode Type | RISC86 Operations |
|------------------------|----------------|-------------|-------------|-------------|-------------------|
| PF2IW mmreg1, mmreg2 | 0Fh, 0Fh | 1Ch | 11-xxx-xxx | short | meu |
| PF2IW mmreg, mem64 | 0Fh, 0Fh | 1Ch | mm-xxx-xxx | short | mload, meu |
| PFNACC mmreg1, mmreg2 | 0Fh, 0Fh | 8Ah | 11-xxx-xxx | short | meu |
| PFNACC mmreg, mem64 | 0Fh, 0Fh | 8Ah | mm-xxx-xxx | short | mload, meu |
| PPFNACC mmreg1, mmreg2 | 0Fh, 0Fh | 8Eh | 11-xxx-xxx | short | meu |
| PPFNACC mmreg, mem64 | 0Fh, 0Fh | 8Eh | mm-xxx-xxx | short | mload, meu |
| PI2FW mmreg1, mmreg2 | 0Fh, 0Fh | 0Ch | 11-xxx-xxx | short | meu |
| PI2FW mmreg, mem64 | 0Fh, 0Fh | 0Ch | mm-xxx-xxx | short | mload, meu |
| PSWAPD mmreg1, mmreg2 | 0Fh, 0Fh | BBh | 11-xxx-xxx | short | meu |
| PSWAPD mmreg, mem64 | 0Fh, 0Fh | BBh | mm-xxx-xxx | short | mload, meu |

4 Logic Symbol Diagram



Notes:

1. The signals are grouped by function. The arrows show the direction of the signal, either into or out of the processor. Signals with double-headed arrows are bidirectional. Signals with pound signs (#) are active Low.
2. The VID[4:0] outputs are supported on low-power versions only. The VCC2DET and VCC2H/L# outputs are supported on the CPGA package only.

5 Signal Descriptions

This chapter includes a detailed description of each signal supported on the AMD-K6-III+ processor. This chapter also provides tables listing the signals grouped by type, beginning on page 140.

The logic symbol diagram on page 91 shows the signals grouped by function.

Connection diagrams and pins listed by high-level function are included in Chapter 18, “Pin Designations” on page 323.

5.1 Signal Terminology

The following terminology is used in this chapter:

- *Driven*—The processor actively pulls the signal up to the High-voltage state or pulls the signal down to the Low-voltage state.
- *Floated*—The signal is not being driven by the processor (high-impedance state), which allows another device to drive this signal.
- *Asserted*—For all active High signals, the term *asserted* means the signal is in the High-voltage state. For all active Low signals, the term *asserted* means the signal is in the Low-voltage state. See Table 19 on page 140 for information on asserting signals synchronously and asynchronously.
- *Negated*—For all active High signals, the term *negated* means the signal is in the Low-voltage state. For all active Low signals, the term *negated* means the signal is in the High-voltage state.
- *Sampled*—The processor has measured the state of a signal at predefined points in time and will take the appropriate action based on the state of the signal. If a signal is not sampled by the processor, its assertion or negation has no effect on the operation of the processor.

5.2 A20M# (Address Bit 20 Mask)

Pin Attribute Input

Summary A20M# is used to simulate the behavior of the 8086 when running in Real mode. The assertion of A20M# causes the processor to force bit 20 of the physical address to 0 prior to accessing the caches or driving out a memory bus cycle. The clearing of address bit 20 maps addresses that extend above the 8086 1-Mbyte limit to below 1 Mbyte.

Sampled The processor samples A20M# as a level-sensitive input on every clock edge. The system logic can drive the signal either synchronously or asynchronously. If it is asserted asynchronously, it must be asserted for a minimum pulse width of two clocks.

The following list explains the effects of the processor sampling A20M# asserted under various conditions:

- Inquire cycles and writeback cycles are not affected by the state of A20M#.
- The assertion of A20M# in System Management Mode (SMM) is ignored.
- When A20M# is sampled asserted in Protected mode, it causes unpredictable processor operation. A20M# is only defined in Real mode.
- To ensure that A20M# is recognized before the first ADS# occurs following the negation of RESET, A20M# must be sampled asserted on the same clock edge that RESET is sampled negated or on one of the two subsequent clock edges.
- To ensure A20M# is recognized before the execution of an instruction, a serializing instruction must be executed between the instruction that asserts A20M# and the targeted instruction. (A *serializing instruction* is an instruction inserted between operations to enforce program order. It forces the processor to finish all modifications to flags, registers, and memory before the next instruction is executed.)

5.3 A[31:3] (Address Bus)

Pin Attribute A[31:5] Bidirectional, A[4:3] Output

Summary A[31:3] contain the physical address for the current bus cycle. The processor drives addresses on A[31:3] during memory and I/O cycles, and cycle definition information during special bus cycles. The processor samples addresses on A[31:5] during inquire cycles.

Driven, Sampled, and Floated *As Outputs:* A[31:3] are driven valid off the same clock edge as ADS# and remain in the same state until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted. A[31:3] are driven during memory cycles, I/O cycles, special bus cycles, and interrupt acknowledge cycles. The processor continues to drive the address bus while the bus is idle.

As Inputs: The processor samples A[31:5] during inquire cycles on the clock edge on which EADS# is sampled asserted. Even though A4 and A3 are not used during the inquire cycle, they must be driven to a valid state and must meet the same timings as A[31:5].

A[31:3] are floated off the clock edge that AHOLD or BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in recognition of HOLD.

The processor resumes driving A[31:3] off the clock edge on which the processor samples AHOLD or BOFF# negated and off the clock edge on which the processor negates HLDA.

5.4 ADS# (Address Strobe)

Pin Attribute Output

Summary The assertion of ADS# indicates the beginning of a new bus cycle. The address bus and all cycle definition signals corresponding to this bus cycle are driven valid off the same clock edge as ADS#.

Driven and Floated ADS# is asserted for one clock at the beginning of each bus cycle. For non-pipelined cycles, ADS# can be asserted as early as the clock edge after the clock edge on which the last expected BRDY# of the cycle is sampled asserted, resulting in a single idle state between cycles. For pipelined cycles if the processor is prepared to start a new cycle, ADS# can be asserted as early as one clock edge after NA# is sampled asserted.

If AHOLD is sampled asserted, ADS# is only driven in order to perform a writeback cycle due to an inquire cycle that hits a modified cache line.

The processor floats ADS# off the clock edge that BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in recognition of HOLD.

5.5 ADSC# (Address Strobe Copy)

Pin Attribute Output

Summary ADSC# has the identical function and timing as ADS#. In the event ADS# becomes too heavily loaded due to a large fanout in a system, ADSC# can be used to split the load across two outputs, which can improve system timing.

5.6 AHOLD (Address Hold)

Pin Attribute Input

Summary AHOLD can be asserted by the system to initiate one or more inquire cycles. To allow the system to drive the address bus during an inquire cycle, the processor floats A[31:3] and AP off the clock edge on which AHOLD is sampled asserted. The data bus and all other control and status signals remain under the control of the processor and are not floated. This allows a bus cycle that is in progress when AHOLD is sampled asserted to continue to completion. The processor resumes driving the address bus off the clock edge on which AHOLD is sampled negated.

If AHOLD is sampled asserted, ADS# is only asserted in order to perform a writeback cycle due to an inquire cycle that hits a modified cache line.

Sampled The processor samples AHOLD on every clock edge. AHOLD is recognized while INIT and RESET are sampled asserted.

5.7 AP (Address Parity)

Pin Attribute Bidirectional

Summary AP contains the even parity bit for cache line addresses driven and sampled on A[31:5]. Even parity means that the total number of 1 bits on AP and A[31:5] is even. (A4 and A3 are not used for the generation or checking of address parity because these bits are not required to address a cache line.) AP is driven by the processor during processor-initiated cycles and is sampled by the processor during inquire cycles. If AP does not reflect even parity during an inquire cycle, the processor asserts APCHK# to indicate an address bus parity check. The processor does not take an internal exception as the result of detecting an address bus parity check, and system logic must respond appropriately to the assertion of this signal.

Driven, Sampled, and Floated *As an Output:* The processor drives AP valid off the clock edge on which ADS# is asserted until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted. AP is driven during memory cycles, I/O cycles, special bus cycles, and interrupt acknowledge cycles. The processor continues to drive AP while the bus is idle.

As an Input: The processor samples AP during inquire cycles on the clock edge on which EADS# is sampled asserted.

The processor floats AP off the clock edge that AHOLD or BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in recognition of HOLD.

The processor resumes driving AP off the clock edge on which the processor samples AHOLD or BOFF# negated and off the clock edge on which the processor negates HLDA.

5.8 APCHK# (Address Parity Check)

Pin Attribute Output

Summary If the processor detects an address parity error during an inquire cycle, APCHK# is asserted for one clock. The processor does not take an internal exception as the result of detecting an address bus parity check, and system logic must respond appropriately to the assertion of this signal.

The processor is designed so that APCHK# does not glitch, enabling the signal to be used as a clocking source for system logic.

Driven APCHK# is driven valid off the clock edge after the clock edge on which the processor samples EADS# asserted. It is negated off the next clock edge.

APCHK# is always driven except in the Three-State Test mode.

5.9 BE[7:0]# (Byte Enables)

Pin Attribute

Output

Summary

BE[7:0]# are used by the processor to indicate the valid data bytes during a write cycle and the requested data bytes during a read cycle. The byte enables can be used to derive address bits A[2:0], which are not physically part of the processor's address bus. The processor checks and generates valid data parity for the data bytes that are valid as defined by the byte enables. The eight byte enables correspond to the eight bytes of the data bus as follows:

- BE7#: D[63:56]
- BE6#: D[55:48]
- BE5#: D[47:40]
- BE4#: D[39:32]
- BE3#: D[31:24]
- BE2#: D[23:16]
- BE1#: D[15:8]
- BE0#: D[7:0]

The processor expects data to be driven by the system logic on all eight bytes of the data bus during a burst cache-line read cycle, independent of the byte enables that are asserted.

The byte enables are also used to distinguish between special bus cycles as defined in Table 24 on page 142.

Driven and Floated

BE[7:0]# are driven off the same clock edge as ADS# and remain in the same state until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted. BE[7:0]# are driven during memory cycles, I/O cycles, special bus cycles, and interrupt acknowledge cycles.

The processor floats BE[7:0]# off the clock edge that BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in recognition of HOLD. Unlike the address bus, BE[7:0]# are not floated in response to AHOLD.

5.10 BF[2:0] (Bus Frequency)

Pin Attribute Inputs, Internal Pullups

Summary BF[2:0] determine the internal operating frequency of the processor. The frequency of the CLK input signal is multiplied internally by a ratio determined by the state of these signals as defined in Table 17. BF[2:0] have weak internal pullups and default to the 3.5 multiplier if left unconnected.

Table 17. Processor-to-Bus Clock Ratios

| State of BF[2:0] Inputs | Processor-Clock to Bus-Clock Ratio |
|-------------------------|------------------------------------|
| 100b | 2.0x |
| 101b | 3.0x |
| 110b | 6.0x |
| 111b | 3.5x |
| 000b | 4.5x |
| 001b | 5.0x |
| 010b | 4.0x |
| 011b | 5.5x |

Sampled BF[2:0] are sampled during the falling transition of RESET. They must meet a minimum setup time of 1.0 ms and a minimum hold time of two clocks relative to the negation of RESET.

5.11 BOFF# (Backoff)

Pin Attribute

Input

Summary

If BOFF# is sampled asserted, the processor unconditionally aborts any cycles in progress and transitions to a bus hold state by floating the following signals: A[31:3], ADS#, ADSC#, AP, BE[7:0]#, CACHE#, D[63:0], D/C#, DP[7:0], LOCK#, M/IO#, PCD, PWT, SCYC, and W/R#. These signals remain floated until BOFF# is sampled negated. This allows an alternate bus master or the system to control the bus.

When BOFF# is sampled negated, any processor cycle that was aborted due to the assertion of BOFF# is restarted from the beginning of the cycle, regardless of the number of transfers that were completed. If BOFF# is sampled asserted on the same clock edge as BRDY# of a bus cycle of any length, then BOFF# takes precedence over the BRDY#. In this case, the cycle is aborted and restarted after BOFF# is sampled negated.

Sampled

BOFF# is sampled on every clock edge. The processor floats its bus signals off the clock edge on which BOFF# is sampled asserted. These signals remain floated until the clock edge on which BOFF# is sampled negated.

BOFF# is recognized while INIT and RESET are sampled asserted.

5.12 BRDY# (Burst Ready)

Pin Attribute

Input, Internal Pullup

Summary

BRDY# is asserted to the processor by system logic to indicate either that the data bus is being driven with valid data during a read cycle or that the data bus has been latched during a write cycle. If necessary, the system logic can insert bus cycle wait states by negating BRDY# until it is ready to continue the data transfer. BRDY# is also used to indicate the completion of special bus cycles.

Sampled

BRDY# is sampled every clock edge within a bus cycle starting with the clock edge after the clock edge that negates ADS#. BRDY# is ignored while the bus is idle. The processor samples the following inputs on the clock edge on which BRDY# is sampled asserted: D[63:0], DP[7:0], and KEN# during read cycles, EWBE# during write cycles (if not masked off), and WB/WT# during read and write cycles. If NA# is sampled asserted prior to BRDY#, then KEN# and WB/WT# are sampled on the clock edge on which NA# is sampled asserted.

The number of times the processor expects to sample BRDY# asserted depends on the type of bus cycle, as follows:

- One time for a single-transfer cycle, a special bus cycle, or each of two cycles in an interrupt acknowledge sequence
- Four times for a burst cycle (once for each data transfer)

BRDY# can be held asserted for four consecutive clocks throughout the four transfers of the burst, or it can be negated to insert wait states.

5.13 BRDYC# (Burst Ready Copy)

Pin Attribute Input, Internal Pullup

Summary BRDYC# has the identical function as BRDY#. In the event BRDY# becomes too heavily loaded due to a large fanout or loading in a system, BRDYC# can be used to reduce this loading, which improves timing.

Sampled BRDYC# is sampled every clock edge within a bus cycle starting with the clock edge after the clock edge that negates ADS#.

5.14 BREQ (Bus Request)

Pin Attribute Output

Summary BREQ is asserted by the processor to request the bus in order to complete an internally pending bus cycle. The system logic can use BREQ to arbitrate among the bus participants. If the processor does not own the bus, BREQ is asserted until the processor gains access to the bus in order to begin the pending cycle or until the processor no longer needs to run the pending cycle. If the processor currently owns the bus, BREQ is asserted with ADS#. The processor asserts BREQ for each assertion of ADS# but does not necessarily assert ADS# for each assertion of BREQ.

Driven BREQ is asserted off the same clock edge on which ADS# is asserted. BREQ can also be asserted off any clock edge, independent of the assertion of ADS#. BREQ can be negated one clock edge after it is asserted.

The processor always drives BREQ except in the Three-State Test mode.

5.15 CACHE# (Cacheable Access)

Pin Attribute Output

Summary For reads, CACHE# is asserted to indicate the cacheability of the current bus cycle. In addition, if the processor samples KEN# asserted, which indicates the driven address is cacheable, the cycle is a 32-byte burst read cycle. For write cycles, CACHE# is asserted to indicate the current bus cycle is a modified cache-line writeback. KEN# is ignored during writebacks. If CACHE# is not asserted, or if KEN# is sampled negated during a read cycle, the cycle is not cacheable and defaults to a single-transfer cycle.

Driven and Floated CACHE# is driven off the same clock edge as ADS# and remains in the same state until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted.

CACHE# is floated off the clock edge that BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in recognition of HOLD.

5.16 CLK (Clock)

Pin Attribute Input

Summary The CLK signal is the bus clock for the processor and is the reference for all signal timings under normal operation (except for TDI, TDO, TMS, and TRST#). BF[2:0] determine the internal frequency multiplier applied to CLK to obtain the processor's core operating frequency. See "BF[2:0] (Bus Frequency)" on page 101 for a list of the processor-to-bus clock ratios.

Sampled The CLK signal must be stable a minimum of 1.0 ms prior to the negation of RESET to ensure the proper operation of the processor. See "CLK Switching Characteristics" on page 298 for details regarding the CLK specifications.

5.17 D/C# (Data/Code)

Pin Attribute Output

Summary The processor drives D/C# during a memory bus cycle to indicate whether it is addressing data or executable code. D/C# is also used to define other bus cycles, including interrupt acknowledge and special cycles. See Table 23 and Table 24 on page 142 for more details.

Driven and Floated D/C# is driven off the same clock edge as ADS# and remains in the same state until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted. D/C# is driven during memory cycles, I/O cycles, special bus cycles, and interrupt acknowledge cycles.

D/C# is floated off the clock edge that BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in recognition of HOLD.

5.18 D[63:0] (Data Bus)

Pin Attribute Bidirectional

Summary D[63:0] represent the processor's 64-bit data bus. Each of the eight bytes of data that comprise this bus is qualified as valid by its corresponding byte enable. See "BE[7:0]# (Byte Enables)" on page 100.

Driven, Sampled, and Floated

As Outputs: For single-transfer write cycles, the processor drives D[63:0] with valid data one clock edge after the clock edge on which ADS# is asserted and D[63:0] remain in the same state until the clock edge on which BRDY# is sampled asserted. If the cycle is a writeback—in which case four 8-byte transfers occur—D[63:0] are driven one clock edge after the clock edge on which ADS# is asserted and are subsequently changed off the clock edge on which each BRDY# assertion of the burst cycle is sampled.

If the assertion of ADS# represents a pipelined write cycle that follows a read cycle, the processor does not drive D[63:0] until it is certain that contention on the data bus will not occur. In this case, D[63:0] are driven the clock edge after the last expected BRDY# of the previous cycle is sampled asserted.

As Inputs: During read cycles, the processor samples D[63:0] on the clock edge on which BRDY# is sampled asserted.

The processor always floats D[63:0] except when they are being driven during a write cycle as described above. In addition, D[63:0] are floated off the clock edge that BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in recognition of HOLD.

5.19 DP[7:0] (Data Parity)

Pin Attribute Bidirectional

Summary DP[7:0] are even parity bits for each valid byte of data—as defined by BE[7:0]#—driven and sampled on the D[63:0] data bus. *Even parity* means that the total number of 1 bits within each byte of data and its respective data parity bit is an even number. DP[7:0] are driven by the processor during write cycles and sampled by the processor during read cycles.

If the processor detects bad parity on any valid byte of data during a read cycle, PCHK# is asserted for one clock beginning the clock edge after BRDY# is sampled asserted. The processor does not take an internal exception as the result of detecting a data parity check, and system logic must respond appropriately to the assertion of this signal.

The eight data parity bits correspond to the eight bytes of the data bus as follows:

- DP7: D[63:56]
- DP6: D[55:48]
- DP5: D[47:40]
- DP4: D[39:32]
- DP3: D[31:24]
- DP2: D[23:16]
- DP1: D[15:8]
- DP0: D[7:0]

For systems that do not support data parity, DP[7:0] should be connected to V_{CC3} through pullup resistors.

Driven, Sampled, and Floated

As Outputs: For single-transfer write cycles, the processor drives DP[7:0] with valid parity one clock edge after the clock edge on which ADS# is asserted and DP[7:0] remain in the same state until the clock edge on which BRDY# is sampled asserted. If the cycle is a writeback, DP[7:0] are driven one clock edge after the clock edge on which ADS# is asserted and are subsequently changed off the clock edge on which each BRDY# assertion of the burst cycle is sampled.

As Inputs: During read cycles, the processor samples DP[7:0] on the clock edge BRDY# is sampled asserted.

The processor always floats DP[7:0] except when they are being driven during a write cycle as described above. In addition, DP[7:0] are floated off the clock edge that BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in recognition of HOLD.

5.20 EADS# (External Address Strobe)

Pin Attribute Input

Summary System logic asserts EADS# during a cache inquire cycle to indicate that the address bus contains a valid address. EADS# can only be driven after the system logic has taken control of the address bus by asserting AHOLD or BOFF# or by receiving HLDA. The processor responds to the sampling of EADS# and the address bus by driving HIT#, which indicates if the inquired cache line exists in the processor's caches, and HITM#, which indicates if it is in the modified state.

Sampled If AHOLD or BOFF# is asserted by the system logic in order to execute a cache inquire cycle, the processor begins sampling EADS# two clock edges after AHOLD or BOFF# is sampled asserted. If the system logic asserts HOLD in order to execute a cache inquire cycle, the processor begins sampling EADS# two clock edges after the clock edge HLDA is asserted by the processor.

EADS# is ignored during the following conditions:

- One clock edge after the clock edge on which EADS# is sampled asserted
- Two clock edges after the clock edge on which ADS# is asserted
- When the processor is driving the address bus
- When the processor asserts HITM#

5.21 EWBE# (External Write Buffer Empty)

Pin Attribute

Input

Summary

The system logic can negate EWBE# to the processor to indicate that its external write buffers are full and that additional data cannot be stored at this time. This causes the processor to delay the following activities until EWBE# is sampled asserted:

- The commitment of write hit cycles to cache lines in the modified state or exclusive state in the processor's caches
- The decode and execution of an instruction that follows a currently-executing serializing instruction
- The assertion or negation of SMIACK#
- The entering of the Halt state and the Stop Grant state

Negating EWBE# does not prevent the completion of any type of cycle that is currently in progress.

Sampled

The processor samples EWBE# on each clock edge that BRDY# is sampled asserted during all memory write cycles (except writeback cycles), I/O write cycles, and special bus cycles.

If EWBE# is sampled negated, it is sampled on every clock edge until it is asserted, and then it is ignored until BRDY# is sampled asserted in the next write cycle or special cycle.

If EFER[3] is set to 1, then EWBE# is ignored by the processor. For more information on the EFER settings and EWBE#, see "EWBE# Control" on page 229.

5.22 FERR# (Floating-Point Error)

Pin Attribute Output

Summary The assertion of FERR# indicates the occurrence of an unmasked floating-point exception resulting from the execution of a floating-point instruction. This signal is provided to allow the system logic to handle this exception in a manner consistent with IBM-compatible PC/AT systems. See “Handling Floating-Point Exceptions” on page 237 for a system logic implementation that supports floating-point exceptions.

The state of the numeric error (NE) bit in CR0 does not affect the FERR# signal.

The processor is designed so that FERR# does not glitch, enabling the signal to be used as a clocking source for system logic.

Driven The processor asserts FERR# on the instruction boundary of the next floating-point instruction, MMX instruction, 3DNow! instruction, or WAIT instruction that occurs following the floating-point instruction that caused the unmasked floating-point exception—that is, FERR# is not asserted at the time the exception occurs. The IGNNE# signal does not affect the assertion of FERR#.

FERR# is negated during the following conditions:

- Following the successful execution of the floating-point instructions FCLEX, FINIT, FSAVE, and FSTENV
- Under certain circumstances, following the successful execution of the floating-point instructions FLDCW, FLDENV, and FRSTOR, which load the floating-point status word or the floating-point control word
- Following the falling transition of RESET

FERR# is always driven except in the Three-State Test mode.

See “IGNNE# (Ignore Numeric Exception)” on page 116 for more details on floating-point exceptions.

5.23 FLUSH# (Cache Flush)

Pin Attribute Input

Summary In response to sampling FLUSH# asserted, the processor writes back any cache lines in the L1 data cache or L2 cache that are in the modified state, invalidates all lines in the L1 and L2 caches, and then executes a flush acknowledge special cycle. See Table 24 on page 142 for the bus definition of special cycles.

In addition, FLUSH# is sampled when RESET is negated to determine if the processor enters the Three-State Test mode. If FLUSH# is 0 during the falling transition of RESET, the processor enters the Three-State Test mode instead of performing the normal RESET functions.

Sampled FLUSH# is sampled and latched as a falling edge-sensitive signal. During normal operation (not RESET), FLUSH# is sampled on every clock edge but is not recognized until the next instruction boundary.

- If FLUSH# is asserted synchronously (see Table 19 on page 140), it can be asserted for a minimum of one clock.
- If FLUSH# is asserted asynchronously, it must have been negated for a minimum of two clocks, followed by an assertion of a minimum of two clocks.

FLUSH# is also sampled during the falling transition of RESET. If RESET and FLUSH# are driven synchronously, FLUSH# is sampled on the clock edge prior to the clock edge on which RESET is sampled negated. If RESET is driven asynchronously, the minimum setup and hold time for FLUSH#, relative to the negation of RESET, is two clocks.

5.24 HIT# (Inquire Cycle Hit)

Pin Attribute Output

Summary The processor asserts HIT# during an inquire cycle to indicate that the cache line is valid within the processor's L1 and/or L2 caches (also known as a cache hit). The cache line can be in the modified, exclusive, or shared state.

Driven HIT# is always driven—except in the Three-State Test mode—and only changes state the clock edge after the clock edge on which EADS# is sampled asserted. It is driven in the same state until the next inquire cycle.

5.25 HITM# (Inquire Cycle Hit To Modified Line)

Pin Attribute Output

Summary The processor asserts HITM# during an inquire cycle to indicate that the cache line exists in the processor's L1 data cache or L2 cache in the modified state. The processor performs a writeback cycle as a result of this cache hit. If an inquire cycle hits a cache line that is currently being written back, the processor asserts HITM# but does not execute another writeback cycle. The system logic must not expect the processor to assert ADS# each time HITM# is asserted.

Driven HITM# is always driven—except in the Three-State Test mode—and, in particular, is driven to represent the result of an inquire cycle the clock edge after the clock edge on which EADS# is sampled asserted. If HITM# is negated in response to the inquire address, it remains negated until the next inquire cycle. If HITM# is asserted in response to the inquire address, it remains asserted throughout the writeback cycle and is negated one clock edge after the last BRDY# of the writeback is sampled asserted.

5.26 HLDA (Hold Acknowledge)

Pin Attribute Output

Summary When HOLD is sampled asserted, the processor completes the current bus cycles, floats the processor bus, and asserts HLDA in an acknowledgment that these events have been completed. The processor does not assert HLDA until the completion of a locked sequence of cycles. While HLDA is asserted, another bus master can drive cycles on the bus, including inquire cycles to the processor. The following signals are floated when HLDA is asserted: A[31:3], ADS#, ADSC#, AP, BE[7:0]#, CACHE#, D[63:0], D/C#, DP[7:0], LOCK#, M/IO#, PCD, PWT, SCYC, and W/R#.

The processor is designed so that HLDA does not glitch.

Driven HLDA is always driven except in the Three-State Test mode. If a processor cycle is in progress while HOLD is sampled asserted, HLDA is asserted one clock edge after the last BRDY# of the cycle is sampled asserted. If the bus is idle, HLDA is asserted one clock edge after HOLD is sampled asserted. HLDA is negated one clock edge after the clock edge on which HOLD is sampled negated.

The assertion of HLDA is independent of the sampled state of BOFF#.

The processor floats the bus every clock in which HLDA is asserted.

5.27 HOLD (Bus Hold Request)

Pin Attribute Input

Summary The system logic can assert HOLD to gain control of the processor's bus. When HOLD is sampled asserted, the processor completes the current bus cycles, floats the processor bus, and asserts HLDA in an acknowledgment that these events have been completed.

Sampled The processor samples HOLD on every clock edge. If a processor cycle is in progress while HOLD is sampled asserted, HLDA is asserted one clock edge after the last BRDY# of the cycle is sampled asserted. If the bus is idle, HLDA is asserted one clock edge after HOLD is sampled asserted. HOLD is recognized while INIT and RESET are sampled asserted.

5.28 IGNNE# (Ignore Numeric Exception)

Pin Attribute Input

Summary

IGNNE#, in conjunction with the numeric error (NE) bit in the CR0 register, is used by the system logic to control the effect of an unmasked floating-point exception on a previous floating-point instruction during the execution of a floating-point instruction, MMX instruction, 3DNow! instruction, or the WAIT instruction—hereafter referred to as the target instruction.

If an unmasked floating-point exception is pending and the target instruction is considered error-sensitive, then the relationship between NE and IGNNE# is as follows:

- If NE = 0, then:
 - If IGNNE# is sampled asserted, the processor ignores the floating-point exception and continues with the execution of the target instruction.
 - If IGNNE# is sampled negated, the processor waits until it samples IGNNE#, INTR, SMI#, NMI, or INIT asserted.
 - If IGNNE# is sampled asserted while waiting, the processor ignores the floating-point exception and continues with the execution of the target instruction.
 - If INTR, SMI#, NMI, or INIT is sampled asserted while waiting, the processor handles its assertion appropriately.
- If NE = 1, the processor invokes the INT 10h exception handler.

If an unmasked floating-point exception is pending and the target instruction is considered error-insensitive, then the processor ignores the floating-point exception and continues with the execution of the target instruction.

FERR# is not affected by the state of the NE bit or IGNNE#. FERR# is always asserted at the instruction boundary of the target instruction that follows the floating-point instruction that caused the unmasked floating-point exception.

This signal is provided to allow the system logic to handle exceptions in a manner consistent with IBM-compatible PC/AT systems.

Sampled

The processor samples IGNNE# as a level-sensitive input on every clock edge. The system logic can drive the signal either synchronously or asynchronously. If it is asserted asynchronously, it must be asserted for a minimum pulse width of two clocks.

5.29 INIT (Initialization)

Pin Attribute

Input

Summary

The assertion of INIT causes the processor to empty its pipelines, to initialize most of its internal state, and to branch to address FFFF_FFF0h—the same instruction execution starting point used after RESET. Unlike RESET, the processor preserves the contents of its caches, the floating-point state, the MMX state, Model-Specific Registers, the CD and NW bits of the CR0 register, and other specific internal resources.

INIT can be used as an accelerator for 80286 code that requires a reset to exit from Protected mode back to Real mode.

Sampled

INIT is sampled and latched as a rising edge-sensitive signal. INIT is sampled on every clock edge but is not recognized until the next instruction boundary. During an I/O write cycle, it must be sampled asserted a minimum of three clock edges before BRDY# is sampled asserted if it is to be recognized on the boundary between the I/O write instruction and the following instruction.

- If INIT is asserted synchronously (see Table 19 on page 140), it can be asserted for a minimum of one clock.
- If it is asserted asynchronously, it must have been negated for a minimum of two clocks, followed by an assertion of a minimum of two clocks.

5.30 INTR (Maskable Interrupt)

Pin Attribute Input

Summary INTR is the system's maskable interrupt input to the processor. When the processor samples and recognizes INTR asserted, the processor executes a pair of interrupt acknowledge bus cycles and then jumps to the interrupt service routine specified by the interrupt number that was returned during the interrupt acknowledge sequence. The processor only recognizes INTR if the interrupt flag (IF) in the EFLAGS register equals 1.

Sampled The processor samples INTR as a level-sensitive input on every clock edge, but the interrupt request is not recognized until the next instruction boundary. The system logic can drive INTR either synchronously or asynchronously. If it is asserted asynchronously, it must be asserted for a minimum pulse width of two clocks. In order to be recognized, INTR must remain asserted until an interrupt acknowledge sequence is complete.

5.31 INV (Invalidation Request)

Pin Attribute Input

Summary During an inquire cycle, the state of INV determines whether an addressed cache line that is found in the processor's L1 and/or L2 caches transitions to the invalid state or the shared state.

If INV is sampled asserted during an inquire cycle, the processor transitions the cache line (if found) to the invalid state, regardless of its previous state. If INV is sampled negated during an inquire cycle, the processor transitions the cache line (if found) to the shared state. In either case, if the cache line is found in the modified state, the processor writes it back to memory before changing its state.

Sampled INV is sampled on the clock edge on which EADS# is sampled asserted.

5.32 KEN# (Cache Enable)

Pin Attribute Input

Summary

If KEN# is sampled asserted, it indicates that the address presented by the processor is cacheable. If KEN# is sampled asserted and the processor intends to perform a cache-line fill (signified by the assertion of CACHE#), the processor executes a 32-byte burst read cycle and expects to sample BRDY# asserted a total of four times. If KEN# is sampled negated during a read cycle, a single-transfer cycle is executed and the processor does not cache the data. For write cycles, CACHE# is asserted to indicate the current bus cycle is a modified cache-line writeback. KEN# is ignored during writebacks.

If PCD is asserted during a bus cycle, the processor does not cache any data read during that cycle, regardless of the state of KEN#. See “PCD (Page Cache Disable)” on page 124 for more details.

If the processor has sampled the state of KEN# during a cycle, and that cycle is aborted due to the sampling of BOFF# asserted, the system logic must ensure that KEN# is sampled in the same state when the processor restarts the aborted cycle.

Sampled

KEN# is sampled on the clock edge on which the first BRDY# or NA# of a read cycle is sampled asserted. If the read cycle is a burst, KEN# is ignored during the last three assertions of BRDY#. KEN# is sampled during read cycles only when CACHE# is asserted.

5.33 LOCK# (Bus Lock)

Pin Attribute Output

Summary

The processor asserts LOCK# during a sequence of bus cycles to ensure that the cycles are completed without allowing other bus masters to intervene. Locked operations consist of two to five bus cycles. LOCK# is asserted during the following operations:

- An interrupt acknowledge sequence
- Descriptor Table accesses
- Page Directory and Page Table accesses
- XCHG instruction
- An instruction with an allowable LOCK prefix

In order to ensure that locked operations appear on the bus and are visible to the entire system, any data operands addressed during a locked cycle that reside in the processor's caches are flushed and invalidated from the caches prior to the locked operation. If the cache line is in the modified state, it is written back and invalidated prior to the locked operation. Likewise, any data read during a locked operation is not cached.

The processor is designed so that LOCK# does not glitch.

Driven and Floated

During a locked cycle, LOCK# is asserted off the same clock edge on which ADS# is asserted and remains asserted until the last BRDY# of the last bus cycle is sampled asserted. The processor negates LOCK# for at least one clock between consecutive sequences of locked operations to allow the system logic to arbitrate for the bus.

LOCK# is floated off the clock edge on which BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in response to HOLD. When LOCK# is floated due to BOFF# sampled asserted, the system logic is responsible for preserving the lock condition while LOCK# is in the high-impedance state.

5.34 M/IO# (Memory or I/O)

Pin Attribute Output

Summary The processor drives M/IO# during a bus cycle to indicate whether it is addressing the memory or I/O space. If M/IO# = 1, the processor is addressing memory or a memory-mapped I/O port as the result of an instruction fetch or an instruction that loads or stores data. If M/IO# = 0, the processor is addressing an I/O port during the execution of an I/O instruction. In addition, M/IO# is used to define other bus cycles, including interrupt acknowledge and special cycles. See Table 23 and Table 24 on page 142 for more details.

Driven and Floated M/IO# is driven off the same clock edge as ADS# and remains in the same state until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted. M/IO# is driven during memory cycles, I/O cycles, special bus cycles, and interrupt acknowledge cycles.

M/IO# is floated off the clock edge on which BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in response to HOLD.

5.35 NA# (Next Address)

Pin Attribute

Input

Summary

System logic asserts NA# to indicate to the processor that it is ready to accept another bus cycle pipelined into the previous bus cycle. ADS#, along with address and status signals, can be asserted as early as one clock edge after NA# is sampled asserted if the processor is prepared to start a new cycle. Because the processor allows a maximum of two cycles to be in progress at a time, the assertion of NA# is sampled while two cycles are in progress, but ADS# is not asserted until the completion of the first cycle.

Sampled

NA# is sampled every clock edge during bus cycles, starting one clock edge after the clock edge that negates ADS#, until the last expected BRDY# of the last executed cycle is sampled asserted (with the exception of the clock edge after the clock edge that negates the ADS# for a second pending cycle). Because the processor latches NA# when sampled, the system logic only needs to assert NA# for one clock.

5.36 NMI (Non-Maskable Interrupt)

Pin Attribute Input

Summary When NMI is sampled asserted, the processor jumps to the interrupt service routine defined by interrupt number 02h. Unlike the INTR signal, software cannot mask the effect of NMI if it is sampled asserted by the processor. However, NMI is temporarily masked upon entering System Management Mode (SMM). In addition, an interrupt acknowledge cycle is not executed because the interrupt number is predefined.

If NMI is sampled asserted while the processor is executing the interrupt service routine for a previous NMI, the subsequent NMI remains pending until the completion of the execution of the IRET instruction at the end of the interrupt service routine.

Sampled NMI is sampled and latched as a rising edge-sensitive signal. During normal operation, NMI is sampled on every clock edge but is not recognized until the next instruction boundary.

- If NMI is asserted synchronously (see Table 19 on page 140), it can be asserted for a minimum of one clock.
- If NMI is asserted asynchronously, it must have been negated for a minimum of two clocks, followed by an assertion of a minimum of two clocks.

5.37 PCD (Page Cache Disable)

Pin Attribute Output

Summary

The processor drives PCD to indicate the operating system's specification of cacheability for the page being addressed. System logic can use PCD to control external caching. If PCD is asserted, the addressed page is not cached. If PCD is negated, the cacheability of the addressed page depends upon the state of CACHE# and KEN#.

The state of PCD depends upon the processor's operating mode and the state of certain bits in its control registers and TLB as follows:

- In Real mode, or in Protected and Virtual-8086 modes while paging is disabled (PG bit in CR0 set to 0):

PCD output = CD bit in CR0

- In Protected and Virtual-8086 modes while caching is enabled (CD bit in CR0 set to 0) and paging is enabled (PG bit in CR0 set to 1):

- For accesses to I/O space, page directory entries, and other non-paged accesses:

PCD output = PCD bit in CR3

- For accesses to 4-Kbyte page table entries or 4-Mbyte pages:

PCD output = PCD bit in page directory entry

- For accesses to 4-Kbyte pages:

PCD output = PCD bit in page table entry

Driven and Floated

PCD is driven off the same clock edge as ADS# and remains in the same state until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted.

PCD is floated off the clock edge that BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in response to HOLD.

5.38 PCHK# (Parity Check)

Pin Attribute Output

Summary The processor asserts PCHK# during read cycles if it detects an even parity error on one or more valid bytes of D[63:0] during a read cycle. (*Even parity* means that the total number of 1 bits within each byte of data and its respective data parity bit is even.) The processor checks data parity for the data bytes that are valid, as defined by BE[7:0]#, the byte enables.

PCHK# is always driven but is only asserted for memory and I/O read bus cycles and the second cycle of an interrupt acknowledge sequence. PCHK# is not driven during any type of write cycles or special bus cycles. The processor does not take an internal exception as the result of detecting a data parity error, and system logic must respond appropriately to the assertion of this signal.

The processor is designed so that PCHK# does not glitch, enabling the signal to be used as a clocking source for system logic.

Driven PCHK# is always driven except in the Three-State Test mode. For each BRDY# returned to the processor during a read cycle with a parity error detected on the data bus, PCHK# is asserted for one clock, one clock edge after BRDY# is sampled asserted.

5.39 PWT (Page Writethrough)

Pin Attribute Output

Summary

The processor drives PWT to indicate the operating system's specification of the writeback state or writethrough state for the page being addressed. PWT, together with WB/WT#, specifies the data cache-line state during cacheable read misses and write hits to shared cache lines. See "WB/WT# (Writeback or Writethrough)" on page 139 for more details.

The state of PWT depends upon the processor's operating mode and the state of certain bits in its control registers and TLB as follows:

- In Real mode, or in Protected and Virtual-8086 modes while paging is disabled (PG bit in CR0 set to 0):
PWT output = 0 (writeback state)
- In Protected and Virtual-8086 modes while paging is enabled (PG bit in CR0 set to 1):
 - For accesses to I/O space, page directory entries, and other non-paged accesses:
PWT output = PWT bit in CR3
 - For accesses to 4-Kbyte page table entries or 4-Mbyte pages:
PWT output = PWT bit in page directory entry
 - For accesses to 4-Kbyte pages:
PWT output = PWT bit in page table entry

Driven and Floated

PWT is driven off the same clock edge as ADS# and remains in the same state until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted.

PWT is floated off the clock edge on which BOFF# is sampled asserted and off the clock edge on which the processor asserts HLDA in response to HOLD.

5.40 RESET (Reset)

Pin Attribute Input

Summary When the processor samples RESET asserted, it immediately flushes and initializes all internal resources and its internal state including its pipelines and caches, the floating-point state, the MMX state, the 3DNow! state, and all registers, and then the processor jumps to address FFFF_FFF0h to start instruction execution.

The FLUSH# signal is sampled during the falling transition of RESET to invoke the Three-State Test mode.

Sampled RESET is sampled as a level-sensitive input on every clock edge. System logic can drive the signal either synchronously or asynchronously.

During the initial power-on reset of the processor, RESET must remain asserted for a minimum of 1.0 ms after CLK and V_{CC} reach specification before it is negated.

During a warm reset, while CLK and V_{CC} are within their specification, RESET must remain asserted for a minimum of 15 clocks prior to its negation.

5.41 RSVD (Reserved)

Pin Attribute Not Applicable

Summary Reserved signals are a special class of pins that can be treated in one of the following ways:

- As no-connect (NC) pins, in which case these pins are left unconnected
- As pins connected to the system logic as defined by the industry-standard Super7 and Socket 7 interface
- Any combination of NC and Socket 7 pins

In any case, if the RSVD pins are treated accordingly, the normal operation of the AMD-K6-III+ processor is not adversely affected in any manner.

5.42 SCYC (Split Cycle)

Pin Attribute Output

Summary The processor asserts SCYC during misaligned, locked transfers on the D[63:0] data bus. The processor generates additional bus cycles to complete the transfer of misaligned data.

For purposes of bus cycles, the term *aligned* means:

- Any 1-byte transfers
- 2-byte and 4-byte transfers that lie within 4-byte address boundaries
- 8-byte transfers that lie within 8-byte address boundaries

Driven and Floated SCYC is asserted off the same clock edge as ADS#, and negated off the clock edge on which NA# or the last expected BRDY# of the entire locked sequence is sampled asserted. SCYC is only valid during locked memory cycles.

SCYC is floated off the clock edge on which BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in response to HOLD.

5.43 SMI# (System Management Interrupt)

Pin Attribute

Input, Internal Pullup

Summary

The assertion of SMI# causes the processor to enter System Management Mode (SMM). Upon recognizing SMI#, the processor performs the following actions, in the order shown:

1. Flushes its instruction pipelines
2. Completes all pending and in-progress bus cycles
3. Acknowledges the interrupt by asserting SMIACT# after sampling EWBE# asserted (if EWBE# is masked off, then SMIACT# is not affected by EWBE#)
4. Saves the internal processor state in SMM memory
5. Disables interrupts by clearing the interrupt flag (IF) in EFLAGS and disables NMI interrupts
6. Jumps to the entry point of the SMM service routine at the SMM base physical address, which defaults to 0003_8000h in SMM memory

See “System Management Mode (SMM)” on page 241 for more details regarding SMM.

Sampled

SMI# is sampled and latched as a falling edge-sensitive signal. SMI# is sampled on every clock edge but is not recognized until the next instruction boundary. If SMI# is to be recognized on the instruction boundary associated with a BRDY#, it must be sampled asserted a minimum of three clock edges before the BRDY# is sampled asserted.

- If SMI# is asserted synchronously (see Table 19 on page 140), it can be asserted for a minimum of one clock.
- If SMI# is asserted asynchronously, it must have been negated for a minimum of two clocks followed by an assertion of a minimum of two clocks.

A second assertion of SMI# while in SMM is latched but is not recognized until the SMM service routine is exited.

5.44 SMIACT# (System Management Interrupt Active)

Pin Attribute Output

Summary The processor acknowledges the assertion of SMI# with the assertion of SMIACT# to indicate that the processor has entered System Management Mode (SMM). The system logic can use SMIACT# to enable SMM memory. See “SMI# (System Management Interrupt)” on page 130 for more details.

See “System Management Mode (SMM)” on page 241 for more details regarding SMM.

Driven The processor asserts SMIACT# after the last BRDY# of the last pending bus cycle is sampled asserted (including all pending write cycles) and after EWBE# is sampled asserted (if EWBE# is masked off, then SMIACT# is not affected by EWBE#). SMIACT# remains asserted until after the last BRDY# of the last pending bus cycle associated with exiting SMM is sampled asserted.

SMIACT# remains asserted during any flush, internal snoop, or writeback cycle due to an inquire cycle.

5.45 STPCLK# (Stop Clock)

Pin Attribute

Input, Internal Pullup

Summary

The assertion of STPCLK# causes the processor to enter the Stop Grant state, during which the processor's internal clock is stopped. From the Stop Grant state, the processor can subsequently transition to the Stop Clock state, in which the bus clock CLK is stopped. Upon recognizing STPCLK#, the processor performs the following actions, in the order shown:

1. Flushes its instruction pipelines
2. Completes all pending and in-progress bus cycles
3. Acknowledges the STPCLK# assertion by executing a Stop Grant special bus cycle (see Table 24 on page 142)
4. Stops its internal clock after BRDY# of the Stop Grant special bus cycle is sampled asserted and after EWBE# is sampled asserted (if EWBE# is masked off, then entry into the Stop Grant state is not affected by EWBE#)
5. Enters the Stop Clock state if the system logic stops the bus clock CLK (optional)

See "Clock Control" on page 277 for more details regarding clock control.

Sampled

STPCLK# is sampled as a level-sensitive input on every clock edge but is not recognized until the next instruction boundary. System logic can drive the signal either synchronously or asynchronously. If it is asserted asynchronously, it must be asserted for a minimum pulse width of two clocks.

STPCLK# must remain asserted until recognized, which is indicated by the completion of the Stop Grant special cycle.

5.46 TCK (Test Clock)

Pin Attribute Input, Internal Pullup

Summary TCK is the clock for boundary-scan testing using the Test Access Port (TAP). See “Boundary-Scan Test Access Port (TAP)” on page 253 for details regarding the operation of the TAP controller.

Sampled The processor always samples TCK, except while TRST# is asserted.

5.47 TDI (Test Data Input)

Pin Attribute Input, Internal Pullup

Summary TDI is the serial test data and instruction input for boundary-scan testing using the Test Access Port (TAP). See “Boundary-Scan Test Access Port (TAP)” on page 253 for details regarding the operation of the TAP controller.

Sampled The processor samples TDI on every rising TCK edge, but only while in the Shift-IR and Shift-DR states.

5.48 TDO (Test Data Output)

Pin Attribute Output

Summary TDO is the serial test data and instruction output for boundary-scan testing using the Test Access Port (TAP). See “Boundary-Scan Test Access Port (TAP)” on page 253 for details regarding the operation of the TAP controller.

Driven and Floated The processor drives TDO on every falling TCK edge, but only while in the Shift-IR and Shift-DR states. TDO is floated at all other times.

5.49 TMS (Test Mode Select)

Pin Attribute Input, Internal Pullup

Summary TMS specifies the test function and sequence of state changes for boundary-scan testing using the Test Access Port (TAP). See “Boundary-Scan Test Access Port (TAP)” on page 253 for details regarding the operation of the TAP controller.

Sampled The processor samples TMS on every rising TCK edge. If TMS is sampled High for five or more consecutive clocks, the TAP controller enters its Test-Logic-Reset state, regardless of the controller state. This action is the same as that achieved by asserting TRST#.

5.50 TRST# (Test Reset)

Pin Attribute Input, Internal Pullup

Summary The assertion of TRST# initializes the Test Access Port (TAP) by resetting its state machine to the Test-Logic-Reset state. See “Boundary-Scan Test Access Port (TAP)” on page 253 for details regarding the operation of the TAP controller.

Sampled TRST# is a completely asynchronous input that does not require a minimum setup and hold time relative to TCK. See Table 70 on page 310 for the minimum pulse width requirement.

5.51 VCC2DET (VCC2 Detect)

Pin Attribute Output (supported on the CPGA package only)

Summary VCC2DET is internally tied to V_{SS} (logic level 0) to indicate to the system logic that it must supply the specified dual-voltage requirements to the V_{CC2} and V_{CC3} pins. The V_{CC2} pins supply voltage to the processor core, independent of the voltage supplied to the I/O buffers on the V_{CC3} pins. Upon sampling VCC2DET Low, system logic should sample VCC2H/L# to identify core voltage requirements

Note that this pin is not supported on the OBGA package.

Driven VCC2DET always equals 0 and is never floated—even during the Three-State Test mode.

5.52 VCC2H/L# (VCC2 High/Low)

Pin Attribute Output (supported on the CPGA package only)

Summary VCC2H/L# is internally tied to V_{SS} (logic level 0) to indicate to the system logic that it must supply the specified processor core voltage to the V_{CC2} pins. The V_{CC2} pins supply voltage to the processor core, independent of the voltage supplied to the I/O buffers on the V_{CC3} pins. Upon sampling VCC2DET Low to identify dual-voltage processor requirements, system logic should sample VCC2H/L# to identify the core voltage requirements: 2.9-V and 3.2-V products (High) or 2.4-V and lower products (Low).

- VCC2H/L# is only driven High on older legacy (0.35-micron process technology) AMD-K6 processors to indicate core voltages of 2.9 V and 3.2 V.
- VCC2H/L# is driven Low for all AMD-K6 processors with a core voltage requirement of 2.4 V or less. Note that all AMD products based on the 0.18-micron process technology, including the AMD-K6-III+ processor, are 2.0 V or less.

Note that this pin is not supported on the OBGA package.

Driven VCC2H/L# always equals 0 and is never floated for 2.4-V and lower products—even during the Three-State Test mode. To ensure proper operation for 2.9-V and 3.2-V products, system logic that samples VCC2H/L# should design a weak pullup resistor for this signal.

Table 18. Output Pin Float Conditions for VCC2 High/Low

| Name | Floated At: |
|-----------------------|---------------|
| VCC2DET ¹ | Always Driven |
| VCC2H/L# ¹ | Always Driven |

Notes:

1. All outputs except VCC2DET, VCC2H/L#, and TDO float during the Three-State Test mode.

5.53 VID[4:0] (Voltage Identification)

Pin Attribute Output

Summary For AMD PowerNow! technology-enabled processors, the VID[4:0] signals are used to drive the VID inputs of the DC/DC regulator that generates the core voltage for the processor. The processor VID[4:0] outputs default to 01010b when RESET is sampled asserted.

Note that these pins are supported on the low-power versions only of the AMD-K6-III+ processor. For more information about these signals, see the *Embedded AMD-K6™ Processors BIOS Design Guide Application Note*, order# 23913.

Driven VID[4:0] are initialized to the default state after RESET is sampled asserted, the CPU input clock is running, and the core and I/O voltages are applied. Thereafter, the VID [4:0] outputs are always driven.

5.54 W/R# (Write/Read)

Pin Attribute Output

Summary The processor drives W/R# to indicate whether it is performing a write or a read cycle on the bus. In addition, W/R# is used to define other bus cycles, including interrupt acknowledge and special cycles. See Table 23 and Table 24 on page 142 for more details.

Driven and Floated W/R# is driven off the same clock edge as ADS# and remains in the same state until the clock edge on which NA# or the last expected BRDY# of the cycle is sampled asserted. W/R# is driven during memory cycles, I/O cycles, special bus cycles, and interrupt acknowledge cycles.

W/R# is floated off the clock edge on which BOFF# is sampled asserted and off the clock edge that the processor asserts HLDA in response to HOLD.

5.55 WB/WT# (Writeback or Writethrough)

Pin Attribute Input

Summary WB/WT#, together with PWT, specifies the data cache-line state during cacheable read misses and write hits to shared cache lines.

- If WB/WT# = 0 or PWT = 1 during a cacheable read miss or write hit to a shared cache line, the accessed line is cached in the shared state. This is referred to as the *writethrough state* because all write cycles to this cache line are driven externally on the bus.
- If WB/WT# = 1 and PWT = 0 during a cacheable read miss or a write hit to a shared cache line, the accessed line is cached in the exclusive state. Subsequent write hits to the same line cause its state to transition from exclusive to modified. This is referred to as the *writeback state* because the L1 data cache and the L2 cache can contain modified cache lines that are subject to be written back—referred to as a writeback cycle—as the result of an inquire cycle, an internal snoop, a flush operation, or the WBINVD instruction.

Sampled WB/WT# is sampled on the clock edge that the first BRDY# or NA# of a bus cycle is sampled asserted. If the cycle is a burst read, WB/WT# is ignored during the last three assertions of BRDY#. WB/WT# is sampled during memory read and non-writeback write cycles and is ignored during all other types of cycles.

5.56 Pin Tables by Type

Table 19. Input Pin Types

| Name | Type | Name | Type |
|-----------------------|--------------|----------------------|--------------|
| A20M# ¹ | Asynchronous | IGNNE# ¹ | Asynchronous |
| AHOLD | Synchronous | INIT ² | Asynchronous |
| BF[2:0] ³ | Synchronous | INTR ¹ | Asynchronous |
| BOFF# | Synchronous | INV | Synchronous |
| BRDY# | Synchronous | KEN# | Synchronous |
| BRDYC# | Synchronous | NA# | Synchronous |
| CLK | Clock | NMI ² | Asynchronous |
| EADS# | Synchronous | RESET ^{4,5} | Asynchronous |
| EWBE# ⁶ | Synchronous | SMI# ² | Asynchronous |
| FLUSH# ^{2,7} | Asynchronous | STPCLK# ¹ | Asynchronous |
| HOLD | Synchronous | WB/WT# | Synchronous |

Notes:

1. These level-sensitive signals can be asserted synchronously or asynchronously. To be sampled on a specific clock edge, setup and hold times must be met. If asserted asynchronously, they must be asserted for a minimum pulse width of two clocks.
2. These edge-sensitive signals can be asserted synchronously or asynchronously. To be sampled on a specific clock edge, setup and hold times must be met. If asserted asynchronously, they must have been negated at least two clocks prior to assertion and must remain asserted at least two clocks.
3. BF[2:0] are sampled during the falling transition of RESET. They must meet a minimum setup time of 1.0 ms and a minimum hold time of two clocks relative to the negation of RESET.
4. During the initial power-on reset of the processor, RESET must remain asserted for a minimum of 1.0 ms after CLK and V_{CC} reach specification before it is negated.
5. During a warm reset, while CLK and V_{CC} are within their specification, RESET must remain asserted for a minimum of 15 clocks prior to its negation.
6. When register bit EFER[3] is set to 1, EWBE# is ignored by the processor.
7. FLUSH# is also sampled during the falling transition of RESET and can be asserted synchronously or asynchronously. To be sampled on a specific clock edge, setup and hold times must be met relative to the clock edge before the clock edge on which RESET is sampled negated. If asserted asynchronously, FLUSH# must meet a minimum setup and hold time of two clocks relative to the negation of RESET.

Table 20. Output Pin Float Conditions

| Name | Floated At: ¹ | Name | Floated At: ¹ |
|-----------------------|--------------------------|-----------------------|--------------------------|
| A[4:3] ^{2,3} | HLDA, AHOLD, BOFF# | LOCK# ² | HLDA, BOFF# |
| ADS# ² | HLDA, BOFF# | M/IO# ² | HLDA, BOFF# |
| ADSC# ² | HLDA, BOFF# | PCD ² | HLDA, BOFF# |
| APCHK# | Always Driven | PCHK# | Always Driven |
| BE[7:0]# ² | HLDA, BOFF# | PWT ² | HLDA, BOFF# |
| BREQ | Always Driven | SCYC ² | HLDA, BOFF# |
| CACHE# ² | HLDA, BOFF# | SMIACK# | Always Driven |
| D/C# ² | HLDA, BOFF# | VCC2DET | Always Driven |
| FERR# | Always Driven | VCC2H/L# | Always Driven |
| HIT# | Always Driven | VID[4:0] ⁴ | Always Driven |
| HITM# | Always Driven | W/R# ² | HLDA, BOFF# |
| HLDA | Always Driven | | |

Notes:

1. All outputs except VCC2DET, VCC2H/L#, and TDO float during the Three-State Test mode.
2. Floated off the clock edge that BOFF# is sampled asserted and off the clock edge that HLDA is asserted.
3. Floated off the clock edge that AHOLD is sampled asserted.
4. Supported on the low-power versions only.

Table 21. Input/Output Pin Float Conditions

| Name | Floated At: ¹ |
|------------------------|--------------------------|
| A[31:5] ^{2,3} | HLDA, AHOLD, BOFF# |
| AP ^{2,3} | HLDA, AHOLD, BOFF# |
| D[63:0] ² | HLDA, BOFF# |
| DP[7:0] ² | HLDA, BOFF# |

Notes:

1. All outputs except VCC2DET and TDO float during the Three-State Test mode.
2. Floated off the clock edge that BOFF# is sampled asserted and off the clock edge that HLDA is asserted.
3. Floated off the clock edge that AHOLD is sampled asserted.

Table 22. Test Pin Types

| Name | Type | Comment |
|-------|--------|-----------------------------------|
| TCK | Clock | |
| TDI | Input | Sampled on the rising edge of TCK |
| TDO | Output | Driven on the falling edge of TCK |
| TMS | Input | Sampled on the rising edge of TCK |
| TRST# | Input | Asynchronous (Independent of TCK) |

5.57 Bus Cycle Definitions

Table 23. Bus Cycle Definition

| Bus Cycle Initiated | Generated by the CPU | | | | Generated by System Logic |
|--|----------------------|------|------|--------|---------------------------|
| | M/IO# | D/C# | W/R# | CACHE# | KEN# |
| Code Read, L1 Instruction Cache and L2 Cache Line Fill | 1 | 0 | 0 | 0 | 0 |
| Code Read, Noncacheable | 1 | 0 | 0 | 1 | x ¹ |
| Code Read, Noncacheable | 1 | 0 | 0 | x | 1 |
| Encoding for Special Cycle | 0 | 0 | 1 | 1 | x |
| Interrupt Acknowledge | 0 | 0 | 0 | 1 | x |
| I/O Read | 0 | 1 | 0 | 1 | x |
| I/O Write | 0 | 1 | 1 | 1 | x |
| Memory Read, L1 Data Cache and L2 Cache Line Fill | 1 | 1 | 0 | 0 | 0 |
| Memory Read, Noncacheable | 1 | 1 | 0 | 1 | x |
| Memory Read, Noncacheable | 1 | 1 | 0 | x | 1 |
| Memory Write, L1 Data Cache or L2 Cache Writeback | 1 | 1 | 1 | 0 | x |
| Memory Write, Noncacheable | 1 | 1 | 1 | 1 | x |

Notes:

1. x means "don't care"

Table 24. Special Cycles

| Special Cycle | A4 | BE7# | BE6# | BE5# | BE4# | BE3# | BE2# | BE1# | BE0# | M/IO# | D/C# | W/R# | CACHE# | KEN# |
|---|----|------|------|------|------|------|------|------|------|-------|------|------|--------|----------------|
| Stop Grant | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | x ¹ |
| Enhanced Power Management (EPM) Stop Grant ² | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | x |
| Flush Acknowledge (FLUSH# sampled asserted) | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | x |
| Writeback (WBINVD instruction) | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | x |
| Halt | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | x |
| Flush (INVD, WBINVD instruction) | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | x |
| Shutdown | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | x |

Notes:

1. x means "don't care".
2. Supported on the low-power versions only.

6 AMD PowerNow!™ Technology

The AMD PowerNow!™ technology is an advanced second-generation power management feature that reduces the overall power consumed by the processor through control of voltage and frequency. This power saving technology is designed to be dynamic and flexible by enabling instant “on-the-fly” and independent control of both the processor’s core voltage and frequency.

AMD PowerNow! technology can be used in conjunction with the existing power management schemes in an embedded system to provide a better combination of performance and power savings than previously possible.

6.1 Enhanced Power Management Features

AMD PowerNow! technology-enabled processors include two new features specifically designed to enhance power management functionality:

- Dynamic core frequency control
- Dynamic core voltage control

These enhanced power management features are accessed and controlled through an I/O block and two registers:

- An aligned 16-byte block of I/O address space is defined by the Enhanced Power Management Register (EPMR).
- The Enhanced Power Management Register (EPMR) is supported on low-power versions of the processor only.
- The Processor State Observability Register (PSOR) is defined differently on the low-power versions of the AMD-K6-III+ processor to support AMD PowerNow! technology features.

The EPMR and PSOR registers and the I/O block are defined in this section, followed by a discussion of how to implement and use the AMD PowerNow! technology features (see “Dynamic Core Frequency and Core Voltage Control” on page 150). The *Embedded AMD-K6™ Processors BIOS Design Guide Application Note*, order# 23913, contains additional information.

Enhanced Power Management Register (EPMR)

The EPMR register allows software to access the aligned Enhanced Power Management (EPM) 16-byte block of I/O address space, which contains bits for enabling, controlling, and monitoring the enhanced power management features. All accesses to the EPM 16-byte I/O block must be aligned dword accesses. Valid accesses to the EPM 16-byte block do not generate I/O cycles on the host bus, while non-aligned and non-dword accesses are passed to the host bus.

The EPMR is MSR C000_0086h.

Figure 54 and Table 25 define the EPMR register. An assertion of RESET clears all of the bits of the 16-byte I/O block to zero (excluding the Voltage ID Output bits which default to 01010b). BIOS must always initialize the EPMR register and enhanced power management features whenever RESET is asserted.

For more information about the EPMR register, see the *Embedded AMD-K6™ Processors BIOS Design Guide Application Note*, order# 23913.

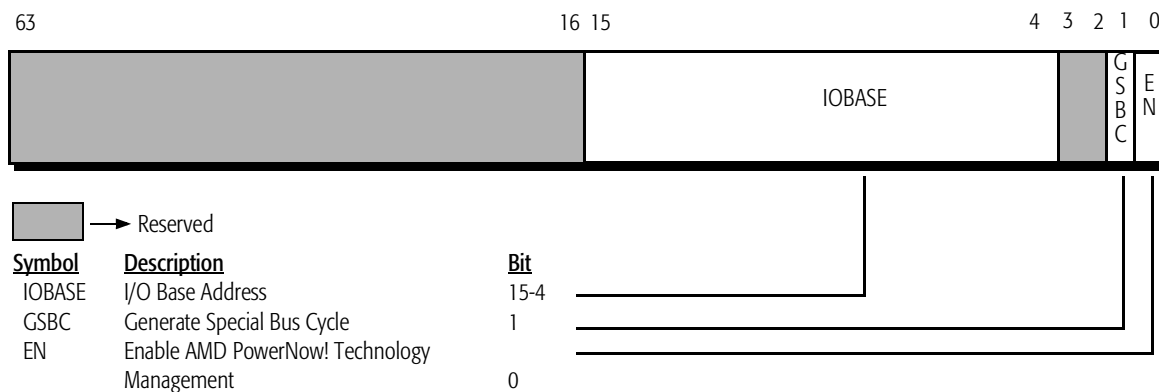


Figure 54. Enhanced Power Management Register (EPMR)

Table 25. Enhanced Power Management Register (EPMR) Definition

| Bit | Description | R/W | Function ¹ |
|-------|---|-----|---|
| 63–16 | Reserved | R | All reserved bits are always read as 0. |
| 15–4 | I/O BASE Address (IOBASE) | R/W | IOBASE defines a base address for a 16-byte block of I/O address space accessible for enabling, controlling, and monitoring the EPM features. |
| 3–2 | Reserved | R | All reserved bits are always read as 0. |
| 1 | Generate Special Bus Cycle (GSBC) | R/W | This bit controls whether a special bus cycle is generated upon dword accesses within the EPM 16-byte I/O block. If set to 1, an EPM special bus cycle is generated, where BE[7:0]# = BFh and A[4:3] = 00b. |
| 0 | Enable AMD PowerNow! Technology Management (EN) | R/W | This bit controls access to the I/O-mapped address space for the AMD PowerNow! EPM features. Clearing this bit to zero does not affect the state of bits defined in the EPM 16-byte I/O block. |

Notes:

1. All bits default to 0 when RESET is asserted.

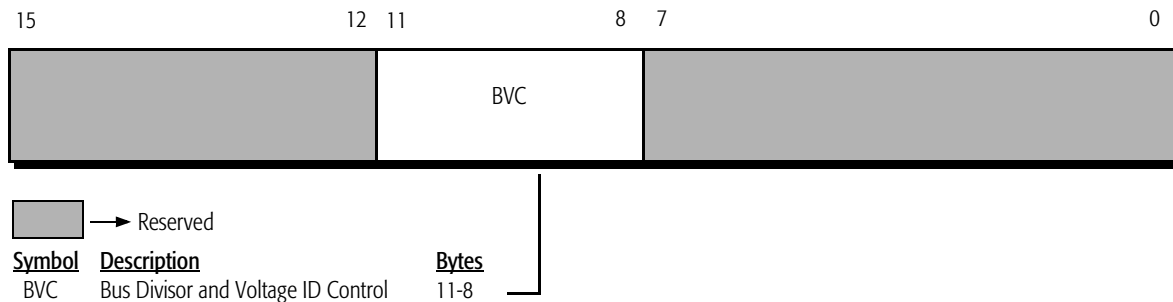
IOBASE Field. The IOBASE field is initialized during POST to an I/O address range used by an SMM handler to access the enhanced power management features. Because the I/O range is only enabled and accessed by the SMM handler during SMM, the EPM features are hidden from all other software (OS included)—BIOS does not need to report the I/O range to the operating system.

GSBC Bit. If the GSBC bit is enabled (set to 1), a special bus cycle is generated upon a dword access within the EPM 16-byte I/O block. The EPM special bus cycle is defined as the processor driving D/C# = 0, M/IO# = 0, and W/R# = 1, BE[7:0]# = BFh and A[31:3] = 0000h. The system logic must return BRDY# in response to all processor special cycles.

EN Bit. The EN bit should only be enabled (set to 1) by an SMM handler when the SMM handler accesses the EPM features. Upon exiting, the SMM handler should disable the EN bit and thereby protect the EPM 16-byte I/O block from unwanted accesses. When the EN bit is disabled, accesses to the EPM block 16-byte I/O block are passed to the host bus.

EPM 16-Byte I/O Block

The EPM 16-byte I/O block contains one 4-byte field—Bus Divisor and Voltage ID Control (BVC)—for enabling, controlling, and monitoring the enhanced power management features (see Figure 55). Table 26 defines the function of the BVC field within the EPM 16-byte I/O block mapped by the EPMR.

**Figure 55. EPM 16-Byte I/O Block****Table 26. EPM 16-Byte I/O Block Definition**

| Byte | Description | R/W | Function ¹ |
|-------|--|-----|--|
| 15-12 | Reserved | R | All reserved bits are always read as 0. |
| 11-8 | Bus Divisor and Voltage ID Control (BVC) | R/W | The bit fields within the BVC bytes allow software to change the processor bus divisor and core voltage. |
| 7-0 | Reserved | R | All reserved bits are always read as 0. |

Notes:

1. All bits default to 0 when RESET is asserted.

BVC. Figure 56 on page 147 shows the format, and Table 27 on page 147 defines the function of each bit of the BVC field located within the EPM 16-byte I/O block.

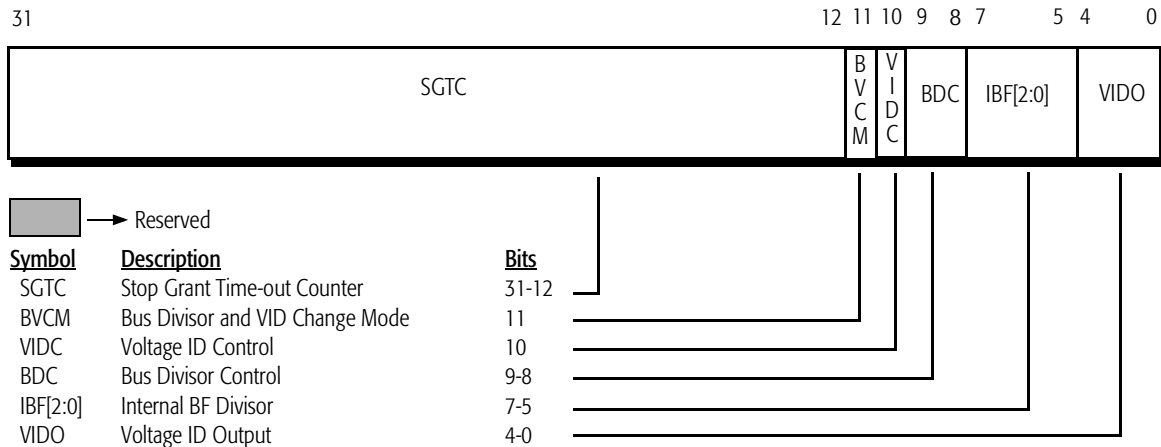


Figure 56. Bus Divisor and Voltage ID Control (BVC) Field

Table 27. Bus Divisor and Voltage ID Control (BVC) Definition

| Bit | Description | R/W | Function ¹ |
|-------|--|-----|--|
| 31-12 | Stop Grant Time-Out Counter (SGTC) | W | Writing a non-zero value to this field causes the processor to enter the EPM Stop Grant state internally. This 20-bit value is multiplied by 4096 to determine the duration of the EPM Stop Grant state, measured in processor bus clocks. |
| 11 | Bus Divisor and VID Change Mode (BVCM) | R/W | This bit controls the mode in which the bus-divisor and the voltage control bits are allowed to change. If BVCM=0, the Bus Divisor and Voltage ID changes take effect only upon entering the EPM Stop Grant state as a result of the SGTC field being programmed. BVCM=1 is reserved. |
| 10 | Voltage ID Control (VIDC) | R/W | This bit controls the mode of Voltage ID control. If VIDC=0, the processor VID[4:0] pins are unchanged upon entering the EPM Stop Grant state. If VIDC=1, the processor VID[4:0] pins are programmed to the VIDO value upon entering the EPM Stop Grant state. BIOS should initialize this bit to 1 during the POST routine. |
| 9-8 | Bus Divisor Control (BDC) | R/W | This 2-bit field controls the mode of Bus Divisor control. If BDC[1:0]=00b, the BF[2:0] pins are sampled at the falling edge of RESET. If BDC[1:0]=1xb, the IBF[2:0] field is sampled upon entering the EPM Stop Grant state. BDC[1:0]=01b is reserved. BIOS should initialize these bits to 10b during the POST routine. |
| 7-5 | Internal BF Divisor (IBF[2:0]) | R/W | If BDC[1:0]=1xb, the processor EBF[2:0] field of the PSOR is programmed to the IBF[2:0] value upon entering the EPM Stop Grant state. |
| 4-0 | Voltage ID Output (VIDO) | R/W | This 5-bit value is driven out on the processor VID[4:0] pins upon entering the EPM Stop Grant state if the VIDC bit=1. These bits are initialized to 01010b and driven on the processor VID[4:0] pins at RESET. |

Notes:

1. All bits default to 0 when RESET is asserted, except the VIDO bits which default to 01010b.

Processor State Observability Register (PSOR)

To support AMD PowerNow! technology, all low-power versions of the AMD-K6-III+ processor provide a different version of the Processor State Observability Register (PSOR), as shown in Figure 57 and fully described in this section. All standard-power versions of the processor support the PSOR register as defined on page 49. The PSOR register is MSR C000_0087h.

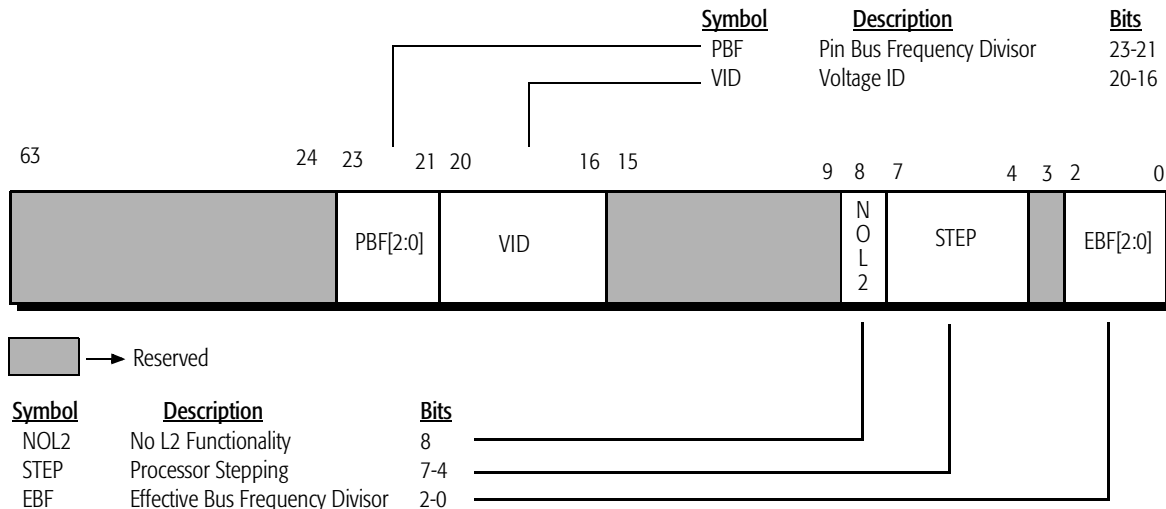


Figure 57. Processor State Observability Register (PSOR)—Low-Power Versions of the Processor

PBF[2:0] Field. This read-only field contains the BF divisor values externally applied to the processor BF[2:0] pins. These input BF values are sampled by the processor during the falling transition of RESET.

Note: This BF divisor value may be different than the BF divisor value supplied to the processor’s internal PLL.

VID Field. This read-only field contains the Voltage ID bits driven to the processor VID[4:0] pins at RESET. These bits are initialized to 01010b and driven on the VID[4:0] pins at RESET.

Note: Low-power AMD-K6-III+ processors support AMD PowerNow! technology, which enables dynamic alteration of the processor’s core voltage. See “Enhanced Power Management Register (EPMR)” on page 144 for information on programming the VID[4:0] pins.

NOL2 Bit. This read-only bit indicates whether the processor contains an L2 cache. This bit is always set to 0 for the AMD-K6-III+ processor.

STEP Field. This read-only field contains the stepping ID. This is identical to the value returned by CPUID standard function 1 in EAX[3:0].

EBF[2:0] Field. This read-only field contains the effective value of the BF divisor supplied to the processor's internal PLL, which allows the BIOS to determine the frequency of the host bus. The core frequency must first be determined using the Time Stamp Counter (TSC) method (see "Time Stamp Counter" on page 46). The core frequency is then divided by the processor-to-bus clock ratio as determined by the EBF field (see Table 28). The result is the frequency of the processor bus.

Table 28. Processor-to-Bus Clock Ratios

| State of EBF[2:0] | Processor-to-Bus Clock Ratio |
|-------------------|------------------------------|
| 100b | 2.0x ¹ |
| 101b | 3.0x |
| 110b | 6.0x |
| 111b | 3.5x |
| 000b | 4.5x |
| 001b | 5.0x |
| 010b | 4.0x |
| 011b | 5.5x |

Notes:

1. The AMD-K6-III+ processor does not support the 2.5x ratio supported by earlier AMD-K6 processors. Instead, a ratio of 2.0x is selected when EBF[2:0] equals 100b.

6.2 Dynamic Core Frequency and Core Voltage Control

AMD PowerNow! technology-enabled processors support the ability to change the bus frequency divisor and core voltage transparently to the user during run-time. These features are implemented in conjunction with a new clock control state—the EPM Stop Grant state.

For AMD PowerNow! technology state transitions, the EPMR register is accessed using an SMM handler.

- The SMM handler initiates core voltage and frequency transitions by writing a non-zero value to the Stop Grant Time-Out Counter (SGTC) field.
- This action automatically places the processor into the EPM Stop Grant State and transitions the CPU core voltage and frequency to the values specified in the Voltage ID Output (VIDO) and Internal BF Divisor (IBF) fields of the BVC field.
- Once the timer of the SGTC has expired, the EPM Stop Grant State is exited and the AMD PowerNow! technology state transition is completed.

See “Clock Control” on page 277 for more information about the EPM Stop Grant State.

Effective Bus Frequency Divisor (EBF[2:0])

The processor core frequency is controlled by the Effective Bus Frequency Divisor—EBF[2:0]—which dictates the processor-to-bus clock ratio supplied to the processor’s internal PLL. This processor-to-bus clock ratio is multiplied by the external bus frequency to set the frequency of operation for the processor core.

- At the fall of RESET, the EBF[2:0] value is determined by the state of the processor BF[2:0] input pins.
- Afterwards, the EBF[2:0] value can be dynamically controlled through AMD PowerNow! technology state transitions.

Table 28 on page 149 lists valid EBF[2:0] states and equivalent processor-to-bus clock ratios.

**Dynamic Core
Frequency Control**

For AMD PowerNow! technology core frequency transitions, the BVC field of the EPM 16-byte I/O block is accessed through an SMM handler.

- To change the processor core frequency, the SMM handler initiates core voltage and frequency transitions by writing a non-zero value to the SGTC field.
- This action automatically places the processor into the EPM Stop Grant state and transitions the CPU core voltage and frequency to the values specified in the VIDO and IBF fields of the BVC field.

Note: System-initiated inquire (snoop) cycles are not supported and must be prevented during the EPM Stop Grant state.

**Voltage Identification
(VID) Outputs**

AMD PowerNow! technology-enabled processors feature Voltage ID (VID) outputs to support dynamic control of the core voltage.

- These outputs serve as inputs to a DC/DC regulator that supplies the processor core voltage.
- Based on its VID[4:0] inputs, the regulator outputs a corresponding voltage.
- For those regulators that do not support VID inputs, the processor VID[4:0] outputs must be used to manipulate the regulator's feedback voltage to vary the regulator output voltage.

7 Bus Cycles

The following sections describe and illustrate the timing and relationship of bus signals during various types of bus cycles. A representative set of bus cycles is illustrated.

7.1 Timing Diagrams

The timing diagrams illustrate the signals on the external local bus as a function of time, as measured by the bus clock (CLK).

Bus Clock (CLK)

Throughout this chapter, the term *clock* refers to a single bus-clock cycle. A clock extends from one rising CLK edge to the next rising CLK edge. The processor samples and drives most signals relative to the rising edge of CLK. The exceptions to this rule include the following:

- BF[2:0]—Sampled on the falling edge of RESET
- FLUSH#—Sampled on the falling edge of RESET, also sampled on the rising edge of CLK
- All inputs and outputs are sampled relative to TCK in Boundary-Scan Test Mode. Inputs are sampled on the rising edge of TCK, outputs are driven off of the falling edge of TCK.

Waveform Definitions

For each signal in the timing diagrams, the High level represents 1, the Low level represents 0, and the Middle level represents the floating (high-impedance) state.

When both the High and Low levels are shown, the meaning depends on the signal:

- A single signal indicates ‘don’t care’.
- In the case of bus activity, if both High and Low levels are shown, it indicates that the processor, alternate master, or system logic is driving a value, but this value may or may not be valid. (For example, the value on the address bus is valid only during the assertion of ADS#, but addresses are also driven on the bus at other times.)

Figure 58 on page 154 defines the different waveform representations.

Active High Signals

For all active High signals, the term *asserted* means the signal is in the High-voltage state and the term *negated* means the signal is in the Low-voltage state.

Active Low Signals

For all active Low signals, the term *asserted* means the signal is in the Low-voltage state and the term *negated* means the signal is in the High-voltage state.


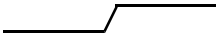


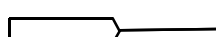
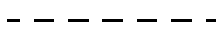
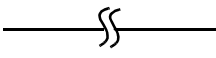
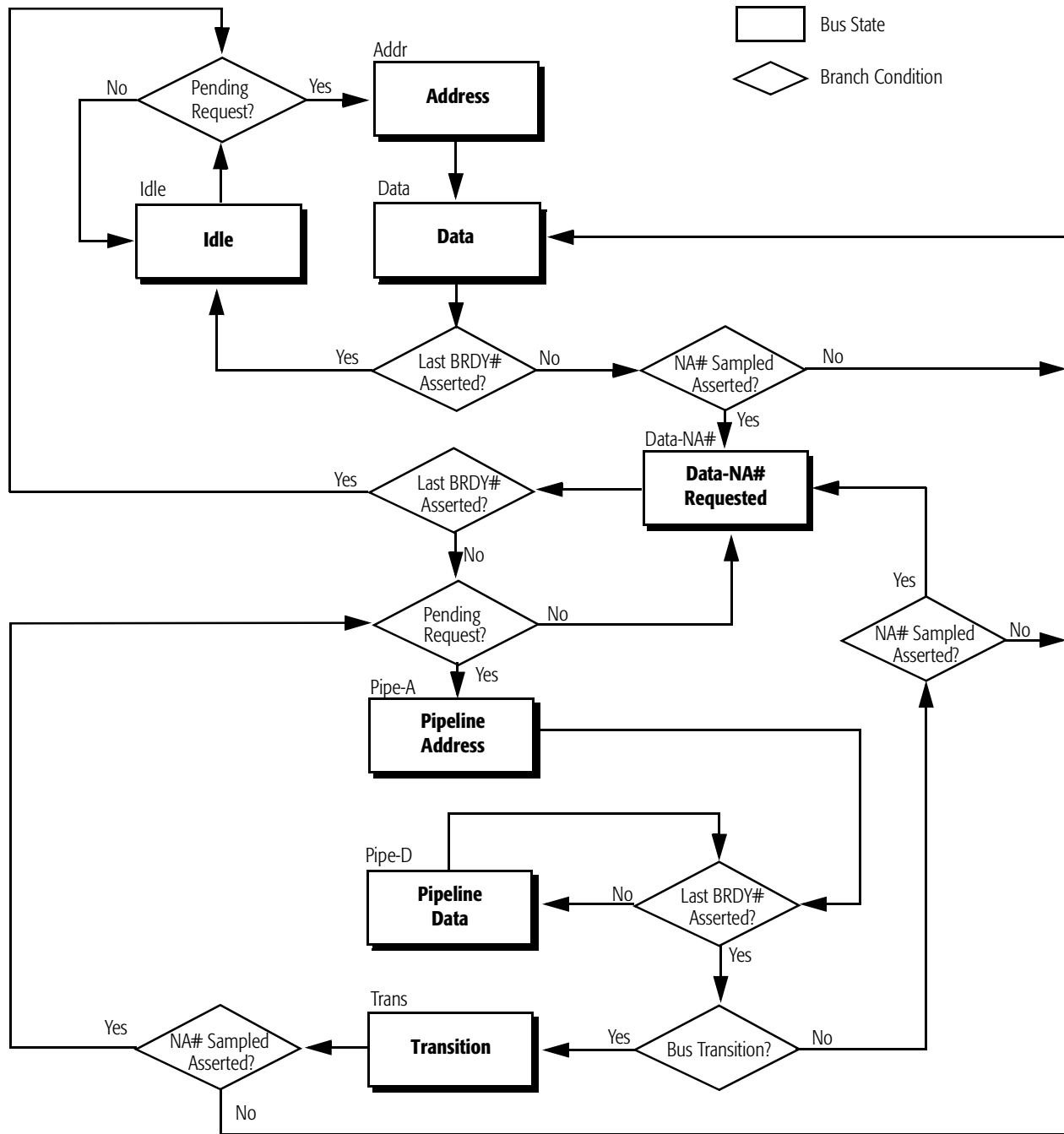
| Waveform | Description |
|---|--|
|  | Don't care or bus is driven |
|  | Signal or bus is changing from Low to High |
|  | Signal or bus is changing from High to Low |
|  | Bus is changing |
|  | Bus is changing from valid to invalid |
|  | Signal or bus is floating |
|  | Denotes multiple clock periods |

Figure 58. Waveform Definitions

7.2 Bus States

The bus states illustrated in Figure 59 are described in this section.



Note: The processor transitions to the IDLE state on the clock edge on which BOFF# or RESET is sampled asserted.

Figure 59. Bus State Machine Diagram

- Idle** The processor does not drive the system bus in the Idle state and remains in this state until a new bus cycle is requested. The processor enters this state off the clock edge on which the last BRDY# of a cycle is sampled asserted during the following conditions:
- The processor is in the Data state
 - The processor is in the Data-NA# Requested state and no internal pending cycle is requested
- In addition, the processor is forced into this state when the system logic asserts RESET or BOFF#. The transition to this state occurs on the clock edge on which RESET or BOFF# is sampled asserted.
- Address** In this state, the processor drives ADS# to indicate the beginning of a new bus cycle by validating the address and control signals. The processor remains in this state for one clock and unconditionally enters the Data state on the next clock edge.
- Data** In the Data state, the processor drives the data bus during a write cycle or expects data to be returned during a read cycle. The processor remains in this state until either NA# or the last BRDY# is sampled asserted. If the last BRDY# is sampled asserted or both the last BRDY# and NA# are sampled asserted on the same clock edge, the processor enters the Idle state. If NA# is sampled asserted first, the processor enters the Data-NA# Requested state.
- Data-NA# Requested** If the processor samples NA# asserted while in the Data state and the current bus cycle is not completed (the last BRDY# is not sampled asserted), it enters the Data-NA# Requested state. The processor remains in this state until either the last BRDY# is sampled asserted or an internal pending cycle is requested. If the last BRDY# is sampled asserted before the processor drives a new bus cycle, the processor enters the Idle state (no internal pending cycle is requested) or the Address state (processor has a internal pending cycle).
- Pipeline Address** In this state, the processor drives ADS#, indicating the beginning of a new bus cycle and validating the address and control signals. In this state, the processor is still waiting for the current bus cycle to be completed (until the last BRDY# is

sampled asserted). If the last BRDY# is not sampled asserted, the processor enters the Pipeline Data state.

If the processor samples the last BRDY# asserted in this state, it determines if a bus transition is required between the current bus cycle and the pipelined bus cycle. A bus transition is required when the data bus direction changes between bus cycles, such as a memory write cycle followed by a memory read cycle. If a bus transition is required, the processor enters the Transition state for one clock to prevent data bus contention. If a bus transition is not required, the processor enters the Data state.

The processor does not transition to the Data-NA# Requested state from the Pipeline Address state because the processor does not begin sampling NA# until it has exited the Pipeline Address state.

Pipeline Data

Two bus cycles are executing concurrently in this state. The processor cannot issue any additional bus cycles until the current bus cycle is completed. The processor drives the data bus during write cycles or expects data to be returned during read cycles for the current bus cycle until the last BRDY# of the current bus cycle is sampled asserted.

If the processor samples the last BRDY# asserted in this state, it determines if a bus transition is required between the current bus cycle and the pipelined bus cycle. If the bus transition is required, the processor enters the Transition state for one clock to prevent data bus contention. If a bus transition is not required, the processor enters the Data state (NA# was not sampled asserted) or the Data-NA# Requested state (NA# was sampled asserted).

Transition

The processor enters this state for one clock during data bus transitions and enters the Data state on the next clock edge if NA# is not sampled asserted. The sole purpose of this state is to avoid bus contention caused by bus transitions during pipeline operation.

7.3 Memory Reads and Writes

The AMD-K6-III+ processor performs single or burst-memory bus cycles.

- The single-transfer memory bus cycle transfers 1, 2, 4, or 8 bytes and requires a minimum of two clocks.
- Misaligned instructions or operands result in a split cycle, which requires multiple transactions on the bus.
- A burst cycle consists of four back-to-back 8-byte (64-bit) transfers on the data bus.

Single-Transfer Memory Read and Write

Figure 60 on page 159 shows a single-transfer read from memory, followed by two single-transfer writes to memory. For the memory read cycle, the processor asserts ADS# for one clock to validate the bus cycle and also drives A[31:3], BE[7:0]#, D/C#, W/R#, and M/IO# to the bus. The processor then waits for the system logic to return the data on D[63:0] (with DP[7:0] for parity checking) and assert BRDY#. The processor samples BRDY# on every clock edge starting with the clock edge after the clock edge that negates ADS#. See “BRDY# (Burst Ready)” on page 103.

During the read cycle, the processor drives PCD, PWT, and CACHE# to indicate its caching and cache-coherency intent for the access. The system logic returns KEN# and WB/WT# to either confirm or change this intent. If the processor asserts PCD and negates CACHE#, the accesses are noncacheable, even though the system logic asserts KEN# during the BRDY# to indicate its support for cacheability. The processor (which drives CACHE#) and the system logic (which drives KEN#) must agree in order for an access to be cacheable.

The processor can drive another cycle (in this example, a write cycle) by asserting ADS# off the next clock edge after BRDY# is sampled asserted. Therefore, an idle clock is guaranteed between any two bus cycles. The processor drives D[63:0] with valid data one clock edge after the clock edge on which ADS# is asserted. To minimize processor idle times, the system logic stores the address and data in write buffers, returns BRDY#, and performs the store to memory later. If the processor samples EWBE# negated during a write cycle, it suspends certain activities until EWBE# is sampled asserted. See “EWBE# (External Write Buffer Empty)” on page 110. In

Figure 60, the second write cycle occurs during the execution of a serializing instruction. The processor delays the following cycle until EWBE# is sampled asserted.

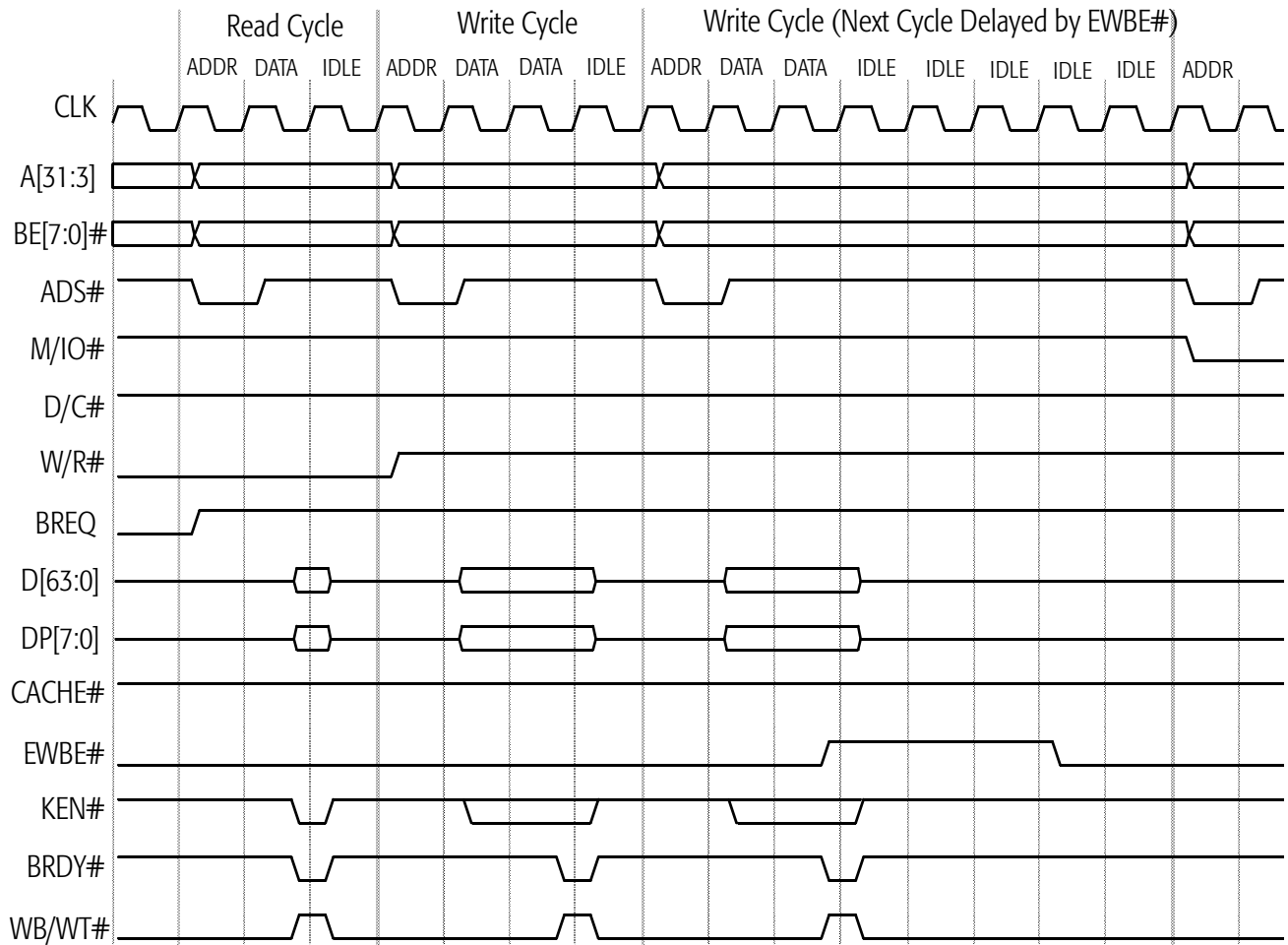


Figure 60. Non-Pipelined Single-Transfer Memory Read/Write and Write Delayed by EWBE#

Misaligned Single-Transfer Memory Read and Write

Figure 61 on page 161 shows a misaligned (split) memory read followed by a misaligned memory write. Any cycle that is not aligned as defined in “SCYC (Split Cycle)” on page 129 is considered misaligned. When the processor encounters a misaligned access, it determines the appropriate pair of bus cycles—each with its own ADS# and BRDY#—required to complete the access.

The AMD-K6-III+ processor performs misaligned memory reads and memory writes using least-significant bytes (LSBs) first followed by most-significant bytes (MSBs). Table 29 shows the order. In the first memory read cycle in Figure 61, the processor reads the least-significant bytes. Immediately after the processor samples BRDY# asserted, it drives the second bus cycle to read the most-significant bytes to complete the misaligned transfer.

Table 29. Bus-Cycle Order During Misaligned Memory Transfers

| Type of Access | First Cycle | Second Cycle |
|----------------|-------------|--------------|
| Memory Read | LSBs | MSBs |
| Memory Write | LSBs | MSBs |

Similarly, the misaligned memory write cycle in Figure 61 transfers the LSBs to the memory bus first. In the next cycle, after the processor samples BRDY# asserted, the MSBs are written to the memory bus.

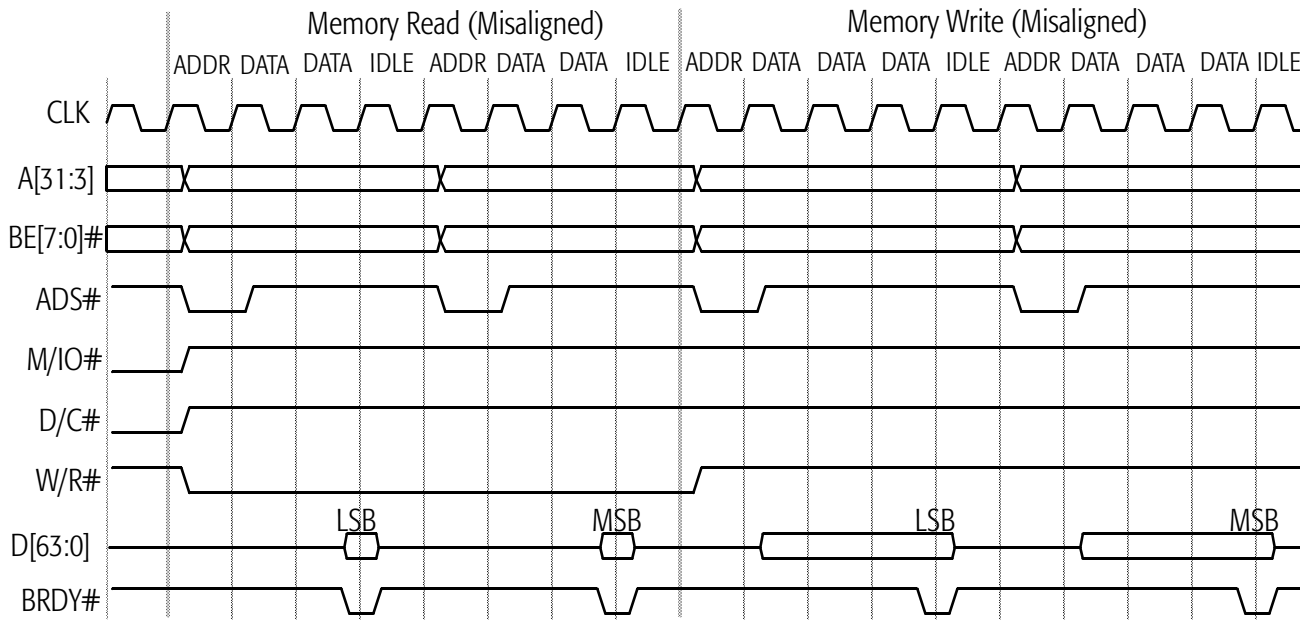


Figure 61. Misaligned Single-Transfer Memory Read and Write

Burst Reads and Pipelined Burst Reads

Figure 62 on page 163 shows normal burst read cycles and a pipelined burst read cycle. The AMD-K6-III+ processor drives CACHE# and ADS# together to specify that the current bus cycle is a burst cycle. If the processor samples KEN# asserted with the first BRDY#, it performs burst transfers. During the burst transfers, the system logic must ignore BE[7:0]# and must return all eight bytes beginning at the starting address the processor asserts on A[31:3]. Depending on the starting address, the system logic must determine the successive quadword addresses (A[4:3]) for each transfer in a burst, as shown in Table 30. The processor expects the second, third, and fourth quadwords to occur in the sequences shown in Table 30.

Table 30. A[4:3] Address-Generation Sequence During Bursts

| Address Driven By Processor on A[4:3] | A[4:3] Addresses of Subsequent Quadwords ¹ Generated by System Logic | | | |
|---------------------------------------|---|------------|------------|------------|
| | Quadword 1 | Quadword 2 | Quadword 3 | Quadword 4 |
| 00b | 01b | 10b | 11b | |
| 01b | 00b | 11b | 10b | |
| 10b | 11b | 00b | 01b | |
| 11b | 10b | 01b | 00b | |

Notes:

1. Quadword = 8 bytes.

In Figure 62, the processor drives CACHE# throughout all burst read cycles. In the first burst read cycle, the processor drives ADS# and CACHE#, then samples BRDY# on every clock edge starting with the clock edge after the clock edge that negates ADS#. The processor samples KEN# asserted on the clock edge on which the first BRDY# is sampled asserted, executes a 32-byte burst read cycle, and expects a total of four BRDY# signals. An ideal no-wait state access is shown in Figure 62, whereas most system logic solutions add wait states between the transfers.

The second burst read cycle illustrates a similar sequence, but the processor samples NA# asserted on the same clock edge that the first BRDY# is sampled asserted. NA# assertion indicates the system logic is requesting the processor to output the next address early (also known as a pipeline transfer request). Without waiting for the current cycle to complete, the processor drives ADS# and related signals for the next burst cycle. Pipelining can reduce processor cycle-to-cycle idle times.

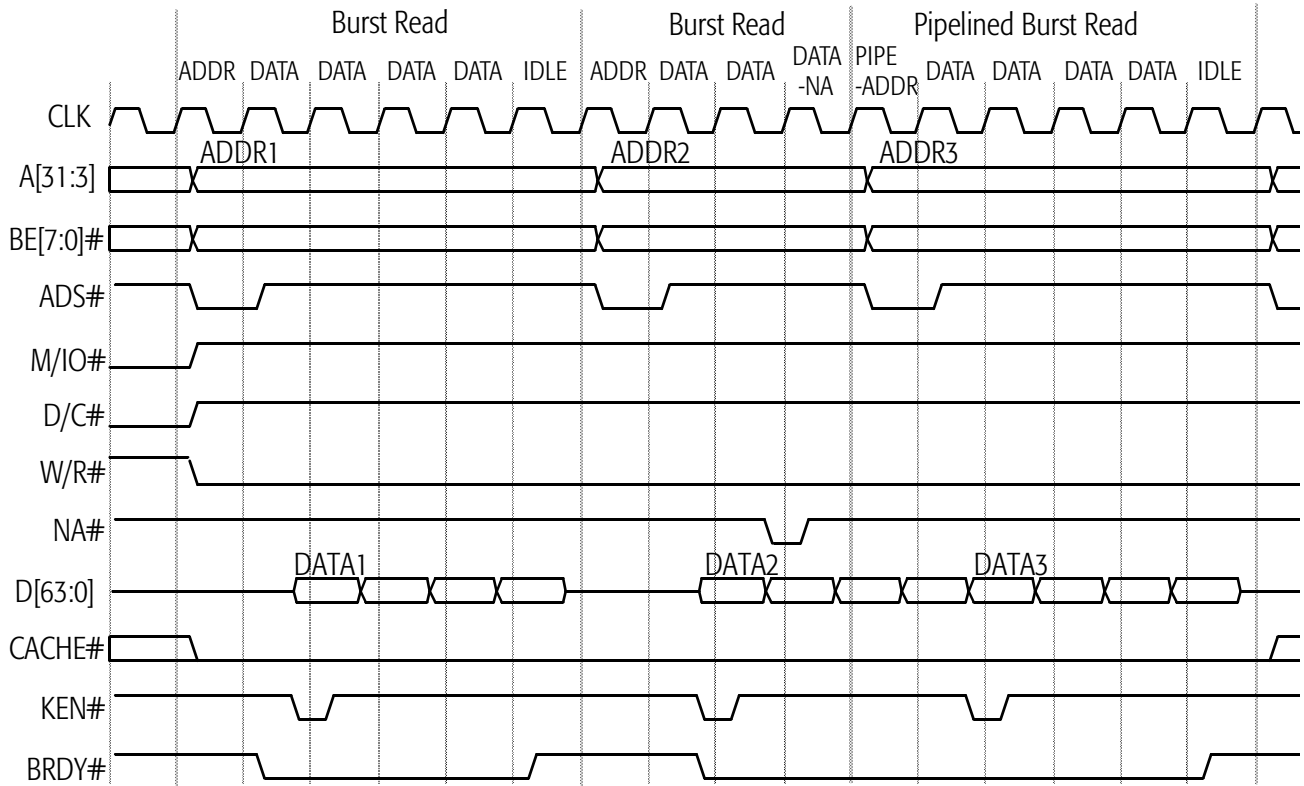


Figure 62. Burst Reads and Pipelined Burst Reads

Burst Writeback

Figure 63 on page 165 shows a burst read followed by a writeback transaction. The AMD-K6-III+ processor initiates writebacks under the following conditions:

- *Replacement*—If a cache-line fill is initiated for a cache line currently filled with valid entries, the processor selects a line for replacement based on a least-recently-used (LRU) algorithm for the L1 instruction cache and the L2 cache, and a least-recently-allocated (LRA) algorithm for the L1 data cache. Before a replacement is made to a L1 data cache or L2 cache line that is in the modified state, the modified line is scheduled to be written back to memory.
- *Internal Snoop*—The processor snoops its L1 instruction cache during read or write misses to its L1 data cache, and it snoops its L1 data cache during read misses to its L1 instruction cache. This snooping is performed to determine whether the same address is stored in both caches, a situation that is taken to imply the occurrence of self-modifying code. If an internal snoop hits a L1 data cache line in the modified state, the line is written back to memory before being invalidated.
- *WBINVD Instruction*—When the processor executes a WBINVD instruction, it writes back all modified lines in the L1 data cache and L2 cache, and then invalidates all lines in all caches.
- *Cache Flush*—When the processor samples FLUSH# asserted, it executes a flush acknowledge special cycle and writes back all modified lines in the L1 data cache and L2 cache, and then invalidates all lines in all caches.

The processor drives writeback cycles during inquire or cache flush cycles. The writeback shown in Figure 63 is caused by a cache-line replacement. The processor completes the burst read cycle that fills the cache line. Immediately following the burst read cycle is the burst writeback cycle that represents the modified line to be written back to memory. D[63:0] are driven one clock edge after the clock edge on which ADS# is asserted and are subsequently changed off the clock edge on which each of the four BRDY# signals of the burst cycle are sampled asserted.

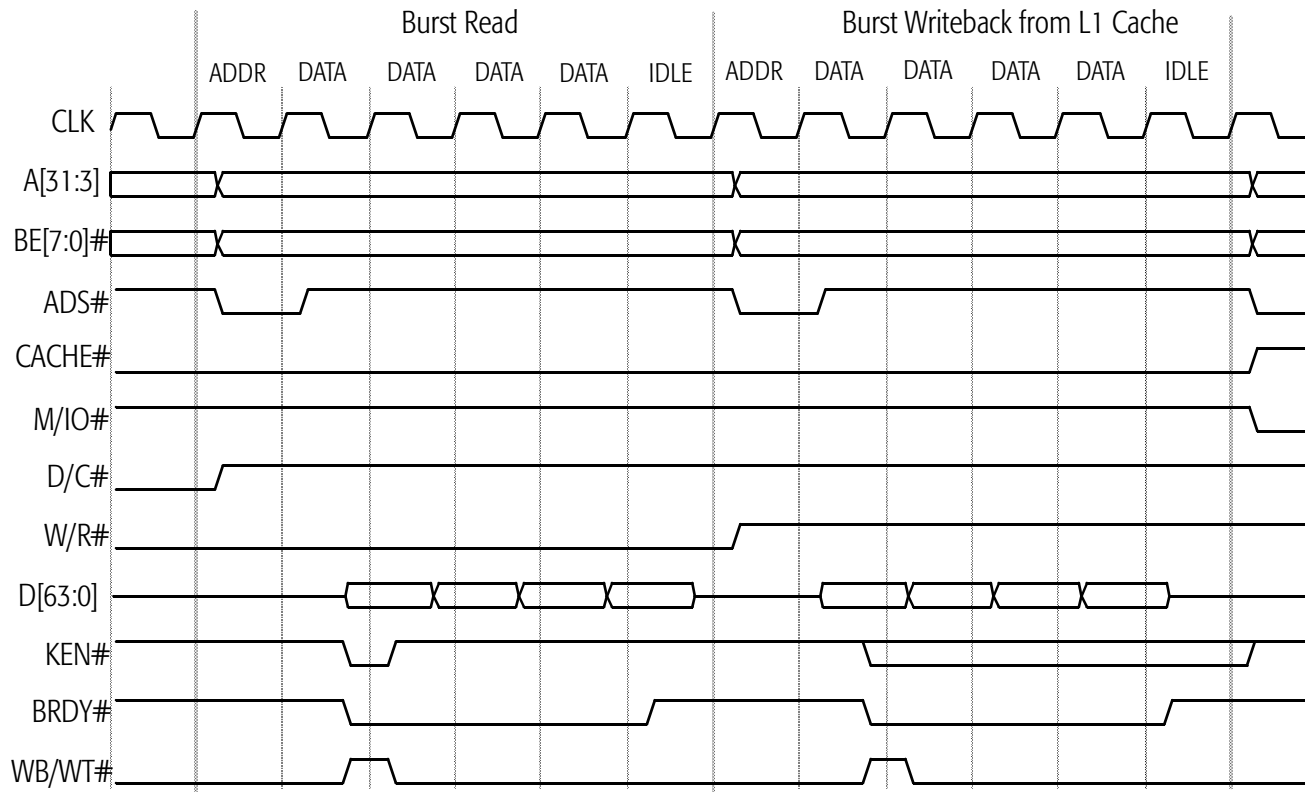


Figure 63. Burst Writeback due to Cache-Line Replacement

7.4 I/O Read and Write

Basic I/O Read and Write

The processor accesses I/O when it executes an I/O instruction (for example, IN or OUT). Figure 64 shows an I/O read followed by an I/O write. The processor drives M/IO# Low and D/C# High during I/O cycles. In this example, the first cycle shows a single wait state I/O read cycle. It follows the same sequence as a single-transfer memory read cycle. The processor drives ADS# to initiate the bus cycle, then it samples BRDY# on every clock edge starting with the clock edge after the clock edge that negates ADS#. The system logic must return BRDY# to complete the cycle. When the processor samples BRDY# asserted, it can assert ADS# for the next cycle off the next clock edge. (In this example, an I/O write cycle.)

The I/O write cycle is similar to a memory write cycle, but the processor drives M/IO# low during an I/O write cycle. The processor asserts ADS# to initiate the bus cycle. The processor drives D[63:0] with valid data one clock edge after the clock edge on which ADS# is asserted. The system logic must assert BRDY# when the data is properly stored to the I/O destination. The processor samples BRDY# on every clock edge starting with the clock edge after the clock edge that negates ADS#. In this example, two wait states are inserted while the processor waits for BRDY# to be asserted.

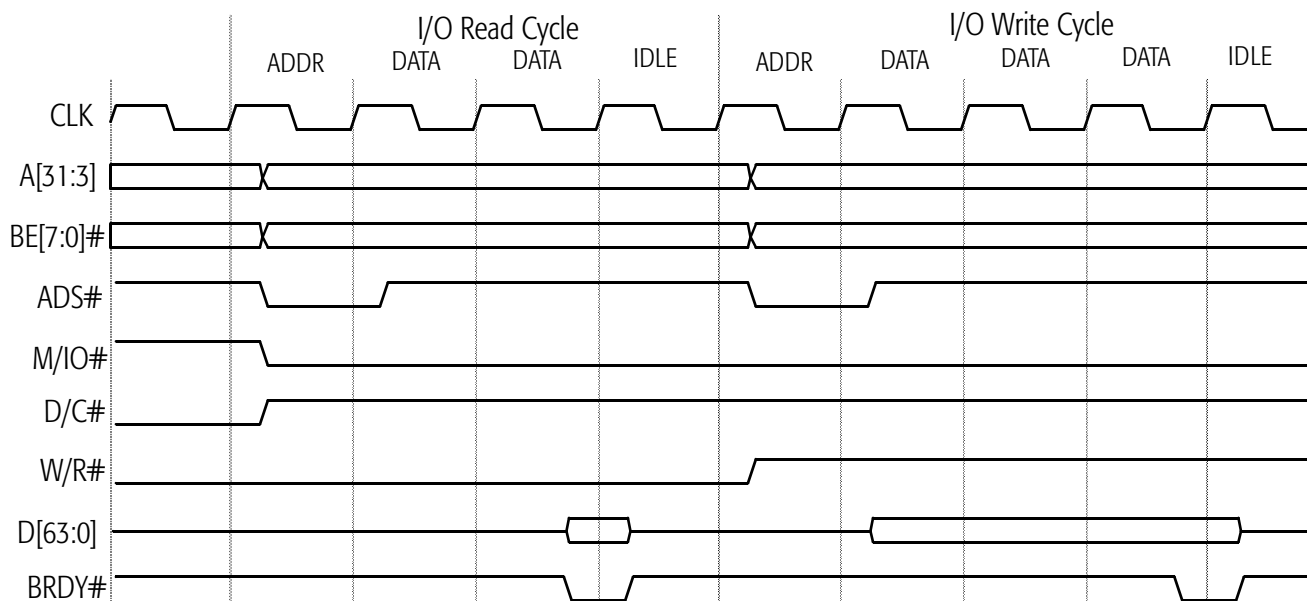


Figure 64. Basic I/O Read and Write

Misaligned I/O Read and Write

Table 31 shows the misaligned I/O read and write cycle order executed by the AMD-K6-III+ processor. In Figure 65, the least-significant bytes (LSBs) are transferred first. Immediately after the processor samples BRDY# asserted, it drives the second bus cycle to transfer the most-significant bytes (MSBs) to complete the misaligned bus cycle.

Table 31. Bus-Cycle Order During Misaligned I/O Transfers

| Type of Access | First Cycle | Second Cycle |
|----------------|-------------|--------------|
| I/O Read | LSBs | MSBs |
| I/O Write | LSBs | MSBs |

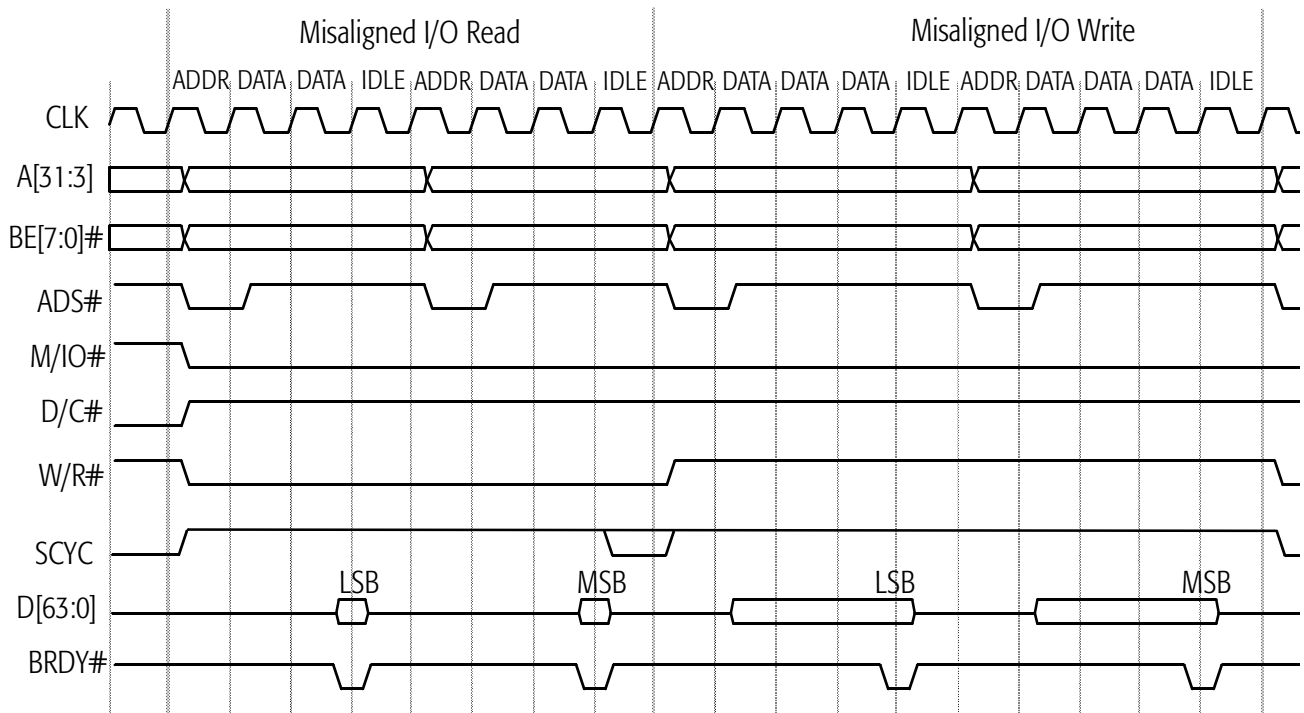


Figure 65. Misaligned I/O Transfer

7.5 Inquire and Bus Arbitration Cycles

The AMD-K6-III+ processor provides built-in level-one (L1) data and instruction caches, and a unified level-two (L2) cache. Each L1 cache is 32 Kbytes and two-way set-associative. The L2 cache is 256 Kbytes and four-way set-associative. The system logic or other bus master devices can initiate an inquire cycle to maintain cache/memory coherency. In response to the inquire cycle, the processor compares the inquire address with its cache tag addresses in all caches, and, if necessary, updates the MESI state of the cache line and performs writebacks to memory.

An inquire cycle can be initiated by asserting AHOLD, BOFF#, or HOLD. AHOLD is exclusively used to support inquire cycles. During AHOLD-initiated inquire cycles, the processor only floats the address bus. BOFF# provides the fastest access to the bus because it aborts any processor cycle that is in-progress, whereas AHOLD and HOLD both permit an in-progress bus cycle to complete. During HOLD-initiated and BOFF#-initiated inquire cycles, the processor floats all of its bus-driving signals.

The AMD-K6-III+ processor does not support system-initiated inquire cycles during the Enhanced Power Management (EPM) Stop Grant State. For more information on the EPM Stop Grant State, see “Clock Control” on page 277.

Hold and Hold Acknowledge Cycle

The system logic or another bus device can assert HOLD to initiate an inquire cycle or to gain full control of the bus. When the AMD-K6-III+ processor samples HOLD asserted, it completes any in-progress bus cycle and asserts HLDA to acknowledge release of the bus. The processor floats the following signals off the same clock edge on which HLDA is asserted:

- | | |
|------------|-----------|
| ■ A[31:3] | ■ DP[7:0] |
| ■ ADS# | ■ LOCK# |
| ■ AP# | ■ M/IO# |
| ■ BE[7:0]# | ■ PCD |
| ■ CACHE# | ■ PWT |
| ■ D[63:0] | ■ SCYC |
| ■ D/C# | ■ W/R# |

Figure 66 on page 169 shows a basic HOLD/HLDA operation. In this example, the processor samples HOLD asserted during the memory read cycle. It continues the current memory read cycle until BRDY# is sampled asserted. The processor drives HLDA and floats its outputs one clock edge after the last BRDY# of the cycle is sampled asserted. The system logic can assert HOLD for as long as it needs to utilize the bus. The processor samples HOLD on every clock edge but does not assert HLDA until any in-progress cycle or sequence of locked cycles is completed.

When the processor samples HOLD negated during a hold acknowledge cycle, it negates HLDA off the next clock edge. The processor regains control of the bus and can assert ADS# off the same clock edge on which HLDA is negated.

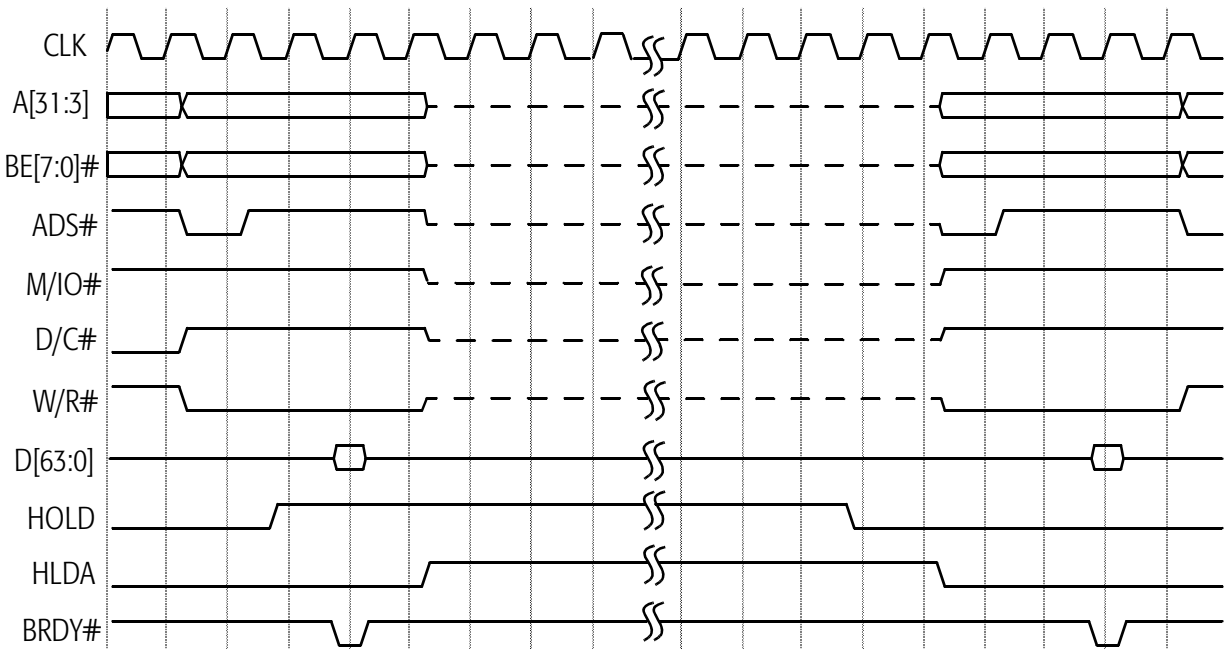


Figure 66. Basic HOLD/HLDA Operation

**HOLD-Initiated
Inquire Hit to Shared
or Exclusive Line**

Figure 67 on page 171 shows a HOLD-initiated inquire cycle. In this example, the processor samples HOLD asserted during the burst memory read cycle. The processor completes the current cycle (until the last expected BRDY# is sampled asserted), asserts HLDA and floats its outputs as described in “Hold and Hold Acknowledge Cycle” on page 168.

The system logic drives an inquire cycle within the hold acknowledge cycle. It asserts EADS#, which validates the inquire address on A[31:5]. If EADS# is sampled asserted before HOLD is sampled negated, the processor recognizes it as a valid inquire cycle.

In Figure 67, the processor asserts HIT# and negates HITM# on the clock edge after the clock edge on which EADS# is sampled asserted, indicating the current inquire cycle hit a shared or exclusive cache line. (Shared and exclusive cache lines have not been modified and do not need to be written back.) During an inquire cycle, the processor samples INV to determine whether the addressed cache line found in the processor’s caches transitions to the invalid state or the shared state. In this example, the processor samples INV asserted with EADS#, which invalidates the cache line.

The system logic can negate HOLD off the same clock edge on which EADS# is sampled asserted. The processor continues driving HIT# in the same state until the next inquire cycle. HITM# is not asserted unless HIT# is asserted.

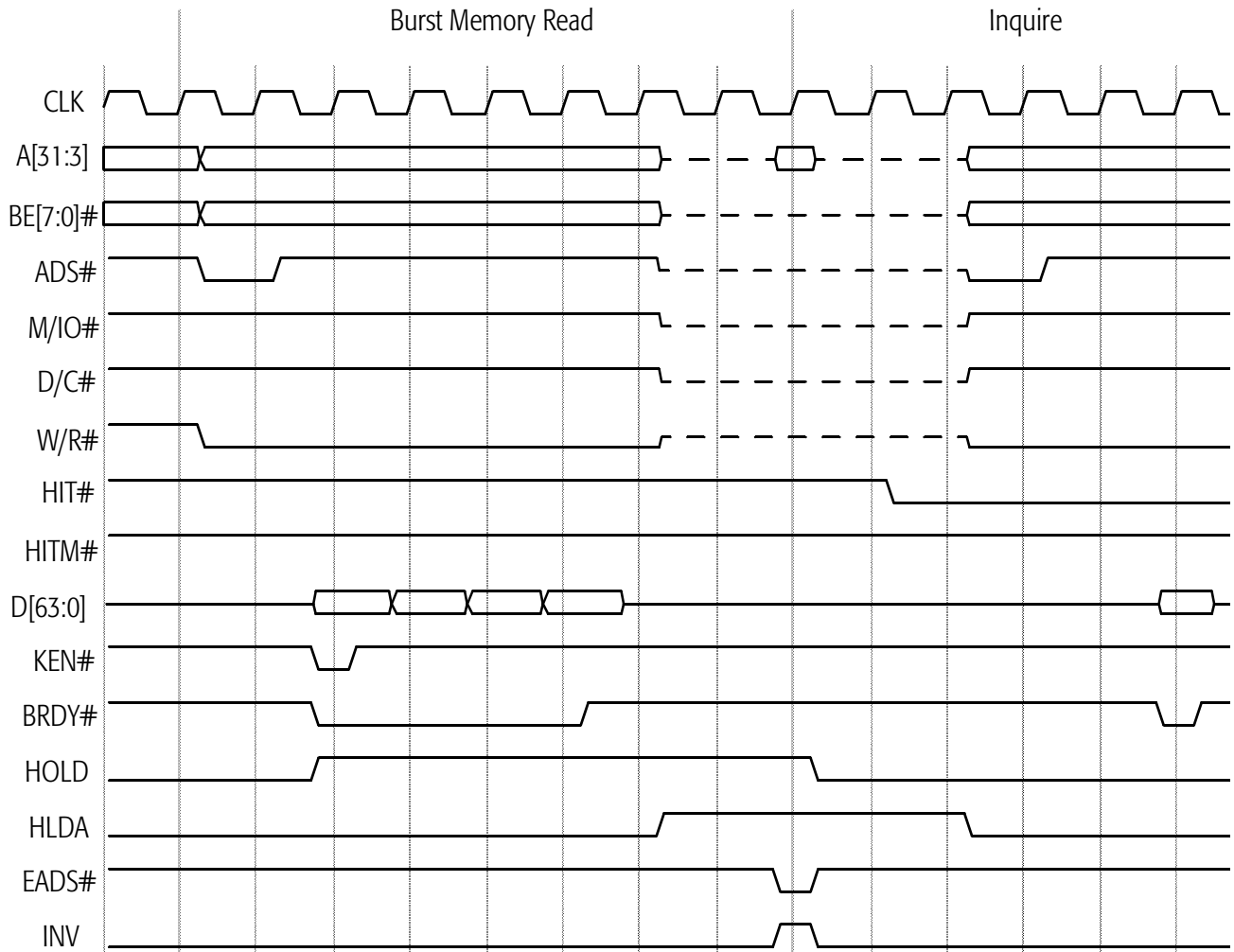


Figure 67. HOLD-Initiated Inquire Hit to Shared or Exclusive Line

**HOLD-Initiated
Inquire Hit to
Modified Line**

Figure 68 on page 173 shows the same sequence as Figure 67 on page 171, but in Figure 68 the inquire cycle hits a modified line and the processor asserts both HIT# and HITM#. In this example, the processor performs a writeback cycle immediately after the inquire cycle. It updates the modified cache line to external memory (normally, external cache or DRAM). The processor uses the address (A[31:5]) that was latched during the inquire cycle to perform the writeback cycle. The processor asserts HITM# throughout the writeback cycle and negates HITM# one clock edge after the last expected BRDY# of the writeback is sampled asserted.

When the processor samples EADS# during the inquire cycle, it also samples INV to determine the cache line MESI state after the inquire cycle. If INV is sampled asserted during an inquire cycle, the processor transitions the line (if found) to the invalid state, regardless of its previous state. The cache line invalidation operation is not visible on the bus. If INV is sampled negated during an inquire cycle, the processor transitions the line (if found) to the shared state. In Figure 68 the processor samples INV asserted during the inquire cycle.

In a HOLD-initiated inquire cycle, the system logic can negate HOLD off the same clock edge on which EADS# is sampled asserted. The processor drives HIT# and HITM# on the clock edge after the clock edge on which EADS# is sampled asserted.

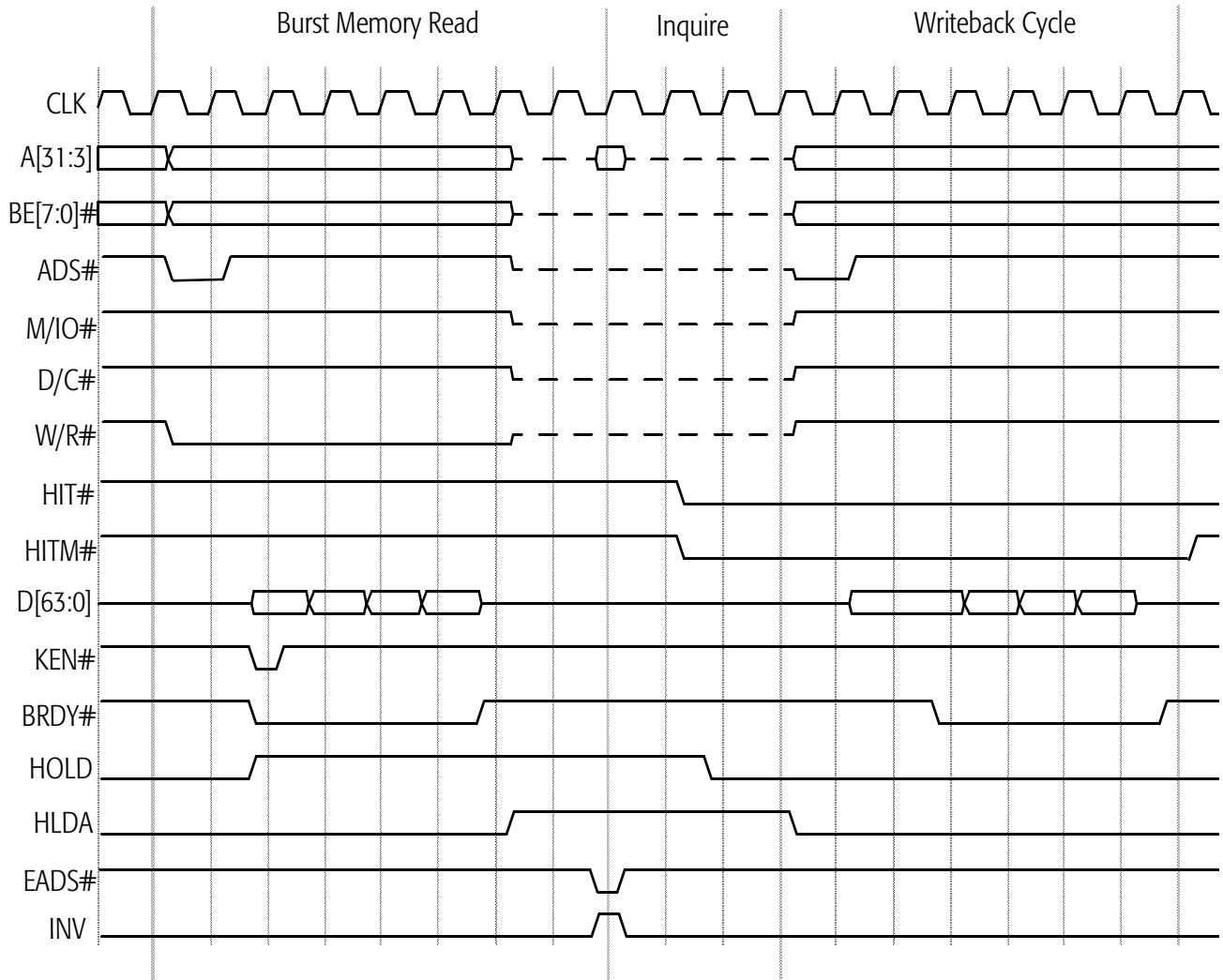


Figure 68. HOLD-Initiated Inquire Hit to Modified Line

**AHOLD-Initiated
Inquire Miss**

AHOLD can be asserted by the system to initiate one or more inquire cycles. To allow the system to drive the address bus during an inquire cycle, the processor floats A[31:3] and AP off the clock edge on which AHOLD is sampled asserted. The data bus and all other control and status signals remain under the control of the processor and are not floated. This functionality allows a bus cycle in progress when AHOLD is sampled asserted to continue to completion. The processor resumes driving the address bus off the clock edge on which AHOLD is sampled negated.

In Figure 69 on page 175, the processor samples AHOLD asserted during the memory burst read cycle, and it floats the address bus off the same clock edge on which it samples AHOLD asserted. While the processor still controls the bus, it completes the current cycle until the last expected BRDY# is sampled asserted. The system logic drives EADS# with an inquire address on A[31:5] during an inquire cycle. The processor samples EADS# asserted and compares the inquire address to its tag address in the L1 instruction and data caches, and in the L2 cache. In Figure 69, the inquire address misses the tag address in the processor (both HIT# and HITM# are negated). Therefore, the processor proceeds to the next cycle when it samples AHOLD negated. (The processor can drive a new cycle by asserting ADS# off the same clock edge that it samples AHOLD negated.)

For an AHOLD-initiated inquire cycle to be recognized, the processor must sample AHOLD asserted for at least two consecutive clocks before it samples EADS# asserted. If the processor detects an address parity error during an inquire cycle, APCHK# is asserted for one clock. The system logic must respond appropriately to the assertion of this signal.

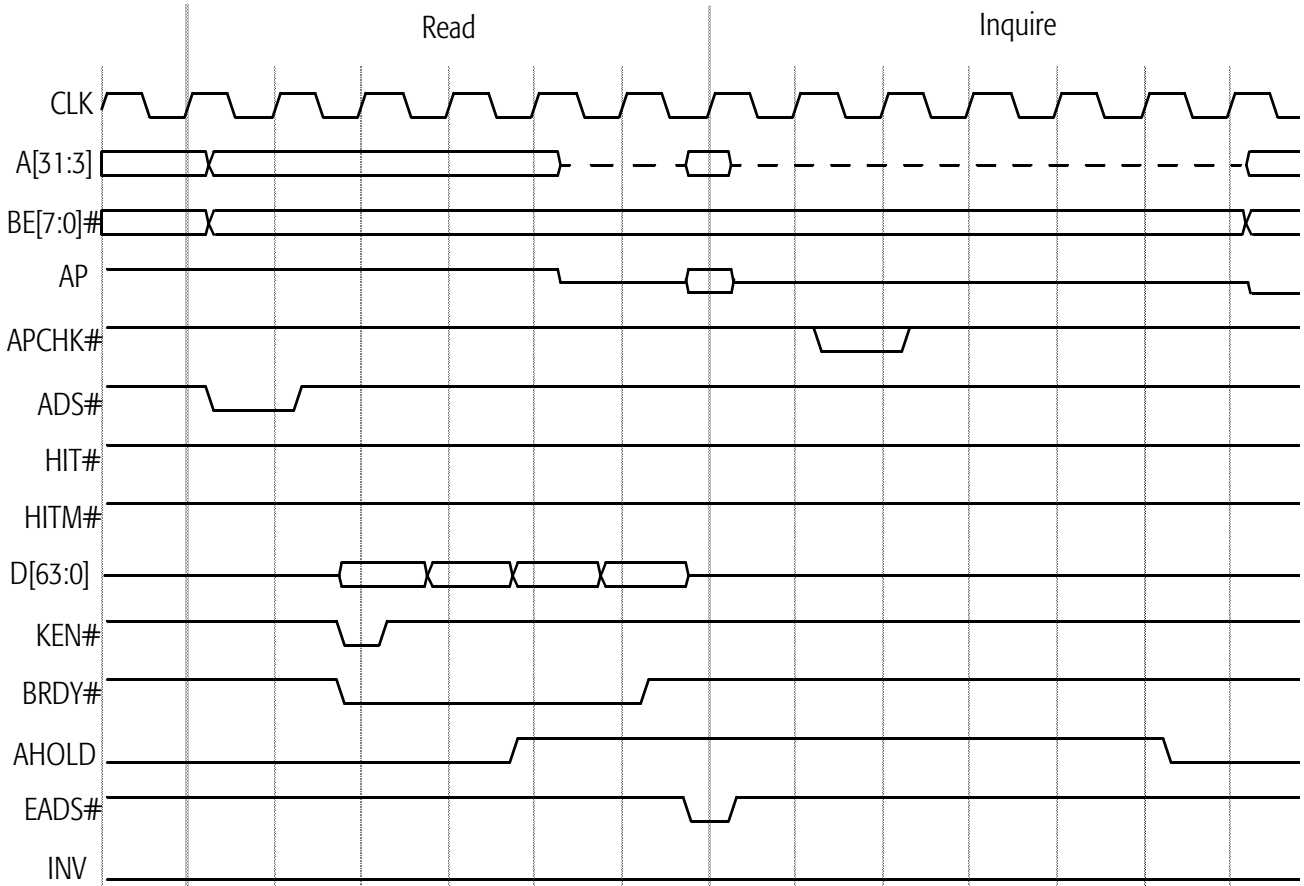


Figure 69. AHOLD-Initiated Inquire Miss

**AHOLD-Initiated
Inquire Hit to Shared
or Exclusive Line**

In Figure 70 on page 177, the processor asserts HIT# and negates HITM# off the clock edge after the clock edge on which EADS# is sampled asserted, indicating the current inquire cycle hits either a shared or exclusive line. (HIT# is driven in the same state until the next inquire cycle.) The processor samples INV asserted during the inquire cycle and transitions the line to the invalid state regardless of its previous state.

During an AHOLD-initiated inquire cycle, the processor samples AHOLD on every clock edge until it is negated. In Figure 70, the processor asserts ADS# off the same clock on which AHOLD is sampled negated. If the inquire cycle hits a modified line, the processor performs a writeback cycle before it drives a new bus cycle. The next section describes the AHOLD-initiated inquire cycle that hits a modified line.

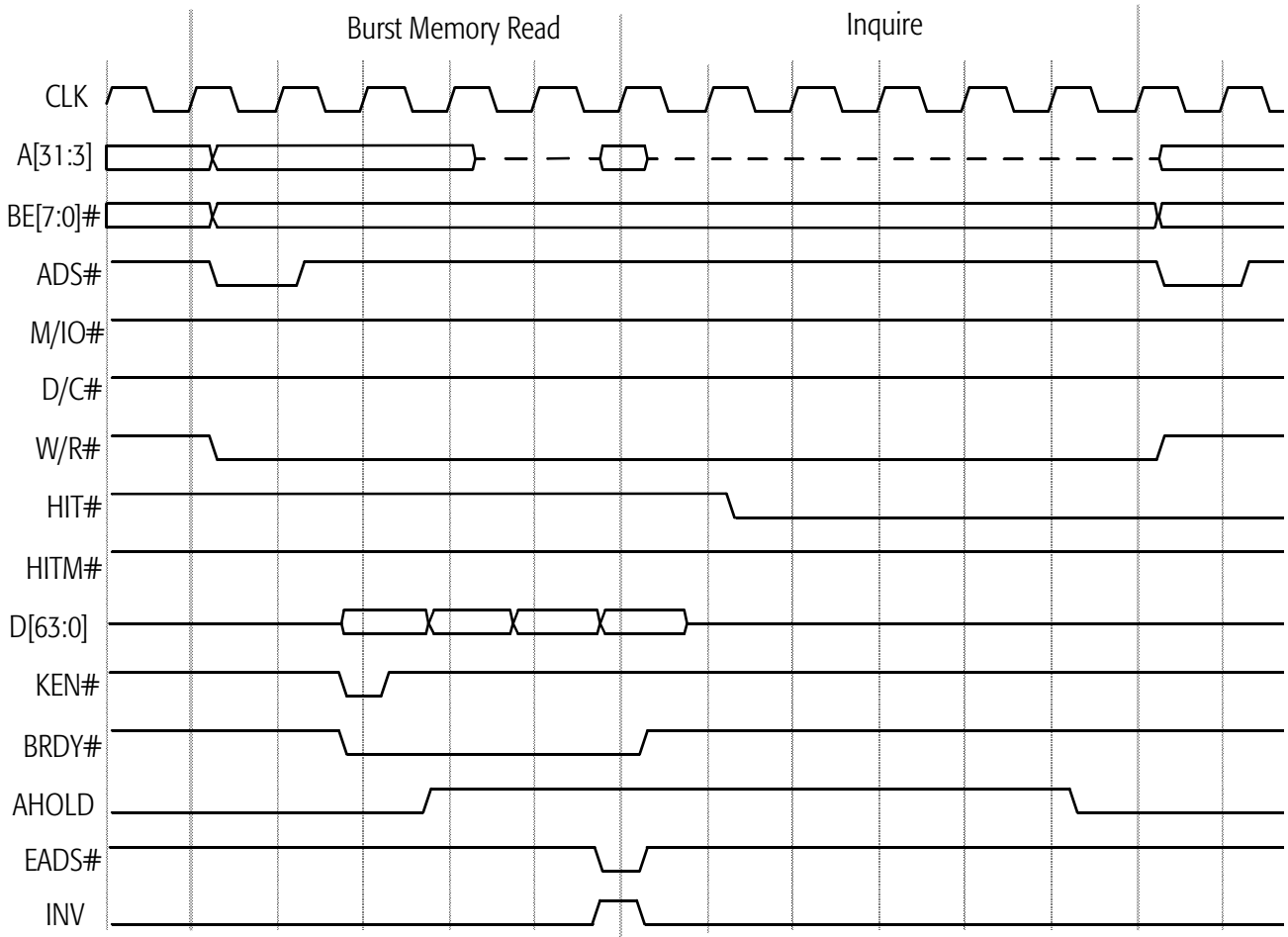


Figure 70. AHOLD-Initiated Inquire Hit to Shared or Exclusive Line

**AHOLD-Initiated
Inquire Hit to
Modified Line**

Figure 71 on page 179 shows an AHOLD-initiated inquire cycle that hits a modified line. During the inquire cycle in this example, the processor asserts both HIT# and HITM# on the clock edge after the clock edge that it samples EADS# asserted. This condition indicates that the cache line exists in the processor's L1 data cache or L2 cache in the modified state.

If the inquire cycle hits a modified line, the processor performs a writeback cycle immediately after the inquire cycle to update the modified cache line to shared memory (normally external cache or DRAM). In Figure 71, the system logic holds AHOLD asserted throughout the inquire cycle and the processor writeback cycle. In this case, the processor is not driving the address bus during the writeback cycle because AHOLD is sampled asserted. The system logic writes the data to memory by using its latched copy of the inquire cycle address. If the processor samples AHOLD negated before it performs the writeback cycle, it drives the writeback cycle by using the address (A[31:5]) that it latched during the inquire cycle.

If INV is sampled asserted during an inquire cycle, the processor transitions the line (if found) to the invalid state, regardless of its previous state (the cache invalidation operation is not visible on the bus). If INV is sampled negated during an inquire cycle, the processor transitions the line (if found) to the shared state. In either case, if the line is found in the modified state, the processor writes it back to memory before changing its state. Figure 71 shows that the processor samples INV asserted during the inquire cycle and invalidates the cache line after the inquire cycle.

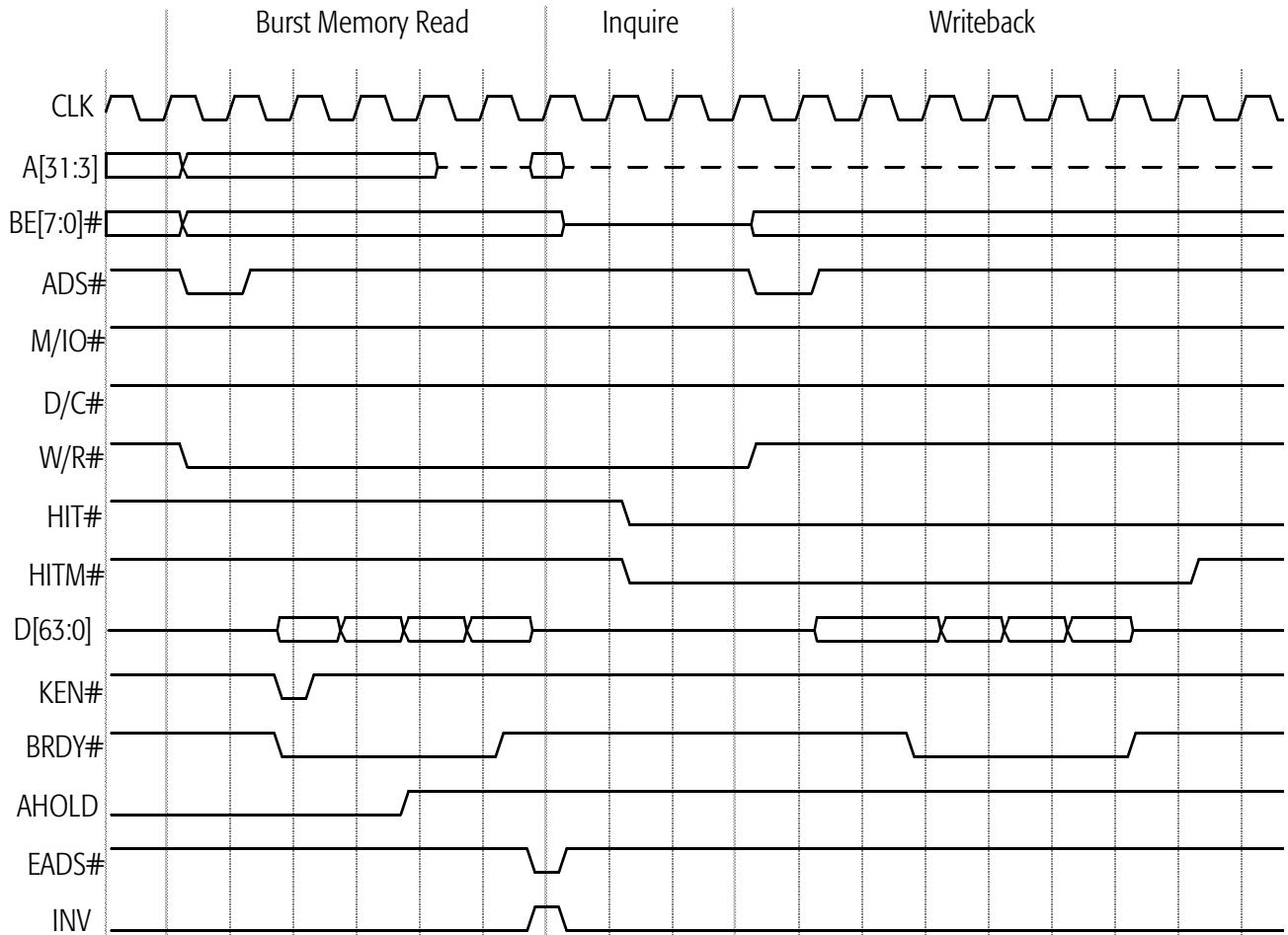


Figure 71. AHOLD-Initiated Inquire Hit to Modified Line

AHOLD Restriction

When the system logic drives an AHOLD-initiated inquire cycle, it must assert AHOLD for at least two clocks before it asserts EADS#. This requirement guarantees the processor recognizes and responds to the inquire cycle properly. The processor's 32 address bus drivers turn on almost immediately after AHOLD is sampled negated. If the processor switches the data bus (D[63:0] and DP[7:0]) during a write cycle off the same clock edge that switches the address bus (A[31:3] and AP), the processor switches 102 drivers simultaneously, which can lead to ground-bounce spikes. Therefore, before negating AHOLD the following restrictions must be observed by the system logic:

- When the system logic negates AHOLD during a write cycle, it must ensure that AHOLD is not sampled negated on the clock edge on which BRDY# is sampled asserted (See Figure 72 on page 181).
- When the system logic negates AHOLD during a writeback cycle, it must ensure that AHOLD is not sampled negated on the clock edge on which ADS# is negated (See Figure 72).
- When a write cycle is pipelined into a read cycle, AHOLD must not be sampled negated on the clock edge after the clock edge on which the last BRDY# of the read cycle is sampled asserted to avoid the processor simultaneously driving the data bus (for the pending write cycle) and the address bus off this same clock edge.

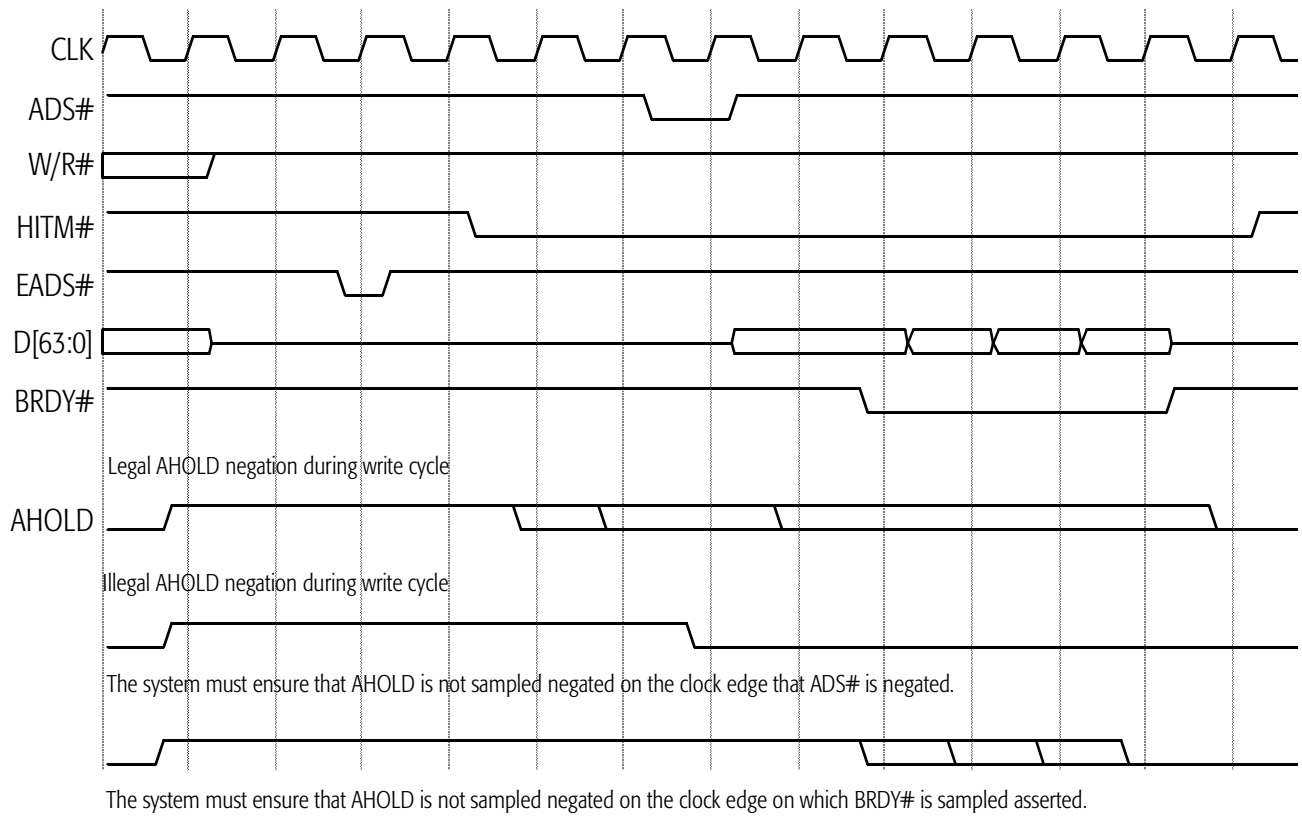


Figure 72. AHOLD Restriction

Bus Backoff (BOFF#)

BOFF# provides the fastest response among bus-hold inputs. Either the system logic or another bus master can assert BOFF# to gain control of the bus immediately. BOFF# is also used to resolve potential deadlock problems that arise as a result of inquire cycles. The processor samples BOFF# on every clock edge. If BOFF# is sampled asserted, the processor unconditionally aborts any cycles in progress and transitions to a bus hold state. (See “BOFF# (Backoff)” on page 102.) Figure 73 on page 183 shows a read cycle that is aborted when the processor samples BOFF# asserted even though BRDY# is sampled asserted on the same clock edge. The read cycle is restarted after BOFF# is sampled negated (KEN# must be in the same state during the restarted cycle as its state during the aborted cycle).

During a BOFF#-initiated inquire cycle that hits a shared or exclusive line, the processor samples BOFF# negated and restarts any bus cycle that was aborted when BOFF# was asserted. If a BOFF#-initiated inquire cycle hits a modified line, the processor performs a writeback cycle before it restarts the aborted cycle.

If the processor samples BOFF# asserted on the same clock edge that it asserts ADS#, ADS# is floated but the system logic may erroneously interpret ADS# as asserted. In this case, the system logic must properly interpret the state of ADS# when BOFF# is negated.

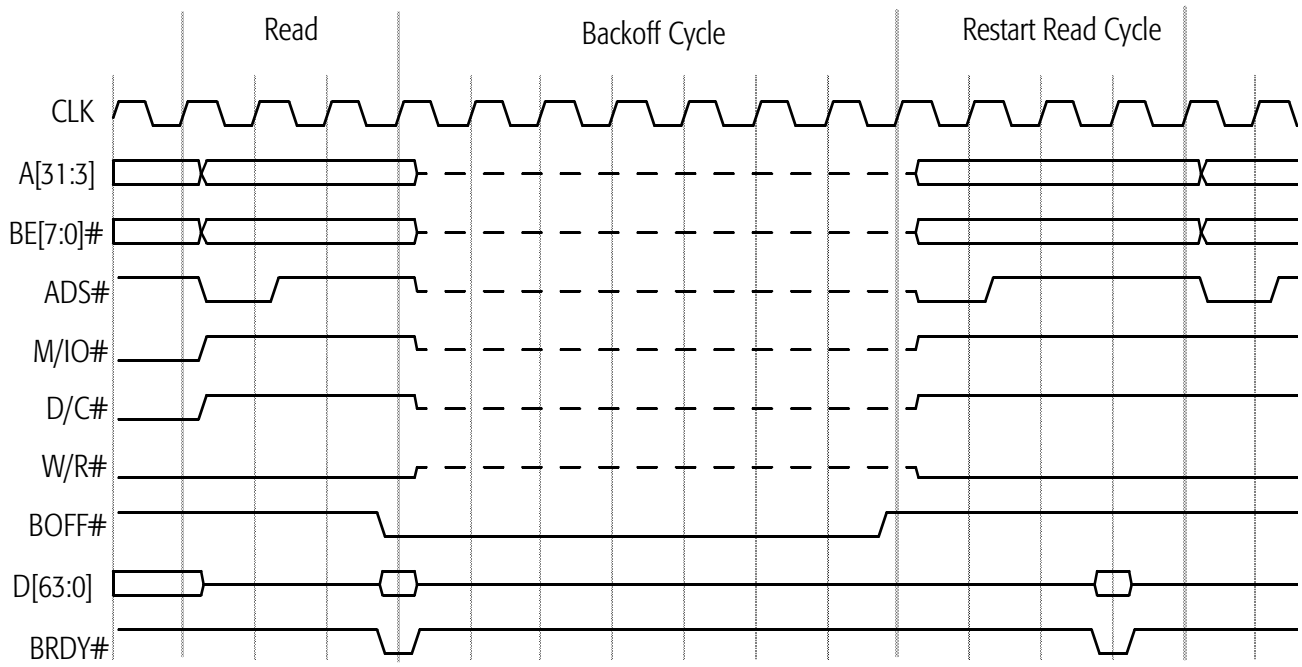


Figure 73. BOFF# Timing

Locked Cycles

The processor asserts LOCK# during a sequence of bus cycles to ensure the cycles are completed without allowing other bus masters to intervene. Locked operations can consist of two to five cycles. LOCK# is asserted during the following operations:

- An interrupt acknowledge sequence
- Descriptor Table accesses
- Page Directory and Page Table accesses
- XCHG instruction
- An instruction with an allowable LOCK prefix

In order to ensure that locked operations appear on the bus and are visible to the entire system, any data operands addressed during a locked cycle that reside in the processor's caches are flushed and invalidated from the caches prior to the locked operation. If the cache line is in the modified state, it is written back and invalidated prior to the locked operation. Likewise, any data read during a locked operation is not cached. The processor negates LOCK# for at least one clock between consecutive sequences of locked operations to allow the system logic to arbitrate for the bus.

The processor asserts SCYC during misaligned locked transfers on the D[63:0] data bus. The processor generates additional bus cycles to complete the transfer of misaligned data.

Basic Locked Operation

Figure 74 on page 185 shows a pair of read-write bus cycles. It represents a typical read-modify-write locked operation. The processor asserts LOCK# off the same clock edge that it asserts ADS# of the first bus cycle in the locked operation and holds it asserted until the last expected BRDY# of the last bus cycle in the locked operation is sampled asserted. (The processor negates LOCK# off of the same clock edge.)

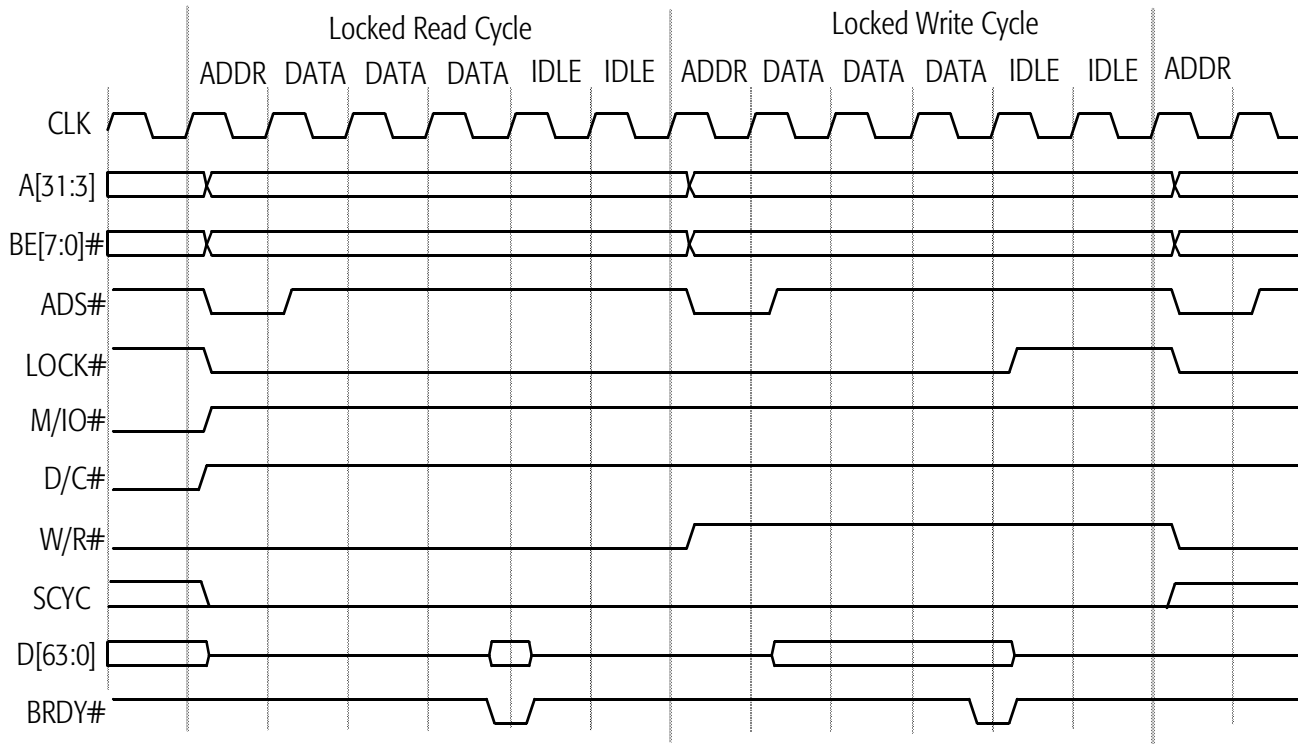


Figure 74. Basic Locked Operation

**Locked Operation
with BOFF#
Intervention**

Figure 75 on page 187 shows BOFF# asserted within a locked read-write pair of bus cycles. In this example, the processor asserts LOCK# with ADS# to drive a locked memory read cycle followed by a locked memory write cycle. During the locked memory write cycle in this example, the processor samples BOFF# asserted. The processor immediately aborts the locked memory write cycle and floats all its bus-driving signals, including LOCK#. The system logic or another bus master can initiate an inquire cycle or drive a new bus cycle one clock edge after the clock edge on which BOFF# is sampled asserted. If the system logic drives a BOFF#-initiated inquire cycle and hits a modified line, the processor performs a writeback cycle before it restarts the locked cycle (the processor asserts LOCK# during the writeback cycle).

In Figure 75, the processor immediately restarts the aborted locked write cycle by driving the bus off the clock edge on which BOFF# is sampled negated. The system logic must ensure the processor results for interrupted and uninterrupted locked cycles are consistent. That is, the system logic must guarantee the memory accessed by the processor is not modified during the time another bus master controls the bus.

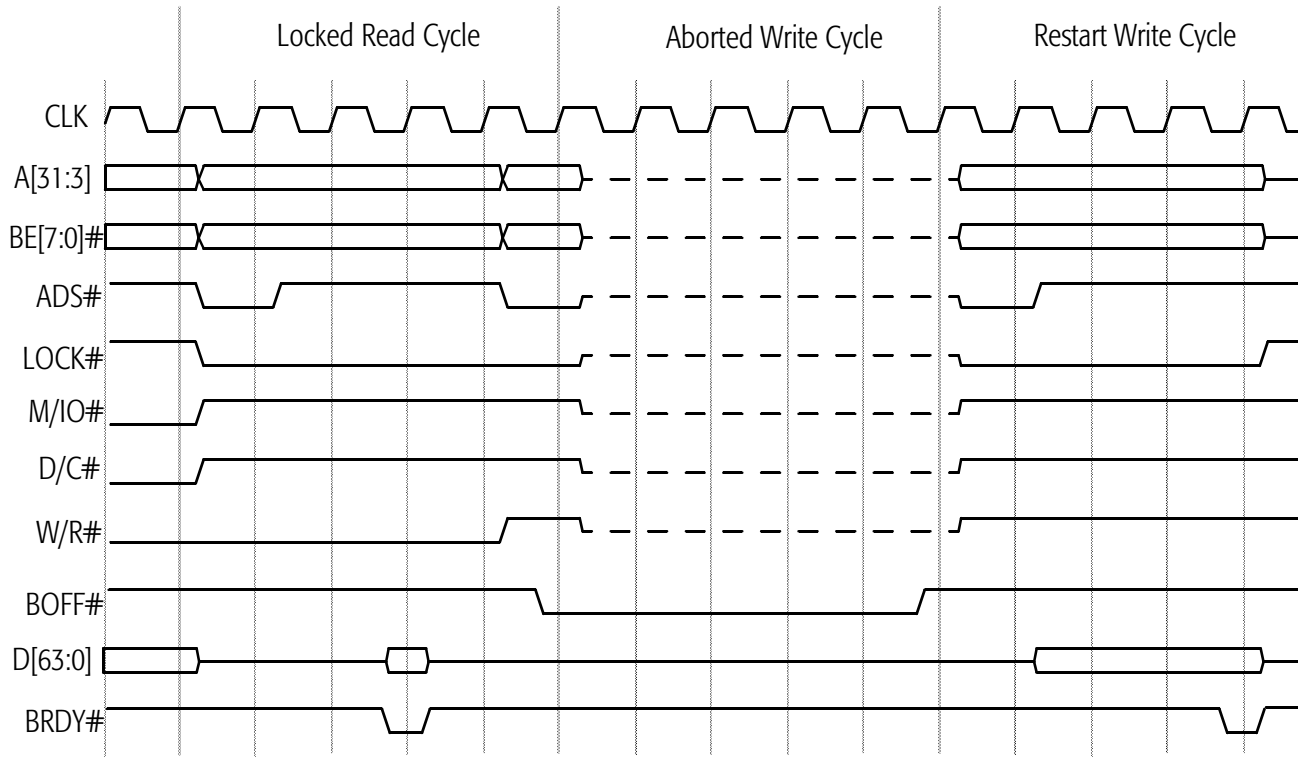


Figure 75. Locked Operation with BOFF# Intervention

Interrupt Acknowledge

In response to recognizing the system's maskable interrupt (INTR), the processor drives an interrupt acknowledge cycle at the next instruction boundary. During an interrupt acknowledge cycle, the processor drives a locked pair of read cycles as shown in Figure 76 on page 189. The first read cycle is not functional, and the second read cycle returns the interrupt number on D[7:0] (00h–FFh). Table 32 shows the state of the signals during an interrupt acknowledge cycle.

Table 32. Interrupt Acknowledge Operation Definition

| Processor Outputs | First Bus Cycle | Second Bus Cycle |
|-------------------|-----------------|---|
| D/C# | Low | Low |
| M/IO# | Low | Low |
| W/R# | Low | Low |
| BE[7:0]# | EFh | FEh (low byte enabled) |
| A[31:3] | 0000_0000h | 0000_0000h |
| D[63:0] | (ignored) | Interrupt number expected from interrupt controller on D[7:0] |

The system logic can drive INTR either synchronously or asynchronously. If it is asserted asynchronously, it must be asserted for a minimum pulse width of two clocks. To ensure it is recognized, INTR must remain asserted until an interrupt acknowledge sequence is complete.

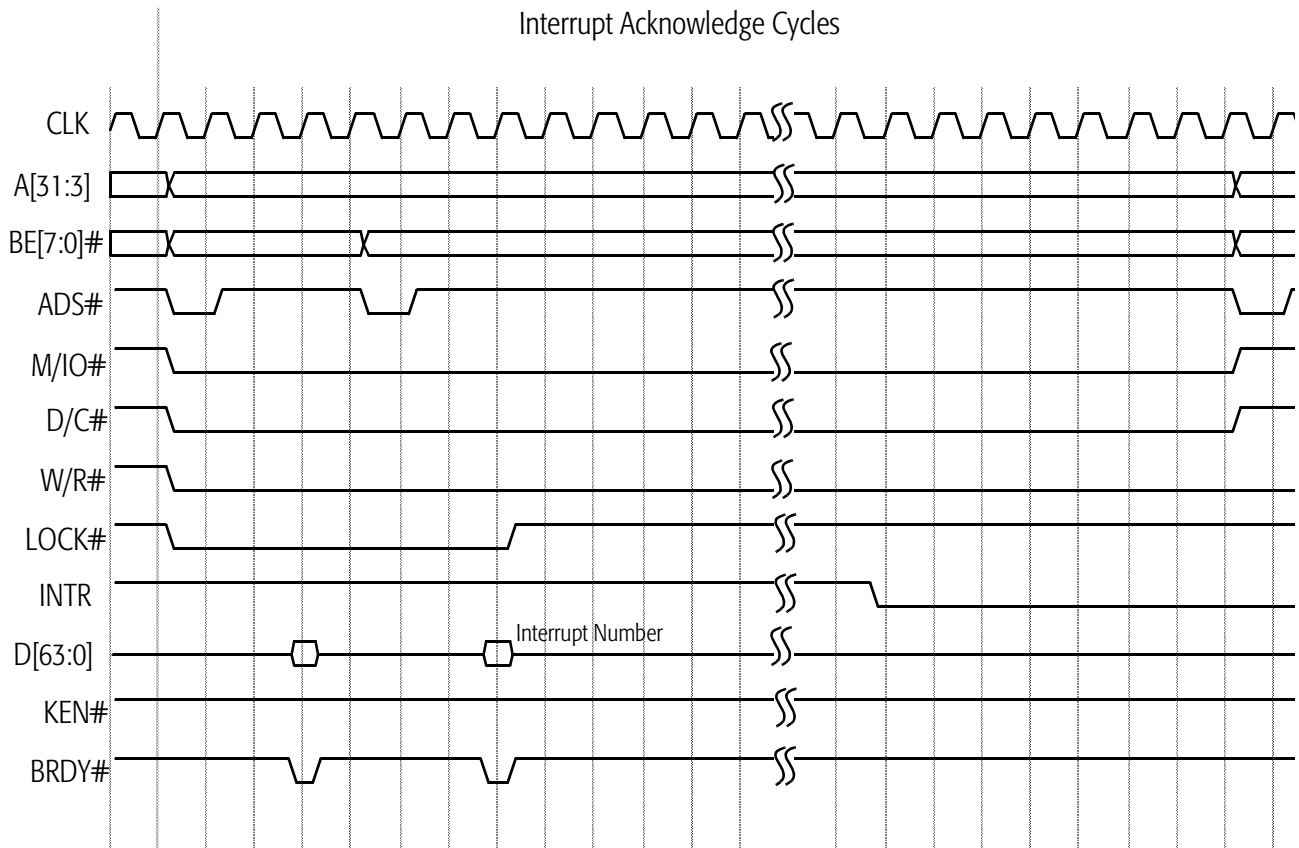


Figure 76. Interrupt Acknowledge Operation

7.6 Special Bus Cycles

The AMD-K6-III+ processor drives special bus cycles that include the following:

- Stop grant
- Enhanced power management
- Flush acknowledge
- Cache writeback invalidation
- Halt
- Cache invalidation
- Shutdown

During all special cycles, D/C# = 0, M/IO# = 0, and W/R# = 1. BE[7:0]# and A[31:3] are driven to differentiate among the special cycles, as shown in Table 33.

Note that the system logic must return BRDY# in response to all processor special cycles.

Table 33. Encodings for Special Bus Cycles

| BE[7:0]# | A[4:3] ¹ | Special Bus Cycle | Cause |
|----------|---------------------|-----------------------------|---|
| FBh | 10b | Stop Grant | STPCLK# sampled asserted |
| BFh | 00b | EPM Stop Grant ² | A dword access is made to the EPM 16-byte I/O block and the GSBC bit of the EPMR register is set to 1 |
| EFh | 00b | Flush Acknowledge | FLUSH# sampled asserted |
| F7h | 00b | Writeback | WBINVD instruction |
| FBh | 00b | Halt | HLT instruction |
| FDh | 00b | Flush | INVD,WBINVD instruction |
| FEh | 00b | Shutdown | Triple fault |

Notes:

1. A[31:5] = 0
2. Supported on the low-power versions only.

Basic Special Bus Cycle

Figure 77 on page 191 shows a basic special bus cycle.

The processor drives D/C# = 0, M/IO# = 0, and W/R# = 1 off the same clock edge that it asserts ADS#.

In this example, BE[7:0]# = FBh and A[31:3] = 0000_0000h, which indicates that the special cycle is a halt special cycle (See

Table 33). A halt special cycle is generated after the processor executes the HLT instruction.

If the processor samples FLUSH# asserted, it writes back any L1 data cache and L2 cache lines that are in the modified state and invalidates all lines in all caches. The processor then drives a flush acknowledge special cycle.

If the processor executes a WBINVD instruction, it drives a writeback special cycle after the processor completes invalidating and writing back the cache lines.

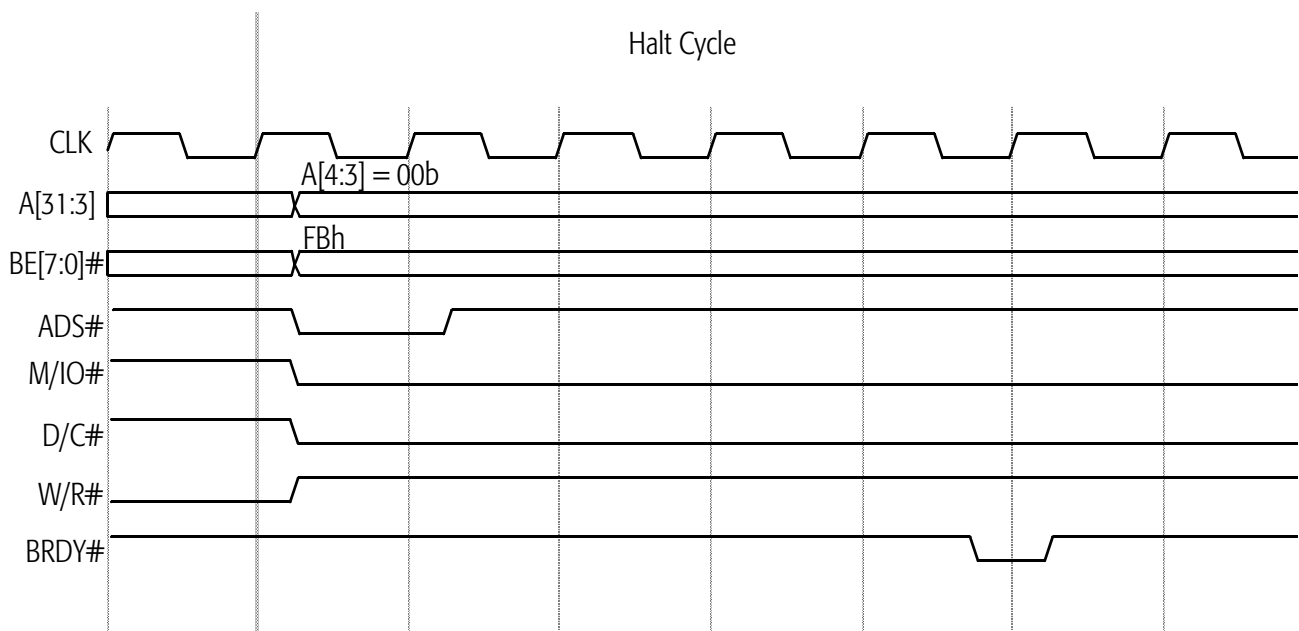


Figure 77. Basic Special Bus Cycle (Halt Cycle)

Shutdown Cycle

In Figure 78 on page 192, a shutdown (triple fault) occurs in the first half of the waveform, and a shutdown special cycle follows in the second half. The processor enters shutdown when an interrupt or exception occurs during the handling of a double fault (INT 8), which amounts to a triple fault. When the processor encounters a triple fault, it stops its activity on the bus and generates the shutdown special bus cycle (BE[7:0]# = FEh).

The system logic must assert NMI, INIT, RESET, or SMI# to get the processor out of the shutdown state.

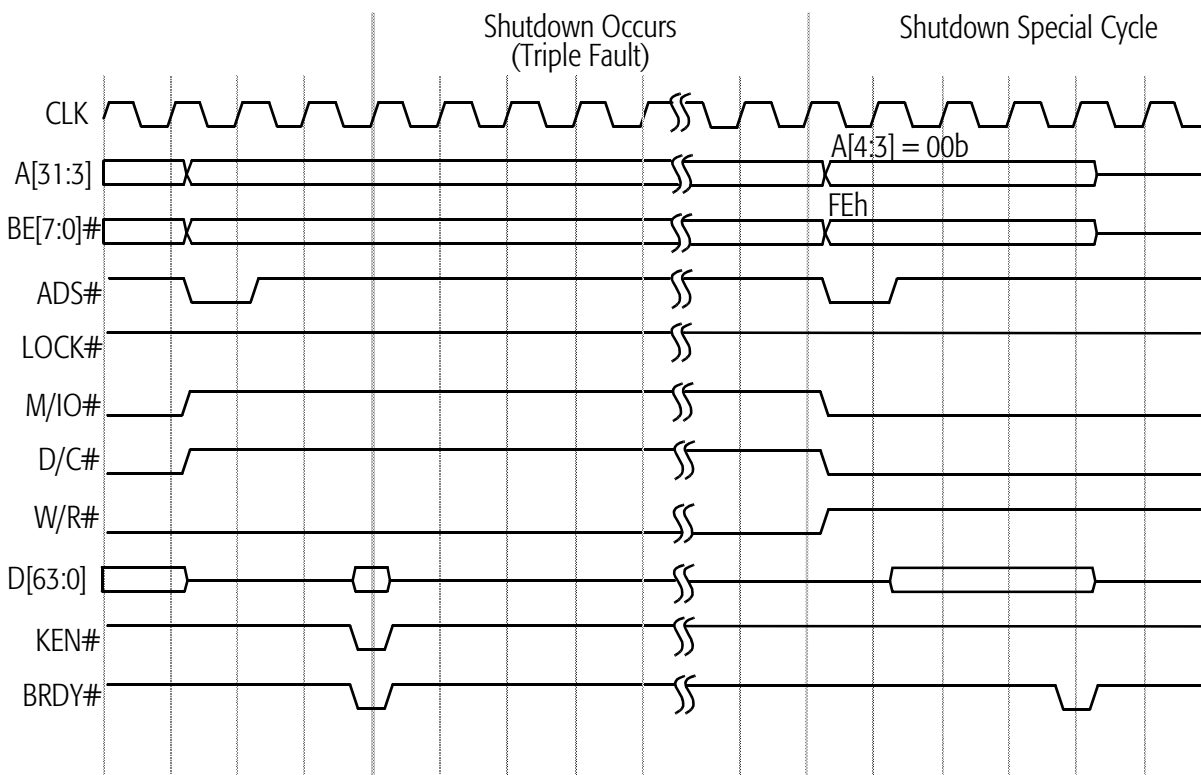


Figure 78. Shutdown Cycle

Stop Grant and Stop Clock States

Figure 79 on page 194 and Figure 80 on page 195 show the processor transition from normal execution to the Stop Grant state, then to the Stop Clock state, back to the Stop Grant state, and finally back to normal execution. The series of transitions begins when the processor samples STPCLK# asserted. On recognizing a STPCLK# interrupt at the next instruction retirement boundary, the processor performs the following actions, in the order shown:

1. Its instruction pipelines are flushed.
2. All pending and in-progress bus cycles are completed.
3. The STPCLK# assertion is acknowledged by executing a Stop Grant special bus cycle.
4. Its internal clock is stopped after BRDY# of the Stop Grant special bus cycle is sampled asserted (if EWBE# is masked off, then entry into the Stop Grant state is not affected by EWBE#) and after EWBE# is sampled asserted.
5. The Stop Clock state is entered if the system logic stops the bus clock CLK (optional).

STPCLK# is sampled as a level-sensitive input on every clock edge but is not recognized until the next instruction boundary. The system logic drives the signal either synchronously or asynchronously. If it is asserted asynchronously, it must be asserted for a minimum pulse width of two clocks. STPCLK# must remain asserted until recognized, which is indicated by the completion of the Stop Grant special cycle.

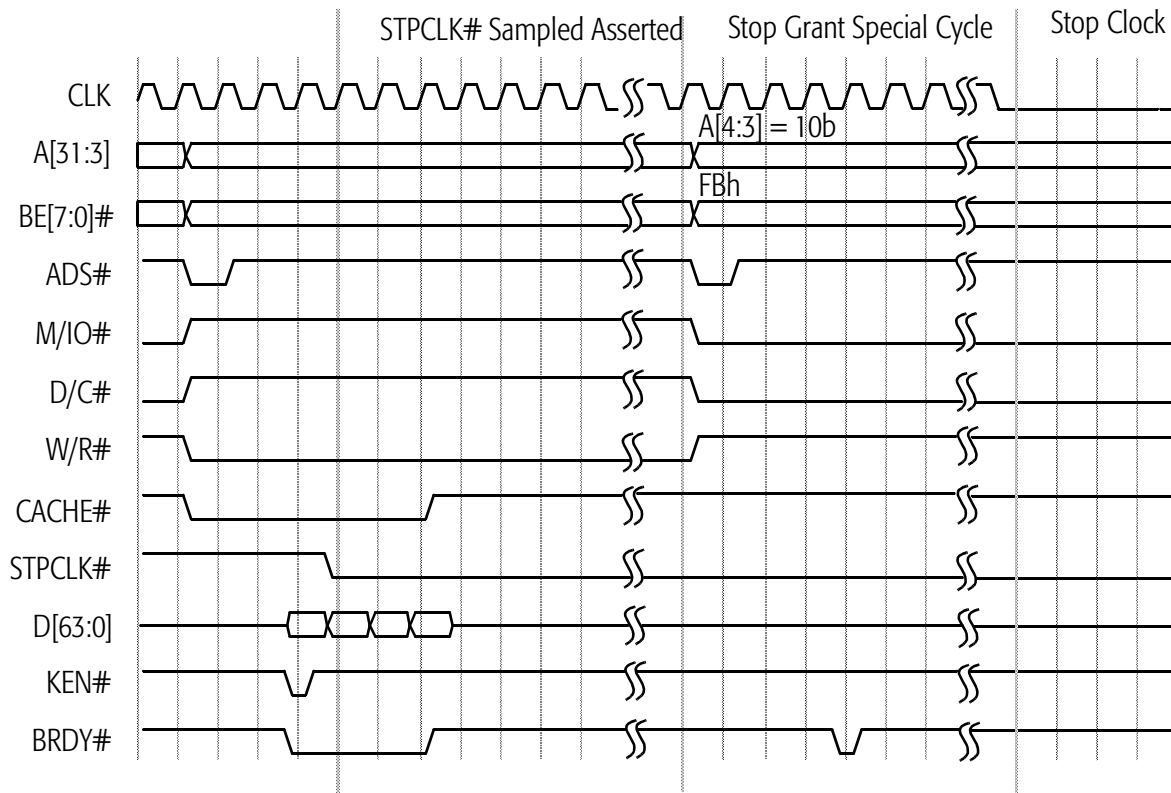


Figure 79. Stop Grant and Stop Clock Modes, Part 1

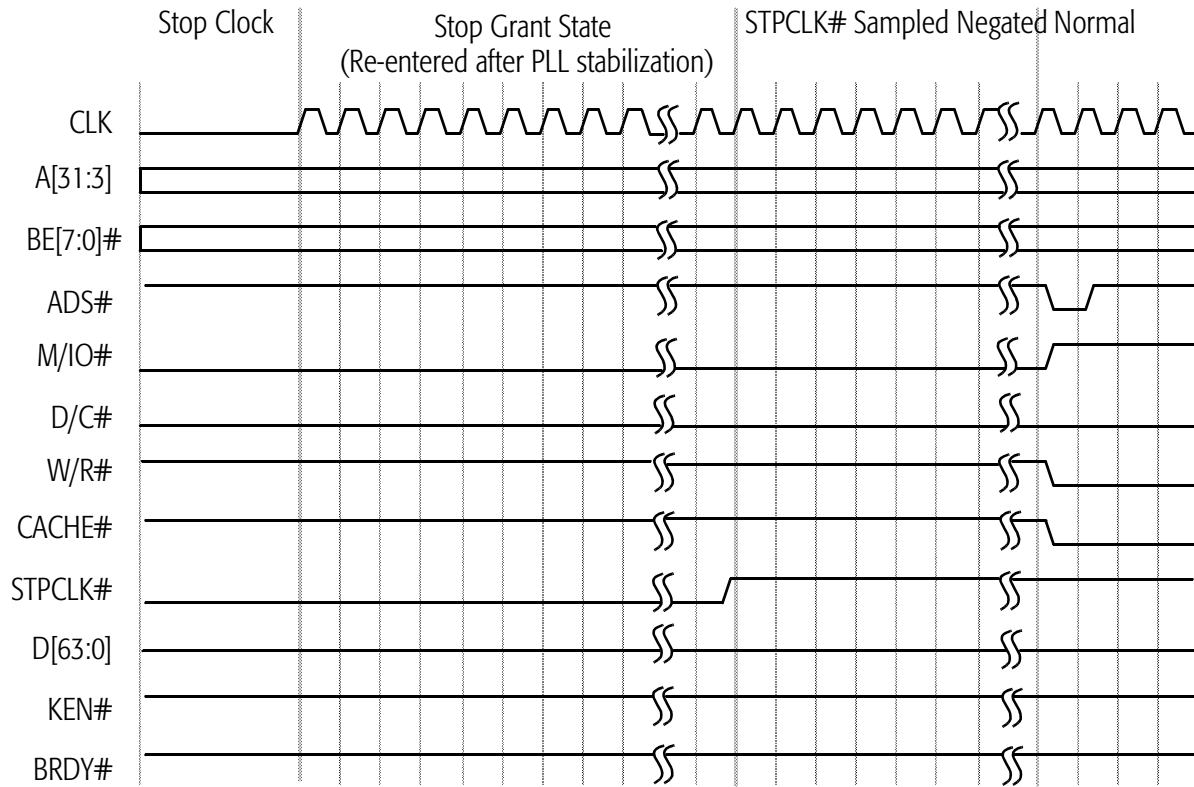


Figure 80. Stop Grant and Stop Clock Modes, Part 2

**INIT-Initiated
Transition from
Protected Mode to
Real Mode**

INIT is typically asserted in response to a BIOS interrupt that writes to an I/O port. This interrupt is often in response to a Ctrl-Alt-Del keyboard input. The BIOS writes to a port (similar to port 64h in the keyboard controller) that asserts INIT. INIT is also used to support 80286 software that must return to Real mode after accessing extended memory in Protected mode.

The assertion of INIT causes the processor to empty its pipelines, initialize most of its internal state, and branch to address FFFF_FFF0h—the same instruction execution starting point used after RESET. Unlike RESET, the processor preserves the contents of its caches, the floating-point state, the MMX state, Model-Specific Registers (MSRs), the CD and NW bits of the CR0 register, the time stamp counter, and other specific internal resources.

Figure 81 on page 197 shows an example in which the operating system writes to an I/O port, causing the system logic to assert INIT. The sampling of INIT asserted starts an extended microcode sequence that terminates with a code fetch from FFFF_FFF0h, the reset location. INIT is sampled on every clock edge but is not recognized until the next instruction boundary. During an I/O write cycle, it must be sampled asserted a minimum of three clock edges before BRDY# is sampled asserted if it is to be recognized on the boundary between the I/O write instruction and the following instruction. If INIT is asserted synchronously, it can be asserted for a minimum of one clock. If it is asserted asynchronously, it must have been negated for a minimum of two clocks, followed by an assertion of a minimum of two clocks.

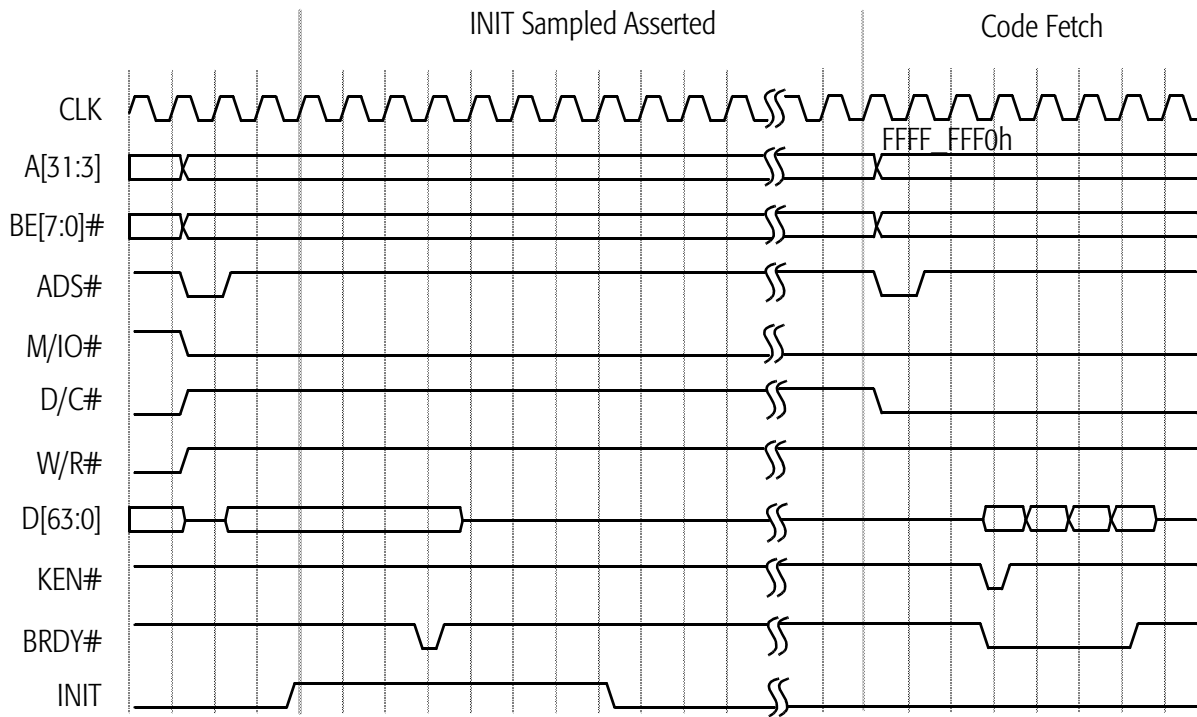


Figure 81. INIT-Initiated Transition from Protected Mode to Real Mode

8 Power-on Configuration and Initialization

On power-on the system logic must reset the AMD-K6-III+ processor by asserting the RESET signal. When the processor samples RESET asserted, it immediately flushes and initializes all internal resources and its internal state, including its pipelines and caches, the floating-point state, the MMX and 3DNow! states, and all registers. Then the processor jumps to address FFFF_FFF0h to start instruction execution.

8.1 Signals Sampled During the Falling Transition of RESET

FLUSH#

FLUSH# is sampled on the falling transition of RESET to determine if the processor begins normal instruction execution or enters Three-State Test mode.

- If FLUSH# is High during the falling transition of RESET, the processor unconditionally runs its Built-In Self Test (BIST), performs the normal reset functions, then jumps to address FFFF_FFF0h to start instruction execution. (See “Built-In Self-Test (BIST)” on page 251 for more details.)
- If FLUSH# is Low during the falling transition of RESET, the processor enters Three-State Test mode. (See “Three-State Test Mode” on page 252 and “FLUSH# (Cache Flush)” on page 112 for more details.)

BF[2:0]

The internal operating frequency of the processor is determined by the state of the bus frequency signals BF[2:0] when they are sampled during the falling transition of RESET. The frequency of the CLK input signal is multiplied internally by a ratio defined by BF[2:0]. (See “BF[2:0] (Bus Frequency)” on page 101 for the processor-clock to bus-clock ratios.)

8.2 RESET Requirements

During the initial power-on reset of the processor, RESET must remain asserted for a minimum of 1.0 ms after CLK and V_{CC} reach specification. (See “CLK Switching Characteristics” on page 298 for clock specifications. “Electrical Data” beginning on page 287 for V_{CC} specifications.)

During a warm reset while CLK and V_{CC} are within specification, RESET must remain asserted for a minimum of 15 clocks prior to its negation.

8.3 State of Processor After RESET

Output Signals

Table 34 shows the state of all processor outputs and bidirectional signals immediately after RESET is sampled asserted.

Table 34. Output Signal State After RESET

| Signal | State | Signal | State |
|------------------|----------|-----------------------|----------|
| A[31:3], AP | Floating | LOCK# | High |
| ADS#, ADSC# | High | M/IO# | Low |
| APCHK# | High | PCD | Low |
| BE[7:0]# | Floating | PCHK# | High |
| BREQ | Low | PWT | Low |
| CACHE# | High | SCYC | Low |
| D/C# | Low | SMIACK# | High |
| D[63:0], DP[7:0] | Floating | TDO | Floating |
| FERR# | High | VCC2DET | Low |
| HIT# | High | VCC2H/L# | Low |
| HITM# | High | VID[4:0] ¹ | 01010b |
| HLDA | Low | W/R# | Low |

Notes:

1. Supported on low-power versions only.

Registers

Table 35 on page 201 shows the state of all architecture registers and Model-Specific Registers (MSRs) after the processor has completed its initialization due to the recognition of the assertion of RESET.

Table 35. Register State After RESET

| Register | State (hex) |
|--------------------------------------|------------------------------|
| GDTR | base:0000_0000h limit:0FFFFh |
| IDTR | base:0000_0000h limit:0FFFFh |
| TR | 0000h |
| LDTR | 0000h |
| EIP | FFFF_FFF0h |
| EFLAGS | 0000_0002h |
| EAX ¹ | 0000_0000h |
| EBX | 0000_0000h |
| ECX | 0000_0000h |
| EDX ² | 0000_059Xh |
| ESI | 0000_0000h |
| EDI | 0000_0000h |
| EBP | 0000_0000h |
| ESP | 0000_0000h |
| CS | F000h |
| SS | 0000h |
| DS | 0000h |
| ES | 0000h |
| FS | 0000h |
| GS | 0000h |
| FPU Stack R7–R0 ³ | 0000_0000_0000_0000_0000h |
| FPU Control Word ³ | 0040h |
| FPU Status Word ³ | 0000h |
| FPU Tag Word ³ | 5555h |
| FPU Instruction Pointer ³ | 0000_0000_0000h |
| FPU Data Pointer ³ | 0000_0000_0000h |
| FPU Opcode Register ³ | 000_0000_0000b |
| CR0 ⁴ | 6000_0010h |
| CR2 | 0000_0000h |
| CR3 | 0000_0000h |
| CR4 | 0000_0000h |
| DR7 | 0000_0400h |
| DR6 | FFFF_0FF0h |
| DR3 | 0000_0000h |

Table 35. Register State After RESET (continued)

| Register | State (hex) |
|---------------------|----------------------|
| DR2 | 0000_0000h |
| DR1 | 0000_0000h |
| DR0 | 0000_0000h |
| MCAR ³ | 0000_0000_0000_0000h |
| MCTR ³ | 0000_0000_0000_0000h |
| TR12 ³ | 0000_0000_0000_0000h |
| TSC ³ | 0000_0000_0000_0000h |
| EFER ³ | 0000_0000_0000_0002h |
| STAR ³ | 0000_0000_0000_0000h |
| WHCR ³ | 0000_0000_0000_0000h |
| UWCCR ³ | 0000_0000_0000_0000h |
| PSOR ⁵ | 0000_0000_0000_01SBh |
| PFIR ^{3,5} | 0000_0000_0000_0000h |
| EPMR ^{3,6} | 0000_0000_0000_0000h |

Notes:

1. The contents of EAX indicate if BIST was successful. If EAX = 0000_0000h, BIST was successful. If EAX is non-zero, BIST failed.
2. EDX contains the AMD-K6-III+ processor signature, where X indicates the processor Stepping ID.
3. The contents of these registers are preserved following the recognition of INIT.
4. The CD and NW bits of CRO are preserved following the recognition of INIT.
5. "S" represents the Stepping. "B" represents PSOR[3:0], where PSOR[3] equals 0, and PSOR[2:0] is equal to the value of the BF[2:0] signals sampled during the falling transition of RESET.
6. Supported on low-power versions only.

8.4 State of Processor After INIT

The recognition of the assertion of INIT causes the processor to empty its pipelines, to initialize most of its internal state, and to branch to address FFFF_FFF0h—the same instruction execution starting point used after RESET.

Unlike RESET, the processor preserves the contents of its caches, the floating-point state, the MMX and 3DNow! states, MSRs, and the CD and NW bits of the CR0 register.

The edge-sensitive interrupts FLUSH# and SMI# are sampled and preserved during the INIT process and are handled accordingly after the initialization is complete. However, the processor resets any pending NMI interrupt upon sampling INIT asserted.

INIT can be used as an accelerator for 80286 code that requires a reset to exit from Protected mode back to Real mode.

9 Cache Organization

The following sections describe the basic architecture and resources of the AMD-K6-III+ processor internal caches.

The performance of the AMD-K6-III+ processor is enhanced by writeback level-one (L1) and level-two (L2) caches.

- The L1 cache is organized as separate 32-Kbyte instruction and data caches, each with two-way set associativity.
- The L2 cache is 256 Kbytes, and is organized as a unified, four-way set-associative cache (See Figure 82 on page 206).

The cache line size is 32 bytes, and lines are fetched from external memory using an efficient pipelined burst transaction.

As the L1 instruction cache is filled from the L2 cache or from external memory, each instruction byte is analyzed for instruction boundaries using predecode logic. Predecoding annotates each instruction byte in the L1 instruction cache with information that later enables the decoders to efficiently decode multiple instructions simultaneously.

Translation lookaside buffers (TLB) are used in conjunction with the L1 cache to translate linear addresses to physical addresses. The L1 instruction cache is associated with a 64-entry TLB, while the L1 data cache is associated with a 128-entry TLB.

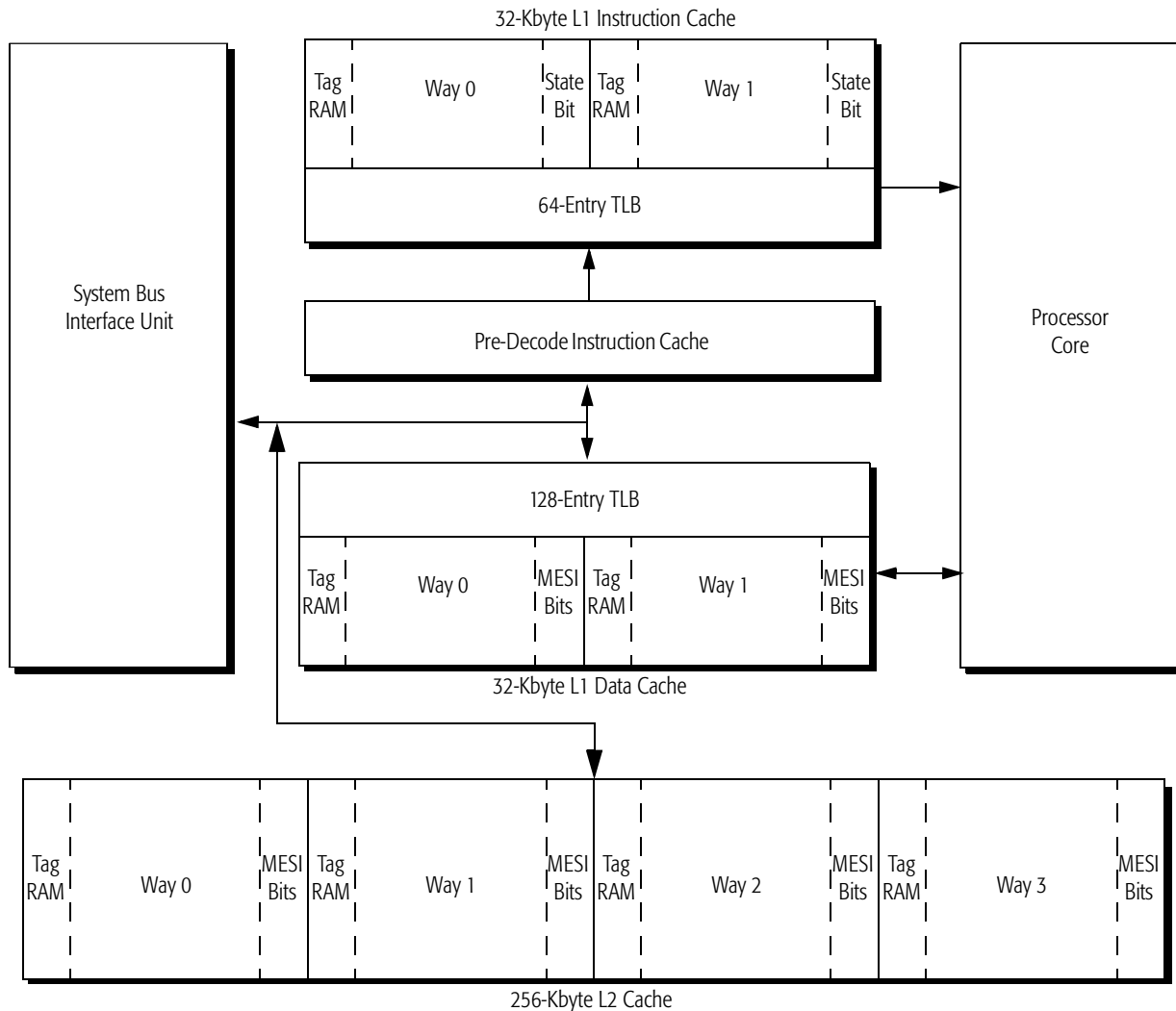


Figure 82. L1 and L2 Cache Organization for the AMD-K6™-III+ Processor

The processor cache design takes advantage of a sectored organization (See Figure 83). Each *sector* consists of 64 bytes configured as two 32-byte cache lines. The two cache lines of a sector share a common tag but have separate MESI (modified, exclusive, shared, invalid) bits that track the state of each cache line.

L1 Instruction Cache Line

| | | | | | | | | | | |
|---------|--------------|---------|----------------|---------|----------------|-------|-------|--------|----------------|------------|
| Tag | Cache Line 0 | Byte 31 | Predecode Bits | Byte 30 | Predecode Bits | | | Byte 0 | Predecode Bits | 1 MESI Bit |
| Address | Cache Line 1 | Byte 31 | Predecode Bits | Byte 30 | Predecode Bits | | | Byte 0 | Predecode Bits | 1 MESI Bit |

L1 Data Cache Line and L2 Cache Line

| | | | | | | | |
|---------|--------------|---------|---------|-------|-------|--------|-------------|
| Tag | Cache Line 0 | Byte 31 | Byte 30 | | | Byte 0 | 2 MESI Bits |
| Address | Cache Line 1 | Byte 31 | Byte 30 | | | Byte 0 | 2 MESI Bits |

Note: L1 instruction-cache lines have only two coherency states (valid or invalid) rather than the four MESI coherency states of L1 data-cache and L2 cache lines. Only two states are needed for the L1 instruction cache because these lines are read-only.

Figure 83. L1 Cache Sector Organization

9.1 MESI States in the L1 Data Cache and L2 Cache

The state of each line in the caches is tracked by the MESI bits. The coherency of these states or MESI bits is maintained by internal processor snoops and external inquire cycles by the system logic. The following four states are defined for the L1 data cache and the L2 cache:

- *Modified*—This line has been modified and is different from external memory.
- *Exclusive*—In general, an exclusive line in the L1 data cache or the L2 cache is not modified and is the same as external memory. The exception is the case where a line exists in the modified state in the L1 data cache and also resides in the L2 cache. By design, the line in the L2 cache must be in the exclusive state.
- *Shared*—If a cache line is in the shared state it means that the same line can exist in more than one cache system.
- *Invalid*—The information in this line is not valid.

9.2 Predecode Bits

Decoding x86 instructions is particularly difficult because the instructions vary in length, ranging from 1 to 15 bytes long. Predecode logic supplies the predecode bits associated with each instruction byte.

Predecode bits indicate the number of bytes to the start of the next x86 instruction. The predecode bits are passed with the instruction bytes to the decoders where they assist with parallel x86 instruction decoding. The predecode bits use memory separate from the 32-Kbyte L1 instruction cache. The predecode bits are stored in an extended L1 instruction cache alongside each x86 instruction byte as shown in Figure 83 on page 207.

The L2 cache does not store predecode bits. As an instruction cache line is fetched from the L2 cache, the predecode bits are generated and stored alongside the cache line in the L1 instruction cache in the same manner as if the cache line were fetched from the processor's system bus.

9.3 Cache Operation

The operating modes for the caches are configured by software using the not writethrough (NW) and cache disable (CD) bits of control register 0 (CR0 bits 29 and 30, respectively). These bits are used in all operating modes.

- When the CD and NW bits are both set to 0, the cache is fully enabled. This is the standard operating mode for the cache. If a L1 cache read miss occurs, the processor determines if the read hits the L2 cache, in which case the cache line is supplied from the L2 cache to the L1 cache. If a read misses both the L1 and the L2 caches, a line fill (32-byte burst read) on the system bus occurs in order to fetch the cache line. The cache line is then filled in both the L1 and the L2 caches. Write hits to the L1 and L2 caches are updated, while write misses and writes to shared lines cause external memory updates. Refer to Table 39 on page 221 for a summary of cache read and write cycles and the effect of these operations on the cache MESI state.

Note: A write allocate operation can modify the behavior of write misses to the caches. See "Write Allocate" on page 215.

The AMD-K6-III+ processor does not enforce any rules of inclusion or exclusion as part of the protocol defined for the L1 and L2 caches. However, there are certain restrictions imposed by design on the allowable MESI states of a cache line that exists in both the L1 cache and the L2 cache. Refer to Table 40 on page 225 for a list of the valid cache-line states allowed.

- When CD is set to 0 and NW is set to 1, an invalid mode of operation exists that causes a general protection fault to occur.
- When CD is set to 1 (disabled) and NW is set to 0, the cache fill mechanism is disabled but the contents of the cache are still valid. The processor reads from the caches if the read hits the L1 or the L2 cache. If a read misses both the L1 and the L2 caches, a line fill does not occur on the system bus. Write hits to the L1 or L2 cache are updated, while write misses and writes to shared lines cause external memory updates. If PWT is driven Low and WB/WT# is sampled High, a write hit to a shared line changes the cache-line state to exclusive.
- When the CD and NW bits are both set to 1, the cache is fully disabled. Even though the cache is disabled, the contents are not necessarily invalid. The processor reads from the caches if the read hits the L1 or the L2 cache. If a read misses both the L1 and the L2 caches, a line fill does not occur on the system bus. If a write hits the L1 or the L2 cache, the cache is updated but an external memory update does not occur. If a cache line is in the exclusive state during a write hit, the cache-line state is changed to modified. Cache lines in the shared state remain in the shared state after a write hit. Write misses access external memory directly.

The operating system can control the cacheability of a page. The paging mechanism is controlled by CR3, the Page Directory Entry (PDE), and the Page Table Entry (PTE). Within CR3, PDE, and PTE are Page Cache Disable (PCD) and Page Writethrough (PWT) bits. The values of the PCD and PWT bits used in Table 36 on page 210 and Table 37 on page 210 are taken from either the PTE or PDE. For more information on PCD and PWT, see “PCD (Page Cache Disable)” on page 124 and “PWT (Page Writethrough)” on page 126, respectively.

Table 36 describes how the PWT signal is driven based on the values of the PWT bits and the PG bit of CR0.

Table 36. PWT Signal Generation

| PWT Bit ¹ | PG Bit of CR0 | PWT Signal |
|----------------------|---------------|------------|
| 1 | 1 | High |
| 0 | 1 | Low |
| 1 | 0 | Low |
| 0 | 0 | Low |

Notes:

1. PWT is taken from PTE or PDE.

Table 37 describes how the PCD signal is driven based on the values of the CD bit of CR0, the PCD bits, and the PG bit of CR0.

Table 37. PCD Signal Generation

| CD Bit of CR0 | PCD Bit ¹ | PG Bit of CR0 | PCD Signal |
|---------------|----------------------|---------------|------------|
| 1 | X | X | High |
| 0 | 1 | 1 | High |
| 0 | 0 | 1 | Low |
| 0 | 1 | 0 | Low |
| 0 | 0 | 0 | Low |

Notes:

1. PCD is taken from PTE or PDE.

Table 38 describes how the CACHE# signal is driven based on the cycle type, the CI bit of TR12, the PCD signal, and the UWCCR model-specific register.

Table 38. CACHE# Signal Generation

| Cycle Type | CI Bit of TR12 | PCD Signal | Access Within WC/UC Range ¹ | CACHE# |
|-----------------------------|----------------|------------|--|--------|
| Writebacks | X | X | X | Low |
| Unlocked Reads | 0 | 0 | 0 | Low |
| Locked Reads | X | X | X | High |
| Single Writes | X | X | X | High |
| Any Cycle Except Writebacks | 1 | X | X | High |
| Any Cycle Except Writebacks | X | 1 | X | High |
| Any Cycle Except Writebacks | X | X | 1 | High |

Notes:

1. WC and UC refer to Write-Combining and Uncacheable Memory Ranges as defined in the UWCCR.

Cache-Related Signals

Complete descriptions of the signals that control cacheability and cache coherency are given on the following pages:

- CACHE#—page 105
- EADS#—page 109
- FLUSH#—page 112
- HIT#—page 113
- HITM#—page 113
- INV—page 118
- KEN#—page 119
- PCD—page 124
- PWT—page 126
- WB/WT#—page 139

9.4 Cache Disabling and Flushing

L1 and L2 Cache Disabling

To completely disable all accesses to the L1 and the L2 caches, the CD bit must be set to 1 and the caches must be completely flushed. There are three different methods for flushing the caches. The first method relies on the system logic and the other two methods rely on software.

- For the system logic to flush the caches, the processor must sample FLUSH# asserted. In this method, the processor writes back any L1 data cache and L2 cache lines that are in

the modified state, invalidates all lines in all caches, and then executes a flush acknowledge special cycle (See Table 24 on page 142).

- The second method for flushing the caches is for software to execute the WBINVD instruction, which causes all modified lines to first be written back to memory, then marks all cache lines as invalid. Alternatively, if writing modified lines back to memory is not necessary, the INVD instruction can be used to invalidate all cache lines.
- The third method for flushing the caches is to make use of the Page Flush/Invalidate Register (PFIR), which allows cache invalidation and optional flushing of a specific 4-Kbyte page from the linear address space (see “Page Flush/Invalidate Register (PFIR)” on page 223). Unlike the previous two methods of flushing the caches, this particular method requires the software to be aware of which specific pages must be flushed and invalidated.

L2 Cache Disabling

The L2 cache in the AMD-K6-III+ processor can be completely disabled by setting the L2 Disable (L2D) bit (EFER[4]) to 1 (see “Extended Feature Enable Register (EFER)” on page 47). If disabled in this manner, the processor does not access the L2 cache for any purpose, including allocations, read hits, write hits, snoops, inquire cycles, flushing, and read/write attempts by means of the L2AAR. (See “L2 Cache Testing” on page 213.) The L1 cache operation is not affected by disabling the L2 cache.

The L2D bit is provided for debug and testing purposes only. For normal operation and maximum performance, this bit must be set to 0, which is the default setting following reset.

The AMD-K6-III+ processor does not provide a method for disabling the L1 cache while the L2 cache remains enabled.

9.5 L2 Cache Testing

The AMD-K6-III+ processor provides the L2AAR MSR that allows for direct access to the L2 cache and L2 tag arrays. For more detailed information, refer to “L2 Cache and Tag Array Testing” on page 264.

9.6 Cache-Line Fills

The processor performs a cache-line fill for any area of system memory defined as cacheable. If an area of system memory is not explicitly defined as uncacheable by the software or system logic, or implicitly treated as uncacheable by the processor, then the memory access is assumed to be cacheable.

Software can prevent caching of certain pages by setting the PCD bit in the PDE or PTE. Additionally, software can define regions of memory as uncacheable or write combinable by programming the MTRRs in the UWCCR MSR (see “Memory Type Range Registers” on page 231). Write-combinable memory is defined as uncacheable.

The system logic also has control of the cacheability of bus cycles. If it determines the address is not cacheable, system logic negates the KEN# signal when asserting the first BRDY# or NA# of a cycle.

The processor does not cache certain memory accesses such as locked operations. In addition, the processor does not cache PDE or PTE memory reads in the L1 cache (referred to as *page table walks*). However, page table walks are cached in the L2 cache if the PDE or PTE is determined to be cacheable.

When the processor needs to read memory, the processor drives a read cycle onto the bus. If the cycle is cacheable, the processor asserts CACHE#. If the cycle is not cacheable, a non-burst, single-transfer read takes place. The processor waits for the system logic to return the data and assert a single BRDY# (See Figure 60 on page 159). If the cycle is cacheable, the processor executes a 32-byte burst read cycle. The processor expects a total of four BRDY# signals for a burst read cycle to take place (See Figure 62 on page 163).

Cache-line fills initiate 32-byte burst read cycles from memory on the system bus for the L1 instruction cache and the L1 data cache. All L1 cache-line fills supplied from the system bus are also filled in the L2 cache.

9.7 Cache-Line Replacements

As programs execute and task switches occur, some cache lines eventually require replacement.

When a cache miss occurs in the L1 cache, the required cache line is filled from either the L2 cache, if the cache line is present (L2 cache hit), or from external memory, if the cache line is not present (L2 cache miss). If the cache line is filled from external memory, the cache line is filled in both the L1 and the L2 caches.

Two forms of cache misses and associated cache fills can take place—a tag-miss cache fill and a tag-hit cache fill.

- In the case of a *tag-miss* cache fill, the level-one cache miss is due to a tag mismatch, in which case the required cache line is filled either from the level-two cache or from external memory, and the level-one cache line within the sector that was not required is marked as invalid.
- In the case of a *tag-hit* cache fill, the address matches the tag, but the requested cache line is marked as invalid. The required level-one cache line is filled from the level-two cache or from external memory, and the level-one cache line within the sector that is not required remains in the same cache state.

If a L1 data-cache line being filled replaces a modified line, the modified line is written back to the L2 cache if the cache line is present (L2 cache hit). By design, if a cache line is in the modified state in the L1 cache, this cache line can only exist in the L2 cache in the exclusive state. During the writeback, the L2 cache-line state is changed from exclusive to modified, and the writeback does not occur on the system bus. If the replacement writeback does not hit the L2 cache (L2 cache miss), then the modified L1 cache line is written back on the system bus, and the L2 cache is not updated. If the other cache line in this sector is in the modified state, it is also written back in the same manner.

L1 instruction-cache lines and L2 cache lines are replaced using a Least Recently Used (LRU) algorithm. If a line replacement is required, lines are replaced when read cache misses occur.

The L1 data cache uses a slightly different approach to line replacement. If a miss occurs, and a replacement is required, lines are replaced by using a Least Recently Allocated (LRA) algorithm.

9.8 Write Allocate

Write allocate, if enabled, occurs when the processor has a pending memory write cycle to a cacheable line and the line does not currently reside in the L1 data cache. If the line does not exist in the L2 cache, the processor performs a 32-byte burst read cycle on the system bus to fetch the data-cache line addressed by the pending write cycle. If the line does exist in the L2 cache, the data is supplied directly from the L2 cache, in which case a system bus cycle is not executed. The data associated with the pending write cycle is merged with the recently-allocated data-cache line and stored in the processor's L1 data cache. If the data-cache line was fetched from memory (because of a L2 cache miss), the data is stored, without modification, in the L2 cache. The final MESI state of the cache lines depends on the state of the WB/WT# and PWT signals during the burst read cycle and the subsequent L1 data cache write hit (See Table 39 on page 221 to determine the cache-line states and the access types following a cache write miss). If the L1 data cache line is stored in the modified state, then the same cache line is stored in the L2 cache in the exclusive state. If the L1 data cache line is stored in the shared state, then the same cache line is stored in the L2 cache in the shared state.

If a data-cache line fetch from memory is attempted because the write allocate misses the L2 cache, and KEN# is sampled negated, the processor does not perform an allocation. In this case, the pending write cycle is executed as a single write cycle on the system bus.

During write allocates that miss the L2 cache, a 32-byte burst read cycle is executed in place of a non-burst write cycle. While the burst read cycle generally takes longer to execute than the non-burst write cycle, performance gains are realized on subsequent write cycle hits to the write-allocated cache line.

Due to the nature of software, memory accesses tend to occur in proximity of each other (principle of locality). The likelihood of additional write hits to the write-allocated cache line is high.

Write allocates that hit the L2 cache increase performance by avoiding accesses to the system bus.

The following is a description of three mechanisms by which the AMD-K6-III+ processor performs write allocations. A write allocate is performed when any one or more of these mechanisms indicates that a pending write is to a cacheable area of memory.

Write to a Cacheable Page

Every time the processor completes a L1 cache line fill, the address of the page in which the cache line resides is saved in the Cacheability Control Register (CCR). The page address of subsequent write cycles is compared with the page address stored in the CCR. If the two addresses are equal, then the processor performs a write allocate because the page has already been determined to be cacheable.

When the processor performs a L1 cache line fill from a different page than the address saved in the CCR, the CCR is updated with the new page address.

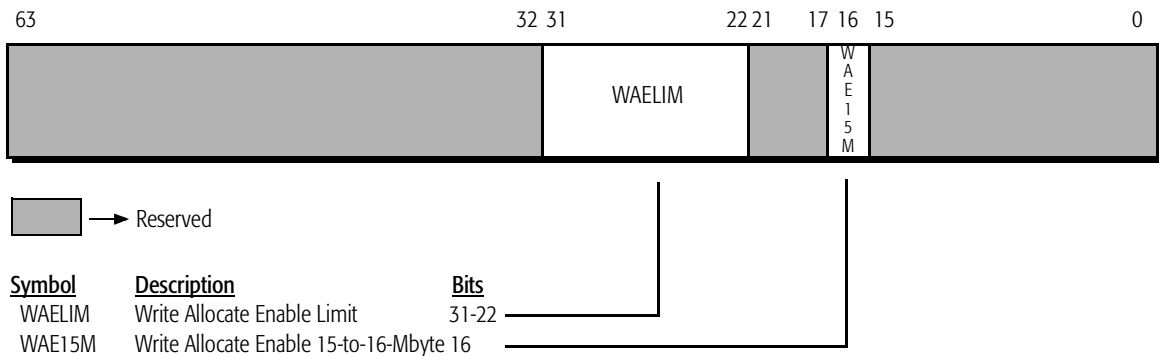
Write to a Sector

If the address of a pending write cycle matches the tag address of a valid L1 cache sector, but the addressed cache line within the sector is marked invalid (a sector hit but a cache line miss), then the processor performs a write allocate. The pending write cycle is determined to be cacheable because the sector hit indicates the presence of at least one valid cache line in the sector. The two cache lines within a sector are guaranteed by design to be within the same page.

Write Allocate Limit

The AMD-K6-III+ processor uses two mechanisms that are programmable within the Write Handling Control Register (WHCR) to enable write allocations for write cycles that address a definable area, or a special 1-Mbyte memory area.

The WHCR contains two fields—the Write Allocate Enable Limit (WAELIM) field, and the Write Allocate Enable 15-to-16-Mbyte (WAE15M) bit (see Figure 84).



Notes: Hardware RESET initializes this MSR to all zeros.

Figure 84. Write Handling Control Register (WHCR)

Write Allocate Enable Limit Field. The WAE15M field is 10 bits wide. This field, multiplied by 4 Mbytes, defines an upper memory limit. Any pending write cycle that misses the L1 cache and that addresses memory below this limit causes the processor to perform a write allocate (assuming the address is not within a range where write allocates are disallowed). Write allocate is disabled for memory accesses at and above this limit unless the processor determines a pending write cycle is cacheable by means of one of the other write allocate mechanisms—“Write to a Cacheable Page” and “Write to a Sector.” The maximum value of this limit is $((2^{10}-1) \cdot 4 \text{ Mbytes}) = 4092 \text{ Mbytes}$. When all the bits in this field are set to 0, all memory is above this limit and write allocates due to this mechanism is disabled (even if all bits in the WAE15M field are set to 0, write allocates can still occur due to the “Write to a Cacheable Page” and “Write to a Sector” mechanisms).

Write Allocate Enable 15-to-16-Mbyte Bit. The Write Allocate Enable 15-to-16-Mbyte (WAE15M) bit is used to enable write allocations for memory write cycles that address the 1 Mbyte of memory between 15 Mbytes and 16 Mbytes. This bit must be set to 1 to allow write allocate in this memory area. This bit is provided to account for a small number of uncommon memory-mapped I/O adapters that use this particular memory address space. If the system contains one of these peripherals, the bit should be set to 0 (even if the WAE15M bit is set to 0, write allocates can still occur between 15 Mbytes and 16 Mbytes due to the “Write to a Cacheable Page” and “Write to a

Sector” mechanisms). The WAE15M bit is ignored if the value in the WAELIM field is set to less than 16 Mbytes.

By definition a write allocate is not performed in the memory area between 640 Kbytes and 1 Mbyte unless the processor determines a pending write cycle is cacheable by means of one of the other write allocate mechanisms—“Write to a Cacheable Page” and “Write to a Sector.” It is not considered safe to perform write allocations between 640 Kbytes and 1 Mbyte (000A_0000h to 000F_FFFFh) because it is considered a noncacheable region of memory.

If a memory region is defined as write combinable or uncacheable by a MTRR, write allocates are not performed in that region.

Write Allocate Logic Mechanisms and Conditions

Figure 85 shows the logic flow for all the mechanisms involved with write allocate for memory bus cycles. The left side of the diagram (the text) describes the conditions that need to be true in order for the value of that line to be a 1. Items 1 to 4 of the diagram are related to general cache operation and items 5 to 10 are related to the write allocate mechanisms.

For more information about write allocate, see the *Implementation of Write Allocate in the K86™ Processors Application Note*, order# 21326.

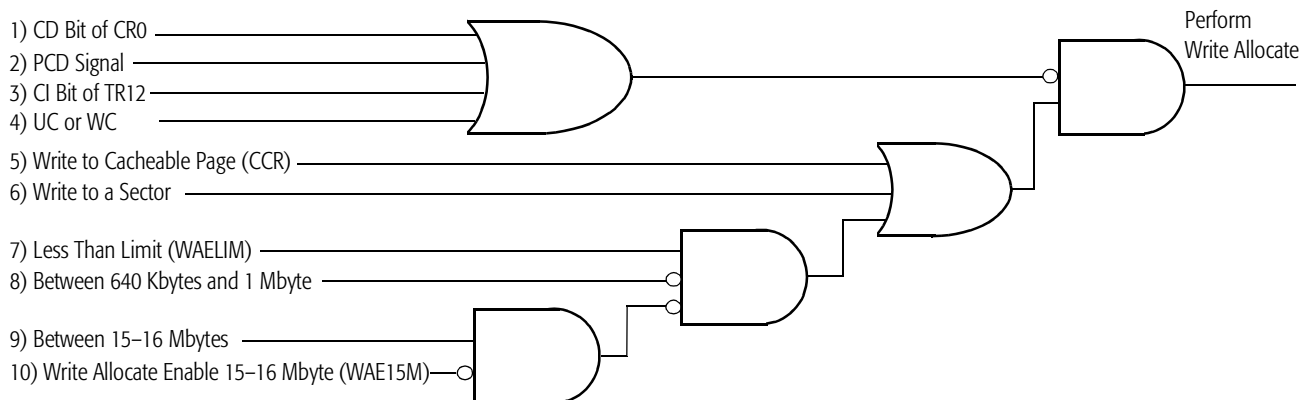


Figure 85. Write Allocate Logic Mechanisms and Conditions

The following list describes the corresponding items in Figure 85:

1. *CD Bit of CR0*—When the cache disable (CD) bit within control register 0 (CR0) is set to 1, the cache fill mechanism for both reads and writes is disabled and write allocate does not occur.
2. *PCD Signal*—When the PCD (page cache disable) signal is driven High, caching for that page is disabled, even if KEN# is sampled asserted, and write allocate does not occur.
3. *CI Bit of TR12*—When the cache inhibit bit of Test Register 12 is set to 1, L1 and L2 cache fills are disabled and write allocate does not occur.
4. *UC or WC*—If a pending write cycle addresses a region of memory defined as write combinable or uncacheable by an MTRR, write allocates are not performed in that region.
5. *Write to a Cacheable Page (CCR)*—A write allocate is performed if the processor knows that a page is cacheable. The CCR is used to store the page address of the last L1 cache fill for a read miss. See “Write to a Cacheable Page” on page 216 for a detailed description of this condition.
6. *Write to a Sector*—A write allocate is performed if the address of a pending write cycle matches the tag address of a valid L1 cache sector but the addressed cache line within the sector is invalid. See “Write to a Sector” on page 216 for a detailed description of this condition.
7. *Less Than Limit (WAELIM)*—The write allocate limit mechanism determines if the memory area being addressed is less than the limit set in the WAELIM field of WHCR. If the address is less than the limit, write allocate for that memory address is performed as long as conditions 8 through 10 do not prevent write allocate (even if conditions 8 and 10 attempt to prevent write allocate, condition 5 or 6 allows write allocate to occur).
8. *Between 640 Kbytes and 1 Mbyte*—Write allocate is not performed in the memory area between 640 Kbytes and 1 Mbyte. It is not considered safe to perform write allocations between 640 Kbytes and 1 Mbyte (000A_0000h to 000F_FFFFh) because this area of memory is considered a noncacheable region of memory (even if condition 8 attempts to prevent write allocate, condition 5 or 6 allows write allocate to occur).

9. *Between 15–16 Mbytes*—If the address of a pending write cycle is in the 1 Mbyte of memory between 15 Mbytes and 16 Mbytes, and the WAE15M bit is set to 1, write allocate for this cycle is enabled.
10. *Write Allocate Enable 15–16 Mbytes (WAE15M)*—This condition is associated with the Write Allocate Limit mechanism and affects write allocate only if the limit specified by the WAELIM field is greater than or equal to 16 Mbytes. If the memory address is between 15 Mbytes and 16 Mbytes, and the WAE15M bit in the WHCR is set to 0, write allocate for this cycle is disabled (even if condition 10 attempts to prevent write allocate, condition 5 or 6 allows write allocate to occur).

9.9 Prefetching

Hardware Prefetching

The AMD-K6-III+ processor conditionally performs cache prefetching, which results in the filling of the required cache line first, and a prefetch of the second cache line making up the other half of the sector. From the perspective of the external bus, the two cache-line fills typically appear as two 32-byte burst read cycles occurring back-to-back or, if allowed, as pipelined cycles. The burst read cycles do not occur back-to-back (wait states occur) if the processor is not ready to start a new cycle, if higher priority data read or write requests exist, or if NA# (next address) was sampled negated. Wait states can also exist between burst cycles if the processor samples AHOLD or BOFF# asserted.

Software Prefetching

The 3DNow! technology includes an instruction called PREFETCH that allows a cache line to be prefetched into the L1 data cache and the L2 cache. Unlike prefetching under hardware control, software prefetching only fetches the cache line specified by the operand of the PREFETCH instruction, and does not attempt to fetch the other cache line in the sector. The PREFETCH instruction format is defined in Table 15, “3DNow!™ Instructions,” on page 89. For more detailed information, see the *3DNow!™ Technology Manual*, order# 21928.

9.10 Cache States

Table 39 shows all the possible cache-line states before and after program-generated accesses to individual cache lines.

Table 39. L1 and L2 Cache States for Read and Write Accesses

| Type | | Cache State Before Access ¹ | | Access Type | Cache State After Access | |
|-------------|--------------------------------|--|----|---|-----------------------------------|---------------------|
| | | L1 | L2 | | MESI State ² | |
| | | | | | L1 | L2 |
| Cache Read | Read Miss L1, Read Miss L2 | I | I | Single read from bus | I | I |
| | | I | I | Burst read from bus, fill L1 and L2 ³ | S or E ⁴ | S or E ⁴ |
| | Read Hit L1 | E | – | – | E | – |
| | | S | – | – | S | – |
| | | M | – | – | M | – |
| | Read Miss L1, Read Hit L2 | I | E | Fill L1 | E | E |
| | | I | S | Fill L1 | S | S |
| | | I | M | Fill L1 | M | E |
| | | | I | M | Fill L1 | E ⁵ |
| Cache Write | Write Miss L1 Write Miss L2 | I | I | Single write to bus ⁶ | I | I |
| | | I | I | Burst read from bus, fill L1 and L2, write to L1 ⁷ | M ⁸ | E ⁸ |
| | | I | I | Burst read from bus, fill L1 and L2, write to L1 and L2, single write to bus ⁷ | S ⁹ | S ⁹ |
| | Write Hit L1 | S | I | Write to L1, single write to bus | S or E ⁴ | I |
| | | S | S | Write to L1 and L2, single write to bus | S or E ⁴ | S or E ⁴ |
| | | E or M | – | Write to L1 | M | – |
| | Write Miss L1 Write Hit L2 | I | E | Write to L2 ⁶ | I | M |
| | | I | S | Write to L2, single write to bus ⁶ | I | S or E ⁴ |
| | | I | M | Write to L2 ⁶ | I | M |
| | | I | E | Fill L1, write to L1 ⁷ | M | E |
| | | I | S | Write to L2, single write to bus ⁷ | S or E ⁴ | S or E ⁴ |
| | | | I | M | Fill L1, write to L1 ⁷ | M |

Notes:

1. M = Modified, E = Exclusive, S = Shared, I = Invalid. The exclusive and shared states are indistinguishable in the L1 instruction cache and are treated as “valid” states.
2. The final MESI state assumes that the state of the WB/WT# signal remains the same for all accesses to a particular cache line.
3. If CACHE# is driven Low and KEN# is sampled asserted.
4. If PWT is driven Low and WB/WT# is sampled High, the line is cached in the exclusive (writeback) state. If PWT is driven High or WB/WT# is sampled Low, the line is cached in the shared (writethrough) state.

5. This entry only applies to the L1 instruction cache. By design, a cache line cannot exist in the exclusive state in the L1 data cache and in the modified state in the L2 cache.
6. Assumes the write allocate conditions as specified in "Write Allocate" on page 215 are not met.
7. Assumes the write allocate conditions as specified in "Write Allocate" on page 215 are met.
8. Assumes PWT is driven Low and WB/WT# is sampled High.
9. Assumes PWT is driven High or WB/WT# is sampled Low.
- Not applicable or none.

9.11 Cache Coherency

Different ways exist to maintain coherency between the system memory and cache memories. Inquire cycles, internal snoops, FLUSH#, WBINVD, INV, and line replacements all prevent inconsistencies between memories.

Inquire Cycles

Inquire cycles are bus cycles initiated by system logic that ensure coherency between the caches and main memory. In systems with multiple bus masters, system logic maintains cache coherency by driving inquire cycles to the processor. System logic initiates inquire cycles by asserting AHOLD, BOFF#, or HOLD to obtain control of the address bus and then driving EADS#, INV (optional), and an inquire address (A[31:5]).

This type of bus cycle causes the processor to compare the tags for its L1 instruction and L1 data caches, and L2 cache, with the inquire address.

- If there is a hit to a shared or exclusive line in the L1 data cache or the L2 cache, or a valid line in the L1 instruction cache, the processor asserts HIT#.
- If the compare hits a modified line in the L1 data cache or the L2 cache, the processor asserts HIT# and HITM#. If HITM# is asserted, the processor writes the modified line back to memory.
- If INV was sampled asserted with EADS#, a hit invalidates the line.
- If INV was sampled negated with EADS#, a hit leaves the line in the shared state or transitions it from the exclusive or modified state to the shared state.

Table 40 on page 225 lists valid combinations of MESI states permitted for a cache line in the L1 and L2 caches, and shows the effects of inquire cycles performed with INV equal to 0 (non-invalidating) and INV equal to 1 (invalidating).

Internal Snooping

Internal snooping is initiated by the processor (rather than system logic) during certain cache accesses. It is used to maintain coherency between the L1 instruction cache and the L1 data cache.

The processor automatically snoops its L1 instruction cache during read or write misses to its L1 data cache, and it snoops its L1 data cache during read misses to its L1 instruction cache. The L2 cache is not snooped during misses to either of the L1 caches. Table 41 on page 226 summarizes the actions taken during this internal snooping.

If an internal snoop hits its target, the processor does the following:

- **L1 Data Cache Snoop During an L1 Instruction-cache Read Miss**—If modified, the line in the L1 data cache is written back. If the writeback hits the L2 cache, the cache line is stored in the L2 cache in the modified state and no writeback occurs on the system bus. If the writeback misses the L2 cache, the cache line is written back on the system bus to external memory. Regardless of its state, the L1 data-cache line is invalidated and the L1 instruction cache performs a read from either the L2 cache (if a L2 hit occurs) or external memory (if a L2 miss occurs).
- **L1 Instruction Cache Snoop During an L1 Data Cache Miss**—The line in the instruction cache is marked invalid, and the L1 data-cache read or write is performed as defined in Table 39 on page 221.

FLUSH#

In response to sampling FLUSH# asserted, the processor writes back any L1 data cache lines and L2 cache lines that are in the modified state and then marks all lines in the L1 instruction cache, the L1 data cache, and the L2 cache as invalid.

Page Flush/Invalidate Register (PFIR)

The AMD-K6-III+ processor contains the Page Flush/Invalidate Register (PFIR) that allows cache invalidation and optional flushing of a specific 4-Kbyte page from the linear address space (see Figure 86 on page 224). When the PFIR is written to (using the WRMSR instruction), the invalidation and, optionally, the flushing begins. The total amount of cache in the AMD-K6-III+ processor is 320 Kbytes. Using this register can result in a much lower cycle count for flushing particular pages versus flushing the entire cache.

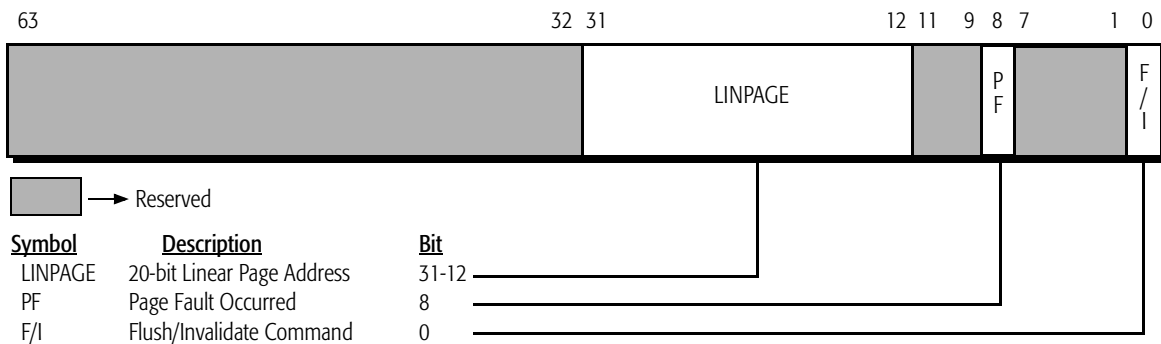


Figure 86. Page Flush/Invalidate Register (PFIR)

LINPAGE Field. This 20-bit field must be written with bits 31:12 of the linear address of the 4-Kbyte page that is to be invalidated and optionally flushed from the L1 or the L2 cache.

PF Bit. If an attempt to invalidate or flush a page results in a page fault, the processor sets the PF bit to 1, and the invalidate or flush operation is not performed (even though invalidate operations do not normally generate page faults). In this case, an actual page fault exception is not generated. If the PF bit equals 0 after an invalidate or flush operation, then the operation executed successfully. The PF bit must be read after every write to the PFIR register to determine if the invalidate or flush operation executed successfully.

F/I Bit. This bit is used to control the type of action that occurs to the specified linear page. If a 0 is written to this bit, the operation is a flush, in which case all cache lines in the modified state within the specified page are written back to memory, after which the entire page is invalidated. If a 1 is written to this bit, the operation is an invalidation, in which case the entire page is invalidated without the occurrence of any writebacks.

WBINVD and INVD

These x86 instructions cause all cache lines to be marked as invalid. WBINVD writes back modified lines before marking all cache lines invalid. INVD does not write back modified lines.

Cache-Line Replacement

Replacing lines in the L1 cache and the L2 cache, according to the line replacement algorithms described in “Cache-Line Fills” on page 213, ensures coherency between external memory and the caches.

Table 40 shows all possible cache-line states before and after inquire cycles.

Table 40. Valid L1 and L2 Cache States and Effect of Inquire Cycles

| Cache State Before Inquire ¹ | | Memory Access ² | Cache State After Inquire | | | |
|---|----------------|----------------------------|---------------------------|----|---------|----|
| | | | INV = 0 | | INV = 1 | |
| L1 | L2 | | L1 | L2 | L1 | L2 |
| I | M | Writeback L2 to bus | I | S | I | I |
| I | E | – | I | S | I | I |
| I | S | – | I | S | I | I |
| I | I | – | I | I | I | I |
| E ³ | M ³ | Writeback L2 to bus | S | S | I | I |
| E | E | – | S | S | I | I |
| E | I | – | S | I | I | I |
| M | E | Writeback L1 to bus | S | I | I | I |
| M | I | Writeback L1 to bus | S | I | I | I |
| S | S | – | S | S | I | I |
| S | I | – | S | I | I | I |

Notes:

1. M = Modified, E = Exclusive, S = Shared, I = Invalid. The exclusive and shared states are indistinguishable in the L1 instruction cache and are treated as "valid" states.
2. Writeback cycles to the bus are 32-byte burst writes.
3. This entry only applies to the L1 instruction cache. By design, a cache line cannot exist in the exclusive state in the L1 data cache and in the modified state in the L2 cache.

Table 41 shows all possible cache-line states before and after various cache-related operations.

Table 41. L1 and L2 Cache States for Snoops, Flushes, and Invalidation

| Operation Type | Cache State Before Operation ¹ | | Access Type ² | Cache State After Operation | |
|--------------------|---|----------------|--------------------------|-----------------------------|----|
| | L1 | L2 | | L1 | L2 |
| Internal Snoop | I | M | – | I | M |
| | I | E | – | I | E |
| | I | S | – | I | S |
| | I | I | – | I | I |
| | E ³ | M ³ | – | I | M |
| | E | E | – | I | E |
| | E | I | – | I | I |
| | M | E | Writeback L1 to L2 | I | M |
| | M | I | Writeback L1 to bus | I | I |
| | S | S | – | I | S |
| FLUSH# Signal | S or E | – | – | I | I |
| | M | – | Writeback L1 to bus | I | I |
| | – | M | Writeback L2 to bus | I | I |
| PFIR (F/I = 0) | S or E | – | – | I | I |
| | M | – | Writeback L1 to bus | I | I |
| | – | M | Writeback L2 to bus | I | I |
| PFIR (F/I = 1) | – | – | – | I | I |
| WBINVD Instruction | S or E | – | – | I | I |
| | M | – | Writeback L1 to bus | I | I |
| | – | M | Writeback L2 to bus | I | I |
| INVD Instruction | – | – | – | I | I |

Notes:

1. M = Modified, E = Exclusive, S = Shared, I = Invalid. The exclusive and shared states are indistinguishable in the L1 instruction cache and are treated as "valid" states.
 2. Writeback cycles to the bus are 32-byte burst writes.
 3. This entry only applies to the L1 instruction cache. By design, a cache line cannot exist in the exclusive state in the L1 data cache and in the modified state in the L2 cache.
- Not applicable or none.

9.12 Writethrough and Writeback Coherency States

The terms *writethrough* and *writeback* apply to two related concepts in a read-write cache like the AMD-K6-III+ processor L1 data cache and the L2 cache. The following conditions apply to both the writethrough and writeback modes:

- **Memory Writes**—A relationship exists between external memory writes and their concurrence with cache updates:
 - An external memory write that occurs concurrently with a cache update to the same location is a *writethrough*. Writethroughs are driven as single cycles on the bus.
 - An external memory write that occurs after the processor has modified a cache line is a *writeback*. Writebacks are driven as burst cycles on the bus.
- **Coherency State**—A relationship exists between MESI coherency states and writethrough-writeback coherency states of lines in the cache as follows:
 - Shared and invalid MESI lines are in the writethrough state.
 - Modified and exclusive MESI lines are in the writeback state.

9.13 A20M# Masking of Cache Accesses

Although the processor samples A20M# as a level-sensitive input on every clock edge, it should only be asserted in Real mode. The processor applies the A20M# masking to its tags, through which all programs access the caches. Therefore, assertion of A20M# affects all addresses (cache and external memory), including the following:

- Cache-line fills (caused by read misses or write allocates)
- Cache writethroughs (caused by write misses or write hits to lines in the shared state)

However, A20M# does not mask writebacks or invalidations caused by the following actions:

- Internal snoops
- Inquire cycles
- The FLUSH# signal

- Writing to the Page Flush/Invalidate Register (PFIR)
- The WBINVD instruction

10 Write Merge Buffer

The AMD-K6-III+ processor contains an 8-byte write merge buffer that allows the processor to conditionally combine data from multiple noncacheable write cycles into this merge buffer. The merge buffer operates in conjunction with the Memory Type Range Registers (MTRRs). Refer to “Memory Type Range Registers” on page 231 for a description of the MTRRs.

Merging multiple write cycles into a single write cycle reduces processor bus utilization and processor stalls, thereby increasing the overall system performance.

10.1 EWBE# Control

The presence of the merge buffer creates the potential to perform out-of-order write cycles relative to the processor’s caches. In general, the ordering of write cycles that are driven externally on the system bus and those that hit the processor’s cache can be controlled by the EWBE# signal. See “EWBE# (External Write Buffer Empty)” on page 110 for more information.

If EWBE# is sampled negated, the processor delays the commitment of write cycles to cache lines in the modified state or exclusive state in the processor’s caches. Therefore, the system logic can enforce strong ordering by negating EWBE# until the external write cycle is complete, thereby ensuring that a subsequent write cycle that hits a cache does not complete ahead of the external write cycle.

However, the addition of the write merge buffer introduces the potential for out-of-order write cycles to occur between writes to the merge buffer and writes to the processor’s caches. Because these writes occur entirely within the processor and are not sent out to the processor bus, the system logic is not able to enforce strong ordering with the EWBE# signal.

The EWBE# control (EWBEC) bits in the EFER register provide a mechanism for enforcing three different levels of write ordering in the presence of the write merge buffer:

- EFER[3] is defined as the Global EWBE# Disable (GEWBED). When GEWBED equals 1, the processor does not attempt to enforce any write ordering internally or externally (the EWBE# signal is ignored). This is the maximum performance setting.
- EFER[2] is defined as the Speculative EWBE# Disable (SEWBED). SEWBED only affects the processor when GEWBED equals 0. If GEWBED equals 0 and SEWBED equals 1, the processor enforces strong ordering for all internal write cycles with the exception of write cycles addressed to a range of memory defined as uncacheable (UC) or write-combining (WC) by the MTRRs. In addition, the processor samples the EWBE# signal. If EWBE# is sampled negated, the processor delays the commitment of write cycles to processor cache lines in the modified state or exclusive state until EWBE# is sampled asserted.

This setting provides performance comparable to, but slightly less than, the performance obtained when GEWBED equals 1 because some degree of write ordering is maintained.

- If GEWBED equals 0 and SEWBED equals 0, the processor enforces strong ordering for all internal and external write cycles. In this setting, the processor assumes, or *speculates*, that strong order must be maintained between writes to the merge buffer and writes that hit the processor's caches. Once the merge buffer is written out to the processor's bus, the EWBE# signal is sampled. If EWBE# is sampled negated, the processor delays the commitment of write cycles to processor cache lines in the modified state or exclusive state until EWBE# is sampled asserted.

This setting is the default after RESET and provides the lowest performance of the three settings because full write ordering is maintained.

Table 42 summarizes the three settings of the EWBEC field for the EFER register, along with the effect of write ordering and performance. For more information on the EFER register, see “Extended Feature Enable Register (EFER)” on page 47.

Table 42. EWBEC Settings and Performance

| EFER[3] (GEWBED) | EFER[2] (SEWBED) | Write Ordering | Performance |
|---------------------|---------------------|-------------------|---------------|
| 1 | 0 or 1 | None | Best |
| 0 | 1 | All except UC/WC | Close-to-Best |
| 0 | 0 | All | Slowest |

10.2 Memory Type Range Registers

The AMD-K6-III+ processor provides two variable-range Memory Type Range Registers (MTRRs)—MTRR0 and MTRR1—that each specify a range of memory. Each range can be defined as one of the following memory types:

- **Uncacheable (UC) Memory**—Memory read cycles are sourced directly from the specified memory address and the processor does not allocate a cache line. Memory write cycles are targeted at the specified memory address and a write allocation does not occur.
- **Write-Combining (WC) Memory**—Memory read cycles are sourced directly from the specified memory address and the processor does not allocate a cache line. The processor conditionally combines data from multiple noncacheable write cycles that are addressed within this range into a merge buffer. Merging multiple write cycles into a single write cycle reduces processor bus utilization and processor stalls, thereby increasing the overall system performance. This memory type is applicable for linear video frame buffers.

UC/WC Cacheability Control Register (UWCCR)

The MTRRs are accessed by addressing the 64-bit MSR known as the UC/WC Cacheability Control Register (UWCCR). The MSR address of the UWCCR is C000_0085h. Following reset, all bits in the UWCCR register are set to 0. MTRR0 (lower 32 bits of the UWCCR register) defines the size and memory type of range 0 and MTRR1 (upper 32 bits) defines the size and memory type of range 1 (see Figure 87).

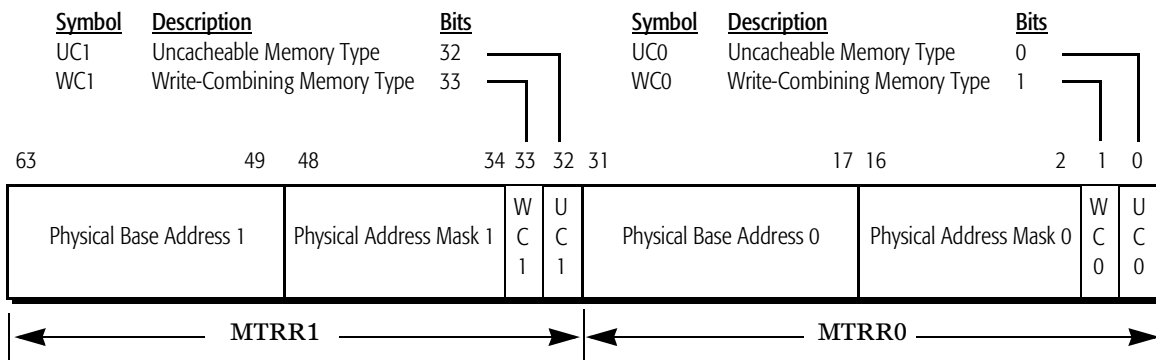


Figure 87. UC/WC Cacheability Control Register (UWCCR)

Physical Base Address n (n=0, 1). This address is the 15 most-significant bits of the physical base address of the memory range. The least-significant 17 bits of the base address are not needed because the base address is by definition always aligned on a 128-Kbyte boundary.

Physical Address Mask n (n=0, 1). This value is the 15 most-significant bits of a physical address mask that is used to define the size of the memory range. This mask is logically ANDed with both the physical base address field of the UWCCR register and the physical address generated by the processor. If the results of the two AND operations are equal, then the generated physical address is considered within the range.

That is, if:

$$\text{Mask} \ \& \ \text{Physical Base Address} = \text{Mask} \ \& \ \text{Physical Address Generated}$$

then, the physical address generated by the processor is in the range.

WC_n (n=0, 1). When set to 1, this memory range is defined as write combinable (see Table 43). Write-combinable memory is uncacheable.

UC_n (n=0, 1). When set to 1, this memory range is defined as uncacheable (see Table 43).

Table 43. WC/UC Memory Type

| WC _n | UC _n | Memory Type |
|-----------------|-----------------|--|
| 0 | 0 | No effect on cacheability or write combining |
| 1 | 0 | Write-combining memory range (uncacheable) |
| 0 or 1 | 1 | Uncacheable memory range |

10.3 Memory-Range Restrictions

The following rules regarding the address alignment and size of each range must be adhered to when programming the physical base address and physical address mask fields of the UWCCR register:

- The minimum size of each range is 128 Kbytes.
- The physical base address must be aligned on a 128-Kbyte boundary.
- The physical base address must be *range-size aligned*. For example, if the size of the range is 1 Mbyte, then the physical base address must be aligned on a 1-Mbyte boundary.
- All bits set to 1 in the physical address mask must be contiguous. Likewise, all bits set to 0 in the physical address mask must be contiguous. For example:

111_1111_1100_0000b is a valid physical address mask.

111_1111_1101_0000b is invalid.

Table 44 lists the valid physical address masks and the resulting range sizes that can be programmed in the UWCCR register.

Table 44. Valid Masks and Range Sizes for UWCCR Register

| Masks | Size |
|---------------------|------------|
| 111_1111_1111_1111b | 128 Kbytes |
| 111_1111_1111_1110b | 256 Kbytes |
| 111_1111_1111_1100b | 512 Kbytes |
| 111_1111_1111_1000b | 1 Mbyte |
| 111_1111_1111_0000b | 2 Mbytes |
| 111_1111_1110_0000b | 4 Mbytes |
| 111_1111_1100_0000b | 8 Mbytes |
| 111_1111_1000_0000b | 16 Mbytes |
| 111_1111_0000_0000b | 32 Mbytes |
| 111_1110_0000_0000b | 64 Mbytes |
| 111_1100_0000_0000b | 128 Mbytes |
| 111_1000_0000_0000b | 256 Mbytes |
| 111_0000_0000_0000b | 512 Mbytes |
| 110_0000_0000_0000b | 1 Gbyte |
| 100_0000_0000_0000b | 2 Gbytes |
| 000_0000_0000_0000b | 4 Gbytes |

10.4 Examples

Suppose that the range of memory from 16 Mbytes to 32 Mbytes is uncacheable, and the 8-Mbyte range of memory on top of 1 Gbyte is write-combinable. Range 0 is defined as the uncacheable range, and range 1 is defined as the write-combining range.

- Extracting the 15 most-significant bits of the 32-bit physical base address that corresponds to 16 Mbytes (0100_0000h) yields a physical base address 0 field of 000_0000_1000_0000b. Because the uncacheable range size is 16 Mbytes, the physical mask value 0 field is 111_1111_1000_0000b, according to Table 44 on page 234. Bit 1 of the UWCCR register (WC0) is set to 0 and bit 0 of the UWCCR register is set to 1 (UC0).
- Extracting the 15 most-significant bits of the 32-bit physical base address that corresponds to 1 Gbyte (4000_0000h) yields a physical base address 1 field of 010_0000_0000_0000b. Because the write-combining range size is 8 Mbytes, the physical mask value 1 field is 111_1111_1100_0000b, according to Table 44 on page 234. Bit 33 of the UWCCR register (WC1) is set to 1 and bit 32 of the UWCCR register is set to 0 (UC1).

11 Floating-Point and Multimedia Execution Units

11.1 Floating-Point Execution Unit

The AMD-K6-III+ processor contains an IEEE 754-compatible and 854-compatible floating-point execution unit designed to accelerate the performance of software that utilizes the x86 floating-point instruction set.

Floating-point software is typically written to manipulate numbers that are very large or very small, that require a high degree of precision, or that result from complex mathematical operations such as transcendentals. Applications that take advantage of floating-point operations include geometric calculations for graphics acceleration, scientific, statistical, and engineering applications, and business applications that use large amounts of high-precision data.

The high-performance floating-point execution unit contains an adder unit, a multiplier unit, and a divide/square root unit. These low-latency units can execute floating-point instructions in as few as two processor clocks. To increase performance, the processor is designed to simultaneously decode most floating-point instructions with most short-decodeable instructions.

See “Software Environment” on page 27 for a description of the floating-point data types, registers, and instructions.

Handling Floating-Point Exceptions

The AMD-K6-III+ processor provides the following two types of exception handling for floating-point exceptions:

- If the numeric error (NE) bit in CR0 is set to 1, the processor invokes the interrupt 10h handler. In this manner, the floating-point exception is completely handled by software.
- If the NE bit in CR0 is set to 0, the processor requires external logic to generate an interrupt on the INTR signal in order to handle the exception.

External Logic Support of Floating-Point Exceptions

The processor provides the FERR# (Floating-Point Error) and IGNNE# (Ignore Numeric Error) signals to allow the external logic to generate the interrupt in a manner consistent with IBM-compatible PC/AT systems. The assertion of FERR# indicates the occurrence of an unmasked floating-point exception resulting from the execution of a floating-point instruction. IGNNE# is used by the external hardware to control the effect of an unmasked floating-point exception. Under certain circumstances, if IGNNE# is sampled asserted, the processor ignores the floating-point exception.

Figure 88 on page 239 illustrates an implementation of external logic for supporting floating-point exceptions. The following example explains the operation of the external logic in Figure 88:

1. As the result of a floating-point exception, the processor asserts FERR#.
2. The assertion of FERR# and the sampling of IGNNE# negated indicates the processor has stopped instruction execution and is waiting for an interrupt.
3. The assertion of FERR# leads to the assertion of INTR by the interrupt controller.
4. The processor acknowledges the interrupt and jumps to the corresponding interrupt service routine in which an I/O write cycle to address port F0h leads to the assertion of IGNNE#.
5. When IGNNE# is sampled asserted, the processor ignores the floating-point exception and continues instruction execution.
6. When the processor negates FERR#, the external logic negates IGNNE#.

See “FERR# (Floating-Point Error)” on page 111 and “IGNNE# (Ignore Numeric Exception)” on page 116 for more details.

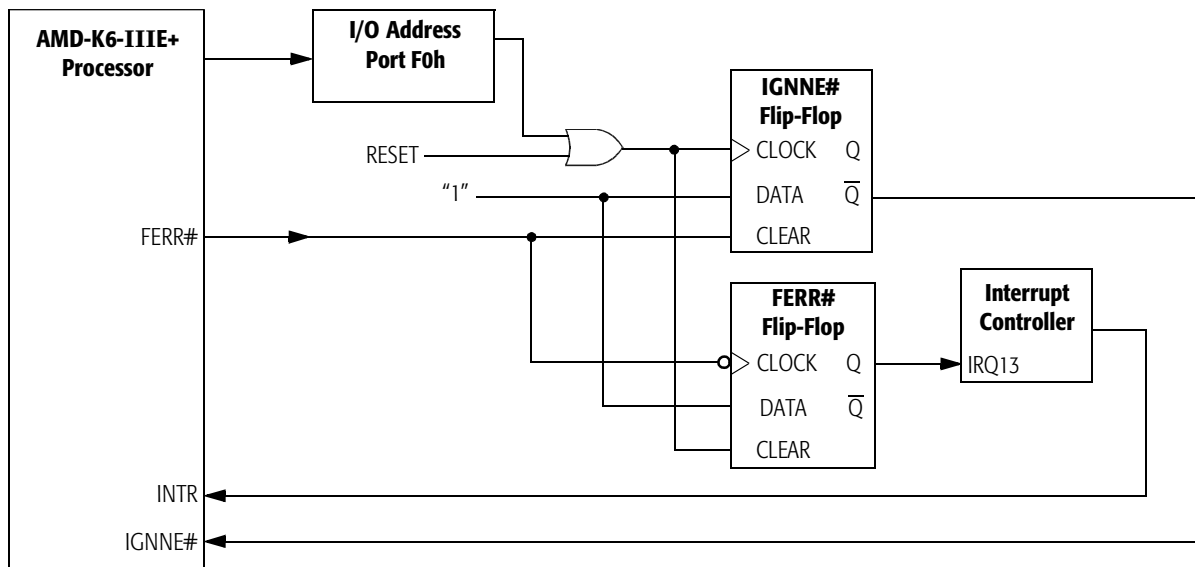


Figure 88. External Logic for Supporting Floating-Point Exceptions

11.2 Multimedia and 3DNow!™ Execution Units

The multimedia and 3DNow! execution units of the AMD-K6-III+ processor are designed to accelerate the performance of software written using the industry-standard MMX instructions and the new 3DNow! instructions. Applications that can take advantage of the MMX and 3DNow! instructions include graphics, video and audio compression and decompression, speech recognition, and telephony applications.

3DNow! technology enables fast frame rates on high-resolution 3D-rendered scenes, realistic physical modeling of real-world environments, sharp and detailed 3D imaging, smooth video playback, and theater-quality audio.

The AMD-K6-III+ processor supports five new digital signal processing (DSP) instructions, developed to enhance the performance of communications applications, including soft xDSL modems, MP3 recording, and Dolby Digital and Surround Sound processing.

For more information on MMX instructions, see the *AMD-K6® Processor Multimedia Technology Manual*, order# 20726. For

more information on 3DNow! instructions, see the *3DNow!™ Technology Manual*, order# 21928. For more information on the 3DNow! technology DSP extensions, see the *AMD Extensions to the 3DNow!™ and MMX™ Instructions Sets Manual*, order# 22466.

The multimedia execution unit can execute MMX instructions in a single processor clock. All MMX and 3DNow! arithmetic instructions are pipelined for higher performance. To increase performance, the processor is designed to simultaneously decode all MMX and 3DNow! instructions with most other instructions.

11.3 Floating-Point and MMX™/3DNow!™ Instruction Compatibility

Registers

The eight 64-bit MMX registers (which are also utilized by 3DNow! instructions) are mapped on the floating-point stack. This enables backward compatibility with all existing software. For example, the register saving event that is performed by operating systems during task switching requires no changes to the operating system. The same support provided in an operating system's interrupt 7 handler (Device Not Available) for saving and restoring the floating-point registers also supports saving and restoring the MMX registers.

Exceptions

There are no new exceptions defined for supporting the MMX and 3DNow! instructions. All exceptions that occur while decoding or executing an MMX or 3DNow! instruction are handled in existing exception handlers without modification.

FERR# and IGNNE#

MMX instructions and 3DNow! instructions do not generate floating-point exceptions. However, if an unmasked floating-point exception is pending, the processor asserts FERR# at the instruction boundary of the next floating-point instruction, MMX instruction, 3DNow! instruction or WAIT instruction.

The sampling of IGNNE# asserted only affects processor operation during the execution of an error-sensitive floating-point instruction, MMX instruction, 3DNow! instruction or WAIT instruction when the NE bit in CR0 is set to 0.

12 System Management Mode (SMM)

SMM is an alternate operating mode entered by way of a system management interrupt (SMI#) and handled by an interrupt service routine. SMM is designed for system control activities such as power management. These activities appear transparent to conventional operating systems like DOS and Windows. SMM is targeted for use by the Basic Input Output System (BIOS), specialized low-level device drivers, and the operating system. The code and data for SMM are stored in the SMM memory area, which is isolated from main memory.

The processor enters SMM by the assertion of the SMI# interrupt and the processor's acknowledgment by the assertion of SMIACT#. At this point the processor saves its state into the SMM memory state-save area and jumps to the SMM service routine. The processor returns from SMM when it executes the resume (RSM) instruction from within the SMM service routine. Subsequently, the processor restores its state from the SMM save area, negates SMIACT#, and resumes execution with the instruction following the point where it entered SMM.

The following sections summarize the SMM state-save area, entry into and exit from SMM, exceptions and interrupts in SMM, memory allocation and addressing in SMM, and the SMI# and SMIACT# signals.

12.1 SMM Operating Mode and Default Register Values

The software environment within SMM has the following characteristics:

- Addressing and operation in real mode
- 4-Gbyte segment limits
- Default 16-bit operand, address, and stack sizes, although instruction prefixes can override these defaults
- Control transfers that do not override the default operand size truncate the EIP to 16 bits
- Far jumps or calls cannot transfer control to a segment with a base address requiring more than 20 bits, as in real mode segment-base addressing

- A20M# is masked
- Interrupt vectors use the real-mode interrupt vector table
- The IF flag in EFLAGS is cleared (INTR not recognized)
- The TF flag in EFLAGS is cleared
- The NMI and INIT interrupts are disabled
- Debug register DR7 is cleared (debug traps disabled)

Figure 89 shows the default map of the SMM memory area. It consists of a 64-Kbyte area, between 0003_0000h and 0003_FFFFh, of which the top 32 Kbytes (0003_8000h to 0003_FFFFh) must be populated with RAM. The default code-segment (CS) base address for the area—called the SMM base address—is at 0003_0000h. The top 512 bytes (0003_FE00h to 0003_FFFFh) contain a fill-down SMM state-save area. The default entry point for the SMM service routine is 0003_8000h.

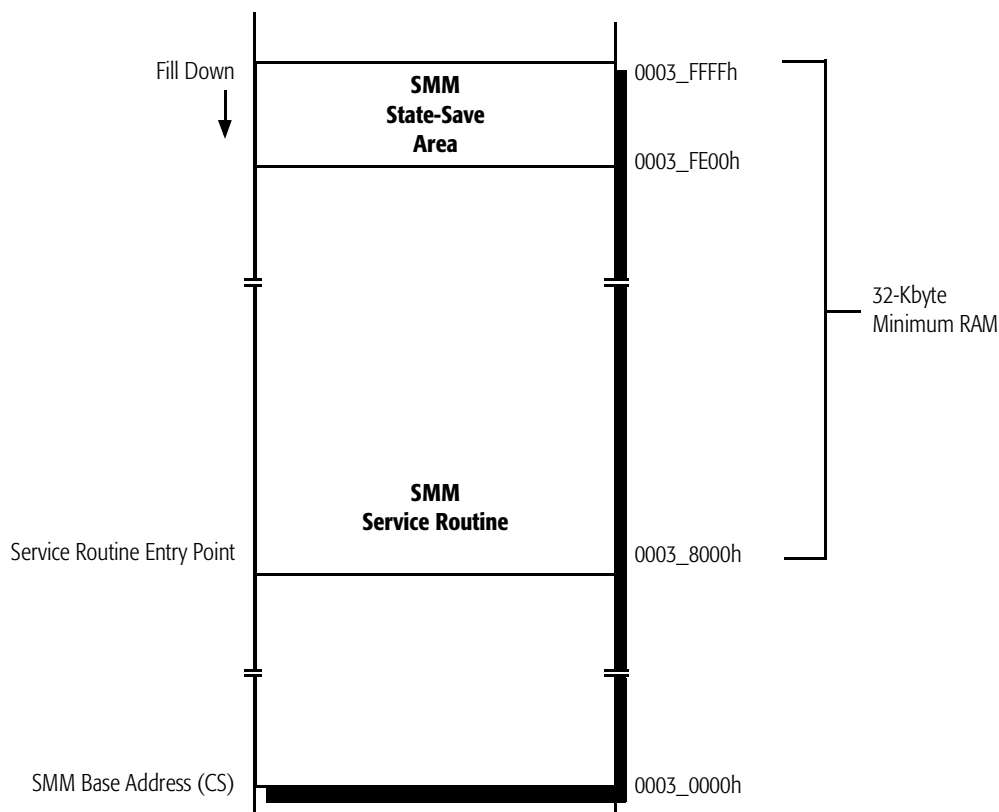


Figure 89. SMM Memory

Table 45 shows the initial state of registers when entering SMM.

Table 45. Initial State of Registers in SMM

| Registers | SMM Initial State |
|-----------------------------|---|
| General Purpose Registers | unmodified |
| EFLAGS | 0000_0002h |
| CR0 | PE, EM, TS, and PG are cleared (bits 0, 2, 3, and 31). The other bits are unmodified. |
| DR7 | 0000_0400h |
| GDTR, LDTR, IDTR, TSSR, DR6 | unmodified |
| EIP | 0000_8000h |
| CS | 0003_0000h |
| DS, ES, FS, GS, SS | 0000_0000h |

12.2 SMM State-Save Area

When the processor acknowledges an SMI# interrupt by asserting SMI $\#$, it saves its state in a 512-byte SMM state-save area shown in Table 46. The save begins at the top of the SMM memory area (SMM base address + FFFFh) and fills down to SMM base address + FE00h.

Table 46 shows the offsets in the SMM state-save area relative to the SMM base address. The SMM service routine can alter any of the read/write values in the state-save area.

Table 46. SMM State-Save Area Map

| Address Offset | Contents Saved |
|----------------|----------------|
| FFFCh | CR0 |
| FFF8h | CR3 |
| FFF4h | EFLAGS |
| FFF0h | EIP |
| FFECh | EDI |
| FFE8h | ESI |
| FFE4h | EBP |
| FFE0h | ESP |
| FFDCh | EBX |
| FFD8h | EDX |
| FFD4h | ECX |
| FFD0h | EAX |

Table 46. SMM State-Save Area Map (continued)

| Address Offset | Contents Saved |
|----------------|------------------------------|
| FFCCh | DR6 |
| FFC8h | DR7 |
| FFC4h | TR |
| FFC0h | LDTR Base |
| FFBCh | GS |
| FFB8h | FS |
| FFB4h | DS |
| FFB0h | SS |
| FFACh | CS |
| FFA8h | ES |
| FFA4h | I/O Trap Doubleword |
| FFA0h | No data dump at this address |
| FF9Ch | I/O Trap EIP ¹ |
| FF98h | No data dump at this address |
| FF94h | No data dump at this address |
| FF90h | IDT Base |
| FF8Ch | IDT Limit |
| FF88h | GDT Base |
| FF84h | GDT Limit |
| FF80h | TSS Attr |
| FF7Ch | TSS Base |
| FF78h | TSS Limit |
| FF74h | No data dump at this address |
| FF70h | LDT High |
| FF6Ch | LDT Low |
| FF68h | GS Attr |
| FF64h | GS Base |
| FF60h | GS Limit |
| FF5Ch | FS Attr |
| FF58h | FS Base |
| FF54h | FS Limit |
| FF50h | DS Attr |
| FF4Ch | DS Base |
| FF48h | DS Limit |
| FF44h | SS Attr |
| FF40h | SS Base |

Table 46. SMM State-Save Area Map (continued)

| Address Offset | Contents Saved |
|----------------|------------------------------|
| FF3Ch | SS Limit |
| FF38h | CS Attr |
| FF34h | CS Base |
| FF30h | CS Limit |
| FF2Ch | ES Attr |
| FF28h | ES Base |
| FF24h | ES Limit |
| FF20h | No data dump at this address |
| FF1Ch | No data dump at this address |
| FF18h | No data dump at this address |
| FF14h | CR2 |
| FF10h | CR4 |
| FF0Ch | I/O Restart ESI ¹ |
| FF08h | I/O Restart ECX ¹ |
| FF04h | I/O Restart EDI ¹ |
| FF02h | HALT Restart Slot |
| FF00h | I/O Trap Restart Slot |
| FEFCh | SMM RevID |
| FEF8h | SMM Base |
| FEF7h–FE00h | No data dump at this address |

Notes:

1. Only contains information if SMI# is asserted during a valid I/O bus cycle.

12.3 SMM Revision Identifier

The SMM revision identifier at offset FEFCh in the SMM state-save area specifies the version of SMM and the extensions that are available on the processor. The SMM revision identifier fields are as follows:

- *Bits 31–18*—Reserved
- *Bit 17*—SMM base address relocation (1 = enabled)
- *Bit 16*—I/O trap restart (1 = enabled)
- *Bits 15–0*—SMM revision level for the AMD-K6-III+ processor= 0002h

Table 47 shows the format of the SMM Revision Identifier.

Table 47. SMM Revision Identifier

| 31–18 | 17 | 16 | 15–0 |
|----------|---------------------|--------------------|--------------------|
| Reserved | SMM Base Relocation | I/O Trap Extension | SMM Revision Level |
| 0 | 1 | 1 | 0002h |

12.4 SMM Base Address

During RESET, the processor sets the base address of the code-segment (CS) for the SMM memory area—the SMM base address—to its default, 0003_0000h. The SMM base address at offset FEF8h in the SMM state-save area can be changed by the SMM service routine to any address that is aligned to a 32-Kbyte boundary. (Locations not aligned to a 32-Kbyte boundary cause the processor to enter the Shutdown state when executing the RSM instruction.)

In some operating environments it may be desirable to relocate the 64-Kbyte SMM memory area to a high memory area in order to provide more low memory for legacy software. During system initialization, the base of the 64-Kbyte SMM memory area is relocated by the BIOS. To relocate the SMM base address, the system enters the SMM handler at the default address. This handler changes the SMM base address location in the SMM state-save area, copies the SMM handler to the new location, and exits SMM.

The next time SMM is entered, the processor saves its state at the new base address. This new address is used for every SMM entry until the SMM base address in the SMM state-save area is changed or a hardware reset occurs.

12.5 Halt Restart Slot

During entry into SMM, the halt restart slot at offset FF02h in the SMM state-save area indicates if SMM was entered from the Halt state. Before returning from SMM, the halt restart slot (offset FF02h) can be written to by the SMM service routine to specify whether the return from SMM takes the processor back to the Halt state or to the next instruction after the HLT instruction.

Upon entry into SMM, the halt restart slot is defined as follows:

- *Bits 15–1*—Reserved
- *Bit 0*—Point of entry to SMM:
 - 1 = entered from Halt state
 - 0 = not entered from Halt state

After entry into the SMI handler and before returning from SMM, the halt restart slot can be written using the following definition:

- *Bits 15–1*—Reserved
- *Bit 0*—Point of return when exiting from SMM:
 - 1 = return to Halt state
 - 0 = return to next instruction after the HLT instruction

If the return from SMM takes the processor back to the Halt state, the HLT instruction is not re-executed, but the Halt special bus cycle is driven on the bus after the return.

12.6 I/O Trap Doubleword

If the assertion of SMI# is recognized during the execution of an I/O instruction, the I/O trap doubleword at offset FFA4h in the SMM state-save area contains information about the instruction. The fields of the I/O trap doubleword are configured as follows:

- *Bits 31–16*—I/O port address
- *Bits 15–4*—Reserved
- *Bit 3*—REP (repeat) string operation (1 = REP string, 0 = not a REP string)
- *Bit 2*—I/O string operation (1 = I/O string, 0 = not an I/O string)
- *Bit 1*—Valid I/O instruction (1 = valid, 0 = invalid)
- *Bit 0*—Input or output instruction (1 = INx, 0 = OUTx)

Table 48 shows the format of the I/O trap doubleword.

Table 48. I/O Trap Doubleword Configuration

| 31–16 | 15–4 | 3 | 2 | 1 | 0 |
|------------------|----------|----------------------|----------------------|-----------------------|-----------------|
| I/O Port Address | Reserved | REP String Operation | I/O String Operation | Valid I/O Instruction | Input or Output |

The I/O trap doubleword is related to the I/O trap restart slot (see “I/O Trap Restart Slot”). If bit 1 of the I/O trap doubleword is set by the processor, it means that SMI# was asserted during the execution of an I/O instruction. The SMI handler tests bit 1 to see if there is a valid I/O instruction trapped. If the I/O instruction is valid, the SMI handler is required to ensure the I/O trap restart slot is set properly. The I/O trap restart slot informs the processor whether it should re-execute the I/O instruction after the RSM or execute the instruction following the trapped I/O instruction.

Note: If SMI# is sampled asserted during an I/O bus cycle a minimum of three clock edges before BRDY# is sampled asserted, the associated I/O instruction is guaranteed to be trapped by the SMI handler.

12.7 I/O Trap Restart Slot

The I/O trap restart slot at offset FF00h in the SMM state-save area specifies whether the trapped I/O instruction should be re-executed on return from SMM. This slot in the state-save area is called the *I/O instruction restart* function. Re-executing a trapped I/O instruction is useful, for example, if an I/O write occurs to a disk that is powered down. The system logic monitoring such an access can assert SMI#. Then the SMM service routine would query the system logic, detect a failed I/O write, take action to power-up the I/O device, enable the I/O trap restart slot feature, and return from SMM.

The fields of the I/O trap restart slot are defined as follows:

- *Bits 31–16*—Reserved
- *Bits 15–0*—I/O instruction restart on return from SMM:
 - 0000h = execute the next instruction after the trapped I/O instruction
 - 00FFh = re-execute the trapped I/O instruction

Table 49 shows the format of the I/O trap restart slot.

Table 49. I/O Trap Restart Slot

| 31–16 | 15–0 |
|----------|---|
| Reserved | I/O Instruction restart on return from SMM: 0000h = execute the next instruction after the trapped I/O 00FFh = re-execute the trapped I/O instruction |

The processor initializes the I/O trap restart slot to 0000h upon entry into SMM. If SMM was entered due to a trapped I/O instruction, the processor indicates the validity of the I/O instruction by setting or clearing bit 1 of the I/O trap doubleword at offset FFA4h in the SMM state-save area. The SMM service routine should test bit 1 of the I/O trap doubleword to determine if a valid I/O instruction was being executed when entering SMM and before writing the I/O trap restart slot. If the I/O instruction is valid, the SMM service routine can safely rewrite the I/O trap restart slot with the value 00FFh, which causes the processor to re-execute the trapped I/O instruction when the RSM instruction is executed. If the I/O instruction is invalid, writing the I/O trap restart slot has undefined results.

If a second SMI# is asserted and a valid I/O instruction was trapped by the first SMM handler, the processor services the second SMI# prior to re-executing the trapped I/O instruction. The second entry into SMM never has bit 1 of the I/O trap doubleword set, and the second SMM service routine must not rewrite the I/O trap restart slot.

During a simultaneous SMI# I/O instruction trap and debug breakpoint trap, the AMD-K6-III+ processor first responds to the SMI# and postpones recognizing the debug exception until after returning from SMM via the RSM instruction. If the debug registers DR3–DR0 are used while in SMM, they must be saved and restored by the SMM handler. The processor automatically saves and restores DR7–DR6. If the I/O trap restart slot in the SMM state-save area contains the value 00FFh when the RSM instruction is executed, the debug trap does not occur until after the I/O instruction is re-executed.

12.8 Exceptions, Interrupts, and Debug in SMM

During an SMI# I/O trap, the exception/interrupt priority of the AMD-K6-III+ processor changes from its normal priority. The normal priority places the debug traps at a priority higher than the sampling of the FLUSH# or SMI# signals. However, during an SMI# I/O trap, the sampling of the FLUSH# or SMI# signals takes precedence over debug traps.

The processor recognizes the assertion of NMI within SMM immediately after the completion of an IRET instruction. Once NMI is recognized within SMM, NMI recognition remains enabled until SMM is exited, at which point NMI masking is restored to the state it was in before entering SMM.

13 Test and Debug

The AMD-K6-III+ processor implements various test and debug modes to enable the functional and manufacturing testing of systems and boards that use the processor. In addition, the debug features of the processor allow designers to debug the instruction execution of software components. This chapter describes the following test and debug features:

- **Built-In Self-Test (BIST)**—The BIST, which is invoked after the falling transition of RESET, runs internal tests that exercise most on-chip RAM structures.
- **Three-State Test Mode**—A test mode that causes the processor to float its output and bidirectional pins.
- **Boundary-Scan Test Access Port (TAP)**—The Joint Test Action Group (JTAG) test access function defined by the *IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE 1149.1-1990)* specification.
- **Cache Inhibit**—A feature that disables the processor's internal L1 and L2 caches.
- **Level-2 Cache Array Access Register (L2AAR)**—The AMD-K6-III+ processor provides the L2AAR that allows for direct access to the L2 cache and L2 tag arrays.
- **Debug Support**—Consists of all x86-compatible software debug features, including the debug extensions.

13.1 Built-In Self-Test (BIST)

Following the falling transition of RESET, the processor unconditionally runs its built-in self test (BIST). The internal resources tested during BIST include the following:

- L1 instruction and data caches
- L2 cache
- Instruction and Data Translation Lookaside Buffers (TLBs)

The contents of the EAX general-purpose register after the completion of reset indicate if the BIST was successful.

- If EAX contains 0000_0000h, then BIST was successful.
- If EAX is non-zero, the BIST failed.

Following the completion of the BIST, the processor jumps to address FFFF_FFF0h to start instruction execution, regardless of the outcome of the BIST.

The BIST takes approximately 5,000,000 processor clocks to complete.

13.2 Three-State Test Mode

The Three-State Test mode causes the processor to float its output and bidirectional pins, which is useful for board-level manufacturing testing. In this mode, the processor is electrically isolated from other components on a system board, allowing automated test equipment (ATE) to test components that drive the same signals as those the processor floats.

If the FLUSH# signal is sampled Low during the falling transition of RESET, the processor enters the Three-State Test mode. (See “FLUSH# (Cache Flush)” on page 112 for the specific sampling requirements.) The signals floated in the Three-State Test mode are as follows:

- | | | |
|------------|-----------|------------|
| ■ A[31:3] | ■ D/C# | ■ M/IO# |
| ■ ADS# | ■ D[63:0] | ■ PCD |
| ■ ADSC# | ■ DP[7:0] | ■ PCHK# |
| ■ AP | ■ FERR# | ■ PWT |
| ■ APCHK# | ■ HIT# | ■ SCYC |
| ■ BE[7:0]# | ■ HITM# | ■ SMIACT# |
| ■ BREQ | ■ HLDA | ■ VID[4:0] |
| ■ CACHE# | ■ LOCK# | ■ W/R# |

The VCC2DET, VCC2H/L#, and TDO signals are the only outputs not floated in the Three-State Test mode.

- VCC2DET and VCC2H/L# must remain Low to ensure the system continues to supply the specified processor core voltage to the V_{CC2} pins.

- TDO is never floated because the Boundary-Scan Test Access Port must remain enabled at all times, including during the Three-State Test mode.

The Three-State Test mode is exited when the processor samples RESET asserted.

13.3 Boundary-Scan Test Access Port (TAP)

The boundary-scan Test Access Port (TAP) is an IEEE standard that defines synchronous scanning test methods for complex logic circuits, such as boards containing a processor. The AMD-K6-III+ processor supports the TAP standard defined in the *IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE 1149.1-1990)* specification.

Boundary scan testing uses a shift register consisting of the serial interconnection of boundary-scan cells that correspond to each I/O buffer of the processor. This non-inverting register chain, called a Boundary Scan Register (BSR), can be used to capture the state of every processor pin and to drive every processor output and bidirectional pin to a known state.

Each BSR of every component on a board that implements the boundary-scan architecture can be serially interconnected to enable component interconnect testing.

Test Access Port

The TAP consists of the following:

- **Test Access Port (TAP) Controller**—The TAP controller is a synchronous, finite state machine that uses the TMS and TDI input signals to control a sequence of test operations. See “TAP Controller State Machine” on page 260 for a list of TAP states and their definition.
- **Instruction Register (IR)**—The IR contains the instructions that select the test operation to be performed and the Test Data Register (TDR) to be selected. See “TAP Registers” on page 255 for more details on the IR.
- **Test Data Registers (TDR)**—The three TDRs are used to process the test data. Each TDR is selected by an instruction in the Instruction Register (IR). See “TAP Registers” on page 255 for a list of these registers and their functions.

TAP Signals

The test signals associated with the TAP controller are as follows:

- **TCK**—The Test Clock for all TAP operations. The rising edge of TCK is used for sampling TAP signals, and the falling edge of TCK is used for asserting TAP signals. The state of the TMS signal sampled on the rising edge of TCK causes the state transitions of the TAP controller to occur. TCK can be stopped in the logic 0 or 1 state.
- **TDI**—The Test Data Input represents the input to the most significant bit of all TAP registers, including the IR and all test data registers. Test data and instructions are serially shifted by one bit into their respective registers on the rising edge of TCK.
- **TDO**—The Test Data Output represents the output of the least significant bit of all TAP registers, including the IR and all test data registers. Test data and instructions are serially shifted by one bit out of their respective registers on the falling edge of TCK.
- **TMS**—The Test Mode Select input specifies the test function and sequence of state changes for boundary-scan testing. If TMS is sampled High for five or more consecutive clocks, the TAP controller enters its reset state.
- **TRST#**—The Test Reset signal is an asynchronous reset that unconditionally causes the TAP controller to enter its reset state.

Refer to “Electrical Data” on page 287 and “Signal Switching Characteristics” on page 297 to obtain the electrical specifications of the test signals.

TAP Registers

The AMD-K6-III+ processor provides an Instruction Register (IR) and three Test Data Registers (TDR) to support the boundary-scan architecture. The IR and one of the TDRs—the Boundary-Scan Register (BSR)—consist of a shift register and an output register. The shift register is loaded in parallel in the Capture states. (See “TAP Controller State Machine” on page 260 for a description of the TAP controller states.) In addition, the shift register is loaded and shifted serially in the Shift states. The output register is loaded in parallel from its corresponding shift register in the Update states.

Instruction Register (IR). The IR is a 5-bit register, without parity, that determines which instruction to run and which test data register to select. When the TAP controller enters the Capture-IR state, the processor loads the following bits into the IR shift register:

- *01b*—Loaded into the two least significant bits, as specified by the IEEE 1149.1 standard
- *000b*—Loaded into the three most significant bits

Loading 00001b into the IR shift register during the Capture-IR state results in loading the SAMPLE/PRELOAD instruction.

For each entry into the Shift-IR state, the IR shift register is serially shifted by one bit toward the TDO pin. During the shift, the most significant bit of the IR shift register is loaded from the TDI pin.

The IR output register is loaded from the IR shift register in the Update-IR state, and the current instruction is defined by the IR output register. See “TAP Instructions” on page 259 for a list and definition of the instructions supported by the AMD-K6-III+ processor.

Boundary Scan Register (BSR). The Boundary Scan Register is a Test Data Register consisting of the interconnection of 152 boundary-scan cells. Each output and bidirectional pin of the processor requires a two-bit cell, where one bit corresponds to the pin and the other bit is the output enable for the pin. When a 0 is shifted into the enable bit of a cell, the corresponding pin is floated, and when a 1 is shifted into the enable bit, the pin is driven valid. Each input pin requires a one-bit cell that corresponds to the pin. The last cell of the BSR is reserved and does not correspond to any processor pin.

The total number of bits that comprise the BSR is 297. Table 50 on page 257 lists the order of these bits, where TDI is the input to bit 296, and TDO is driven from the output of bit 0. The entries listed as *pin_E* (where *pin* is an output or bidirectional signal) are the enable bits.

If the BSR is the register selected by the current instruction and the TAP controller is in the Capture-DR state, the processor loads the BSR shift register as follows:

- If the current instruction is SAMPLE/PRELOAD, then the current state of each input, output, and bidirectional pin is loaded. A bidirectional pin is treated as an output if its enable bit equals 1, and it is treated as an input if its enable bit equals 0.
- If the current instruction is EXTEST, then the current state of each input pin is loaded. A bidirectional pin is treated as an input, regardless of the state of its enable.

While in the Shift-DR state, the BSR shift register is serially shifted toward the TDO pin. During the shift, bit 280 of the BSR is loaded from the TDI pin.

The BSR output register is loaded with the contents of the BSR shift register in the Update-DR state. If the current instruction is EXTEST, the processor's output pins, as well as those bidirectional pins that are enabled as outputs, are driven with their corresponding values from the BSR output register.

Table 50. Boundary Scan Bit Definitions¹

| Bit | Pin/Enable | Bit | Pin/Enable | Bit | Pin/Enable | Bit | Pin/Enable | Bit | Pin/Enable | Bit | Pin/Enable |
|-----|---------------------|-----|---------------------|-----|------------|-----|------------|-----|---------------------|-----|------------|
| 296 | A6_E | 263 | A28_E | 230 | HIT# | 197 | AP | 164 | RSVD | 131 | D40_E |
| 295 | A6 | 262 | A28 | 229 | A27_E | 196 | A20_E | 163 | RSVD | 130 | D40 |
| 294 | VID1_E ² | 261 | ADS_E | 228 | A27 | 195 | A20 | 162 | RSVD | 129 | D59_E |
| 293 | VID1 ² | 260 | ADS# | 227 | A4_E | 194 | BREQ_E | 161 | RSVD | 128 | D59 |
| 292 | A22_E | 259 | A17_E | 226 | A4 | 193 | BREQ | 160 | AHOLD | 127 | D9_E |
| 291 | A22 | 258 | A17 | 225 | A7_E | 192 | A11_E | 159 | INV | 126 | D9 |
| 290 | PCHK_E | 257 | A25_E | 224 | A7 | 191 | A11 | 158 | CLK | 125 | D28_E |
| 289 | PCHK# | 256 | A25 | 223 | A8_E | 190 | A10_E | 157 | VID2_E ² | 124 | D28 |
| 288 | A14_E | 255 | PWT_E | 222 | A8 | 189 | A10 | 156 | VID2 ² | 123 | D56_E |
| 287 | A14 | 254 | PWT | 221 | A15_E | 188 | APCHK_E | 155 | CACHE_E | 122 | D56 |
| 286 | A13_E | 253 | A12_E | 220 | A15 | 187 | APCHK# | 154 | CACHE# | 121 | D44_E |
| 285 | A13 | 252 | A12 | 219 | DC_E | 186 | SMIACT_E | 153 | MIO_E | 120 | D44 |
| 284 | A24_E | 251 | A9_E | 218 | D/C# | 185 | SMIACT# | 152 | M/IO# | 119 | D11_E |
| 283 | A24 | 250 | A9 | 217 | A16_E | 184 | RSVD | 151 | FERR_E | 118 | D11 |
| 282 | RESET | 249 | A26_E | 216 | A16 | 183 | A5_E | 150 | FERR# | 117 | DP3_E |
| 281 | A18_E | 248 | A26 | 215 | A19_E | 182 | A5 | 149 | D0_E | 116 | DP3 |
| 280 | A18 | 247 | A30_E | 214 | A19 | 181 | INTR | 148 | D0 | 115 | D39_E |
| 279 | A21_E | 246 | A30 | 213 | SCYC_E | 180 | NMI | 147 | D1_E | 114 | D39 |
| 278 | A21 | 245 | VID0_E ² | 212 | SCYC | 179 | INIT | 146 | D1 | 113 | DP6_E |
| 277 | PCD_E | 244 | VID0 ² | 211 | ADSC_E | 178 | HOLD | 145 | D61_E | 112 | DP6 |
| 276 | PCD | 243 | HITM_E | 210 | ADSC# | 177 | IGNNE# | 144 | D61 | 111 | D8_E |
| 275 | BE4_E | 242 | HITM# | 209 | BE6_E | 176 | SMI# | 143 | D62_E | 110 | D8 |
| 274 | BE4# | 241 | A20M# | 208 | BE6 | 175 | WB/WT# | 142 | D62 | 109 | D32_E |
| 273 | BE7_E | 240 | FLUSH# | 207 | BE3_E | 174 | BF0 | 141 | DP0_E | 108 | D32 |
| 272 | BE7# | 239 | A3_E | 206 | BE3 | 173 | BOFF# | 140 | DP0 | 107 | D36_E |
| 271 | A23_E | 238 | A3 | 205 | HLDA_E | 172 | NA# | 139 | D21_E | 106 | D36 |
| 270 | A23 | 237 | A31_E | 204 | HLDA | 171 | BF1 | 138 | D21 | 105 | D51_E |
| 269 | LOCK_E | 236 | A31 | 203 | BE1_E | 170 | BRDYC# | 137 | D57_E | 104 | D51 |
| 268 | LOCK# | 235 | A29_E | 202 | BE1# | 169 | BRDY# | 136 | D57 | 103 | D15_E |
| 267 | BE0_E | 234 | A29 | 201 | EADS# | 168 | STPCLK# | 135 | D5_E | 102 | D15 |
| 266 | BE0# | 233 | WR_E | 200 | BE2_E | 167 | BF2 | 134 | D5 | 101 | D37_E |
| 265 | BE5_E | 232 | W/R# | 199 | BE2# | 166 | KEN# | 133 | D24_E | 100 | D37 |
| 264 | BE5# | 231 | HIT_E | 198 | AP_E | 165 | EWBE# | 132 | D24 | 99 | D41_E |

Table 50. Boundary Scan Bit Definitions¹ (continued)

| Bit | Pin/Enable | Bit | Pin/Enable | Bit | Pin/Enable | Bit | Pin/Enable | Bit | Pin/Enable | Bit | Pin/Enable |
|-----|------------|-----|------------|-----|------------|-----|---------------------|-----|---------------------|-----|------------|
| 98 | D41 | 81 | D49 | 64 | D20_E | 47 | D35 | 30 | D43_E | 13 | D45 |
| 97 | D52_E | 80 | D17_E | 63 | D20 | 46 | D10_E | 29 | D43 | 12 | D60_E |
| 96 | D52 | 79 | D17 | 62 | D13_E | 45 | D10 | 28 | D58_E | 11 | D60 |
| 95 | D14_E | 78 | D19_E | 61 | D13 | 44 | D53_E | 27 | D58 | 10 | D22_E |
| 94 | D14 | 77 | D19 | 60 | DP5_E | 43 | D53 | 26 | D26_E | 9 | D22 |
| 93 | D29_E | 76 | D48_E | 59 | DP5 | 42 | D34_E | 25 | D26 | 8 | D63_E |
| 92 | D29 | 75 | D48 | 58 | D31_E | 41 | D34 | 24 | D3_E | 7 | D63 |
| 91 | D33_E | 74 | D47_E | 57 | D31 | 40 | VID4_E ² | 23 | D3 | 6 | DP7_E |
| 90 | D33 | 73 | D47 | 56 | D27_E | 39 | VID4 ² | 22 | D55_E | 5 | DP7 |
| 89 | RSVD | 72 | D16_E | 55 | D27 | 38 | D7_E | 21 | D55 | 4 | D4_E |
| 88 | D18_E | 71 | D16 | 54 | D12_E | 37 | D7 | 20 | D42_E | 3 | D4 |
| 87 | D18 | 70 | DP1_E | 53 | D12 | 36 | DP4_E | 19 | D42 | 2 | D2_E |
| 86 | D23_E | 69 | DP1 | 52 | D50_E | 35 | DP4 | 18 | VID3_E ² | 1 | D2 |
| 85 | D23 | 68 | D46_E | 51 | D50 | 34 | D54_E | 17 | VID3 ² | 0 | Reserved |
| 84 | D25_E | 67 | D46 | 50 | D38_E | 33 | D54 | 16 | D6_E | | |
| 83 | D25 | 66 | DP2_E | 49 | D38 | 32 | D30_E | 15 | D6 | | |
| 82 | D49_E | 65 | DP2 | 48 | D35_E | 31 | D30 | 14 | D45_E | | |

Notes:

1. TDI is the input to bit 296, and TDO is driven from the output of bit 0. The entries listed as pin_E (where pin is an output or bidirectional signal) are the enable bits.
2. Supported on low-power versions only.

Device Identification Register (DIR). The DIR is a 32-bit Test Data Register selected during the execution of the IDCODE instruction. The fields of the DIR and their values are shown in Table 51 on page 259 and are defined as follows:

- *Version Code*—This 4-bit field is incremented by AMD manufacturing for each major revision of silicon.
- *Part Number*—This 16-bit field identifies the specific processor model.
- *Manufacturer*—This 11-bit field identifies the manufacturer of the component (AMD).

- **LSB**—The least significant bit (LSB) of the DIR is always set to 1, as specified by the IEEE 1149.1 standard.

Table 51. Device Identification Register

| Version Code (Bits 31–28) | Part Number (Bits 27–12) | Manufacturer (Bits 11–1) | LSB (Bit 0) |
|------------------------------|-----------------------------|-----------------------------|----------------|
| Xh | 05D0h | 00000000001b | 1b |

Bypass Register (BR). The BR is a Test Data Register consisting of a 1-bit shift register that provides the shortest path between TDI and TDO. When the processor is not involved in a test operation, the BR can be selected by an instruction to allow the transfer of test data through the processor without having to serially scan the test data through the BSR. This functionality preserves the state of the BSR and significantly reduces test time.

The BR register is selected by the **BYPASS** and **HIGHZ** instructions as well as by any instructions not supported by the AMD-K6-III+ processor.

TAP Instructions

The processor supports the three instructions required by the IEEE 1149.1 standard—**EXTEST**, **SAMPLE/PRELOAD**, and **BYPASS**—as well as two additional optional instructions—**IDCODE** and **HIGHZ**.

Table 52 shows the complete set of TAP instructions supported by the processor along with the 5-bit Instruction Register encoding and the register selected by each instruction.

Table 52. Supported TAP Instructions

| Instruction | Encoding | Register | Description |
|---------------------|---------------|----------|---|
| EXTEST ¹ | 00000b | BSR | Sample inputs and drive outputs |
| SAMPLE / PRELOAD | 00001b | BSR | Sample inputs and outputs, then load the BSR |
| IDCODE | 00010b | DIR | Read DIR |
| HIGHZ | 00011b | BR | Float outputs and bidirectional pins |
| BYPASS ² | 00100b–11110b | BR | Undefined instruction, execute the BYPASS instruction |
| BYPASS ³ | 11111b | BR | Connect TDI to TDO to bypass the BSR |

Notes:

1. Following the execution of the **EXTEST** instruction, the processor must be reset in order to return to normal, non-test operation.
2. These instruction encodings are undefined on the AMD-K6-III+ processor and default to the **BYPASS** instruction.
3. Because the TDI input contains an internal pullup, the **BYPASS** instruction is executed if the TDI input is not connected or open during an instruction scan operation. The **BYPASS** instruction does not affect the normal operational state of the processor.

EXTEST Instruction. When the EXTEST instruction is executed, the processor loads the BSR shift register with the current state of the input and bidirectional pins in the Capture-DR state and drives the output and bidirectional pins with the corresponding values from the BSR output register in the Update-DR state.

SAMPLE/PRELOAD Instruction. The SAMPLE/PRELOAD instruction performs two functions. These functions are as follows:

- During the Capture-DR state, the processor loads the BSR shift register with the current state of every input, output, and bidirectional pin.
- During the Update-DR state, the BSR output register is loaded from the BSR shift register in preparation for the next EXTEST instruction.

The SAMPLE/PRELOAD instruction does not affect the normal operational state of the processor.

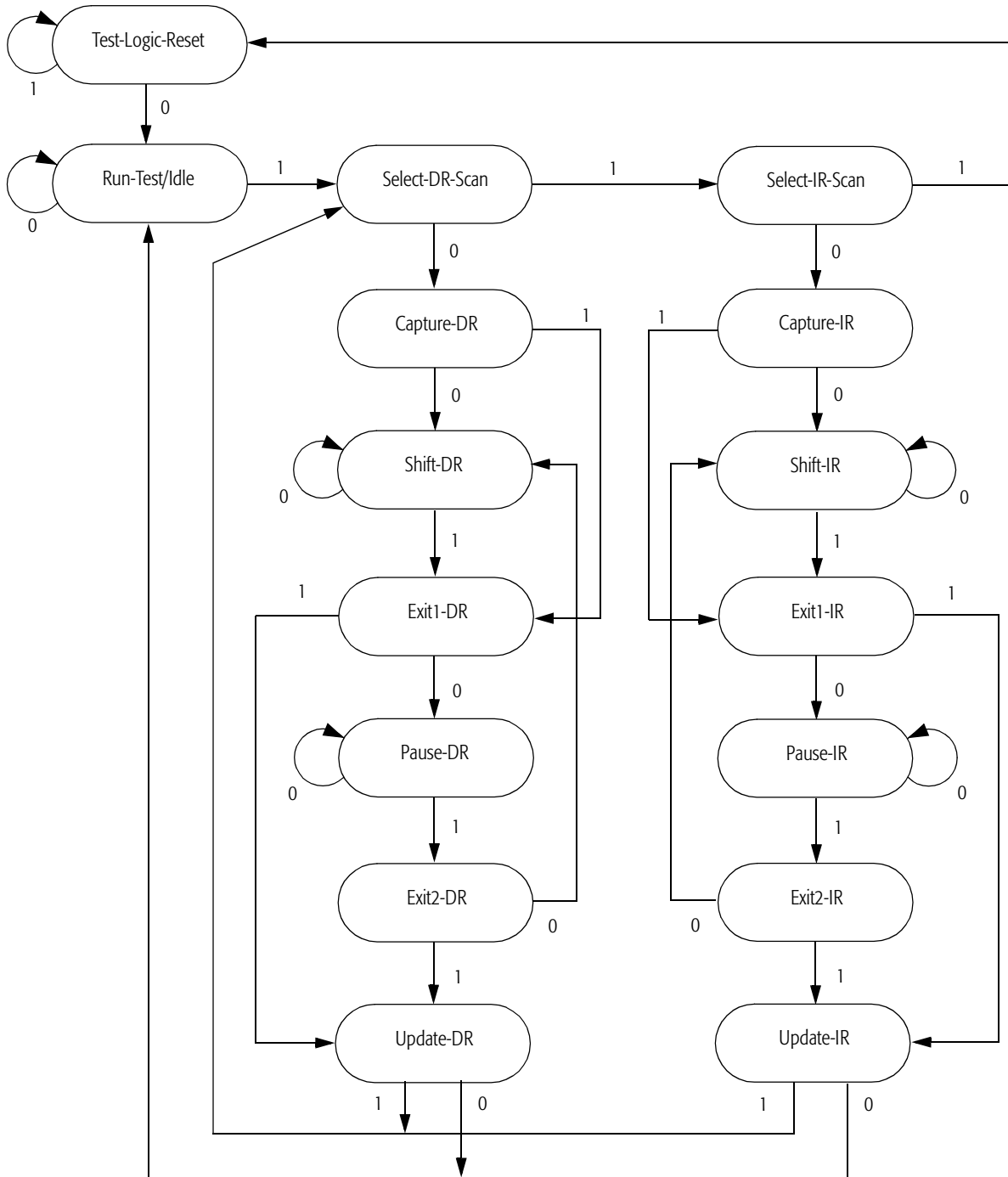
BYPASS Instruction. The BYPASS instruction selects the BR register, which reduces the boundary-scan length through the processor from 297 to one (TDI to BR to TDO). The BYPASS instruction does not affect the normal operational state of the processor.

IDCODE Instruction. The IDCODE instruction selects the DIR register, allowing the device identification code to be shifted out of the processor. This instruction is loaded into the IR when the TAP controller is reset. The IDCODE instruction does not affect the normal operational state of the processor.

HIGHZ Instruction. The HIGHZ instruction forces all output and bidirectional pins to be floated. During this instruction, the BR is selected and the normal operational state of the processor is not affected.

TAP Controller State Machine

The TAP controller state diagram is shown in Figure 90 on page 261. State transitions occur on the rising edge of TCK. The logic 0 or 1 next to the states represents the value of the TMS signal sampled by the processor on the rising edge of TCK.



IEEE Std 1149.1-1990, Copyright © 1990. IEEE. All rights reserved

Figure 90. TAP State Diagram

The states of the TAP controller are described as follows:

Test-Logic-Reset. This state represents the initial reset state of the TAP controller and is entered when the processor samples RESET asserted, when TRST# is asynchronously asserted, and when TMS is sampled High for five or more consecutive clocks. In addition, this state can be entered from the Select-IR-Scan state. The IR is initialized with the IDCODE instruction, and the processor's normal operation is not affected in this state.

Capture-DR. During the SAMPLE/PRELOAD instruction, the processor loads the BSR shift register with the current state of every input, output, and bidirectional pin. During the EXTEST instruction, the processor loads the BSR shift register with the current state of every input and bidirectional pin.

Capture-IR. When the TAP controller enters the Capture-IR state, the processor loads 01b into the two least significant bits of the IR shift register and loads 000b into the three most significant bits of the IR shift register.

Shift-DR. While in the Shift-DR state, the selected TDR shift register is serially shifted toward the TDO pin. During the shift, the most significant bit of the TDR is loaded from the TDI pin.

Shift-IR. While in the Shift-IR state, the IR shift register is serially shifted toward the TDO pin. During the shift, the most significant bit of the IR is loaded from the TDI pin.

Update-DR. During the SAMPLE/PRELOAD instruction, the BSR output register is loaded with the contents of the BSR shift register. During the EXTEST instruction, the output pins, as well as those bidirectional pins defined as outputs, are driven with their corresponding values from the BSR output register.

Update-IR. In this state, the IR output register is loaded from the IR shift register, and the current instruction is defined by the IR output register.

The following states have no effect on the normal or test operation of the processor other than as shown in Figure 90 on page 261:

- Run-Test/Idle—This state is an idle state between scan operations.
- Select-DR-Scan—This is the initial state of the test data register state transitions.
- Select-IR-Scan—This is the initial state of the Instruction Register state transitions.
- Exit1-DR—This state is entered to terminate the shifting process and enter the Update-DR state.
- Exit1-IR—This state is entered to terminate the shifting process and enter the Update-IR state.
- Pause-DR—This state is entered to temporarily stop the shifting process of a Test Data Register.
- Pause-IR—This state is entered to temporarily stop the shifting process of the Instruction Register.
- Exit2-DR—This state is entered in order to either terminate the shifting process and enter the Update-DR state or to resume shifting following the exit from the Pause-DR state.
- Exit2-IR—This state is entered in order to either terminate the shifting process and enter the Update-IR state or to resume shifting following the exit from the Pause-IR state.

13.4 Cache Inhibit

The AMD-K6-III+ processor provides a means for inhibiting the normal operation of its internal L1 and L2 caches while still supporting an external cache. This capability allows system designers to disable the L1 and L2 caches during the testing and debug of an L3 cache.

If the Cache Inhibit bit (bit 3) of Test Register 12 (TR12) is set to 0, the processor's L1 and L2 caches are enabled and operate as described in "Cache Organization" on page 205. If the Cache Inhibit bit is set to 1, the L1 and L2 caches are disabled and no new cache lines are allocated. Even though new allocations do not occur, valid L1 and L2 cache lines remain valid and are read by the processor when a requested address hits a cache line. In addition, the processor continues to support inquire cycles

initiated by the system logic, including the execution of writeback cycles when a modified cache line is hit.

While the L1 and L2 are inhibited, the processor continues to drive the PCD output signal appropriately, which system logic can use to control external L3 caching.

In order to completely disable the L1 and L2 caches so that no valid lines exist in the cache, the Cache Inhibit bit must be set to 1 and the cache must be flushed in one of the following ways:

- Asserting the FLUSH# input signal
- Executing the WBINVD instruction
- Executing the INVD instruction (modified cache lines are not written back to memory)
- Using the Page Flush/Invalidate Register (PFIR) (see “Page Flush/Invalidate Register (PFIR)” on page 223)

13.5 L2 Cache and Tag Array Testing

Level-2 Cache Array Access Register (L2AAR)

The AMD-K6-III+ processor provides the Level-2 Cache Array Access Register (L2AAR) that allows for direct access to the L2 cache and L2 tag arrays. The 256-Kbyte L2 cache in the AMD-K6-III+ is organized as shown in Figure 91 on page 265:

- Four 64-Kbyte ways
- Each way contains 1024 sets
- Each set contains four 64-byte sectors (one sector in each way)
- Each sector contains two 32-byte cache lines
- Each cache line contains four 8-byte octets
- Each octet contains an upper and lower dword (4 bytes)

Each line within a sector contains its own MESI state bits, and associated with each sector is a tag and LRU (least recently used) information.

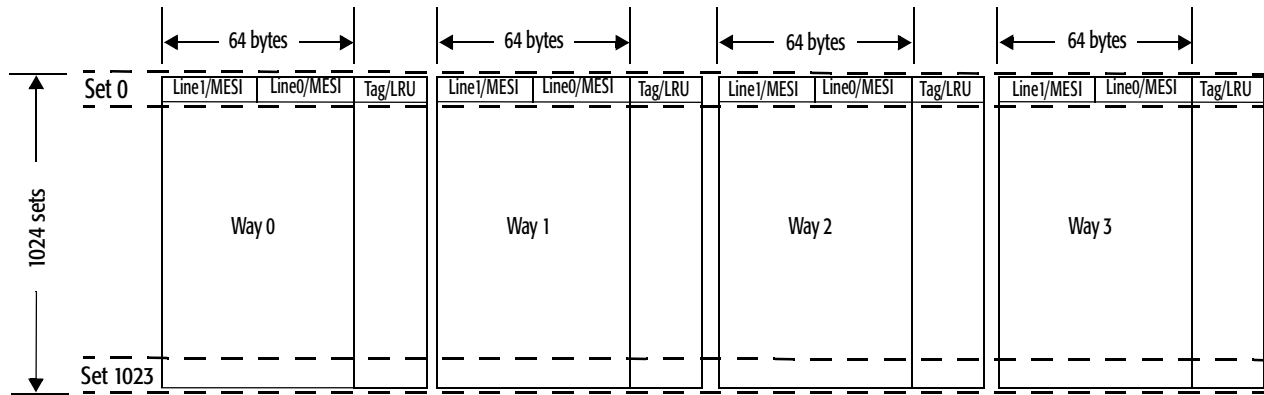


Figure 91. L2 Cache Organization for AMD-K6™-III+ Processor

Figure 92 shows the L2 cache sector and line organization. If bit 5 of the address of a cache line equals 1, then this cache line is stored in Line 1 of a sector. Similarly, if bit 5 of the address of a cache line equals 0, then this cache line is stored in Line 0 of a sector.

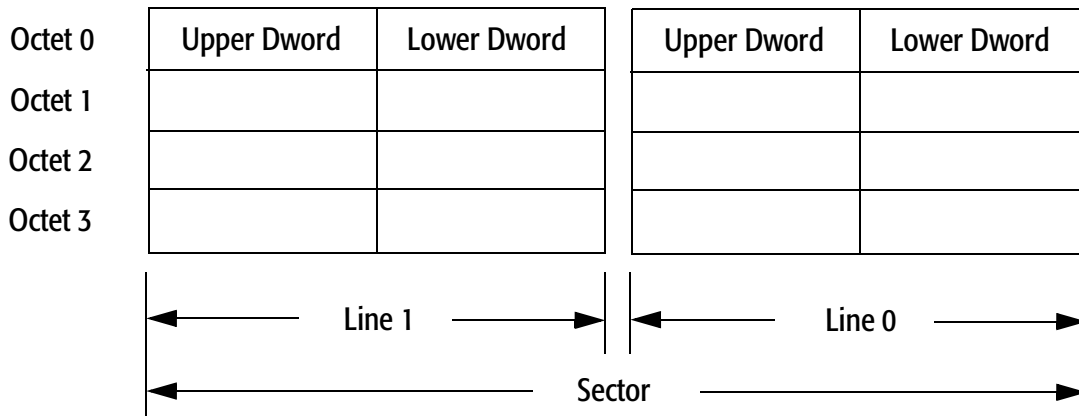


Figure 92. L2 Cache Sector and Line Organization

The L2AAR register is MSR C000_0089h. The operation that is performed on the L2 cache is a function of the instruction executed—RDMSR or WRMSR—and the contents of the EDX register. The EDX register specifies the location of the access, and whether the access is to the L2 cache data or tags (refer to Figure 93 on page 266). Bit 20 of EDX (T/D) determines whether the access is to the L2 cache data or tag. Table 53 on

page 266 describes the operation that is performed based on the instruction and the T/D bit.

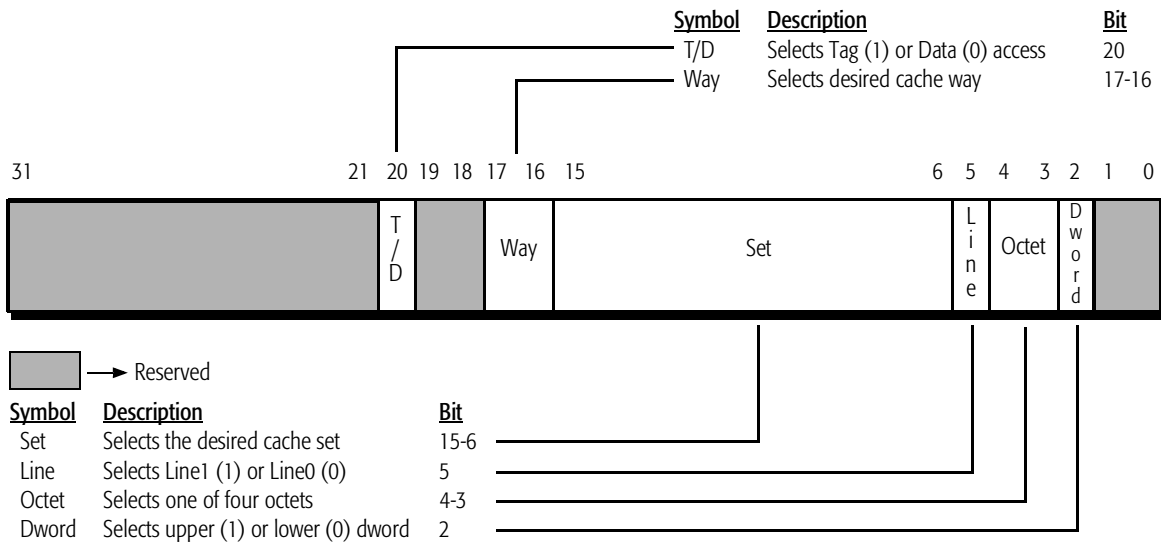


Figure 93. L2 Tag or Data Location for the AMD-K6™-III+ Processor—EDX

Table 53. Tag versus Data Selector

| Instruction | T/D (EDX[20]) | Operation |
|-------------|---------------|--|
| RDMSR | 0 | Read dword from L2 data array into EAX. Dword location is specified by EDX. |
| RDMSR | 1 | Read tag, line state and LRU information from L2 tag array into EAX. Location of tag is specified by EDX. |
| WRMSR | 0 | Write dword to the L2 data array using data in EAX. Dword location is specified by EDX. |
| WRMSR | 1 | Write tag, line state and LRU information into L2 tag array from EAX. Location of tag is specified by EDX. |

When the L2AAR is read or written, EDX is left unchanged. This facilitates multiple accesses when testing the entire cache/tag array.

L2 Cache Data Reads

If the L2 cache *data* is read (as opposed to reading the tag information), the result (dword) is placed in EAX in the format

as illustrated in Figure 94. Similarly, if the L2 cache data is written, the write data is taken from EAX.

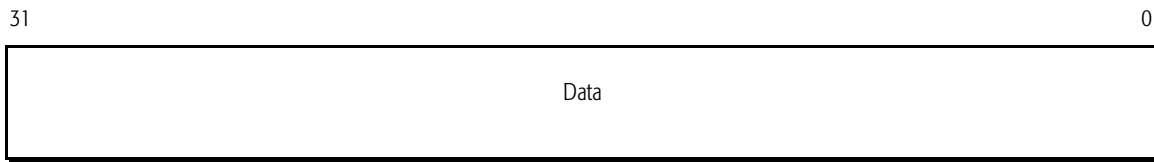


Figure 94. L2 Data - EAX

L2 Tag Reads

If the L2 *tag* is read (as opposed to reading the cache data), the result is placed in EAX in the format as illustrated in Figure 95 on page 268. Similarly, if the L2 tag is written, the write data is taken from EAX. When accessing the L2 tag, the Line, Octet, and Dword fields of the EDX register are ignored.

When writing to the L2 tag, special consideration must be given to the least significant bit of the Tag field of the EAX register—EAX[15]. The length of the L2 tag required to support the 256-Kbyte L2 cache on the AMD-K6-III+ processor is 16 bits, which corresponds to bits 31:16 of the EAX register. However, the processor provides a total of 17 bits for storing the L2 tag—that is, 16 bits for the tag (EAX[31:16]), plus an additional bit for internal purposes (EAX[15]). During normal operation, the processor ensures that this additional bit (bit 15) always corresponds to the set in which the tag resides. Note that bits 15:6 of the address determine the set, in which case if bit 15 is equal to 0, it addresses sets 0 through 511, and if bit 15 is equal to 1, it addresses sets 512 through 1023.

In order to set the full 17-bit L2 tag properly when using the L2AAR register, EAX[15] must likewise correspond to the set in which the tag is being written—that is, EAX[15] must be equal to EDX[15] (refer to Figure 93 on page 266 and Figure 95 on page 268).

It is important to note that this special consideration is required if the processor will subsequently be expected to properly execute instructions or access data from the L2 cache following the setup of the L2 cache by means of the L2AAR register. If the intent of using the L2AAR register is solely to test or debug the L2 cache without the subsequent intent of executing instructions or accessing data from the L2 cache, then this consideration is not required.

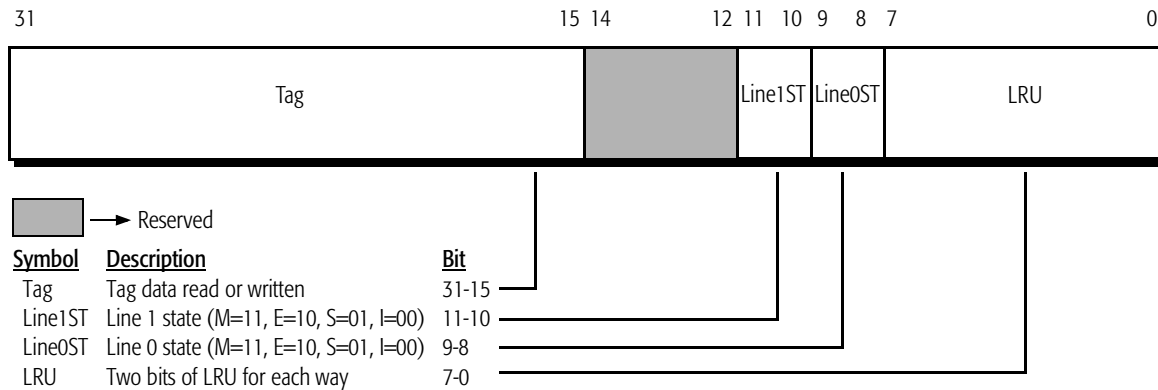
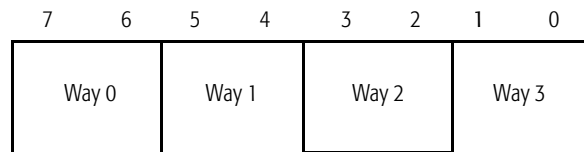


Figure 95. L2 Tag Information for the AMD-K6™-III+ Processor—EAX

LRU (Least Recently Used). For the 4-way set associative L2 cache, each way has a 2-bit LRU field for each sector. Values for the LRU field are 00b, 01b, 10b, and 11b, where 00b indicates that the sector is “most recently used,” and 11b indicates that the sector is “least recently used” (see Figure 96 on page 268). EAX[7:6] indicate LRU information for Way 0, EAX[5:4] for Way 1, EAX[3:2] for Way 2, and EAX[1:0] for Way 3.



LRU Values

- 00b Most Recently Used
- 01b Used More Recent Than 10b, But Less Recent Than 00b
- 10b Used More Recent Than 11b, But Less Recent Than 01b
- 11b Least Recently Used

Figure 96. LRU Byte

13.6 Debug

The AMD-K6-III+ processor implements the standard x86 debug functions, registers, and exceptions. In addition, the processor supports the I/O breakpoint debug extension. The

debug feature assists programmers and system designers during software execution tracing by generating exceptions when one or more events occur during processor execution. The exception handler, or debugger, can be written to perform various tasks, such as displaying the conditions that caused the breakpoint to occur, displaying and modifying register or memory contents, or single-stepping through program execution.

The following sections describe the debug registers and the various types of breakpoints and exceptions that the processor supports.

Debug Registers

Figures 97 through 100 show the 32-bit debug registers supported by the processor.

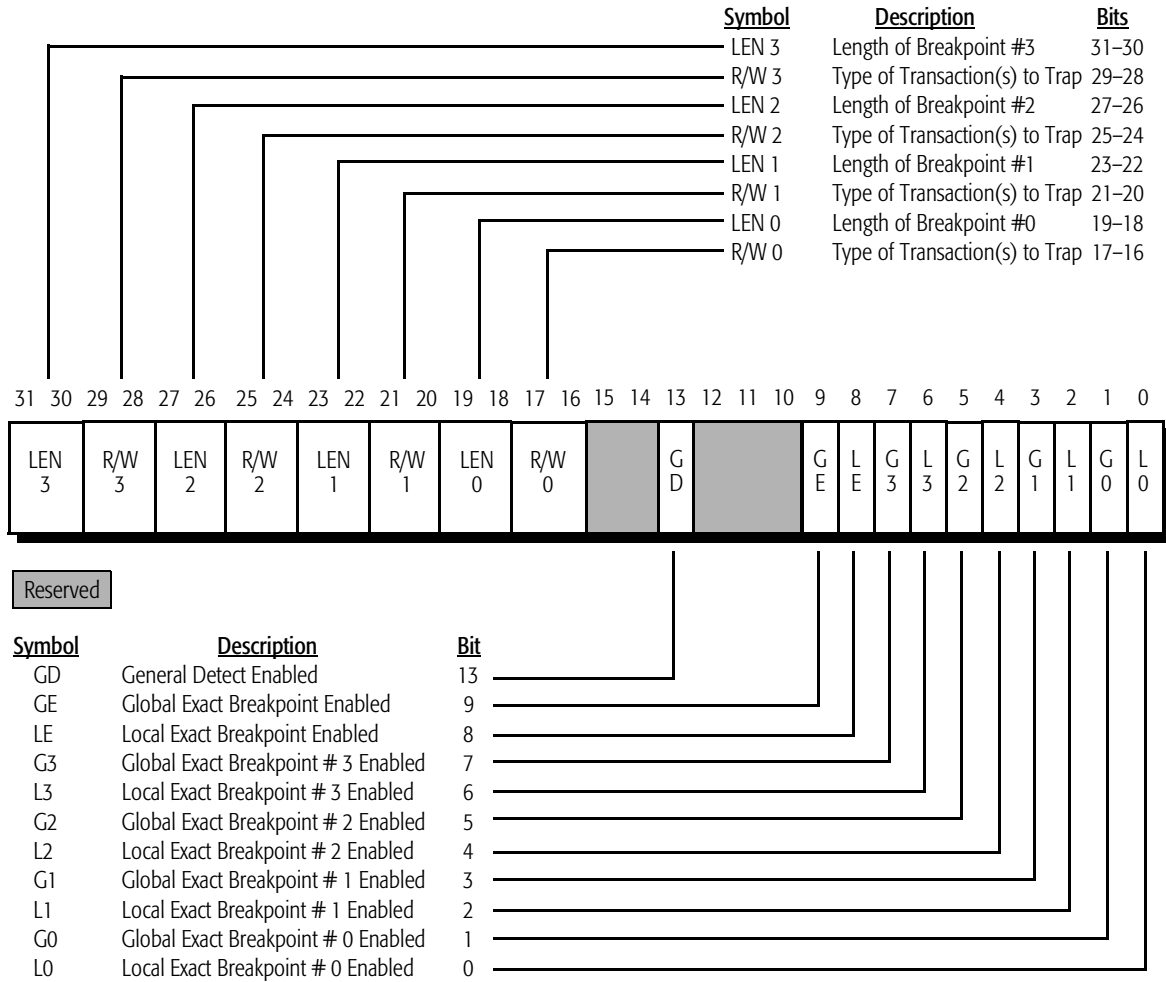


Figure 97. Debug Register DR7

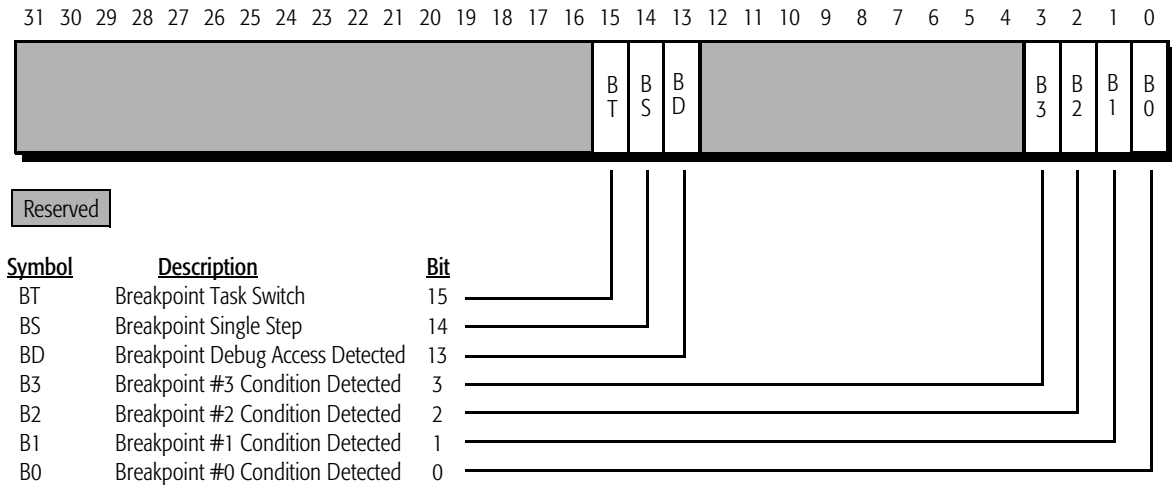
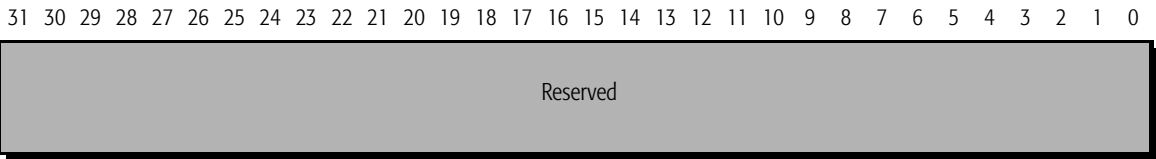


Figure 98. Debug Register DR6

DR5



DR4

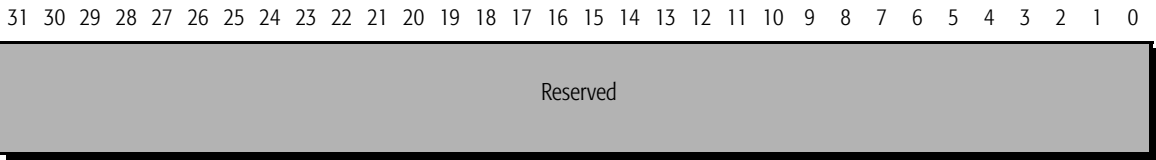


Figure 99. Debug Registers DR5 and DR4

DR3

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Breakpoint 3 32-bit Linear Address

DR2

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Breakpoint 2 32-bit Linear Address

DR1

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Breakpoint 1 32-bit Linear Address

DR0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Breakpoint 0 32-bit Linear Address

Figure 100. Debug Registers DR3, DR2, DR1, and DR0

DR3–DR0. The processor allows the setting of up to four breakpoints. DR3–DR0 contain the linear addresses for breakpoint 3 through breakpoint 0, respectively, and are compared to the linear addresses of processor cycles to determine if a breakpoint occurs. Debug register DR7 defines the specific type of cycle that must occur in order for the breakpoint to occur.

DR5–DR4. When debugging extensions are disabled (bit 3 of CR4 is set to 0), the DR5 and DR4 registers are mapped to DR7 and DR6, respectively, in order to be software compatible with previous generations of x86 processors. When debugging

extensions are enabled (bit 3 of CR4 is set to 1), any attempt to load DR5 or DR4 results in an undefined opcode exception. Likewise, any attempt to store DR5 or DR4 also results in an undefined opcode exception.

DR6. If a breakpoint is enabled in DR7, and the breakpoint conditions as defined in DR7 occur, then the corresponding B bit (B3–B0) in DR6 is set to 1. In addition, any other breakpoints defined using these particular breakpoint conditions are reported by the processor by setting the appropriate B-bits in DR6, regardless of whether these breakpoints are enabled or disabled. However, if a breakpoint is not enabled, a debug exception does not occur for that breakpoint.

If the processor decodes an instruction that writes or reads DR7 through DR0, the BD bit (bit 13) in DR6 is set to 1 (if enabled in DR7) and the processor generates a debug exception. This operation allows control to pass to the debugger prior to debug register access by software.

If the Trap Flag (bit 8) of the EFLAGS register is set to 1, the processor generates a debug exception after the successful execution of every instruction (single-step operation) and sets the BS bit (bit 14) in DR6 to indicate the source of the exception.

When the processor switches to a new task and the debug trap bit (T bit) in the corresponding Task State Segment (TSS) is set to 1, the processor sets the BT bit (bit 15) in DR6 and generates a debug exception.

DR7. When set to 1, L3–L0 locally enable breakpoints 3 through 0, respectively. L3–L0 are set to 0 whenever the processor executes a task switch. Setting L3–L0 to 0 disables the breakpoints and ensures that these particular debug exceptions are only generated for a specific task.

When set to 1, G3–G0 globally enable breakpoints 3 through 0, respectively. Unlike L3–L0, G3–G0 are not set to 0 whenever the processor executes a task switch. Not setting G3–G0 to 0 allows breakpoints to remain enabled across all tasks. If a breakpoint is enabled globally but disabled locally, the global enable overrides the local enable.

The LE (bit 8) and GE (bit 9) bits in DR7 have no effect on the operation of the processor and are provided in order to be software-compatible with previous generations of x86 processors.

When set to 1, the GD bit in DR7 (bit 13) enables the debug exception associated with the BD bit (bit 13) in DR6. This bit is set to 0 when a debug exception is generated.

LEN3–LEN0 and RW3–RW0 are two-bit fields in DR7 that specify the length and type of each breakpoint as defined in Table 54.

Table 54. DR7 LEN and RW Definitions

| LEN Bits ¹ | RW Bits | Breakpoint |
|-----------------------|------------------|------------------------------|
| 00b | 00b ² | Instruction Execution |
| 00b | 01b | One-byte Data Write |
| 01b | | Two-byte Data Write |
| 11b | | Four-byte Data Write |
| 00b | 10b ³ | One-byte I/O Read or Write |
| 01b | | Two-byte I/O Read or Write |
| 11b | | Four-byte I/O Read or Write |
| 00b | 11b | One-byte Data Read or Write |
| 01b | | Two-byte Data Read or Write |
| 11b | | Four-byte Data Read or Write |

Notes:

1. LEN bits equal to 10b is undefined.
2. When RW equals 00b, LEN must be equal to 00b.
3. When RW equals 10b, debugging extensions (DE) must be enabled (bit 3 of CR4 must be set to 1). If DE is set to 0, then RW equal to 10b is undefined.

Debug Exceptions

A debug exception is categorized as either a debug trap or a debug fault.

- A *debug trap* calls the debugger following the execution of the instruction that caused the trap.
- A *debug fault* calls the debugger prior to the execution of the instruction that caused the fault.

All debug traps and faults generate either an Interrupt 01h or an Interrupt 03h exception.

Interrupt 01h. The following events are considered debug traps that cause the processor to generate an Interrupt 01h exception:

- Enabled breakpoints for data and I/O cycles
- Single Step Trap
- Task Switch Trap

The following events are considered debug faults that cause the processor to generate an Interrupt 01h exception:

- Enabled breakpoints for instruction execution
- BD bit in DR6 set to 1

Interrupt 03h. The INT 3 instruction is defined in the x86 architecture as a breakpoint instruction. This instruction causes the processor to generate an Interrupt 03h exception. This exception is a debug trap because the debugger is called following the execution of the INT 3 instruction.

The INT 3 instruction is a one-byte instruction (opcode CCh) typically used to insert a breakpoint in software by writing CCh to the address of the first byte of the instruction to be trapped (the target instruction). Following the trap, if the target instruction is to be executed, the debugger must replace the INT 3 instruction with the first byte of the target instruction.

14 Clock Control

14.1 Clock Control States

The standard-power versions of the AMD-K6-III+ processor support five modes of clock control. The low-power versions of the AMD-K6-III+ processor support six modes of clock control.

The processor can transition between these modes to maximize performance, to minimize power dissipation, or to provide a balance between performance and power. (See “Power Dissipation” on page 291 for the maximum power dissipation of the AMD-K6-III+ processor within the normal and reduced-power states.) The clock-control states supported are:

- **Normal State**—The processor is running in Real Mode, Virtual-8086 Mode, Protected Mode, or System Management Mode (SMM). In this state, all clocks are running—including the external bus clock CLK and the internal processor clock—and the full features and functions of the processor are available.
- **Halt State**—This low-power state is entered following the successful execution of the HLT instruction. During this state, the internal processor clock is stopped.
- **Stop Grant State**—This low-power state is entered following the recognition of the assertion of the STPCLK# signal. During this state, the internal processor clock is stopped.
- **Stop Grant Inquire State**—This state is entered from the Halt state and the Stop Grant state as the result of a system-initiated inquire cycle.
- **Enhanced Power Management (EPM) Stop Grant State**: This low-power state is available on low-power versions of the processor. It is entered following the write of a non-zero value to the SGTC field of the EPM 16-byte I/O block for the purpose of performing dynamic processor core frequency and voltage ID state transitions using AMD PowerNow! technology. During this state, the internal processor clock is stopped.
- **Stop Clock State**—This low-power state is entered from the Stop Grant state when the CLK signal is stopped.

Figure 101 and Figure 102 illustrate the clock control state transitions on the standard-power and low-power versions, respectively, of the AMD-K6-III+ processor. Each of the reduced-power states are described in the following sections.

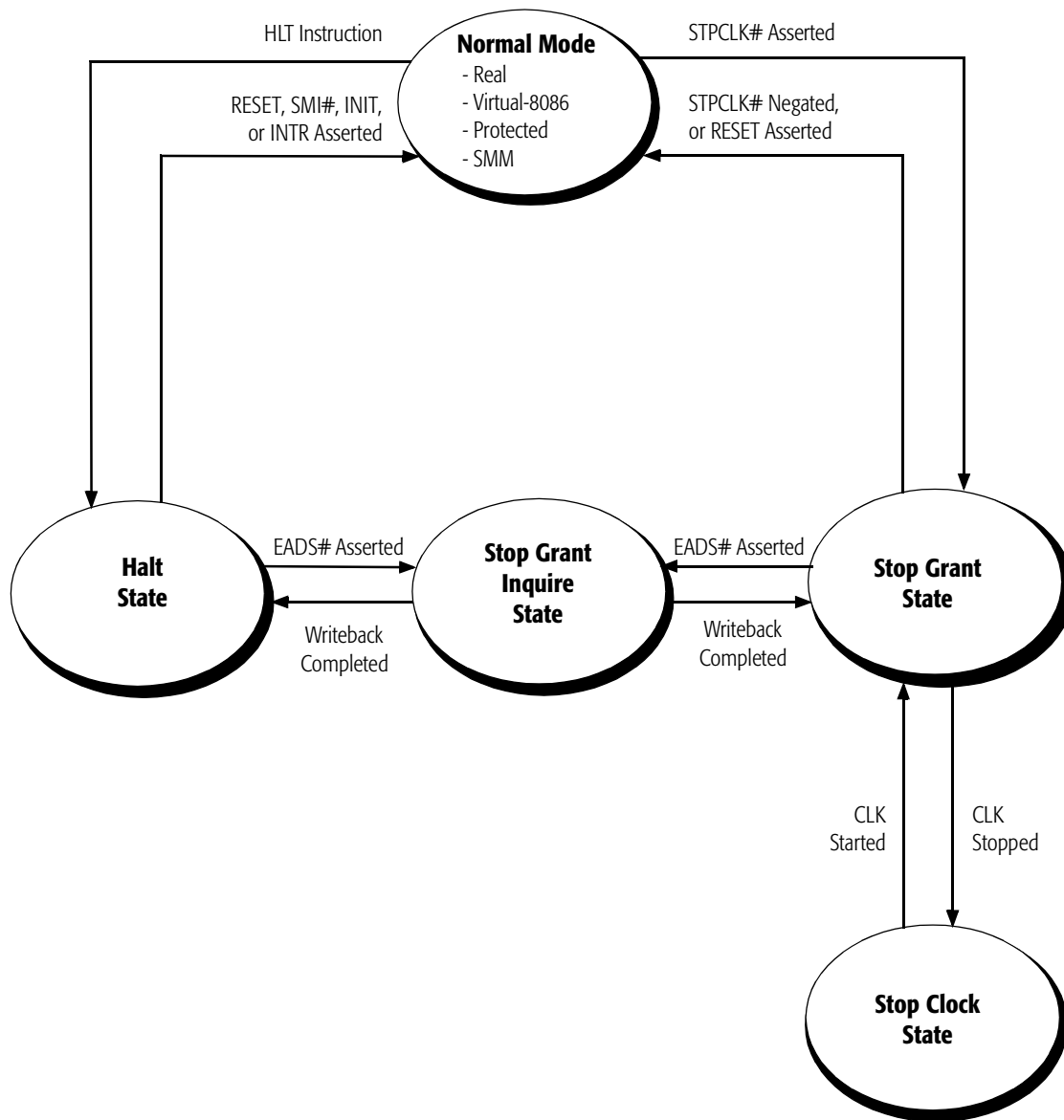


Figure 101. Clock Control State Transitions for Standard-Power Versions of the AMD-K6™-III+ Processor

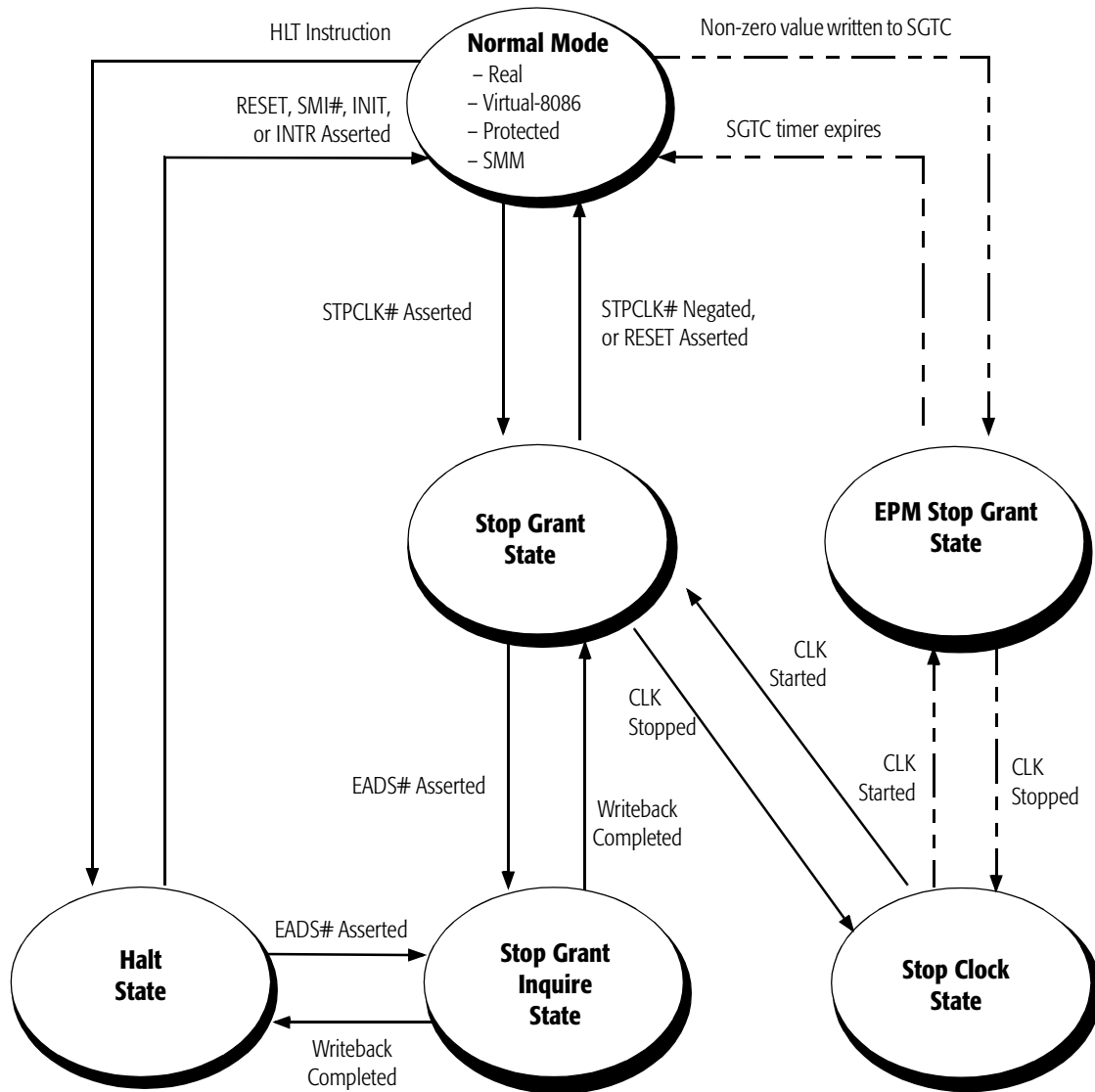


Figure 102. Clock Control State Transitions for Low-Power Versions of the AMD-K6™-III+ Processor

14.2 Halt State

Enter Halt State

During the execution of the HLT instruction, the AMD-K6-III+ processor executes a Halt special cycle. After BRDY# is sampled asserted during this cycle, and then EWBE# is also sampled asserted (if not masked off), the processor enters the Halt state in which the processor disables most of its internal clock distribution. In order to support the following operations, the internal phase-lock loop (PLL) still runs, and some internal resources are still clocked in the Halt state:

- **Inquire Cycles**—The processor continues to sample AHOLD, BOFF#, and HOLD in order to support inquire cycles that are initiated by the system logic. The processor transitions to the Stop Grant Inquire state during the inquire cycle. After returning to the Halt state following the inquire cycle, the processor does not execute another Halt special cycle.
- **Flush Cycles**—The processor continues to sample FLUSH#. If FLUSH# is sampled asserted, the processor performs the flush operation in the same manner as it is performed in the Normal state. Upon completing the flush operation, the processor executes the Halt special cycle, which indicates the processor is in the Halt state.
- **Time Stamp Counter (TSC)**—The TSC continues to count in the Halt state.
- **Signal Sampling**—The processor continues to sample INIT, INTR, NMI, RESET, and SMI#.

After entering the Halt state, all signals driven by the processor retain their state as they existed following the completion of the Halt special cycle.

Exit Halt State

The AMD-K6-III+ processor remains in the Halt state until it samples INIT, INTR (if interrupts are enabled), NMI, RESET, or SMI# asserted. If any of these signals is sampled asserted, the processor returns to the Normal state and performs the corresponding operation. All of the normal requirements for recognition of these input signals apply within the Halt state.

14.3 Stop Grant State

Enter Stop Grant State

After recognizing the assertion of STPCLK#, the AMD-K6-III+ processor flushes its instruction pipelines, completes all pending and in-progress bus cycles, and acknowledges the

STPCLK# assertion by executing a Stop Grant special bus cycle. After BRDY# is sampled asserted during this cycle, and then EWBE# is also sampled asserted (if not masked off), the processor enters the Stop Grant state.

The Stop Grant state is like the Halt state in that the processor disables most of its internal clock distribution in the Stop Grant state.

In order to support the following operations, the internal PLL still runs, and some internal resources are still clocked in the Stop Grant state:

- Inquire cycles—The processor transitions to the Stop Grant Inquire state during an inquire cycle. After returning to the Stop Grant state following the inquire cycle, the processor does not execute another Stop Grant special cycle.
- Time Stamp Counter (TSC)—The TSC continues to count in the Stop Grant state.
- Signal Sampling—The processor continues to sample INIT, INTR, NMI, RESET, and SMI#.

FLUSH# is not recognized in the Stop Grant state (unlike while in the Halt state).

Upon entering the Stop Grant state, all signals driven by the processor retain their state as they existed following the completion of the Stop Grant special cycle.

Exit Stop Grant State

The AMD-K6-III+ processor remains in the Stop Grant state until it samples STPCLK# negated or RESET asserted. If STPCLK# is sampled negated, the processor returns to the Normal state in less than 10 bus clock (CLK) periods. After the transition to the Normal state, the processor resumes execution at the instruction boundary on which STPCLK# was initially recognized.

If STPCLK# is recognized as negated in the Stop Grant state and subsequently sampled asserted prior to returning to the Normal state, the AMD-K6-III+ processor guarantees that a minimum of one instruction is executed prior to re-entering the Stop Grant state.

If INIT, INTR (if interrupts are enabled), FLUSH#, NMI, or SMI# are sampled asserted in the Stop Grant state, the

processor latches the edge-sensitive signals (INIT, FLUSH#, NMI, and SMI#), but otherwise does not exit the Stop Grant state to service the interrupt. When the processor returns to the Normal state due to sampling STPCLK# negated, any pending interrupts are recognized after returning to the Normal state. To ensure their recognition, all of the normal requirements for these input signals apply within the Stop Grant state.

If RESET is sampled asserted in the Stop Grant state, the processor immediately returns to the Normal state and the reset process begins.

14.4 Stop Grant Inquire State

Enter Stop Grant Inquire State

The Stop Grant Inquire state is entered from the Stop Grant state or the Halt state when EADS# is sampled asserted during an inquire cycle initiated by the system logic. The AMD-K6-III+ processor responds to an inquire cycle in the same manner as in the Normal state by driving HIT# and HITM#. If the inquire cycle hits a modified cache line, the processor performs a writeback cycle.

Exit Stop Grant Inquire State

Following the completion of any writeback, the processor returns to the state from which it entered the Stop Grant Inquire state.

14.5 EPM Stop Grant State

Enter EPM Stop Grant State

This state is supported on the low-power versions of the AMD-K6-III+ processor. After receiving a write of a non-zero value to the SGTC (Stop Grant Time-out Counter) field located within the EPM 16-byte I/O block, the processor flushes its instruction pipelines, completes all pending and in-progress bus cycles, and performs the following:

- Drives the processor VID[4:0] output pins to the value stored in the VIDO field of the EPM 16-byte I/O block (see “EPM 16-Byte I/O Block” on page 146) if the VIDC bit is set to 1.
- Forwards the processor-to-bus clock ratio stored in the IBF[2:0] field of the EPM 16-byte I/O block to the internal PLL if the BDC[1:0] value is set to 1xb.

The EPM Stop Grant state is like the Halt state in that the processor disables most of its internal clock distribution in the EPM Stop Grant state. In order to support the following operations, the internal PLL still runs, and some internal resources are still clocked in the EPM Stop Grant state.

- Time Stamp Counter (TSC): The TSC continues to count in the EPM Stop Grant state.
- Signal Sampling: The processor continues to sample INIT, INTR, NMI, RESET, and SMI#.

Unlike the Halt and Stop Grant states, system-initiated inquire cycles are not supported and must be prevented during the EPM Stop Grant state.

FLUSH# is not recognized in the EPM Stop Grant state (unlike while in the Halt state).

Upon entering the EPM Stop Grant state, all signals driven by the processor retain their state as they existed following the completion of the EPM Stop Grant special cycle.

Exit EPM Stop Grant State

The processor remains in the EPM Stop Grant state until the allotted time expires, as determined by the value written to the SGTC field, or until RESET is sampled asserted. Once the allotted time expires, the processor returns to the Normal state. After the transition to the Normal state, the processor resumes execution at the instruction boundary on which the EPM Stop Grant state was entered.

If INIT, INTR (if interrupts are enabled), FLUSH#, NMI, or SMI# are sampled asserted in the EPM Stop Grant state, the processor latches the edge-sensitive signals (INIT, FLUSH#, NMI, and SMI#), but otherwise does not exit the EPM Stop Grant state to service the interrupt. When the processor returns to the Normal state, any pending interrupts are recognized. To ensure their recognition, all of the normal requirements for these input signals apply within the EPM Stop Grant state.

If RESET is sampled asserted in the EPM Stop Grant state, the processor immediately returns to the Normal state and the reset process begins.

14.6 Stop Clock State

Enter Stop Clock State

If the CLK signal is stopped while the AMD-K6-III+ processor is in the Stop Grant state, the processor enters the Stop Clock state. Because all internal clocks and the PLL are not running in the Stop Clock state, the Stop Clock state represents the minimum-power state of all clock control states. The CLK signal must be held Low while it is stopped.

The Stop Clock state cannot be entered from the Halt state.

INTR is the only input signal that is allowed to change states while the processor is in the Stop Clock state. However, INTR is not sampled until the processor returns to the Stop Grant state. All other input signals must remain unchanged in the Stop Clock state.

Exit Stop Clock State

The AMD-K6-III+ processor returns to the Stop Grant state from the Stop Clock state after the CLK signal is started and the internal PLL has stabilized. PLL stabilization is achieved after the CLK signal has been running within its specification for a minimum of 1.0 ms.

The frequency of CLK when exiting the Stop Clock state can be different than the frequency of CLK when entering the Stop Clock state.

The state of the BF[2:0] signals when exiting the Stop Clock state is ignored because the BF[2:0] signals are only sampled during the falling transition of RESET.

15 Electrical Data

This chapter includes specifications for the operating ranges, absolute ratings, and DC characteristics of the AMD-K6-III+ embedded processor. Nominal and maximum power dissipation values for the AMD-K6-III+ processor during normal and reduced power states are listed. The chapter concludes with a discussion of power and grounding requirements.

15.1 Operating Ranges

The AMD-K6-III+ processor is designed to provide functional operation if the voltage and temperature parameters are within the limits defined in Table 55 and Table 56.

Table 55. Operating Ranges for Low-Power AMD-K6™-III+ Devices

| Parameter | Parameter Description | 400 MHz | 450 MHz | 500 MHz |
|-------------------------------|---|--------------------|--------------------|--------------------|
| V _{CC2} ¹ | Core Supply Voltage (Minimum) | 1.5 V ² | 1.6 V ³ | 1.7 V ⁴ |
| V _{CC2} ¹ | Core Supply Voltage (Nominal) | 1.6 V ² | 1.7 V ³ | 1.8 V ⁴ |
| V _{CC2} ¹ | Core Supply Voltage (Maximum) | 1.7 V ² | 1.8 V ³ | 1.9 V ⁴ |
| V _{CC3} ¹ | I/O Supply Voltage (Minimum) | 3.135 V | | |
| V _{CC3} ¹ | I/O Supply Voltage (Nominal) | 3.30 V | | |
| V _{CC3} ¹ | I/O Supply Voltage (Maximum) | 3.6 V | | |
| T _{CASE} | Case Temperature (Minimum) ⁵ | 0°C | | |
| T _{CASE} | Case Temperature (Maximum) ⁵ | 85°C | | |

Notes:

1. V_{CC2} and V_{CC3} are referenced from V_{SS}.
2. V_{CC2} specification for 1.6-V component.
3. V_{CC2} specification for 1.7-V component.
4. V_{CC2} specification for 1.8-V component.
5. Case temperature range required for AMD-K6-III+/400xTZ, AMD-K6-III+/450xPZ, and AMD-K6-III+/500xNZ valid ordering part number combinations, where x represents the package type. See Table 79 on page 336 for a complete list of valid OPNs.

Table 56. Operating Ranges for Standard-Power AMD-K6™-III+ Devices

| Parameter | Parameter Description | 400 MHz | 450 MHz | 500 MHz | 550 MHz |
|-------------------------------|--|---------|---------|---------|---------|
| V _{CC2} ¹ | Core Supply Voltage (Minimum) ² | 1.9 V | | | |
| V _{CC2} ¹ | Core Supply Voltage (Nominal) ² | 2.0 V | | | |
| V _{CC2} ¹ | Core Supply Voltage (Maximum) ² | 2.1 V | | | |
| V _{CC3} ¹ | I/O Supply Voltage (Minimum) | 3.135 V | | | |
| V _{CC3} ¹ | I/O Supply Voltage (Nominal) | 3.30 V | | | |
| V _{CC3} ¹ | I/O Supply Voltage (Maximum) | 3.6 V | | | |
| T _{CASE} | Case Temperature (Minimum) ³ | 0°C | | | |
| T _{CASE} | Case Temperature (Maximum) ³ | 70°C | | | |

Notes:

1. V_{CC2} and V_{CC3} are referenced from V_{SS}.
2. V_{CC2} specification for 2.0-V component.
3. Case temperature range required for AMD-K6-III+/xxxYACR valid ordering part number combinations, where xxx represents the processor core frequency and y represents the package type, as defined in Table 79 on page 336.

15.2 Absolute Ratings

The AMD-K6-III+ processor is not designed to be operated beyond the operating ranges listed in Table 55 and Table 56. Exposure to conditions outside these operating ranges for extended periods of time can affect long-term reliability. Permanent damage can occur if the absolute ratings listed in Table 57 are exceeded.

Table 57. Absolute Ratings

| Parameter | Description | Minimum | Maximum |
|-------------------------|------------------------|---------|--|
| V_{CC2} | Core Supply Voltage | -0.5 V | 2.2 V |
| V_{CC3} | I/O Supply Voltage | -0.5 V | 3.6 V |
| V_{PIN}^1 | Voltage on Any I/O Pin | -0.5 V | $V_{CC3} + 0.4 \text{ V}$ and $\leq 3.8 \text{ V}$ |
| T_{CASE} (under bias) | Case Temperature | -65°C | +110°C |
| $T_{STORAGE}$ | Storage Temperature | -65°C | +150°C |

Notes:

- V_{PIN} (the voltage on any I/O pin) must not be greater than 0.4 V above the voltage being applied to V_{CC3} . In addition, the V_{PIN} voltage must never exceed 3.8 V.

15.3 DC Characteristics

The DC characteristics of the AMD-K6™-III+ processor are shown in Table 58.

Table 58. DC Characteristics for the AMD-K6™-III+ Processor

| Symbol | Parameter Description | Preliminary Data | | Comments |
|------------|-----------------------|------------------|---------------------------|-------------------------------|
| | | Min | Max | |
| V_{IL} | Input Low Voltage | -0.3 V | +0.8 V | |
| V_{IH}^1 | Input High Voltage | 2.0 V | $V_{CC3} + 0.3 \text{ V}$ | |
| V_{OL} | Output Low Voltage | | 0.4 V | $I_{OL} = 4.0\text{-mA}$ load |
| V_{OH} | Output High Voltage | 2.4 V | | $I_{OH} = 3.0\text{-mA}$ load |

Table 58. DC Characteristics for the AMD-K6™-III+ Processor (continued)

| Symbol | Parameter Description | Preliminary Data | | Comments |
|-------------------------------------|--|------------------|---------|--------------------------|
| | | Min | Max | |
| I_{CC2} Low Power | 1.6-V Power Supply Current | | 6.15 A | 400 MHz ^{2,3,4} |
| | 1.7-V Power Supply Current | | 7.60 A | 450 MHz ^{2,3} |
| | 1.8-V Power Supply Current | | 8.60 A | 500 MHz ^{2,3} |
| I_{CC2} Standard Power | 2.0 V Power Supply Current | | 8.70 A | 400 MHz ^{3,4,5} |
| | | | 9.25 A | 450 MHz ^{3,5} |
| | | | 9.75 A | 500 MHz ^{3,5} |
| | | | 10.30 A | 550 MHz ^{3,5} |
| I_{CC3} Standard and Low Power | 3.3 V Power Supply Current | | 0.65 A | 400 MHz ^{3,4,6} |
| | | | 0.66 A | 450 MHz ^{3,6} |
| | | | 0.68 A | 500 MHz ^{3,6} |
| | | | 0.69 A | 550 MHz ^{3,6} |
| I_{LI} ⁷ | Input Leakage Current | | ±15 µA | |
| I_{LO} ⁷ | Output Leakage Current | | ±15 µA | |
| I_{IL} ⁸ | Input Leakage Current Bias with Pullup | | -500 µA | |
| I_{IH} ⁹ | Input Leakage Current Bias with Pulldown | | 500 µA | |
| C_{IN} | Input Capacitance | | 10 pF | |
| C_{OUT} | Output Capacitance | | 15 pF | |
| C_{OUT} | I/O Capacitance | | 20 pF | |
| C_{CLK} | CLK Capacitance | | 10 pF | |
| C_{TIN} | Test Input Capacitance (TDI, TMS, TRST#) | | 10 pF | |
| C_{TOUT} | Test Output Capacitance (TDO) | | 15 pF | |
| C_{TCK} | TCK Capacitance | | 10 pF | |

Notes:

1. V_{CC3} refers to the voltage being applied to V_{CC3} during functional operation.
2. V_{CC2} = Maximum V_{CC2} as listed in Table 55 on page 288— The maximum power supply current must be taken into account when designing a power supply.
3. This specification applies to components using a CLK frequency of 100 MHz.
4. This specification applies to components using a CLK frequency of 66 MHz (66-MHz bus applies to 400-MHz part only).
5. $V_{CC2} = 2.1$ V — The maximum power supply current must be taken into account when designing a power supply.
6. $V_{CC3} = 3.6$ V—The maximum power supply current must be taken into account when designing a power supply.
7. Refers to inputs and I/O without an internal pullup resistor and $0 \leq V_{IN} \leq V_{CC3}$.
8. Refers to inputs with an internal pullup and $V_{IL} = 0.4$ V.
9. Refers to inputs with an internal pulldown and $V_{IH} = 2.4$ V.

15.4 Power Dissipation

Table 59 and Table 60 contain the application power dissipation of the low-power and standard-power AMD-K6-III+ processor during normal and reduced power states. Table 61 on page 292 shows the supported voltages and operating frequencies for low-power versions of AMD-K6-III+ processors enabled with AMD PowerNow! technology.

Table 59. Power Dissipation for Low-Power AMD-K6™-III+ Devices

| Power Dissipation | | 400 MHz ^{1,2} | 450 MHz ¹ | 500 MHz ¹ |
|---|---|------------------------|----------------------|----------------------|
| Application Power | Active ³ | 7.10 W | 8.95 W | 11.40 W |
| | AMD PowerNow! Technology Power Saving Mode ⁴ | 2.95 W | | |
| Thermal Design Power (Maximum) ^{5,6} | | 9.50 W | 12.00 W | 14.50 W |
| Stop Grant/Halt (Maximum) ^{6,7} | V _{CC} Nominal | 2.50 W | | |
| Stop Clock (Maximum) ^{6,8} | V _{CC} Nominal | 1.60 W | 1.90 W | |
| | Lowest Operating V _{CC} ⁹ | 1.30 W | | |

Notes:

1. This specification applies to components using a CLK frequency of 100 MHz.
2. This specification applies to components using a CLK frequency of 66 MHz.
3. The active application power measurements were taken by running a suite of embedded benchmarks covering four major embedded market segments: automotive, office automation, networking, and telecommunications.
4. AMD PowerNow! technology Power Saving Mode represents averaged values measured while running the processor in the lowest settings supported by AMD PowerNow! technology.
5. The maximum power dissipated in the normal clock control state must be taken into account when designing a solution for thermal dissipation for the AMD-K6-2E+ processor.
6. Maximum power is determined for the worst-case instruction sequence or function for the listed clock control states with V_{CC2} = Nominal V_{CC2} as listed in Table 55 on page 288 and V_{CC3} = 3.3 V.
7. The CLK signal and the internal PLL are still running, but most internal clocking has stopped.
8. The CLK signal, the internal PLL, and all internal clocking has stopped.
9. The lowest operating V_{CC} is 1.4 V.

Table 60. Power Dissipation for Standard-Power AMD-K6™-III+ Devices

| Power Dissipation | | 400 MHz ^{1,2} | 450 MHz ¹ | 500 MHz ¹ | 550 MHz ¹ |
|---|---------------------|------------------------|----------------------|----------------------|----------------------|
| Application Power | Active ³ | 11.75 W | 13.00 W | 14.35 W | 15.70 W |
| Thermal Design Power (Maximum) ^{4,5} | | 16.50 W | 17.50 W | 18.50 W | 19.50 W |
| Stop Grant/Halt (Maximum) ^{5,6} | | 4.50 W | | | |
| Stop Clock (Maximum) ^{5,7} | | 4.00 W | | | |

Notes:

1. This specification applies to components using a CLK frequency of 100 MHz.
2. This specification applies to components using a CLK frequency of 66 MHz.
3. The active application power measurements were taken by running a suite of embedded benchmarks covering four major embedded market segments: automotive, office automation, networking, and telecommunications.
4. The maximum power dissipated in the normal clock control state must be taken into account when designing a solution for thermal dissipation for the AMD-K6-2E+ processor.
5. Maximum power is determined for the worst-case instruction sequence or function for the listed clock control states with $V_{CC2} = 2.0\text{ V}$ and $V_{CC3} = 3.3\text{ V}$.
6. The CLK signal and the internal PLL are still running, but most internal clocking has stopped.
7. The CLK signal, the internal PLL, and all internal clocking has stopped.

Table 61. Supported Frequencies and Voltages for Low-Power AMD-K6™-III+ Processors Enabled with AMD PowerNow!™ Technology

| Ordering Part Number ¹ | Core Voltage | Range of Supported Operating Frequencies ² | Active Power ³ |
|-----------------------------------|--------------|---|---------------------------|
| AMD-K6-III+500ANZ | 1.8 V | 500–200 MHz | 11.40–5.80 W |
| | 1.7 V | 450–200 MHz | 8.95–4.90 W |
| | 1.6 V | 400–200 MHz | 7.10–4.20 W |
| | 1.5 V | 350–200 MHz | 5.60–3.70 W |
| | 1.4 V | 300–200 MHz | 4.30–2.95 W |
| AMD-K6-III+450APZ | 1.7 V | 450–200 MHz | 8.95–4.90 W |
| | 1.6 V | 400–200 MHz | 7.10–4.20 W |
| | 1.5 V | 350–200 MHz | 5.60–3.70 W |
| | 1.4 V | 300–200 MHz | 4.30–2.95 W |
| AMD-K6-III+400xTZ | 1.6 V | 400–200 MHz | 7.10–4.20 W |
| | 1.5 V | 350–200 MHz | 5.60–3.70 W |
| | 1.4 V | 300–200 MHz | 4.30–2.95 W |

Notes:

1. An x in this column represents the package type. See Table 79, "AMD-K6™-III+ Embedded Processor Valid Ordering Part Number Combinations," on page 336.
2. AMD PowerNow! technology enables the operating frequency to step down in increments corresponding to the available bus frequency multipliers. Note that 250-MHz operation is not supported due to exclusion of 2.5 bus frequency multiplier.
3. Active application power dissipation for highest and lowest supported frequency at specified voltage.

15.5 Power and Grounding

Power Connections

The AMD-K6-III+ processor is a dual voltage device. Two separate supply voltages are required: V_{CC2} and V_{CC3} .

- V_{CC2} provides the core voltage for the processor.
- V_{CC3} provides the I/O voltage.

See “Operating Ranges” on page 288 for the value and range of V_{CC2} and V_{CC3} .

The power and ground pins for each package are listed in Table 76 on page 327 and Table 78 on page 331. Table 74 on page 323 lists the pin differences between the two packages. The large number of power and ground pins are provided to ensure that the processor and package maintain a clean and stable power distribution network.

For proper operation and functionality, all V_{CC2} , V_{CC3} , and V_{SS} pins must be connected to the appropriate planes in the circuit board. The power planes have been arranged in a pattern to simplify routing and minimize crosstalk on the circuit board. The isolation region between two voltage planes must be at least 0.254mm if they are in the same layer of the circuit board. (See Figure 103 on page 294.) In order to maintain a low-impedance current sink and reference, the ground plane must never be split.

Although the AMD-K6-III+ processor has two separate supply voltages, there are no special power sequencing requirements. The best procedure is to minimize the time between which V_{CC2} and V_{CC3} are either both on or both off.

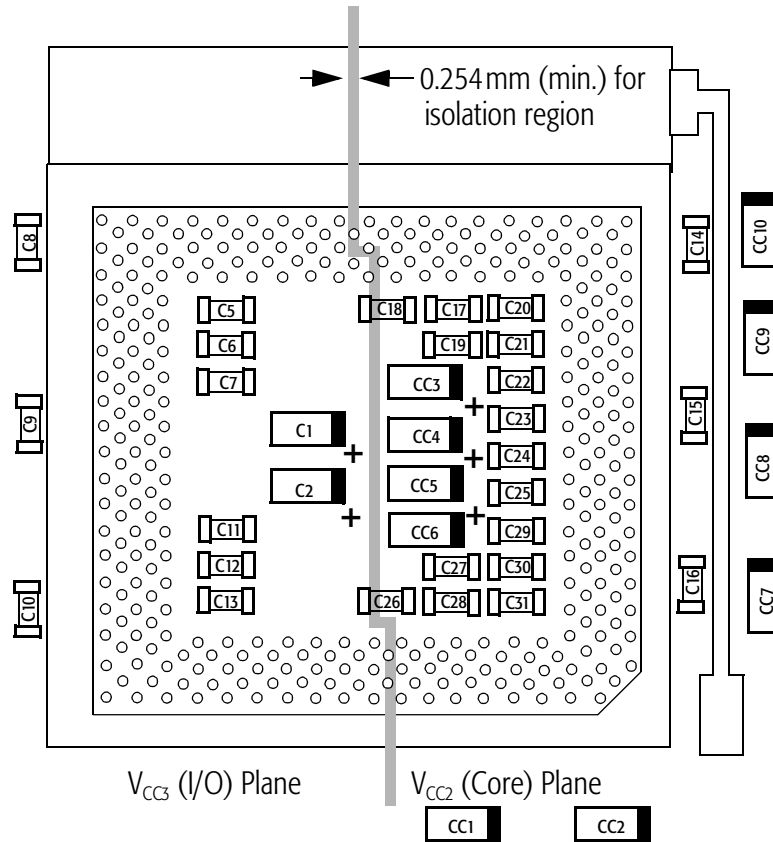


Figure 103. Suggested Component Placement for CPGA Package

Decoupling Recommendations

In addition to the isolation region mentioned in “Power Connections” on page 293, adequate decoupling capacitance is required between the two system power planes and the ground plane to minimize ringing and to provide a low-impedance path for return currents. Suggested decoupling capacitor placement is shown in Figure 103.

Surface-mounted capacitors should be used under the processor’s ZIF socket to minimize resistance and inductance in the lead lengths while maintaining minimal height. For information and recommendations about the specific value, quantity, and location of the capacitors, see the *AMD-K6® Processor Power Supply Design Application Note*, order# 21103.

Pin Connection Requirements

For proper operation, the following requirements for signal pin connections must be met:

- Do not drive address and data signals into large capacitive loads at high frequencies. If necessary, use buffer chips to drive large capacitive loads.
- Leave all NC (no-connect) pins unconnected.
- Unused inputs should always be connected to an appropriate signal level.
 - Active Low inputs that are not being used should be connected to V_{CC3} through a 20-k Ω pullup resistor.
 - Active High inputs that are not being used should be connected to GND through a pulldown resistor.
- Reserved signals can be treated in one of the following ways:
 - As no-connect (NC) pins, in which case these pins are left unconnected
 - As pins connected to the system logic as defined by the industry-standard Super7 and Socket 7 interface
 - Any combination of NC and Socket 7 pins
- Keep trace lengths to a minimum.

16 Signal Switching Characteristics

The AMD-K6-III+ processor signal switching characteristics are presented in Table 62 through Table 71 on the following pages. Valid delay, float, setup, and hold timing specifications are listed. These specifications are provided for the system designer to determine if the timings necessary for the processor to interface with the system logic are met.

- Table 62 on page 298 and Table 63 on page 299 contain the switching characteristics of the CLK input.
- Table 64 on page 300 through Table 67 on page 306 contain the timings for the normal operation signals.
- Table 68 on page 308 and Table 69 on page 309 contain the timings for RESET and the configuration signals.
- Table 70 on page 310 and Table 71 on page 310 contain the timings for the test operation signals.

All signal timings provided are:

- Measured between CLK, TCK, or RESET at 1.5 V and the corresponding signal at 1.5 V—this applies to input and output signals that are switching from Low to High, or from High to Low
- Based on input signals applied at a slew rate of 1 V/ns between 0 V and 3 V (rising) and 3 V to 0 V (falling)
- Valid within the operating ranges given in “Operating Ranges” on page 288
- Based on a load capacitance (C_L) of 0 pF

16.1 CLK Switching Characteristics

Table 62 and Table 63 on page 299 contain the switching characteristics of the CLK input to the AMD-K6-III+ processor for 100-MHz and 66-MHz bus operation, respectively, as measured at the voltage levels indicated by Figure 104 on page 299.

The CLK Period Stability parameter specifies the variance (jitter) allowed between successive periods of the CLK input measured at 1.5 V. This parameter must be considered as one of the elements of clock skew between the AMD-K6-III+ processor and the system logic.

16.2 Clock Switching Characteristics for 100-MHz Bus Operation

Table 62. CLK Switching Characteristics for 100-MHz Bus Operation

| Symbol | Parameter Description | Preliminary Data | | Figure | Comments |
|----------------|-----------------------------------|------------------|----------|--------|----------------|
| | | Min | Max | | |
| | Frequency | 33.3 MHz | 100 MHz | | In Normal Mode |
| t ₁ | CLK Period | 10.0 ns | | 104 | In Normal Mode |
| t ₂ | CLK High Time | 3.0 ns | | 104 | |
| t ₃ | CLK Low Time | 3.0 ns | | 104 | |
| t ₄ | CLK Fall Time | 0.15 ns | 1.5 ns | 104 | |
| t ₅ | CLK Rise Time | 0.15 ns | 1.5 ns | 104 | |
| | CLK Period Stability ¹ | | ± 250 ps | | |

Notes:

1. The jitter frequency power spectrum peaking must occur at frequencies greater than (Frequency of CLK)/3 or less than 500 kHz.

16.3 Clock Switching Characteristics for 66-MHz Bus Operation

Table 63. CLK Switching Characteristics for 66-MHz Bus Operation

| Symbol | Parameter Description | Preliminary Data | | Figure | Comments |
|--------|-----------------------------------|------------------|----------|--------|----------------|
| | | Min | Max | | |
| | Frequency | 33.3 MHz | 66.6 MHz | | In Normal Mode |
| t_1 | CLK Period | 15.0 ns | 30.0 ns | 104 | In Normal Mode |
| t_2 | CLK High Time | 4.0 ns | | 104 | |
| t_3 | CLK Low Time | 4.0 ns | | 104 | |
| t_4 | CLK Fall Time | 0.15 ns | 1.5 ns | 104 | |
| t_5 | CLK Rise Time | 0.15 ns | 1.5 ns | 104 | |
| | CLK Period Stability ¹ | | ± 250 ps | | |

Notes:

1. The jitter frequency power spectrum peaking must occur at frequencies greater than (Frequency of CLK)/3 or less than 500 kHz.

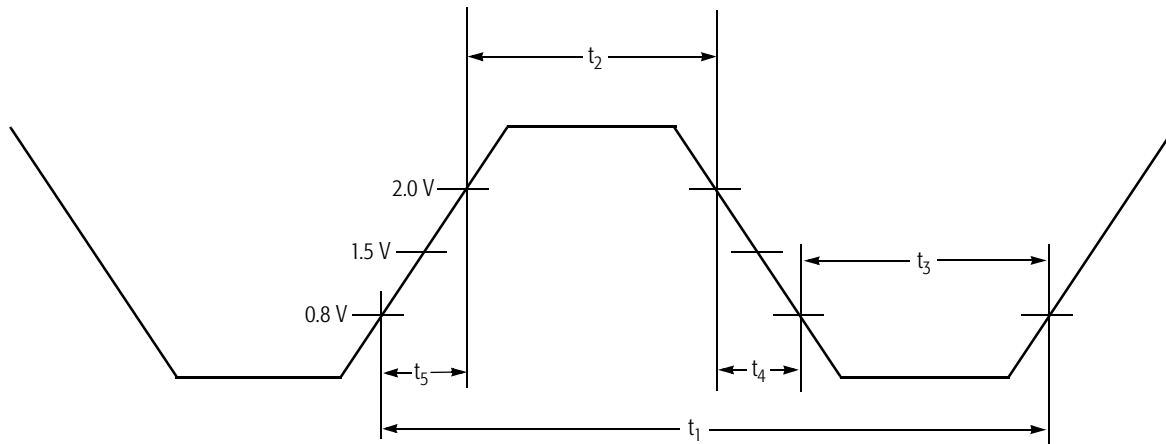


Figure 104. CLK Waveform

16.4 Valid Delay, Float, Setup, and Hold Timings

Valid Delay and Float Timing

The maximum valid delay timings are provided to allow a system designer to determine if setup times to the system logic can be met. Likewise, the minimum valid delay timings are used to analyze hold times to the system logic.

- Valid delay and float timings are given for output signals during functional operation and are given relative to the rising edge of CLK.
- During boundary-scan testing, valid delay and float timings for output signals are with respect to the falling edge of TCK.

Setup and Hold Timing

The setup and hold time requirements for the AMD-K6-III+ processor input signals must be met by the system logic to assure the proper operation of the AMD-K6-III+ processor.

- The setup and hold timings during functional and boundary-scan test mode are given relative to the rising edge of CLK and TCK, respectively.

16.5 Output Delay Timings for 100-MHz Bus Operation

Table 64. Output Delay Timings for 100-MHz Bus Operation

| Symbol | Parameter Description | Preliminary Data | | Figure |
|-----------------|-----------------------|------------------|--------|--------|
| | | Min | Max | |
| t ₆ | A[31:3] Valid Delay | 1.1 ns | 4.0 ns | 106 |
| t ₇ | A[31:3] Float Delay | | 7.0 ns | 107 |
| t ₈ | ADS# Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₉ | ADS# Float Delay | | 7.0 ns | 107 |
| t ₁₀ | ADSC# Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₁₁ | ADSC# Float Delay | | 7.0 ns | 107 |
| t ₁₂ | AP Valid Delay | 1.0 ns | 5.5 ns | 106 |
| t ₁₃ | AP Float Delay | | 7.0 ns | 107 |
| t ₁₄ | APCHK# Valid Delay | 1.0 ns | 4.5 ns | 106 |
| t ₁₅ | BE[7:0]# Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₁₆ | BE[7:0]# Float Delay | | 7.0 ns | 107 |
| t ₁₇ | BREQ Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₁₈ | CACHE# Valid Delay | 1.0 ns | 4.0 ns | 106 |

Table 64. Output Delay Timings for 100-MHz Bus Operation (continued)

| Symbol | Parameter Description | Preliminary Data | | Figure |
|-----------------|--------------------------------|------------------|--------|--------|
| | | Min | Max | |
| t ₁₉ | CACHE# Float Delay | | 7.0 ns | 107 |
| t ₂₀ | D/C# Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₂₁ | D/C# Float Delay | | 7.0 ns | 107 |
| t ₂₂ | D[63:0] Write Data Valid Delay | 1.3 ns | 4.5 ns | 106 |
| t ₂₃ | D[63:0] Write Data Float Delay | | 7.0 ns | 107 |
| t ₂₄ | DP[7:0] Write Data Valid Delay | 1.3 ns | 4.5 ns | 106 |
| t ₂₅ | DP[7:0] Write Data Float Delay | | 7.0 ns | 107 |
| t ₂₆ | FERR# Valid Delay | 1.0 ns | 4.5 ns | 106 |
| t ₂₇ | HIT# Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₂₈ | HITM# Valid Delay | 1.1 ns | 4.0 ns | 106 |
| t ₂₉ | HLDA Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₃₀ | LOCK# Valid Delay | 1.1 ns | 4.0 ns | 106 |
| t ₃₁ | LOCK# Float Delay | | 7.0 ns | 107 |
| t ₃₂ | M/IO# Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₃₃ | M/IO# Float Delay | | 7.0 ns | 107 |
| t ₃₄ | PCD Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₃₅ | PCD Float Delay | | 7.0 ns | 107 |
| t ₃₆ | PCHK# Valid Delay | 1.0 ns | 4.5 ns | 106 |
| t ₃₇ | PWT Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₃₈ | PWT Float Delay | | 7.0 ns | 107 |
| t ₃₉ | SCYC Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₄₀ | SCYC Float Delay | | 7.0 ns | 107 |
| t ₄₁ | SMIACK# Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₄₂ | W/R# Valid Delay | 1.0 ns | 4.0 ns | 106 |
| t ₄₃ | W/R# Float Delay | | 7.0 ns | 107 |

16.6 Input Setup and Hold Timings for 100-MHz Bus Operation

Table 65. Input Setup and Hold Timings for 100-MHz Bus Operation

| Symbol | Parameter Description | Preliminary Data | | Figure |
|------------|------------------------------|------------------|-----|--------|
| | | Min | Max | |
| t_{44} | A[31:5] Setup Time | 3.0 ns | | 108 |
| t_{45} | A[31:5] Hold Time | 1.0 ns | | 108 |
| t_{46}^1 | A20M# Setup Time | 3.0 ns | | 108 |
| t_{47}^1 | A20M# Hold Time | 1.0 ns | | 108 |
| t_{48} | AHOLD Setup Time | 3.5 ns | | 108 |
| t_{49} | AHOLD Hold Time | 1.0 ns | | 108 |
| t_{50} | AP Setup Time | 1.7 ns | | 108 |
| t_{51} | AP Hold Time | 1.0 ns | | 108 |
| t_{52} | BOFF# Setup Time | 3.5 ns | | 108 |
| t_{53} | BOFF# Hold Time | 1.0 ns | | 108 |
| t_{54} | BRDY# Setup Time | 3.0 ns | | 108 |
| t_{55} | BRDY# Hold Time | 1.0 ns | | 108 |
| t_{56} | BRDYC# Setup Time | 3.0 ns | | 108 |
| t_{57} | BRDYC# Hold Time | 1.0 ns | | 108 |
| t_{58} | D[63:0] Read Data Setup Time | 1.7 ns | | 108 |
| t_{59} | D[63:0] Read Data Hold Time | 1.5 ns | | 108 |
| t_{60} | DP[7:0] Read Data Setup Time | 1.7 ns | | 108 |
| t_{61} | DP[7:0] Read Data Hold Time | 1.5 ns | | 108 |
| t_{62} | EADS# Setup Time | 3.0 ns | | 108 |
| t_{63} | EADS# Hold Time | 1.0 ns | | 108 |
| t_{64} | EWBE# Setup Time | 1.7 ns | | 108 |
| t_{65} | EWBE# Hold Time | 1.0 ns | | 108 |
| t_{66}^2 | FLUSH# Setup Time | 1.7 ns | | 108 |
| t_{67}^2 | FLUSH# Hold Time | 1.0 ns | | 108 |
| t_{68} | HOLD Setup Time | 1.7 ns | | 108 |
| t_{69} | HOLD Hold Time | 1.5 ns | | 108 |
| t_{70}^1 | IGNNE# Setup Time | 1.7 ns | | 108 |
| t_{71}^1 | IGNNE# Hold Time | 1.0 ns | | 108 |
| t_{72}^2 | INIT Setup Time | 1.7 ns | | 108 |

Table 65. Input Setup and Hold Timings for 100-MHz Bus Operation (continued)

| Symbol | Parameter Description | Preliminary Data | | Figure |
|------------|-----------------------|------------------|-----|--------|
| | | Min | Max | |
| t_{73}^2 | INIT Hold Time | 1.0 ns | | 108 |
| t_{74}^1 | INTR Setup Time | 1.7 ns | | 108 |
| t_{75}^1 | INTR Hold Time | 1.0 ns | | 108 |
| t_{76} | INV Setup Time | 1.7 ns | | 108 |
| t_{77} | INV Hold Time | 1.0 ns | | 108 |
| t_{78} | KEN# Setup Time | 3.0 ns | | 108 |
| t_{79} | KEN# Hold Time | 1.0 ns | | 108 |
| t_{80} | NA# Setup Time | 1.7 ns | | 108 |
| t_{81} | NA# Hold Time | 1.0 ns | | 108 |
| t_{82}^2 | NMI Setup Time | 1.7 ns | | 108 |
| t_{83}^2 | NMI Hold Time | 1.0 ns | | 108 |
| t_{84}^2 | SMI# Setup Time | 1.7 ns | | 108 |
| t_{85}^2 | SMI# Hold Time | 1.0 ns | | 108 |
| t_{86}^1 | STPCLK# Setup Time | 1.7 ns | | 108 |
| t_{87}^1 | STPCLK# Hold Time | 1.0 ns | | 108 |
| t_{88} | WB/WT# Setup Time | 1.7 ns | | 108 |
| t_{89} | WB/WT# Hold Time | 1.0 ns | | 108 |

Notes:

1. These level-sensitive signals can be asserted synchronously or asynchronously. To be sampled on a specific clock edge, setup and hold times must be met. If asserted asynchronously, they must be asserted for a minimum pulse width of two clocks.
2. These edge-sensitive signals can be asserted synchronously or asynchronously. To be sampled on a specific clock edge, setup and hold times must be met. If asserted asynchronously, they must have been negated at least two clocks prior to assertion and must remain asserted at least two clocks.

16.7 Output Delay Timings for 66-MHz Bus Operation

Table 66. Output Delay Timings for 66-MHz Bus Operation

| Symbol | Parameter Description | Preliminary Data | | Figure |
|-----------------|--------------------------------|------------------|---------|--------|
| | | Min | Max | |
| t ₆ | A[31:3] Valid Delay | 1.1 ns | 6.3 ns | 106 |
| t ₇ | A[31:3] Float Delay | | 10.0 ns | 107 |
| t ₈ | ADS# Valid Delay | 1.0 ns | 6.0 ns | 106 |
| t ₉ | ADS# Float Delay | | 10.0 ns | 107 |
| t ₁₀ | ADSC# Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t ₁₁ | ADSC# Float Delay | | 10.0 ns | 107 |
| t ₁₂ | AP Valid Delay | 1.0 ns | 8.5 ns | 106 |
| t ₁₃ | AP Float Delay | | 10.0 ns | 107 |
| t ₁₄ | APCHK# Valid Delay | 1.0 ns | 8.3 ns | 106 |
| t ₁₅ | BE[7:0]# Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t ₁₆ | BE[7:0]# Float Delay | | 10.0 ns | 107 |
| t ₁₇ | BREQ Valid Delay | 1.0 ns | 8.0 ns | 106 |
| t ₁₈ | CACHE# Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t ₁₉ | CACHE# Float Delay | | 10.0 ns | 107 |
| t ₂₀ | D/C# Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t ₂₁ | D/C# Float Delay | | 10.0 ns | 107 |
| t ₂₂ | D[63:0] Write Data Valid Delay | 1.3 ns | 7.5 ns | 106 |
| t ₂₃ | D[63:0] Write Data Float Delay | | 10.0 ns | 107 |
| t ₂₄ | DP[7:0] Write Data Valid Delay | 1.3 ns | 7.5 ns | 106 |
| t ₂₅ | DP[7:0] Write Data Float Delay | | 10.0 ns | 107 |
| t ₂₆ | FERR# Valid Delay | 1.0 ns | 8.3 ns | 106 |
| t ₂₇ | HIT# Valid Delay | 1.0 ns | 6.8 ns | 106 |
| t ₂₈ | HITM# Valid Delay | 1.1 ns | 6.0 ns | 106 |
| t ₂₉ | HLDA Valid Delay | 1.0 ns | 6.8 ns | 106 |
| t ₃₀ | LOCK# Valid Delay | 1.1 ns | 7.0 ns | 106 |
| t ₃₁ | LOCK# Float Delay | | 10.0 ns | 107 |
| t ₃₂ | M/IO# Valid Delay | 1.0 ns | 5.9 ns | 106 |
| t ₃₃ | M/IO# Float Delay | | 10.0 ns | 107 |
| t ₃₄ | PCD Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t ₃₅ | PCD Float Delay | | 10.0 ns | 107 |

Table 66. Output Delay Timings for 66-MHz Bus Operation (continued)

| Symbol | Parameter Description | Preliminary Data | | Figure |
|----------|-----------------------|------------------|---------|--------|
| | | Min | Max | |
| t_{36} | PCHK# Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t_{37} | PWT Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t_{38} | PWT Float Delay | | 10.0 ns | 107 |
| t_{39} | SCYC Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t_{40} | SCYC Float Delay | | 10.0 ns | 107 |
| t_{41} | SMIACK# Valid Delay | 1.0 ns | 7.3 ns | 106 |
| t_{42} | W/R# Valid Delay | 1.0 ns | 7.0 ns | 106 |
| t_{43} | W/R# Float Delay | | 10.0 ns | 107 |

16.8 Input Setup and Hold Timings for 66-MHz Bus Operation

Table 67. Input Setup and Hold Timings for 66-MHz Bus Operation

| Symbol | Parameter Description | Preliminary Data | | Figure |
|------------|------------------------------|------------------|-----|--------|
| | | Min | Max | |
| t_{44} | A[31:5] Setup Time | 6.0 ns | | 108 |
| t_{45} | A[31:5] Hold Time | 1.0 ns | | 108 |
| t_{46}^1 | A20M# Setup Time | 5.0 ns | | 108 |
| t_{47}^1 | A20M# Hold Time | 1.0 ns | | 108 |
| t_{48} | AHOLD Setup Time | 5.5 ns | | 108 |
| t_{49} | AHOLD Hold Time | 1.0 ns | | 108 |
| t_{50} | AP Setup Time | 5.0 ns | | 108 |
| t_{51} | AP Hold Time | 1.0 ns | | 108 |
| t_{52} | BOFF# Setup Time | 5.5 ns | | 108 |
| t_{53} | BOFF# Hold Time | 1.0 ns | | 108 |
| t_{54} | BRDY# Setup Time | 5.0 ns | | 108 |
| t_{55} | BRDY# Hold Time | 1.0 ns | | 108 |
| t_{56} | BRDYC# Setup Time | 5.0 ns | | 108 |
| t_{57} | BRDYC# Hold Time | 1.0 ns | | 108 |
| t_{58} | D[63:0] Read Data Setup Time | 2.8 ns | | 108 |
| t_{59} | D[63:0] Read Data Hold Time | 1.5 ns | | 108 |
| t_{60} | DP[7:0] Read Data Setup Time | 2.8 ns | | 108 |
| t_{61} | DP[7:0] Read Data Hold Time | 1.5 ns | | 108 |
| t_{62} | EADS# Setup Time | 5.0 ns | | 108 |
| t_{63} | EADS# Hold Time | 1.0 ns | | 108 |
| t_{64} | EWBE# Setup Time | 5.0 ns | | 108 |
| t_{65} | EWBE# Hold Time | 1.0 ns | | 108 |
| t_{66}^2 | FLUSH# Setup Time | 5.0 ns | | 108 |
| t_{67}^2 | FLUSH# Hold Time | 1.0 ns | | 108 |
| t_{68} | HOLD Setup Time | 5.0 ns | | 108 |
| t_{69} | HOLD Hold Time | 1.5 ns | | 108 |
| t_{70}^1 | IGNNE# Setup Time | 5.0 ns | | 108 |
| t_{71}^1 | IGNNE# Hold Time | 1.0 ns | | 108 |
| t_{72}^2 | INIT Setup Time | 5.0 ns | | 108 |

Table 67. Input Setup and Hold Timings for 66-MHz Bus Operation (continued)

| Symbol | Parameter Description | Preliminary Data | | Figure |
|------------|-----------------------|------------------|-----|--------|
| | | Min | Max | |
| t_{73}^2 | INIT Hold Time | 1.0 ns | | 108 |
| t_{74}^1 | INTR Setup Time | 5.0 ns | | 108 |
| t_{75}^1 | INTR Hold Time | 1.0 ns | | 108 |
| t_{76} | INV Setup Time | 5.0 ns | | 108 |
| t_{77} | INV Hold Time | 1.0 ns | | 108 |
| t_{78} | KEN# Setup Time | 5.0 ns | | 108 |
| t_{79} | KEN# Hold Time | 1.0 ns | | 108 |
| t_{80} | NA# Setup Time | 4.5 ns | | 108 |
| t_{81} | NA# Hold Time | 1.0 ns | | 108 |
| t_{82}^2 | NMI Setup Time | 5.0 ns | | 108 |
| t_{83}^2 | NMI Hold Time | 1.0 ns | | 108 |
| t_{84}^2 | SMI# Setup Time | 5.0 ns | | 108 |
| t_{85}^2 | SMI# Hold Time | 1.0 ns | | 108 |
| t_{86}^1 | STPCLK# Setup Time | 5.0 ns | | 108 |
| t_{87}^1 | STPCLK# Hold Time | 1.0 ns | | 108 |
| t_{88} | WB/WT# Setup Time | 4.5 ns | | 108 |
| t_{89} | WB/WT# Hold Time | 1.0 ns | | 108 |

Notes:

1. These level-sensitive signals can be asserted synchronously or asynchronously. To be sampled on a specific clock edge, setup and hold times must be met. If asserted asynchronously, they must be asserted for a minimum pulse width of two clocks.
2. These edge-sensitive signals can be asserted synchronously or asynchronously. To be sampled on a specific clock edge, setup and hold times must be met. If asserted asynchronously, they must have been negated at least two clocks prior to assertion and must remain asserted at least two clocks.

16.9 RESET and Test Signal Timing

Table 68. RESET and Configuration Signals for 100-MHz Bus Operation

| Symbol | Parameter Description | Preliminary Data | | Figure |
|-------------|--|------------------|-----|--------|
| | | Min | Max | |
| t_{90} | RESET Setup Time | 1.7 ns | | 109 |
| t_{91} | RESET Hold Time | 1.0 ns | | 109 |
| t_{92} | RESET Pulse Width, V_{CC} and CLK Stable | 15 clocks | | 109 |
| t_{93} | RESET Active After V_{CC} and CLK Stable | 1.0 ms | | 109 |
| t_{94}^1 | BF[2:0] Setup Time | 1.0 ms | | 109 |
| t_{95}^1 | BF[2:0] Hold Time | 2 clocks | | 109 |
| t_{96} | Intentionally left blank | | | |
| t_{97} | Intentionally left blank | | | |
| t_{98} | Intentionally left blank | | | |
| t_{99}^2 | FLUSH# Setup Time | 1.7 ns | | 109 |
| t_{100}^2 | FLUSH# Hold Time | 1.0 ns | | 109 |
| t_{101}^3 | FLUSH# Setup Time | 2 clocks | | 109 |
| t_{102}^3 | FLUSH# Hold Time | 2 clocks | | 109 |

Notes:

1. BF[2:0] must meet a minimum setup time of 1.0 ms and a minimum hold time of two clocks relative to the negation of RESET.
2. To be sampled on a specific clock edge, setup and hold times must be met the clock edge before the clock edge on which RESET is sampled negated.
3. If asserted asynchronously, these signals must meet a minimum setup and hold time of two clocks relative to the negation of RESET.

Table 69. RESET and Configuration Signals for 66-MHz Bus Operation

| Symbol | Parameter Description | Preliminary Data | | Figure |
|-------------|--|------------------|-----|--------|
| | | Min | Max | |
| t_{90} | RESET Setup Time | 5.0 ns | | 109 |
| t_{91} | RESET Hold Time | 1.0 ns | | 109 |
| t_{92} | RESET Pulse Width, V_{CC} and CLK Stable | 15 clocks | | 109 |
| t_{93} | RESET Active After V_{CC} and CLK Stable | 1.0 ms | | 109 |
| t_{94}^1 | BF[2:0] Setup Time | 1.0 ms | | 109 |
| t_{95}^1 | BF[2:0] Hold Time | 2 clocks | | 109 |
| t_{96} | Intentionally left blank | | | |
| t_{97} | Intentionally left blank | | | |
| t_{98} | Intentionally left blank | | | |
| t_{99}^2 | FLUSH# Setup Time | 5.0 ns | | 109 |
| t_{100}^2 | FLUSH# Hold Time | 1.0 ns | | 109 |
| t_{101}^3 | FLUSH# Setup Time | 2 clocks | | 109 |
| t_{102}^3 | FLUSH# Hold Time | 2 clocks | | 109 |

Notes:

1. BF[2:0] must meet a minimum setup time of 1.0 ms and a minimum hold time of two clocks relative to the negation of RESET.
2. To be sampled on a specific clock edge, setup and hold times must be met the clock edge before the clock edge on which RESET is sampled negated.
3. If asserted asynchronously, these signals must meet a minimum setup and hold time of two clocks relative to the negation of RESET.

Table 70. TCK Waveform and TRST# Timing at 25 MHz

| Symbol | Parameter Description | Preliminary Data | | Figure |
|-----------------|-----------------------|------------------|--------|--------|
| | | Min | Max | |
| | TCK Frequency | | 25 MHz | 110 |
| t_{103} | TCK Period | 40.0 ns | | 110 |
| t_{104} | TCK High Time | 14.0 ns | | 110 |
| t_{105} | TCK Low Time | 14.0 ns | | 110 |
| $t_{106}^{1,2}$ | TCK Fall Time | | 5.0 ns | 110 |
| t_{107} | TCK Rise Time | | 5.0 ns | 110 |
| t_{108}^3 | TRST# Pulse Width | 30.0 ns | | 111 |

Notes:

1. Rise/Fall times can be increased by 1.0 ns for each 10 MHz that TCK is run below its maximum frequency of 25 MHz.
2. Rise/Fall times are measured between 0.8 V and 2.0 V.
3. Asynchronous.

Table 71. Test Signal Timing at 25 MHz

| Symbol | Parameter Description | Preliminary Data | | Figure |
|-------------|------------------------------------|------------------|---------|--------|
| | | Min | Max | |
| t_{109}^1 | TDI Setup Time | 5.0 ns | | 112 |
| t_{110}^1 | TDI Hold Time | 9.0 ns | | 112 |
| t_{111}^1 | TMS Setup Time | 5.0 ns | | 112 |
| t_{112}^1 | TMS Hold Time | 9.0 ns | | 112 |
| t_{113}^2 | TDO Valid Delay | 3.0 ns | 13.0 ns | 112 |
| t_{114}^2 | TDO Float Delay | | 16.0 ns | 112 |
| t_{115}^2 | All Outputs (Non-Test) Valid Delay | 3.0 ns | 13.0 ns | 112 |
| t_{116}^2 | All Outputs (Non-Test) Float Delay | | 16.0 ns | 112 |
| t_{117}^1 | All Inputs (Non-Test) Setup Time | 5.0 ns | | 112 |
| t_{118}^1 | All Inputs (Non-Test) Hold Time | 9.0 ns | | 112 |

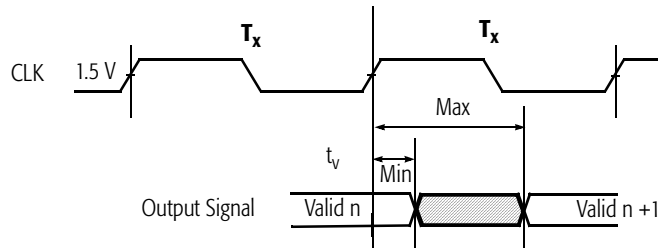
Notes:

1. Parameter is measured from the TCK rising edge.
2. Parameter is measured from the TCK falling edge.

16.10 Timing Diagrams

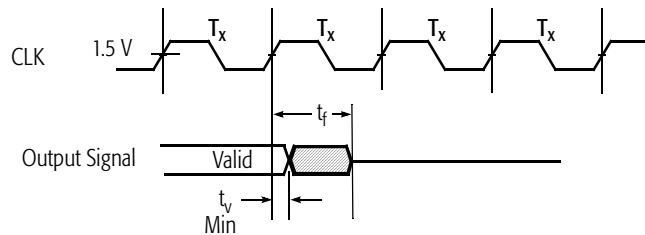
| WAVEFORM | INPUTS | OUTPUTS |
|----------|----------------------------------|-------------------------------------|
| | Must be steady | Steady |
| | Can change from High to Low | Changing from High to Low |
| | Can change from Low to High | Changing from Low to High |
| | Don't care, any change permitted | Changing, State Unknown |
| | (Does not apply) | Center line is high impedance state |

Figure 105. Key to Timing Diagrams



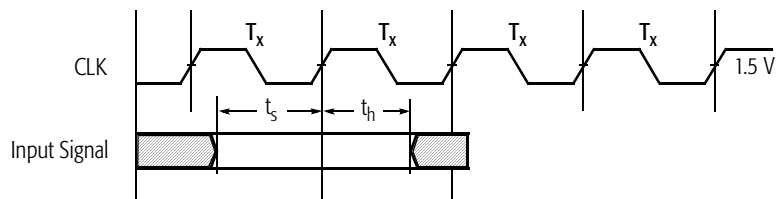
Note: For symbols t_v , listed in Table 64 on page 300 and Table 66 on page 304, where:
 $v = 6, 8, 10, 12, 14, 15, 17, 18, 20, 22, 24, 26, 27, 28, 29, 30, 32, 34, 36, 37, 39, 41, 42$

Figure 106. Output Valid Delay Timing



Note: For symbols t_v and t_f listed in Table 64 on page 300 and Table 66 on page 304, where:
 $v = 6, 8, 10, 12, 15, 18, 20, 22, 24, 30, 32, 34, 37, 39, 42$
 $f = 7, 9, 11, 13, 16, 19, 21, 23, 25, 31, 33, 35, 38, 40, 43$

Figure 107. Maximum Float Delay Timing



Note: For symbols t_s and t_h listed in Table 65 on page 302 and Table 67 on page 306, where:
 $s = 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88$
 $h = 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89$

Figure 108. Input Setup and Hold Timing

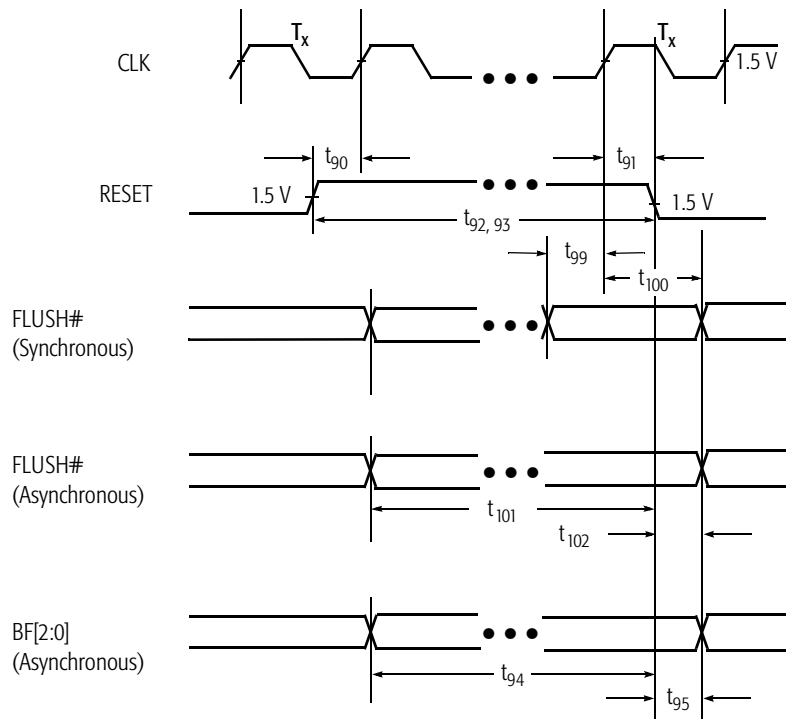


Figure 109. Reset and Configuration Timing

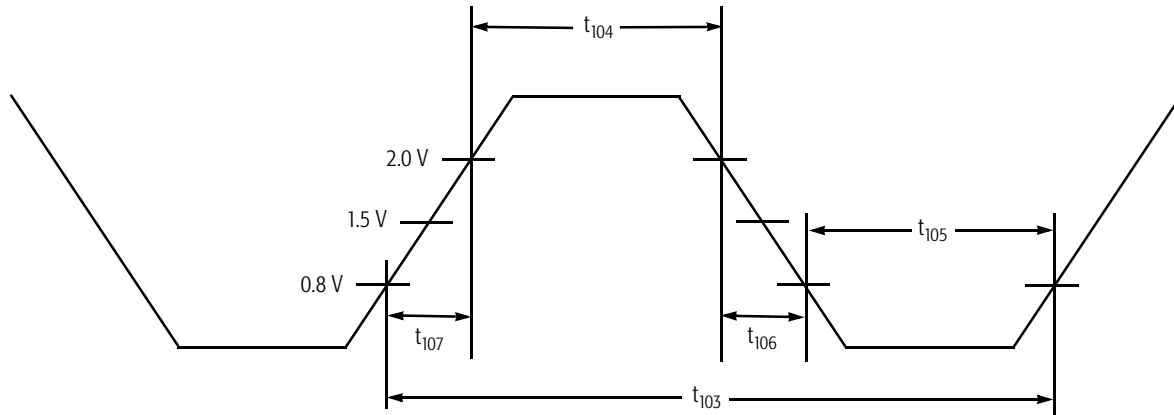


Figure 110. TCK Waveform

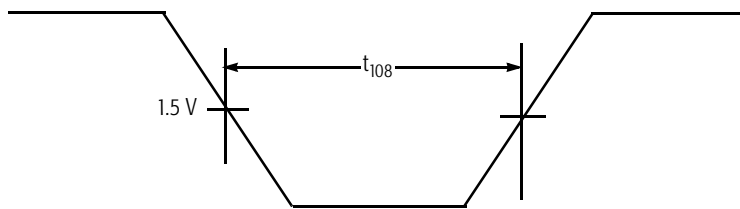


Figure 111. TRST# Timing

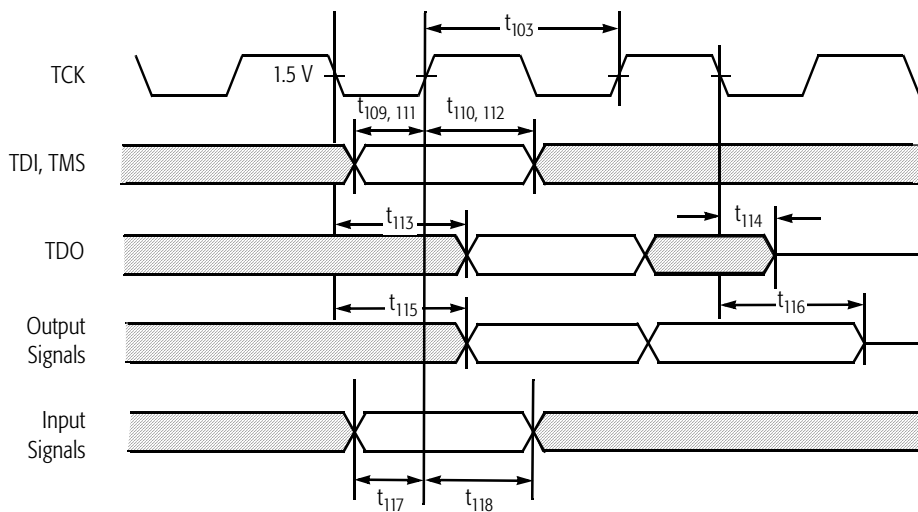


Figure 112. Test Signal Timing Diagram

17 Thermal Design

17.1 Package Thermal Specifications

The AMD-K6-III+ processor operating specification calls for the case temperature (T_C) to be in the range of 0°C to 70°C for standard-power devices and 0°C to 85°C for low-power devices. The ambient temperature (T_A) is not specified as long as the case temperature is not violated. The case temperature must be measured on the top center of the package.

Table 72 and Table 73 on page 316 show the processor thermal specifications for the low-power and standard power AMD-K6-III+ devices, respectively.

Table 72. Package Thermal Specification for Low-Power AMD-K6™-III+ Devices

| θ_{JC} Junction-Case | Maximum Thermal Power | | |
|--------------------------------|-----------------------|---------|---------|
| | 400 MHz | 450 MHz | 500 MHz |
| 1.0° C/W | 9.50 W | 12.00 W | 14.50 W |
| Stop Grant Mode | 2.50 W | | |
| Stop Clock Mode | 1.60 W | 1.90 W | |
| T_C Case Temperature | 0°C–85 | | |

Table 73. Package Thermal Specification for Standard-Power AMD-K6™-III+ Devices

| θ_{JC} Junction-Case | Maximum Thermal Power | | | |
|--------------------------------|-----------------------|---------|---------|---------|
| | 400 MHz | 450 MHz | 500 MHz | 550 MHz |
| 1.0°C/W | 16.50 W | 17.50 W | 18.50 W | 19.50 W |
| Stop Grant Mode | 4.50 W | | | |
| Stop Clock Mode | 4.00 W | | | |
| T_C Case Temperature | 0°C–70°C | | | |

Figure 113 on page 317 shows the thermal model of a processor with a passive thermal solution. The case-to-ambient temperature (T_{CA}) can be calculated from the following equation:

$$\begin{aligned} T_{CA} &= P_{MAX} \cdot \theta_{CA} \\ &= P_{MAX} \cdot (\theta_{IF} + \theta_{SA}) \end{aligned}$$

Where:

$$\begin{aligned} P_{MAX} &= \text{Maximum Power Consumption} \\ \theta_{CA} &= \text{Case-to-Ambient Thermal Resistance} \\ \theta_{IF} &= \text{Interface Material Thermal Resistance} \\ \theta_{SA} &= \text{Sink-to-Ambient Thermal Resistance} \end{aligned}$$

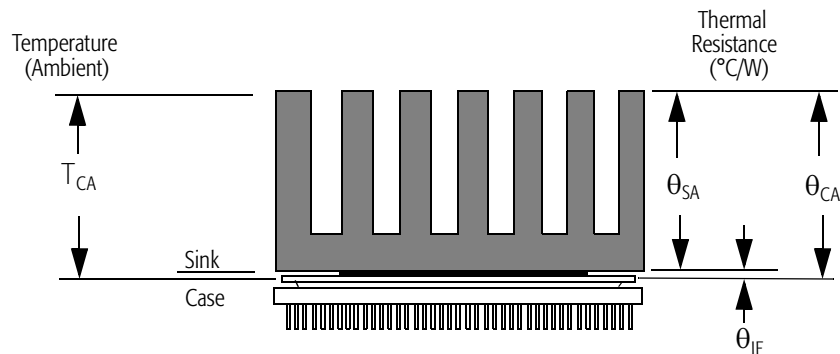


Figure 113. Thermal Model (CPGA Package)

Figure 114 illustrates the case-to-ambient temperature (T_{CA}) in relation to the power consumption (X-axis) and the thermal resistance (Y-axis). If the power consumption and case temperature are known, the thermal resistance (θ_{CA}) requirement can be calculated for a given ambient temperature (T_A) value.

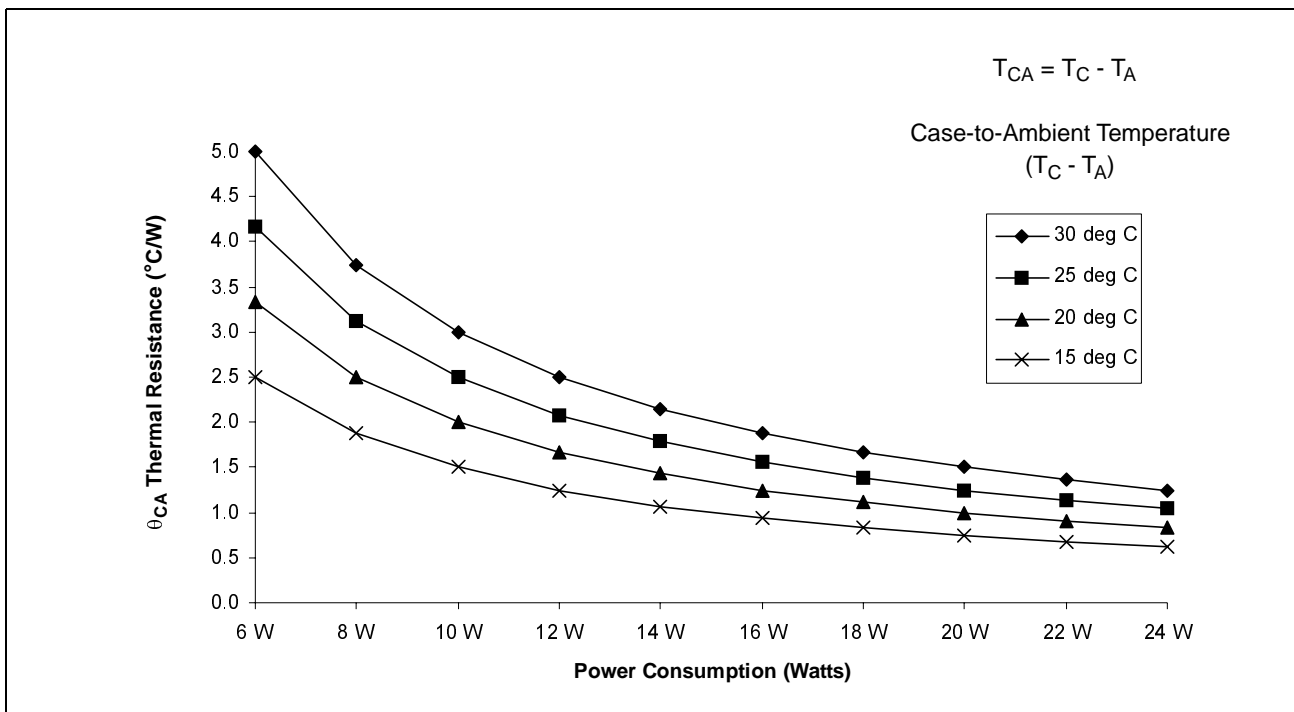


Figure 114. Power Consumption and Thermal Resistance (CPGA Package)

The thermal resistance of a heatsink is determined by the heat dissipation surface area, the material and shape of the heatsink, and the airflow volume across the heatsink. In general, the larger the surface area the lower the thermal resistance.

The required thermal resistance of a heatsink (θ_{SA}) can be calculated using the following example:

If:

$$\begin{aligned} T_C &= 70^\circ\text{C} \\ T_A &= 45^\circ\text{C} \\ P_{MAX} &= 19.50\text{W} \end{aligned}$$

Then:

$$\theta_{CA} \leq \left(\frac{T_C - T_A}{P_{MAX}} \right) = \frac{25^\circ\text{C}}{19.50\text{W}} = 1.28^\circ\text{C/W}$$

Thermal grease is recommended as interface material because it provides the lowest thermal resistance ($\cong 0.20^\circ\text{C/W}$). The required thermal resistance (θ_{SA}) of the heatsink in this example is calculated as follows:

$$\theta_{SA} = \theta_{CA} - \theta_{IF} = 1.28 - 0.20 = 1.08^\circ\text{C/W}$$

Heat Dissipation Path

Figure 115 illustrates the heat dissipation path of the processor. Due to the lower thermal resistance between the processor die junction and case, most of the heat generated by the processor is transferred from the top surface of the case. The small amount of heat generated from the bottom side of the processor where the processor socket blocks the convection can be safely ignored.

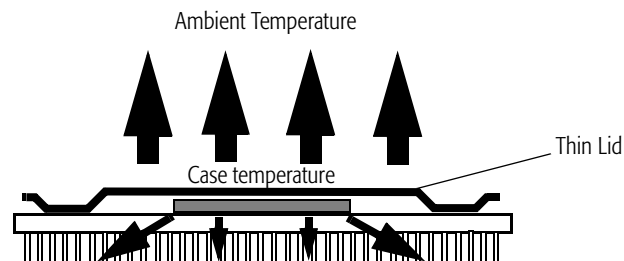


Figure 115. Processor Heat Dissipation Path

17.2 Measuring Case Temperature

The processor case temperature is measured to ensure that the thermal solution meets the processor's operational specification. This temperature should be measured on the top center of the package, where most of the heat is dissipated. Figure 116 shows the correct location for measuring the case temperature. The tip of the thermocouple should be secured to the package surface with a small amount of thermally conductive epoxy. It is also recommended to secure a second location along the thermocouple to avoid any movement during testing.

If a heatsink is installed while measuring, the thermocouple must be installed into the heatsink via a small hole drilled through the heatsink base (for example, 1/16 of an inch). Secure the thermocouple to the base of the heatsink by filling the small hole with thermal epoxy, allowing the tip of the thermocouple to protrude the epoxy and touch the top of the processor case.

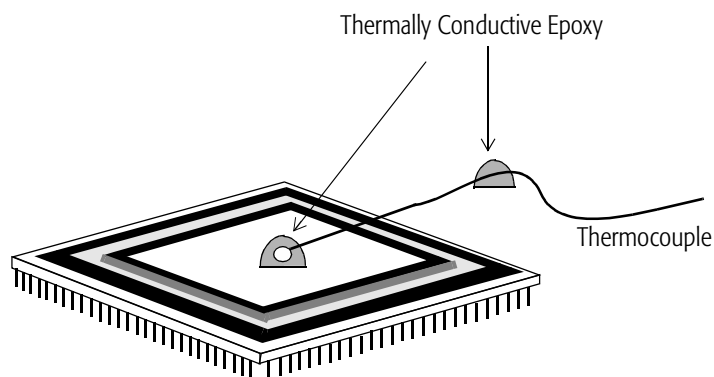


Figure 116. Measuring Case Temperature

17.3 Layout and Airflow Considerations

Voltage Regulator

A voltage regulator is required to support the lower voltage (3.3 V and lower) to the processor. In most applications, the voltage regulator is designed with power transistors. As a result, additional heatsinks are required to dissipate the heat from the power transistors. Figure 117 on page 320 shows the

voltage regulator placed parallel to the processor with the airflow aligned with the devices. With this alignment, the heat generated by the voltage regulator has minimal effect on the processor.

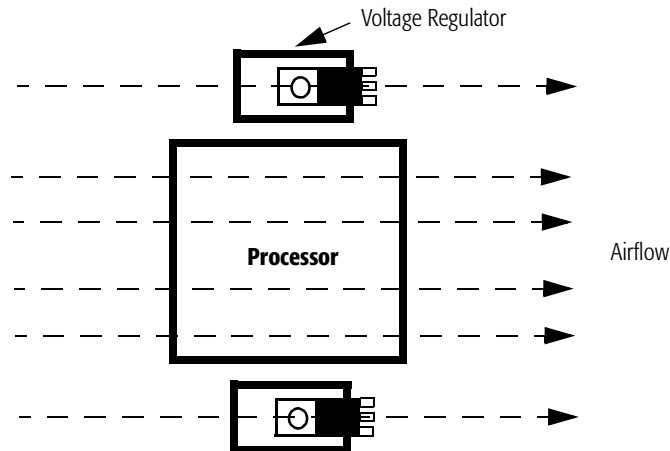


Figure 117. Voltage Regulator Placement

A heatsink and fan combination can deliver much better thermal performance than a heatsink alone. More importantly, with a fan/sink the airflow requirements in a system design are not as critical. A unidirectional heatsink with a fan moves air from the top of the heatsink to the side. In this case, the best location for the voltage regulator is on the side of the processor in the path of the airflow exiting the fan sink (see Figure 118 on page 321). This location guarantees that the heatsinks on both the processor and the regulator receive adequate air circulation.

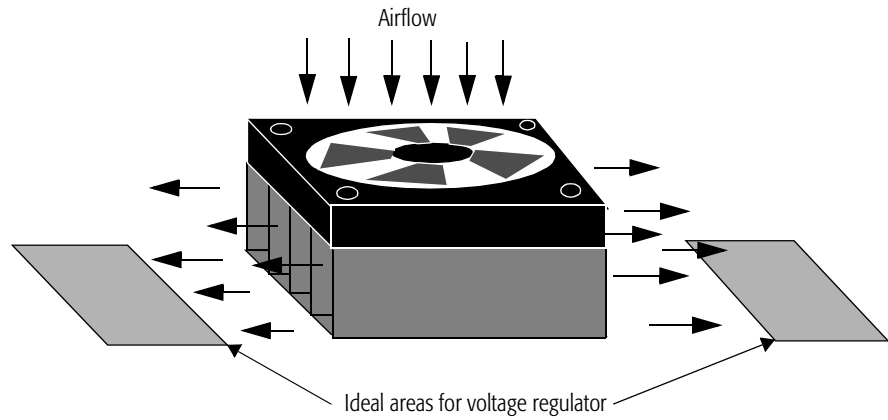


Figure 118. Airflow for a Heatsink with Fan

Airflow Management in a System Design

Complete airflow management in a system is important. In addition to the volume of air, the path of the air is also important. Figure 119 shows the airflow in a dual-fan system. The fan in the front end pulls cool air into the system through intake slots in the chassis. The power supply fan forces the hot air out of the chassis. The thermal performance of the heatsink can be maximized if it is located in the shaded area, where it receives greatest benefit from this air exchange system.

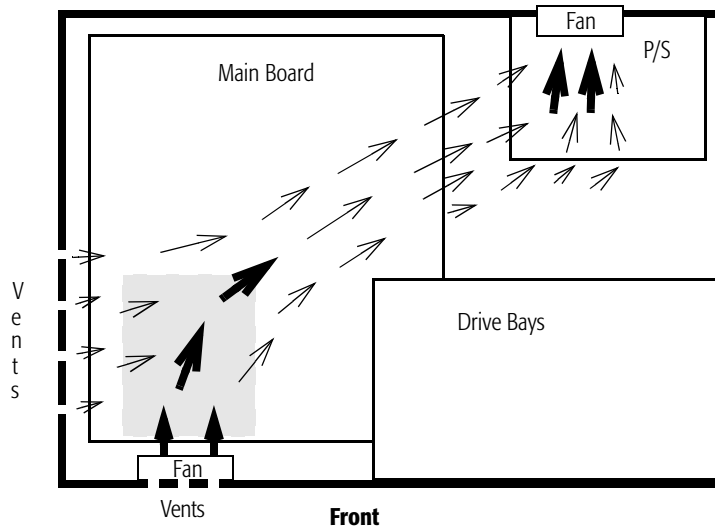


Figure 119. Airflow Path in a Dual-Fan System

Figure 120 shows the airflow management in a system using the ATX form-factor. The orientation of the power supply fan and the motherboard are modified in the ATX platform design. The power supply fan pulls cool air through the chassis and across the processor. The processor is located near the power supply fan, where it can receive adequate airflow without an auxiliary fan. The arrangement significantly improves the airflow across the processor with minimum installation cost.

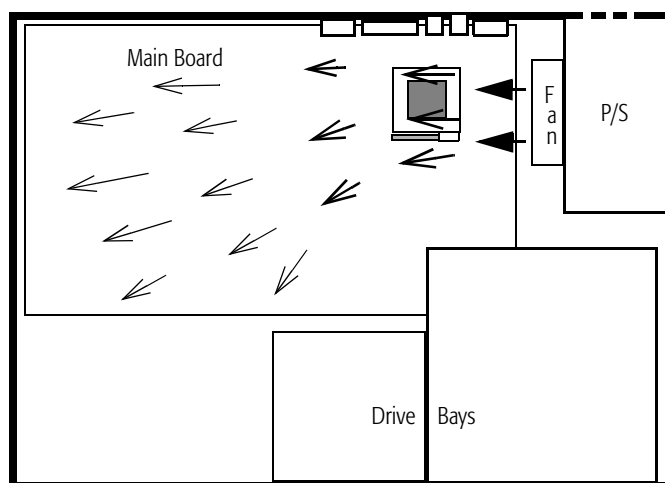


Figure 120. Airflow Path in an ATX Form-Factor System

For more information about thermal design considerations, see the *AMD-K6® Processor Thermal Solution Design Application Note*, order# 21085.

18 Pin Designations

This chapter includes pin connection diagrams and pin designation tables for each of two packages, the Ceramic Pin Grid Array (CPGA) and the Organic Ball Grid Array (OBGA).

The pin designation diagrams include the following annotations:

- Control/Parity Pins
- ◊ V_{SS} Pins
- ▲ V_{CC2} Pins
- △ V_{CC3} Pins
- Data Pins
- Address Pins
- † Test Pins
- NC, INC (Internal No Connect) Pins
- ⊙ RSVD (Reserved) Pins

Note that the OBGA package includes additional pins not supported on the CPGA package. Table 74 shows the pin differences between the two packages.

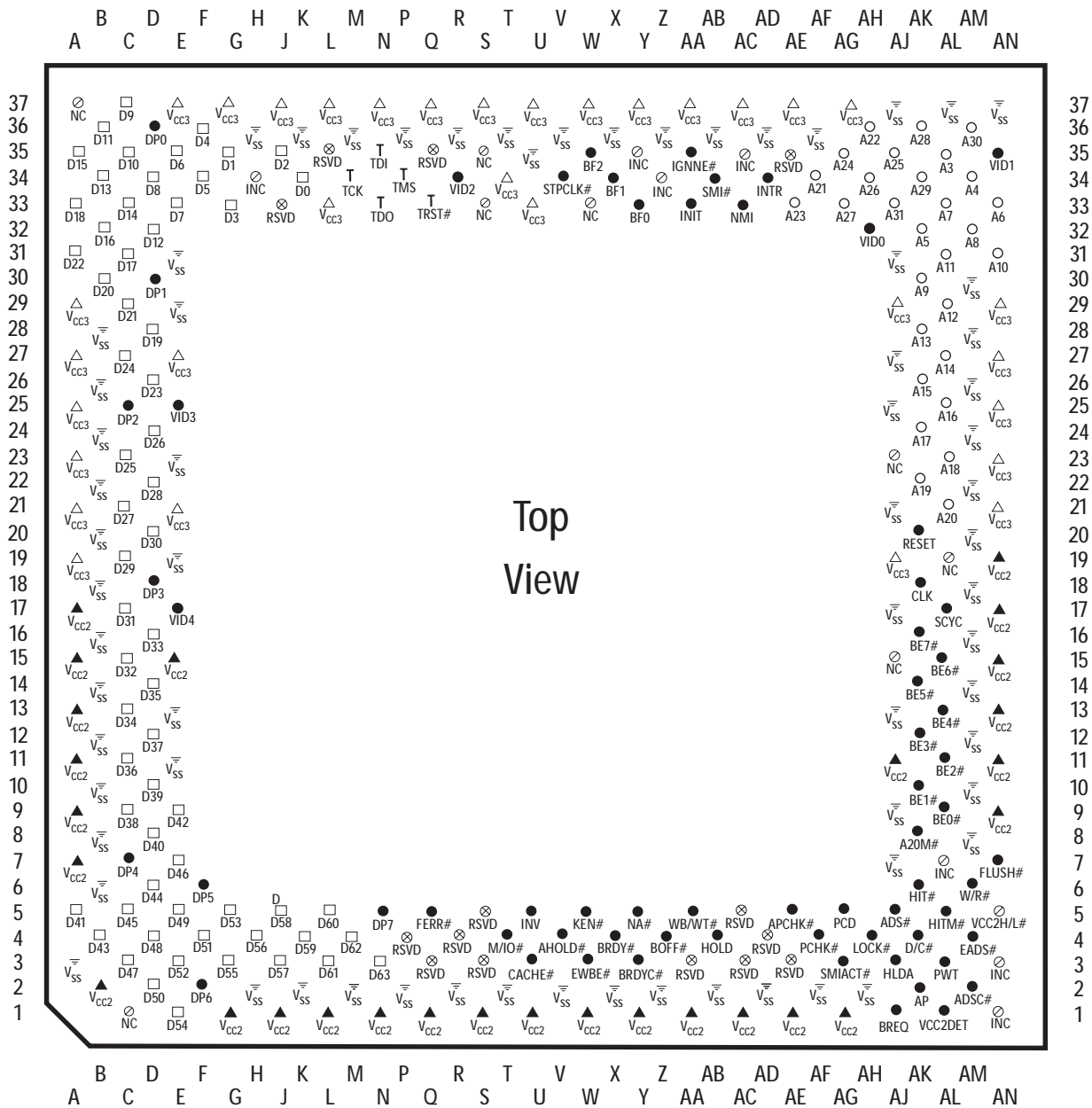
Table 74. Pin Differences Between the CPGA and OBGA Packages

| Pin | CPGA Package | OBGA Package | Comment |
|----------------------|--------------------------------------|--------------------------------------|---|
| VCC2DET | Supported | Not supported | |
| VCC2H/L# | Supported | Not supported | |
| VID[4:0] | Supported on low-power versions only | Supported on low-power versions only | These pins are no-connects (NC) on standard-power versions for both packages. |
| V_{CC2} | 28 | 37 | |
| V_{CC3} | 32 | 26 | |
| V_{SS} | 68 | 99 | |
| No Connects | 13 ¹ 8 ² | 13 ¹ 8 ² | |
| Internal No Connects | 7 | 1 | |
| Reserved | 14 | 16 | |

Notes:

1. Standard-power versions only, since the VID[4:0] outputs are not supported.
2. Low-power versions only.

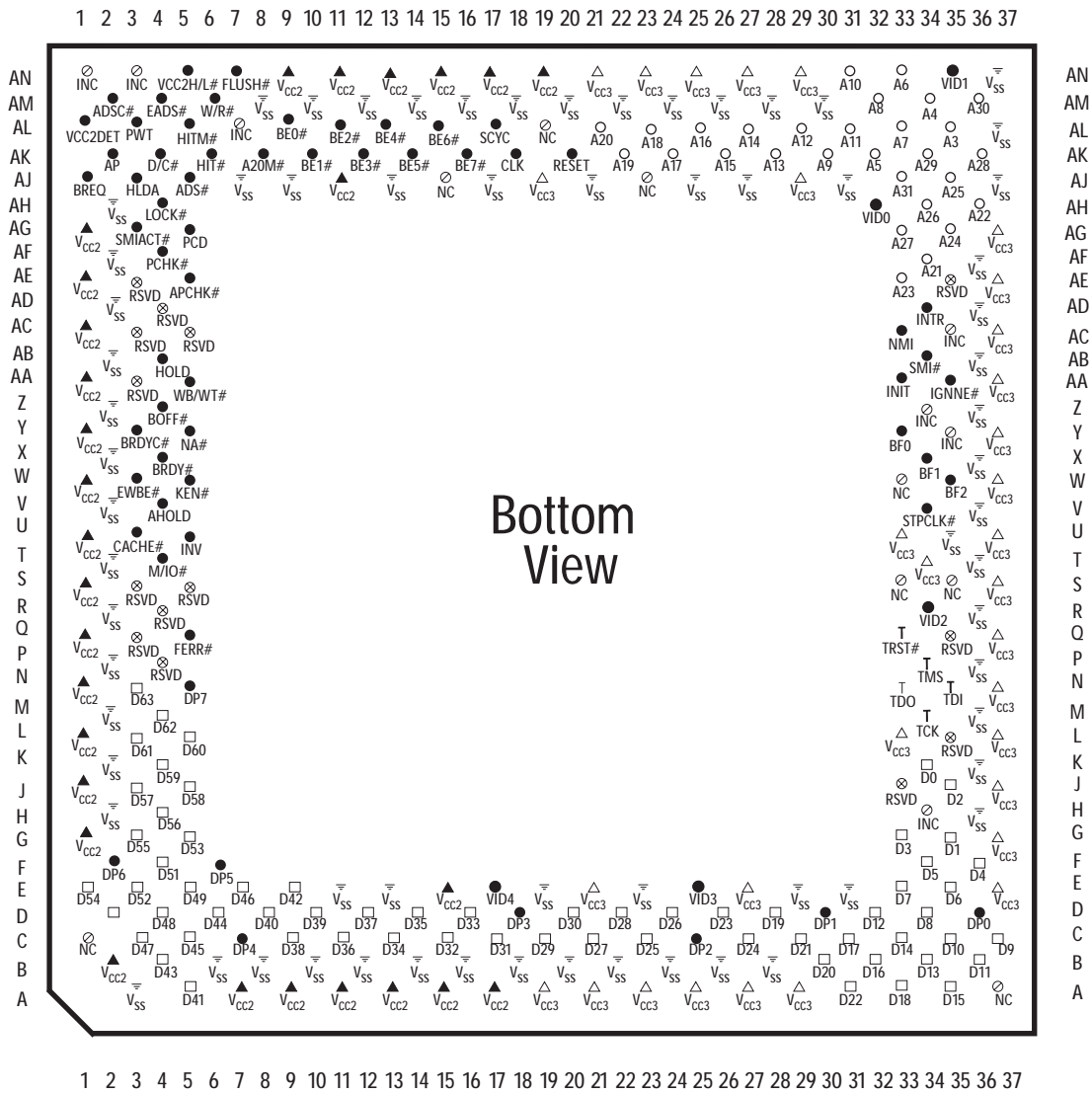
18.1 Pins Designations for CPGA Package



Notes:

The VID[4:0] outputs are supported on low-power versions only. These pins are defined as no-connects on standard-power versions.

Figure 121. CPGA Connection Diagram (Top-Side View)



Notes:

The VID[4:0] outputs are supported on low-power versions only. These pins are defined as no-connects on standard-power versions.

Figure 122. CPGA Connection Diagram (Bottom-Side View)

Table 75. CPGA Pin Designations by Functional Grouping

| Pin Name | Pin Number | Pin Name | Pin Number | Pin Name | Pin Number | Pin Name | Pin Number |
|----------------|------------|----------------|------------|-------------|------------|-------------------------------|------------|
| Control | | Address | | Data | | Data | |
| A20M# | AK-08 | A3 | AL-35 | D0 | K-34 | D52 | E-03 |
| ADS# | AJ-05 | A4 | AM-34 | D1 | G-35 | D53 | G-05 |
| ADSC# | AM-02 | A5 | AK-32 | D2 | J-35 | D54 | E-01 |
| AHOLD | V-04 | A6 | AN-33 | D3 | G-33 | D55 | G-03 |
| APCHK# | AE-05 | A7 | AL-33 | D4 | F-36 | D56 | H-04 |
| BE0# | AL-09 | A8 | AM-32 | D5 | F-34 | D57 | J-03 |
| BE1# | AK-10 | A9 | AK-30 | D6 | E-35 | D58 | J-05 |
| BE2# | AL-11 | A10 | AN-31 | D7 | E-33 | D59 | K-04 |
| BE3# | AK-12 | A11 | AL-31 | D8 | D-34 | D60 | L-05 |
| BE4# | AL-13 | A12 | AL-29 | D9 | C-37 | D61 | L-03 |
| BE5# | AK-14 | A13 | AK-28 | D10 | C-35 | D62 | M-04 |
| BE6# | AL-15 | A14 | AL-27 | D11 | B-36 | D63 | N-03 |
| BE7# | AK-16 | A15 | AK-26 | D12 | D-32 | Test | |
| BF0 | Y-33 | A16 | AL-25 | D13 | B-34 | TCK | M-34 |
| BF1 | X-34 | A17 | AK-24 | D14 | C-33 | TDI | N-35 |
| BF2 | W-35 | A18 | AL-23 | D15 | A-35 | TDO | N-33 |
| BOFF# | Z-04 | A19 | AK-22 | D16 | B-32 | TMS | P-34 |
| BRDY# | X-04 | A20 | AL-21 | D17 | C-31 | TRST# | Q-33 |
| BRDYC# | Y-03 | A21 | AF-34 | D18 | A-33 | Parity | |
| BREQ | AJ-01 | A22 | AH-36 | D19 | D-28 | AP | AK-02 |
| CACHE# | U-03 | A23 | AE-33 | D20 | B-30 | DP0 | D-36 |
| CLK | AK-18 | A24 | AG-35 | D21 | C-29 | DP1 | D-30 |
| D/C# | AK-04 | A25 | AJ-35 | D22 | A-31 | DP2 | C-25 |
| EADS# | AM-04 | A26 | AH-34 | D23 | D-26 | DP3 | D-18 |
| EWBE# | W-03 | A27 | AG-33 | D24 | C-27 | DP4 | C-07 |
| FERR# | Q-05 | A28 | AK-36 | D25 | C-23 | DP5 | F-06 |
| FLUSH# | AN-07 | A29 | AK-34 | D26 | D-24 | DP6 | F-02 |
| HIT# | AK-06 | A30 | AM-36 | D27 | C-21 | DP7 | N-05 |
| HITM# | AL-05 | A31 | AJ-33 | D28 | D-22 | Voltage ID¹ | |
| HLDA | AJ-03 | | | D29 | C-19 | VID4 | E-17 |
| HOLD | AB-04 | | | D30 | D-20 | VID3 | E-25 |
| IGNNE# | AA-35 | | | D31 | C-17 | VID2 | R-34 |
| INIT | AA-33 | | | D32 | C-15 | VID1 | AN-35 |
| INTR | AD-34 | | | D33 | D-16 | VID0 | AH-32 |
| INV | U-05 | | | D34 | C-13 | | |
| KEN# | W-05 | | | D35 | D-14 | | |
| LOCK# | AH-04 | | | D36 | C-11 | | |
| M/IO# | T-04 | | | D37 | D-12 | | |
| NA# | Y-05 | | | D38 | C-09 | | |
| NMI | AC-33 | | | D39 | D-10 | | |
| PCD | AG-05 | | | D40 | D-08 | | |
| PCHK# | AF-04 | | | D41 | A-05 | | |
| PWT | AL-03 | | | D42 | E-09 | | |
| RESET | AK-20 | | | D43 | B-04 | | |
| SCYC | AL-17 | | | D44 | D-06 | | |
| SMI# | AB-34 | | | D45 | C-05 | | |
| SMIACT# | AG-03 | | | D46 | E-07 | | |
| STPCLK# | V-34 | | | D47 | C-03 | | |
| VCC2DET | AL-01 | | | D48 | D-04 | | |
| VCC2H/L# | AN-05 | | | D49 | E-05 | | |
| W/R# | AM-06 | | | D50 | D-02 | | |
| WB/WT# | AA-05 | | | D51 | F-04 | | |

Notes: 1. The VID[4:0] pins are supported on low-power versions only. These pins are defined as no-connects on standard-power versions.

Table 76. CPGA Pin Designations for No Connect, Reserved, Power, and Ground Pins

| Pin Numbers | | | | |
|----------------------------------|------------------|------------------|-----------------|-----------------|
| No Connect (NC) | V _{CC2} | V _{CC3} | V _{SS} | V _{SS} |
| A-37 | A-07 | A-19 | A-03 | AJ-27 |
| C-01 | A-09 | A-21 | B-06 | AJ-31 |
| E-17 ¹ | A-11 | A-23 | B-08 | AJ-37 |
| E-25 ¹ | A-13 | A-25 | B-10 | AL-37 |
| R-34 ¹ | A-15 | A-27 | B-12 | AM-08 |
| S-33 | A-17 | A-29 | B-14 | AM-10 |
| S-35 | B-02 | E-21 | B-16 | AM-12 |
| W-33 | E-15 | E-27 | B-18 | AM-14 |
| AH-32 ¹ | G-01 | E-37 | B-20 | AM-16 |
| AJ-15 | J-01 | G-37 | B-22 | AM-18 |
| AJ-23 | L-01 | J-37 | B-24 | AM-20 |
| AL-19 | N-01 | L-33 | B-26 | AM-22 |
| AN-35 ¹ | Q-01 | L-37 | B-28 | AM-24 |
| Internal No Connect (INC) | S-01 | N-37 | E-11 | AM-26 |
| H-34 | U-01 | Q-37 | E-13 | AM-28 |
| Y-35 | W-01 | S-37 | E-19 | AM-30 |
| Z-34 | Y-01 | T-34 | E-23 | AN-37 |
| AC-35 | AA-01 | U-33 | E-29 | |
| AL-07 | AC-01 | U-37 | E-31 | |
| AN-01 | AE-01 | W-37 | H-02 | |
| AN-03 | AG-01 | Y-37 | H-36 | |
| Reserved (RSVD) | AJ-11 | AA-37 | K-02 | |
| J-33 | AN-09 | AC-37 | K-36 | |
| L-35 | AN-11 | AE-37 | M-02 | |
| P-04 | AN-13 | AG-37 | M-36 | |
| Q-03 | AN-15 | AJ-19 | P-02 | |
| Q-35 | AN-17 | AJ-29 | P-36 | |
| R-04 | AN-19 | AN-21 | R-02 | |
| S-03 | | AN-23 | R-36 | |
| S-05 | | AN-25 | T-02 | |
| AA-03 | | AN-27 | T-36 | |
| AC-03 | | AN-29 | U-35 | |
| AC-05 | | | V-02 | |
| AD-04 | | | V-36 | |
| AE-03 | | | X-02 | |
| AE-35 | | | X-36 | |
| | | | Z-02 | |
| | | | Z-36 | |
| | | | AB-02 | |
| | | | AB-36 | |
| | | | AD-02 | |
| | | | AD-36 | |
| | | | AF-02 | |
| | | | AF-36 | |
| | | | AH-02 | |
| | | | AJ-07 | |
| | | | AJ-09 | |
| | | | AJ-13 | |
| | | | AJ-17 | |
| | | | AJ-21 | |
| | | | AJ-25 | |

Notes: 1. These pins are no-connects on standard-power versions only. They are defined as VID[4:0] on low-power versions. See page 326.

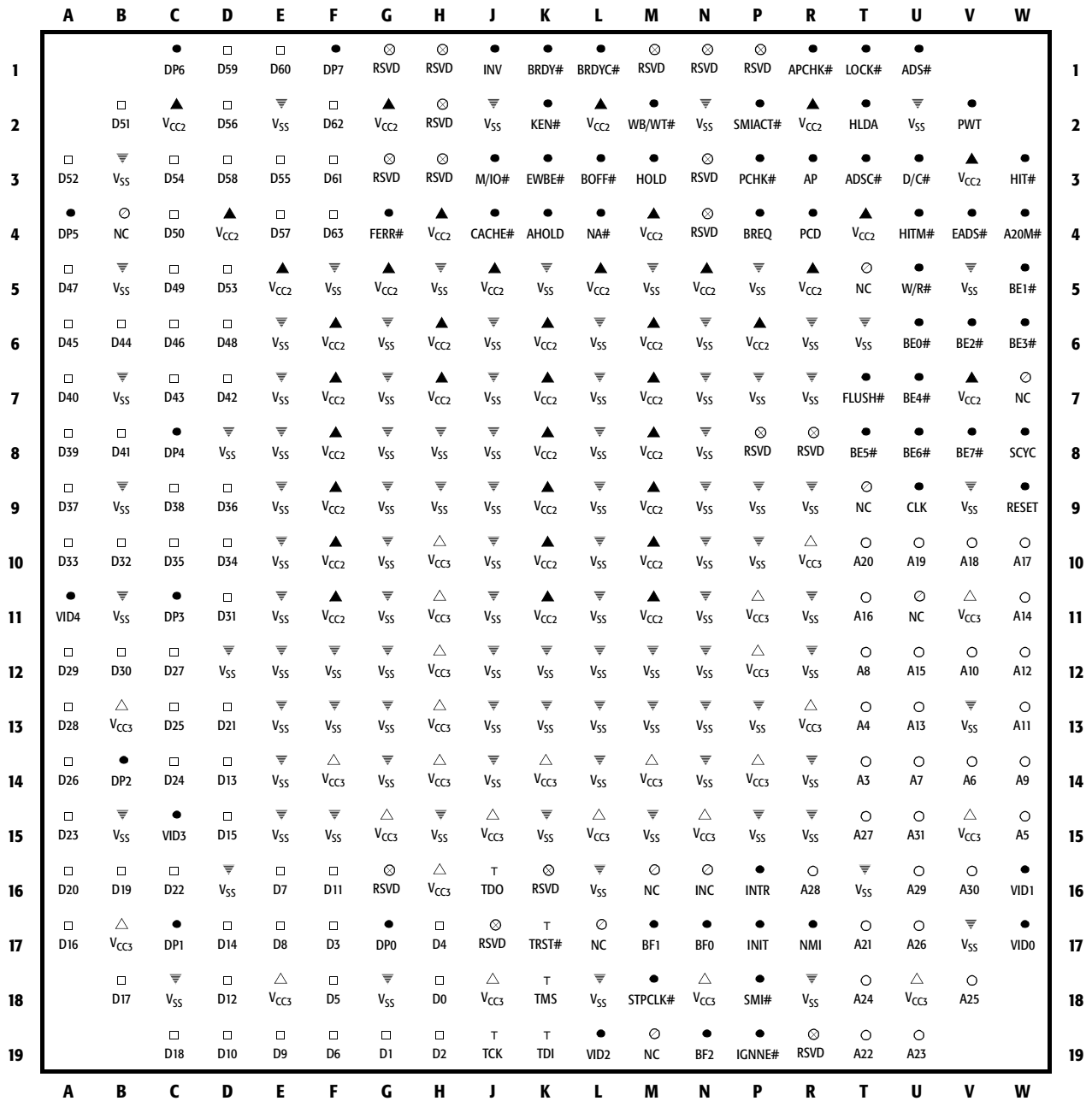
18.2 Pins Designations for OBGA Package

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|---|
| A | | | □ | ● | □ | □ | □ | □ | □ | □ | ● | □ | □ | □ | □ | □ | □ | | | A |
| B | | □ | ▽ | ○ | ▽ | □ | ▽ | □ | ▽ | □ | ▽ | □ | △ | ● | ▽ | □ | △ | □ | | B |
| C | ● | ▲ | □ | □ | □ | □ | □ | ● | □ | □ | ● | □ | □ | □ | ● | □ | ● | ▽ | □ | C |
| D | □ | □ | □ | ▲ | □ | □ | □ | ▽ | □ | □ | □ | ▽ | □ | □ | □ | ▽ | □ | □ | □ | D |
| E | □ | ▽ | □ | □ | ▲ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | □ | □ | △ | □ | E |
| F | ● | □ | □ | □ | ▽ | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ | ▽ | ▽ | △ | ▽ | □ | □ | □ | □ | F |
| G | ⊗ | ▲ | ⊗ | ● | ▲ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | △ | ⊗ | ● | ▽ | □ | G |
| H | ⊗ | ⊗ | ⊗ | ▲ | ▽ | ▲ | ▲ | ▽ | ▽ | △ | △ | △ | △ | △ | ▽ | △ | □ | □ | □ | H |
| J | ● | ▽ | ● | ● | ▲ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | △ | T | ⊗ | △ | T | J |
| K | ● | ● | ● | ● | ▽ | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ | ▽ | ▽ | △ | ▽ | ⊗ | T | T | T | K |
| L | ● | ▲ | ● | ● | ▲ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | △ | ▽ | ○ | ▽ | ● | L |
| M | ⊗ | ● | ● | ▲ | ▽ | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ | ▽ | ▽ | △ | ▽ | ○ | ● | ● | ○ | M |
| N | ⊗ | ▽ | ⊗ | ⊗ | ▲ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | △ | ○ | ● | △ | ● | N |
| P | ⊗ | ● | ● | ● | ▽ | ▲ | ▽ | ⊗ | ▽ | ▽ | △ | △ | ▽ | △ | ▽ | ● | ● | ● | ● | P |
| R | ● | ▲ | ● | ● | ▲ | ▽ | ▽ | ⊗ | ▽ | △ | ▽ | ▽ | △ | ▽ | ▽ | ○ | ● | ▽ | ⊗ | R |
| T | ● | ● | ● | ▲ | ○ | ▽ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ▽ | ○ | ○ | ○ | T |
| U | ● | ▽ | ● | ● | ● | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | △ | ○ | U |
| V | | ● | ▲ | ● | ▽ | ● | ▲ | ● | ▽ | ○ | △ | ○ | ▽ | ○ | △ | ○ | ▽ | ○ | | V |
| W | | | ● | ● | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | | | W |

Notes:

There are three pads missing on each corner of the OBGA package due to manufacturing requirements.
 The VID[4:0] outputs are supported on low-power versions only. These pins are defined as no-connects on standard-power versions.

Figure 123. OBGA Connection Diagram (Top-Side View)



Notes:

There are three pads missing on each corner of the OBGA package due to manufacturing requirements.
 The VID[4:0] outputs are supported on low-power versions only. These pins are defined as no-connects on standard-power versions.

Figure 124. OBGA Connection Diagram (Bottom-Side View)

Table 77. OBGA Pin Designations by Functional Grouping

| Pin Name | Pin Number | Pin Name | Pin Number | Pin Name | Pin Number | Pin Name | Pin Number |
|----------------|------------|----------------|------------|-------------|------------|-------------------------------|------------|
| Control | | Address | | Data | | Data | |
| A20M# | W4 | A3 | T14 | D0 | H18 | D52 | A3 |
| ADS# | U1 | A4 | T13 | D1 | G19 | D53 | D5 |
| ADSC# | T3 | A5 | W15 | D2 | H19 | D54 | C3 |
| AHOLD | K4 | A6 | V14 | D3 | F17 | D55 | E3 |
| APCHK# | R1 | A7 | U14 | D4 | H17 | D56 | D2 |
| BE0# | U6 | A8 | T12 | D5 | F18 | D57 | E4 |
| BE1# | W5 | A9 | W14 | D6 | F19 | D58 | D3 |
| BE2# | V6 | A10 | V12 | D7 | E16 | D59 | D1 |
| BE3# | W6 | A11 | W13 | D8 | E17 | D60 | E1 |
| BE4# | U7 | A12 | W12 | D9 | E19 | D61 | F3 |
| BE5# | T8 | A13 | U13 | D10 | D19 | D62 | F2 |
| BE6# | U8 | A14 | W11 | D11 | F16 | D63 | F4 |
| BE7# | V8 | A15 | U12 | D12 | D18 | Test | |
| BF0 | N17 | A16 | T11 | D13 | D14 | TCK | J19 |
| BF1 | M17 | A17 | W10 | D14 | D17 | TDI | K19 |
| BF2 | N19 | A18 | V10 | D15 | D15 | TDO | J16 |
| BOFF# | L3 | A19 | U10 | D16 | A17 | TMS | K18 |
| BRDY# | K1 | A20 | T10 | D17 | B18 | TRST# | K17 |
| BRDYC# | L1 | A21 | T17 | D18 | C19 | Parity | |
| BREQ | P4 | A22 | T19 | D19 | B16 | AP | R3 |
| CACHE# | J4 | A23 | U19 | D20 | A16 | DP0 | G17 |
| CLK | U9 | A24 | T18 | D21 | D13 | DP1 | C17 |
| D/C# | U3 | A25 | V18 | D22 | C16 | DP2 | B14 |
| EADS# | V4 | A26 | U17 | D23 | A15 | DP3 | C11 |
| EWBE# | K3 | A27 | T15 | D24 | C14 | DP4 | C8 |
| FERR# | G4 | A28 | R16 | D25 | C13 | DP5 | A4 |
| FLUSH# | T7 | A29 | U16 | D26 | A14 | DP6 | C1 |
| HIT# | W3 | A30 | V16 | D27 | C12 | DP7 | F1 |
| HITM# | U4 | A31 | U15 | D28 | A13 | Voltage ID¹ | |
| HLDA | T2 | | | D29 | A12 | VID0 | W17 |
| HOLD | M3 | | | D30 | B12 | VID1 | W16 |
| IGNNE# | P19 | | | D31 | D11 | VID2 | L19 |
| INIT | P17 | | | D32 | B10 | VID3 | C15 |
| INTR | P16 | | | D33 | A10 | VID4 | A11 |
| INV | J1 | | | D34 | D10 | | |
| KEN# | K2 | | | D35 | C10 | | |
| LOCK# | T1 | | | D36 | D9 | | |
| M/IO# | J3 | | | D37 | A9 | | |
| NA# | L4 | | | D38 | C9 | | |
| NMI | R17 | | | D39 | A8 | | |
| PCD | R4 | | | D40 | A7 | | |
| PCHK# | P3 | | | D41 | B8 | | |
| PWT | V2 | | | D42 | D7 | | |
| RESET | W9 | | | D43 | C7 | | |
| SCYC | W8 | | | D44 | B6 | | |
| SMI# | P18 | | | D45 | A6 | | |
| SMIACT# | P2 | | | D46 | C6 | | |
| STPCLK# | M18 | | | D47 | A5 | | |
| W/R# | U5 | | | D48 | D6 | | |
| WB/WT# | M2 | | | D49 | C5 | | |
| | | | | D50 | C4 | | |
| | | | | D51 | B2 | | |

Notes: 1. The VID[4:0] pins are supported on low-power versions only. These pins are defined as no-connects on standard-power versions.

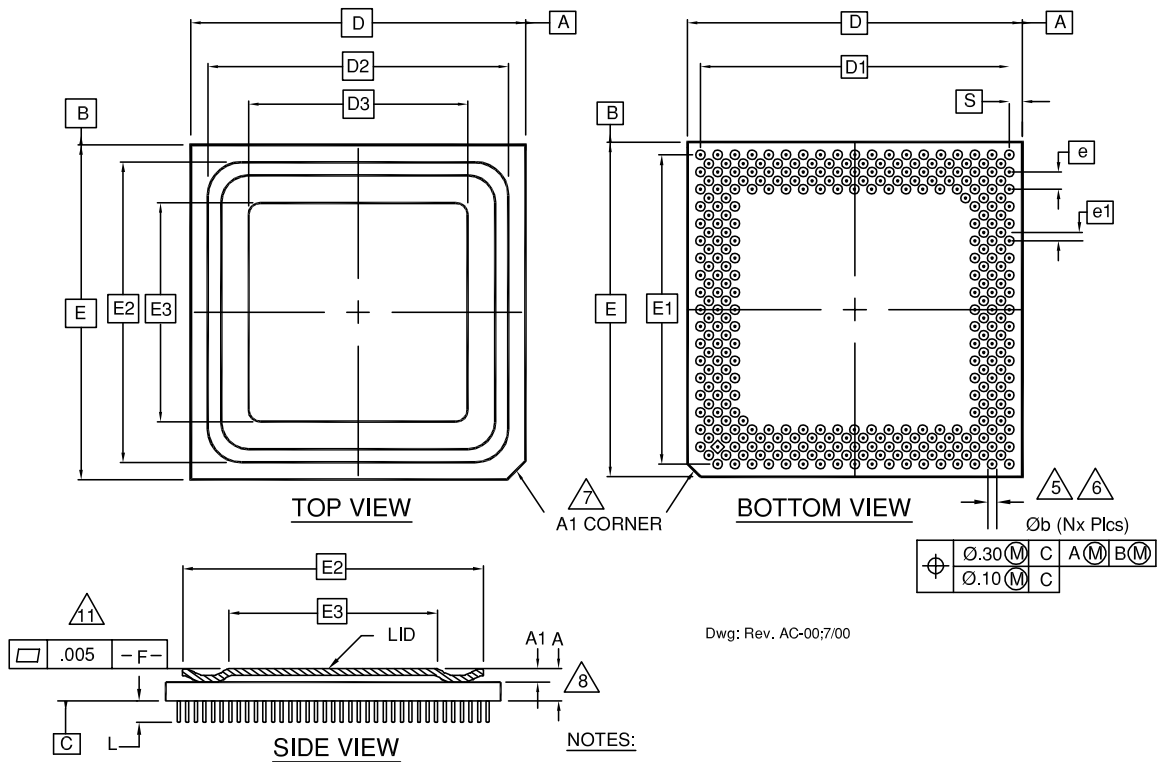
Table 78. OBGA Pin Designations for No Connect, Reserved, Power, and Ground Pins

| Pin Numbers | | | | |
|----------------------------------|------------------|------------------|-----------------|-----------------|
| No Connect (NC) | V _{CC2} | V _{CC3} | V _{SS} | V _{SS} |
| A11 ¹ | C2 | B13 | B3 | K13 |
| B4 | D4 | B17 | B5 | K15 |
| C15 ¹ | E5 | E18 | B7 | L6 |
| L17 | F6 | F14 | B9 | L7 |
| L19 ¹ | F7 | G15 | B11 | L8 |
| M16 | F8 | H10 | B15 | L9 |
| M19 | F9 | H11 | C18 | L10 |
| T5 | F10 | H12 | D8 | L11 |
| T9 | F11 | H13 | D12 | L12 |
| U11 | G2 | H14 | D16 | L13 |
| W7 | G5 | H16 | E2 | L14 |
| W16 ¹ | H4 | J15 | E6 | L16 |
| W17 ¹ | H6 | J18 | E7 | L18 |
| Internal No Connect (INC) | H7 | K14 | E8 | M5 |
| N16 | J5 | L15 | E9 | M12 |
| Reserved (RSVD) | K6 | M14 | E10 | M13 |
| G1 | K7 | N15 | E11 | M15 |
| G3 | K8 | N18 | E12 | N2 |
| G16 | K9 | P11 | E13 | N6 |
| H1 | K10 | P12 | E14 | N7 |
| H2 | K11 | P14 | E15 | N8 |
| H3 | L2 | R10 | F5 | N9 |
| J17 | L5 | R13 | F12 | N10 |
| K16 | M4 | U18 | F13 | N11 |
| M1 | M6 | V11 | F15 | N12 |
| N1 | M7 | V15 | G6 | N13 |
| N3 | M8 | | G7 | N14 |
| N4 | M9 | | G8 | P5 |
| P1 | M10 | | G9 | P7 |
| P8 | M11 | | G10 | P9 |
| R8 | N5 | | G11 | P10 |
| R19 | P6 | | G12 | P13 |
| | R2 | | G13 | P15 |
| | R5 | | G14 | R6 |
| | T4 | | G18 | R7 |
| | V3 | | H5 | R9 |
| | V7 | | H8 | R11 |
| | | | H9 | R12 |
| | | | H15 | R14 |
| | | | J2 | R15 |
| | | | J6 | R18 |
| | | | J7 | T6 |
| | | | J8 | T16 |
| | | | J9 | U2 |
| | | | J10 | V5 |
| | | | J11 | V9 |
| | | | J12 | V13 |
| | | | J13 | V17 |
| | | | J14 | |
| | | | K5 | |
| | | | K12 | |

Notes: 1. These pins are no-connects on standard-power versions only. They are defined as VID[4:0] on low-power versions. See page 330.

19 Package Specifications

19.1 321-Pin Staggered CPGA Package Specification



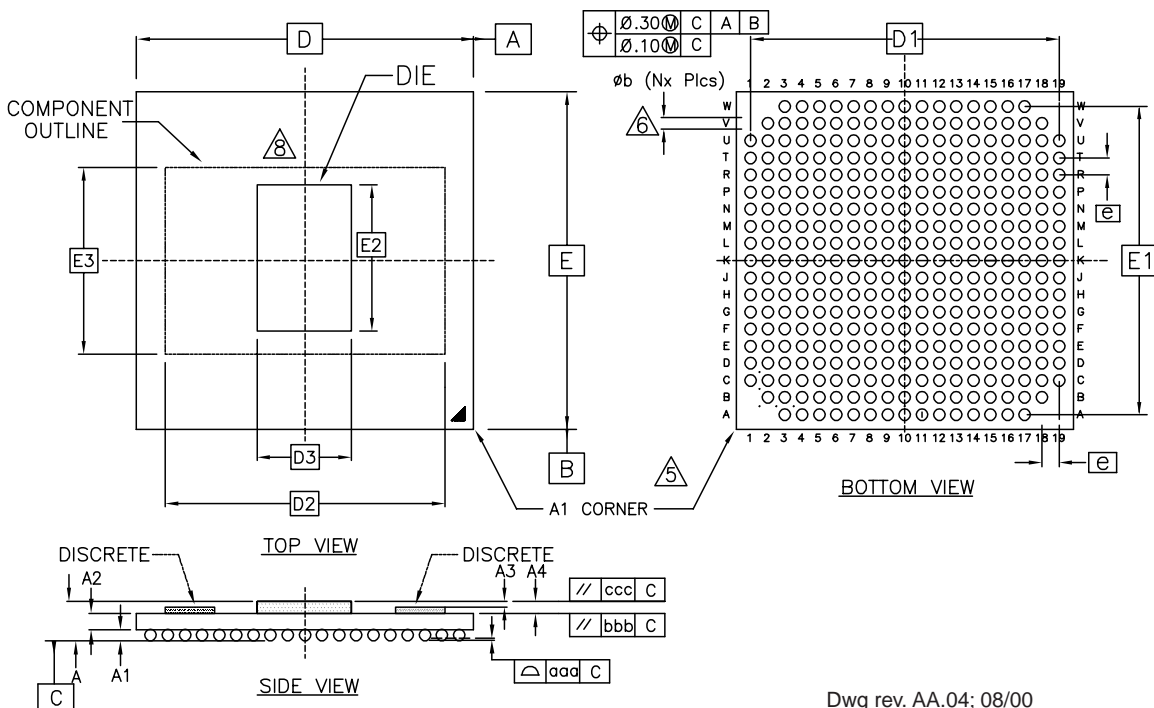
| AMD PACKAGE SYMBOL | VARIATIONS (INCHES.) | |
|--------------------|----------------------|-------|
| | 321 CGF | |
| | MIN. | MAX. |
| D/E | 1.940 | 1.960 |
| D1/E1 | 1.795 | 1.805 |
| D2/E2 | 1.768 | 1.776 |
| D3/E3 | 1.221 | 1.295 |
| A | .115 | .143 |
| A1 | .051 | .060 |
| ⌀b | .017 | .020 |
| ⌀b1 | -- | .064 |
| S | .060 | .100 |
| L | .120 | .130 |
| M | 37 | |
| N | 321 | |

NOTES:

- DIMENSIONING AND TOLERANCING PER ANSI Y14.5M-1982.
- ALL DIMENSIONS ARE SPECIFIED IN INCHES.
- TERMINAL POSITION DESIGNATIONS PER JEDEC PUBLICATIONS 95, SECTION 2.2-5.
- CORNERS OF THE PACKAGE BODY MAY HAVE CHAMFERS FOR HANDLING OR ORIENTATION PURPOSES.
- PIN DIAMETER DIMENSION (⌀b) EXCLUDES SOLDER DIP LEAD FINISH OR CUSTOM PLATING. PIN TIPS SHOULD HAVE RADIUS OR CHAMFER. MINIMUM GOLD PLATING THICKNESS IS 30 MICRO-INCHES.
- SYMBOL "M" DETERMINES PIN MATRIX SIZE AND "N" AS NUMBER OF PINS.
- THIS INDEX CORNER REPRESENTS PIN A1 IDENTIFIER ON BOTH SIDES OF THE PACKAGE AND MAY CONSIST OF NOTCHES, PINS OR METALLIZATIONS AND MAY ALSO VARY FROM THAT SHOWN IN THE DRAWING.
- DIMENSION "A" INCLUDES THE PACKAGE BODY AND LID.
- FOR STAGGERED PIN CONFIGURATION, PINS ON THE SAME ROW ARE ON A .100" GRID. ADJACENT ROWS OFFSET BY .050".
- REFER TO 09-000 SPEC FOR DETAILED PACKAGE LISTS.
- THIS REPRESENTS THE FLATNESS ON TOP OF THE LID.

Figure 125. 321-Pin Staggered CPGA Package Specification

19.2 349-Ball OBGA Package Specification



Dwg rev. AA.04; 08/00

| AMD PACKAGE | OBF349 (0.25µm) | |
|-------------|-----------------|-------|
| SYMBOL | MIN. | MAX. |
| D/E | 24.80 | 25.20 |
| D1/E1 | 22.86 BSC. | |
| D2 | 20.77 | 21.03 |
| D3 | 8.28 | 8.37 |
| E2 | 9.12 | 9.18 |
| E3 | 16.98 | 17.24 |
| A | 2.32 | 2.61 |
| A1 | 0.5 | 0.7 |
| A2 | 0.93 | 1.13 |
| A3 | 0.116 | --- |
| A4 | 0.73 | 0.88 |
| øb | 0.6 | 0.9 |
| e | 1.27 BSC. | |
| M | 19 | |
| N | 349 | |
| aaa | 0.15 | |
| bbb | 0.25 | |
| ccc | 0.35 | |

NOTES:

1. ALL DIMENSIONS ARE SPECIFIED IN MILLIMETER.
2. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M-1994.
3. CORNERS OF THE PACKAGE BODY MAY HAVE CHAMFERS FOR HANDLING OR ORIENTATION PURPOSES.
4. SYMBOL "M" DETERMINES PIN MATRIX SIZE AND "N" AS NUMBER OF BALLS.
5. THIS INDEX CORNER REPRESENTS PIN A1 IDENTIFIER ON BOTH SIDES OF THE PACKAGE AND MAY CONSIST OF NOTCHES, PINS OR METALLIZATIONS AND MAY ALSO VARY FROM THAT SHOWN IN THE DRAWING.
6. DIMENSION "b" IS MEASURED AT THE MAXIMUM SOLDER BALL DIAMETER ON A PLANE PARALLEL TO DATUM C.
7. REFER TO 09-000 SPEC FOR DETAILED PACKAGE LISTS.
8. THERE WILL BE UNDERFILL/EPOXY FILLET AROUND THE DEVICE. UNDERFILL CAN NOT TOUCH THE CHIP CAP PADS AND AMD LOGO. UNDERFILL MUST BE AT LEAST 1.5MM AWAY FROM THE EDGE OF THE PACKAGE.

Figure 126. 349-Ball OBGA Package Specification

20 Ordering Information

Standard AMD-K6™-III E+ Embedded Processor Products

AMD standard products are available in several operating ranges. The ordering part number (OPN) is formed by a combination of the elements below. See Table 79 on page 336 for valid ordering part number combinations.

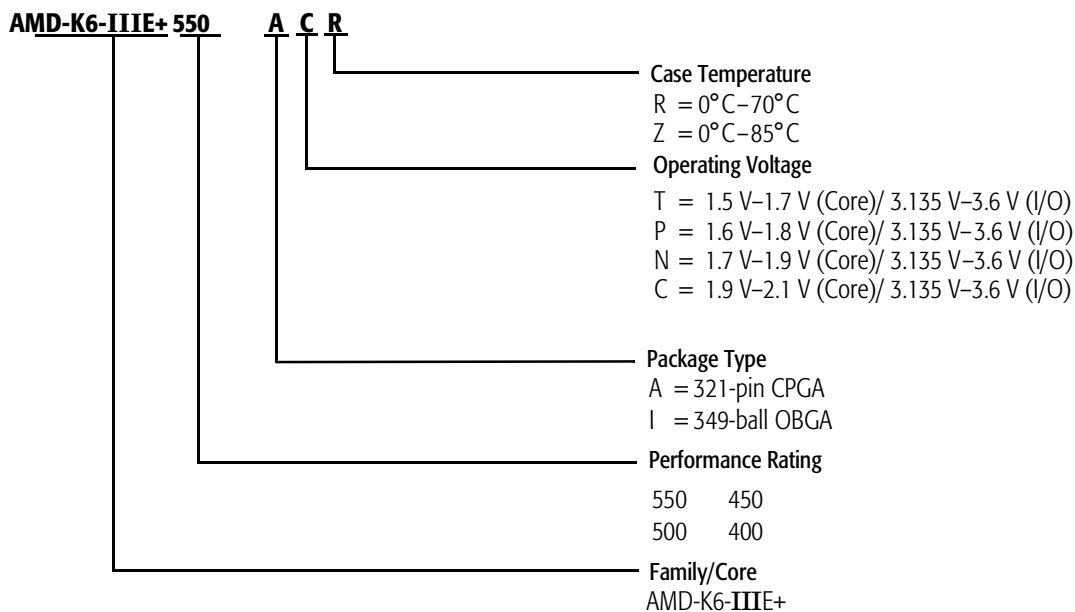


Table 79. AMD-K6™-III+ Embedded Processor Valid Ordering Part Number Combinations

| Device Type | OPN ¹ | Package Type | Operating Voltage | Case Temperature | Maximum CPU/Bus Frequency |
|----------------|-------------------|---------------|---|------------------|---------------------------|
| Low Power | AMD-K6-III+400ATZ | 321-pin CPGA | 1.5 V–1.7 V (Core) 3.135 V–3.6 V (I/O) | 0°C–85°C | 400 MHz/100 MHz |
| | AMD-K6-III+450APZ | 321-pin CPGA | 1.6 V–1.8 V (Core) 3.135 V–3.6 V (I/O) | 0°C–85°C | 450 MHz/100 MHz |
| | AMD-K6-III+500ANZ | 321-pin CPGA | 1.7 V–1.9 V (Core) 3.135 V–3.6 V (I/O) | 0°C–85°C | 500 MHz/100 MHz |
| | AMD-K6-III+400ITZ | 349-ball OBGA | 1.5 V–1.7 V (Core) 3.135 V–3.6 V (I/O) | 0°C–85°C | 400 MHz/100 MHz |
| Standard Power | AMD-K6-III+400ACR | 321-pin CPGA | 1.9 V–2.1 V (Core) 3.135 V–3.6 V (I/O) | 0°C–70°C | 400 MHz/100 MHz |
| | AMD-K6-III+450ACR | 321-pin CPGA | 1.9 V–2.1 V (Core) 3.135 V–3.6 V (I/O) | 0°C–70°C | 450 MHz/100 MHz |
| | AMD-K6-III+500ACR | 321-pin CPGA | 1.9 V–2.1 V (Core) 3.135 V–3.6 V (I/O) | 0°C–70°C | 500 MHz/100 MHz |
| | AMD-K6-III+550ACR | 321-pin CPGA | 1.9 V–2.1 V (Core) 3.135 V–3.6 V (I/O) | 0°C–70°C | 550 MHz/100 MHz |
| | AMD-K6-III+400ICR | 349-ball OBGA | 1.9 V–2.1 V (Core) 3.135 V–3.6 V (I/O) | 0°C–70°C | 400 MHz/100 MHz |
| | AMD-K6-III+450ICR | 349-ball OBGA | 1.9 V–2.1 V (Core) 3.135 V–3.6 V (I/O) | 0°C–70°C | 450 MHz/100 MHz |

Notes:

1. This table lists configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations and to check on newly-released combinations.

Index

Numerics

| | |
|---|---------------------------------|
| 0.18-Micron Process Technology | 7 |
| 100-MHz Bus | |
| clock switching characteristics | 298 |
| frontside | 1, 8 |
| input setup and hold timings | 302 |
| output delay timings | 300 |
| Super7 platform support | 1, 8 |
| 321-Pin Staggered CPGA Package | 2 |
| specification | 333 |
| 349-Ball OBGA Package | |
| specification | 334 |
| 3DNow!™ Technology | 2, 5, 7, 15, 17, 19, 21–24, 127 |
| data types | 37 |
| execution unit | 24 |
| INIT state | 203 |
| instruction compatibility, floating-point and | 240 |
| instructions | 89–90, 240 |
| PREFETCH instruction | 220 |
| register operation | 14 |
| registers | 35 |
| RESET state | 199 |
| software prefetching | 220 |
| 66-MHz Bus | |
| clock switching characteristics | 299 |
| input setup and hold timings | 306 |
| output delay timings | 304 |

A

| | |
|--|---------------|
| A[31:3] | 95 |
| A20M# Signal | 94, 242 |
| masking cache accesses with | 227 |
| Absolute Ratings | 289 |
| Accelerated Graphic Port (AGP) | 8 |
| Acknowledge, Interrupt | 188 |
| Address | |
| bus | 100, 109 |
| A[31:3] signals | 95 |
| address hold signal | 97 |
| address strobe signal | 96 |
| AHOLD restriction | 174, 178, 180 |
| coherency | 222 |
| parity | 98–99 |
| generation sequence during bursts (table) | 162 |
| hold signal | 97 |
| parity check signal | 99 |
| parity signal | 98 |
| ADS# Signal | 96 |
| ADSC# Signal | 96 |
| AGP | 8 |
| AHOLD | |
| -initiated inquire hit to modified line | 178 |
| -initiated inquire hit to shared or exclusive line | 176 |
| -initiated inquire miss | 174 |
| restriction | 180 |
| AHOLD Signal | 97, 280 |

| | |
|---|------------------|
| Airflow | |
| consideration in layout | 319 |
| heatsink with fan (figure) | 321 |
| management | 321 |
| path in a dual-fan system (figure) | 321 |
| path in an ATX form-factor system (figure) | 322 |
| Aligned Transfers | 129 |
| Allocate, Write | 215 |
| AMD PowerNow!™ Technology | 6, 143, 151, 277 |
| disabling | 145 |
| dynamic core voltage control | 151 |
| enabling | 145 |
| enhanced power management register (EPMR) | 144 |
| EPM 16-byte I/O block | 146 |
| EPM stop grant state | 150 |
| I/O base address definition | 145 |
| processor state observability register (PSOR) | 148 |
| SMM handler | 145 |
| voltage identification signals | 137, 151 |
| AP Signal | 98 |
| APCHK# Signal | 99 |
| Asserted signal | 93 |

B

| | |
|---|------------------|
| Backoff | 102 |
| BDC Bit | 147 |
| BE[7:0]# Signal | 100 |
| BF[2:0] | 101, 199, 285 |
| BIOS, enhanced power management | 145 |
| BIST | 251 |
| Block Diagram | 12 |
| BOFF# Signal | 102, 182 |
| locked operation with | 186 |
| Boundary-Scan | |
| bit definitions (table) | 257 |
| register (BSR) | 255 |
| test access port (TAP) | 253 |
| BR | 259 |
| Branch | |
| execution | 26 |
| history table | 25 |
| prediction | 1, 7, 15, 23, 26 |
| target cache | 26 |
| BRDY# Signal | 103 |
| BRDYC# Signal | 104 |
| BREQ Signal | 104 |
| BSR | 255 |
| Built-In Self-Test (BIST) | 251 |
| Burst | |
| pipelined burst reads | 162 |
| reads | 162 |
| ready | 103 |
| ready copy | 104, 200 |
| writeback | 164 |
| writeback due to cache-line replacement | 164–165 |
| Bus | |
| 100-MHz | 1, 8 |
| address | 178 |
| A[31:0] | 95 |
| AHOLD restriction | 180 |

| | |
|--|-------------------|
| coherency | 222 |
| external address strobe signal | 109 |
| hold signal | 97 |
| inquire cycles | 100, 174 |
| parity | 98–99 |
| strobe copy signal | 96 |
| strobe signal | 96 |
| valid | 109 |
| arbitration cycles, inquire and | 168 |
| backoff | 182 |
| byte enable signals | 100 |
| cycles | 153–197 |
| aligned transfers | 129 |
| definitions (table) | 142 |
| order during misaligned I/O transfers (table) | 167 |
| order during misaligned memory transfers (table) | 160 |
| special | 142, 190 |
| data | 97, 103, 174 |
| AHOLD restriction | 180 |
| aligned transfers | 129 |
| byte enables | 100 |
| D[63:0] | 107 |
| memory reads and writes | 158 |
| misaligned transfers | 129, 184 |
| parity | 108, 125 |
| state | 156 |
| transition | 157 |
| frequency signals | 101 |
| hold request signal | 115 |
| lock signal | 120 |
| request signal | 104 |
| states | |
| address | 156 |
| data | 156 |
| data-NA# requested | 156 |
| idle | 156 |
| pipeline address | 156 |
| pipeline data | 157 |
| state machine (figure) | 155 |
| transition | 157 |
| BVC Field | 146 |
| definition (table) | 147 |
| BVCM Bit | 147 |
| BYPASS Instruction | 260 |
| Bypass Register (BR) | 259 |
| C | |
| Cache | 263 |
| branch target | 26 |
| burst writeback cycles | 164 |
| cacheable access | 105 |
| coherency | 222 |
| writeback | 227 |
| writethrough | 227 |
| control | 229 |
| data cache | 16 |
| disabling | 46, 211, 251, 263 |
| enabling | 119 |
| flushing | 112, 191 |
| inhibiting | 251, 263 |
| inquire cycles | 168, 174, 178 |
| inquire cycles (table) | 225 |
| instruction cache | 16 |
| instruction fetch and decode | 17 |
| instruction prefetch | 16–17, 220 |
| L1 cache | |
| cache-line replacement | 214 |
| coherency states | 227 |
| data cache | 16 |
| data cache line (figure) | 207 |
| instruction cache | 16 |
| instruction cache line (figure) | 207 |
| internal snooping | 223 |
| organization | 205 |
| write allocate | 215 |
| L2 cache | |
| cache line (figure) | 207 |
| cache-line replacement | 214 |
| data reads | 266 |
| direct access | 50 |
| disabling for debug | 47 |
| EDX register content | 265 |
| Level-2 Cache Array Access Register (L2AAR) | 50 |
| organization | 205 |
| RDMSR instruction effect | 265 |
| sector and line organization (figure) | 265 |
| tag array testing | 213, 264 |
| tag information (figure) | 52 |
| tag or data location (figure) | 51 |
| tag reads | 267 |
| testing | 264 |
| WRMSR instruction effect | 265 |
| L3 cache | |
| debugging | 263 |
| PCD signal | 264 |
| testing | 263 |
| Level-2 Cache Array Access Register (L2AAR) | 264 |
| -line fills | 213, 264 |
| -line replacement | 214, 224 |
| masking cache accesses with A20M# | 227 |
| MESI states in the data | 207 |
| operation | 208 |
| organization | 205 |
| organization (figure) | 206 |
| predecode bits | 16–17, 208 |
| prefetching | 16–17, 220 |
| sector organization | 16 |
| sector organization (figure) | 16 |
| signals | 211 |
| snooping | 223 |
| states | 221 |
| states (table) | 225 |
| Super7 platform support | 8 |
| total internal | 1 |
| TR12 | 46 |
| translation lookaside buffers (TLBs) | 205 |
| write allocate | 215 |
| write cycle order | 229 |
| write merge buffer | 229 |
| write to a cacheable page | 216 |
| writeback | 12, 16 |
| writethrough | 227 |
| CACHE# Signal | 105 |
| generation (table) | 210 |

| | |
|---|--|
| Capacitance | 290 |
| capacitor placement | 294 |
| large capacitive loads | 295 |
| Capture-DR state | 262 |
| Capture-IR state | 262 |
| Case Temperature | 319 |
| extended | 315 |
| measuring | 319 |
| Centralized Scheduler | 21 |
| CLK | |
| switching characteristics | |
| 100-MHz bus operation | 298 |
| 60-MHz bus operation | 299 |
| CLK Signal | 105 |
| capacitance | 290 |
| Clock Control | 105, 277 |
| states | 277 |
| enhanced power management stop grant | 277 |
| halt | 280 |
| normal | 277 |
| state transitions (figure) | 278–279 |
| stop clock | 193, 285 |
| stop grant | 193, 280, 283 |
| stop grant inquire | 282 |
| switching characteristics | |
| 100-MHz bus operation | 298 |
| 66-MHz bus operation | 299 |
| Coherency | |
| cache | 222 |
| writeback | 227 |
| writethrough | 227 |
| Compatibility, Floating-Point, MMX, and 3DNow! | |
| Instructions | 240 |
| Component Placement | 294 |
| Configuration | |
| power-on initialization | 199 |
| signal timing (figure) | 313 |
| signal timing for 100-MHz bus operation (table) | 308 |
| signal timing for 66-MHz bus operation | 309 |
| VCC pins | 200 |
| Connections | |
| pin requirements | 295 |
| power requirements | 293 |
| Control Register 0 (CR0) | 40 |
| Control Register 1 (CR1) | 40 |
| Control Register 2 (CR2) | 39 |
| Control Register 3 (CR3) | 39 |
| Control Register 4 (CR4) | 39 |
| Counter, Time Stamp | 46 |
| CPGA Package | 2–3 |
| pin designations (figure) | 324 |
| pin designations by function (table) | 326 |
| pin differences (table) | 323 |
| specification | 333 |
| CR4 Register | 46 |
| Customer Service | iii |
| Cycles | |
| bus | 153 |
| hold and hold acknowledge | 168 |
| inquire | 94–99, 109, 113–114, 131, 139, 164, 168, 170, 172, 174, 176–178, 182, 186, 222, 263, 277, 280–282 |
| inquire and bus arbitration | 168 |
| interrupt acknowledge | 95, 98, 100, 106, 123, 138 |
| locked | 184 |
| pipelined | 17, 96 |
| pipelined write | 107 |

| | |
|-----------------------|---|
| shutdown | 192 |
| special bus | 190 |
| writeback | 94, 96–97, 110, 113, 139, 164, 172, 176, 178, 180, 186, 210, 264, 279, 282 |

D

| | |
|--|--------------|
| D/C# Signal | 106 |
| D[63:0] Signals | 107 |
| Data | |
| bus | |
| AHOLD restriction | 97, 180 |
| AHOLD timing | 174 |
| aligned transfers | 129 |
| BRDY# timing | 103 |
| byte enable signals | 100 |
| D[63:0] signals | 107 |
| data state | 156 |
| memory reads and writes | 158 |
| misaligned transfers | 129, 184 |
| parity | 108, 125 |
| split cycles | 129 |
| transition | 157 |
| cache | 16 |
| MESI states | 207 |
| parity | 108 |
| types | |
| 3DNow!™ Technology | 37 |
| floating-point register | 34 |
| integer | 29 |
| MMX technology | 36 |
| Data/Code Signal | 106 |
| Data-NA# Requested State | 156 |
| DC Characteristics | 289 |
| Debug | 268 |
| exceptions | 274 |
| registers | 41, 270 |
| DR3–DR0 | 272 |
| DR5–DR4 | 272 |
| DR6 | 273 |
| DR7 | 273 |
| System Management Mode (SMM) | 249–250 |
| Decoders | 13 |
| Decoupling Recommendations | 294 |
| Descriptors and Gates | 59 |
| Device Identification Register (DIR) | 258 |
| Diagrams | |
| key | 311 |
| timing | 153, 311–314 |
| waveform definitions | 153 |
| Digital Signal Processing Instructions | 90 |
| DIR | 258 |
| Dissipation, Power | 291 |
| Documentation | iii |
| DP[7:0] Signals | 108 |
| DR3–DR0 | 272 |
| DR5–DR4 | 272 |
| DR6 | 273 |
| DR7 | 273 |
| Driven signal | 93 |
| DSP Instructions | 90, 239 |
| Dual Voltage | 293 |

E

| | |
|---|---|
| EADS# Signal | 109 |
| EAS Register | |
| time stamp counter value | 46 |
| EAX Register | 28 |
| BIST results | 202 |
| cache accesses | 51 |
| EBF Field | 150 |
| EBP Register | 28 |
| EBX Register | 28 |
| ECX Register | 28, 46 |
| EDI Register | 28 |
| EDX Register | 28 |
| cache accesses | 50–51 |
| stepping ID | 202 |
| time stamp counter value | 46 |
| EFER | 44, 47, 202, 229 |
| Effective Bus Frequency Divisor Field | 150 |
| EFLAGS Register | 38, 242 |
| EIP Register | 241 |
| Electrical Data | 287 |
| absolute ratings | 289 |
| capacitance | 290 |
| DC characteristics | 289 |
| operating ranges | 287 |
| power and grounding | 293 |
| power dissipation | 291 |
| Embedded Processor Features | 1 |
| EMMS Instruction | 21 |
| EN Bit | 145 |
| Enhanced Power Management | |
| special bus cycle (table) | 142 |
| special bus cycles | 145 |
| stop grant state | 283 |
| Enhanced Power Management Register (EPMR) | 144 |
| EPM 16-Byte I/O Block | 146 |
| EPM Stop Grant State | 142, 150 |
| control | 147 |
| voltage identification output state | 147 |
| EPMR | 44, 144 |
| ESI Register | 28 |
| ESP Register | 28 |
| EWBE# Control (EWBEC) | 229 |
| EWBE# Signal | 110, 229, 280 |
| Exception | 98–99, 108, 125, 192, 240, 249–250, 273–275 |
| debug | 274 |
| flags | 32–33 |
| floating-point | 111, 116, 237–238, 240 |
| handler | 269 |
| handling floating-point | 237 |
| machine check | 45 |
| MMX technology | 240 |
| summary (table) | 62 |
| System Management Mode (SMM) | 250 |
| Execution latency (table) | 23 |
| Execution Units | 1 |
| 3DNow!™ technology | 24 |
| branch | 21, 26 |
| execution latency (table) | 23 |
| floating-point | 3, 21, 237 |
| multimedia | 21, 23–24, 239 |
| register X | 21, 24 |
| register Y | 21, 24 |
| throughput (table) | 23 |

| | |
|---|------------------|
| Extended Feature Enable Register (EFER) | 44, 47, 202, 229 |
| External | |
| address strobe signal | 109 |
| write buffer empty signal | 110 |
| EXTTEST Instruction | 260 |

F

| | |
|--|-------------------------|
| FEMMS Instruction | 21 |
| FERR# Signal | 111, 238, 240 |
| Float Conditions | 136, 141 |
| Floated signal | 93 |
| Floating-Point | |
| and MMX/3DNow! instruction compatibility | 240 |
| and multimedia execution units | 237 |
| error | 111 |
| execution unit | 237 |
| handling exceptions | 237 |
| instructions (table) | 82 |
| register data types | 34 |
| registers | 31 |
| FLUSH# Signal | 112, 199, 223, 252, 280 |
| FPU | |
| control word register | 33 |
| status word register | 32 |
| tag word register | 33 |
| Frequency | 285, 298–299, 310 |
| control | 151 |
| multiplier | 105 |
| operating | 101, 105, 199 |

G

| | |
|--------------------------------|----------|
| Gate Descriptor | 59, 62 |
| General-Purpose Registers | 28 |
| Generate Special Bus Cycle Bit | 145 |
| Global EWBE# Disable (GEWBED) | 230 |
| Ground | |
| pin designations (table) | 327, 331 |
| plane capacitance | 294 |
| pulldown resistor | 295 |
| split planes | 293 |
| unused active high inputs | 295 |
| GSBC Bit | 145 |

H

| | |
|---|--------------|
| Halt | |
| restart slot | 246 |
| state | 280 |
| Heat Dissipation Path | 318 |
| HIGHZ Instruction | 260 |
| Hit to | |
| modified line | 113 |
| modified line, AHOLD-initiated inquire | 178 |
| modified line, HOLD-initiated inquire | 172 |
| shared or exclusive line, AHOLD-initiated inquire | 176 |
| shared or exclusive line, HOLD-initiated inquire | 170 |
| HIT# Signal | 113 |
| HITM# Signal | 113 |
| HLDA Signal | 114 |
| Hold | |
| acknowledge cycle | 168 |
| acknowledge signal | 114, 168–170 |

| | | | |
|--|---|--|--|
| HOLD Signal | 115 | serializing | 94 |
| -initiated inquire hit to modified line | 172 | supported by the processor (table) | 63 |
| -initiated inquire hit to shared or exclusive line | 170 | Test Access Port (TAP) | 259 |
| timing | 297, 312 | WBINVD | 224 |
| I | | Integer | |
| I/O | | data registers | 29 |
| misaligned read and write | 167 | data types | 29 |
| read and write | 166 | instructions (table) | 65 |
| trap doubleword | 247 | Interrupts | 130, 188, 192, 196, 237–238, 240, 242, |
| trap restart slot | 248 | | 250, 274, 282 |
| IBF Field | 147 | 01h | 275 |
| IDCODE Instruction | 260 | 03h | 275 |
| IEEE 1149.1 | 2, 253 | 10h | 237 |
| IEEE 754 | 2, 31, 237 | acknowledge | 95, 103, 106, 118, 120, 125, 184, 188 |
| IEEE 854 | 237 | acknowledge cycle definition (table) | 188 |
| IGNNE# Signal | 116, 238, 240 | acknowledge cycles | 95, 98, 100, 106, 123, 138 |
| Ignore Numeric Exception | 116 | clock grant state | 282 |
| INIT Signal | 117, 242, 280 | descriptor table register | 54 |
| -initiated transition from protected mode to real mode | 196 | flag | 38, 118, 130 |
| processor state after | 203 | floating-point exceptions | 237–238 |
| Initialization | 117 | gate | 61 |
| output signal state | 200 | INIT | 196, 242 |
| power-on configuration | 199 | INTR | 118 |
| processor state after INIT | 203 | IRQ13 | 239 |
| processor state after RESET | 200 | MMX instructions | 240 |
| register state | 200 | NMI | 123, 203, 242 |
| RESET requirements | 200 | redirection bitmap | 55 |
| signals sampled during RESET | 199 | request | 118 |
| Input | | service routine | 118, 123, 238, 241 |
| capacitance | 290 | STPCLK# | 193 |
| leakage current | 290 | summary (table) | 62 |
| pin float conditions (table) | 141 | system management | 241 |
| pin types (table) | 140 | type of | 62 |
| setup and hold timing | | INTR Signal | 118, 280 |
| 100-MHz bus operation | 302 | INV Signal | 118 |
| 66-MHz bus operation | 306 | Invalidation Request | 118 |
| Input/Output (I/O), capacitance | 290 | INVD Instruction | 224 |
| Inquire | 171, 173, 175, 277 | IOBASE Field | 145 |
| bus arbitration cycles | 168 | K | |
| cycle hit | 113 | KEN# Signal | 119 |
| cycle hit to modified line | 113 | L | |
| cycles | 94–99, 109, 113–114, 131, 139, 164, 168, | L1 Cache | 46 |
| | 170, 172, 174, 176–178, 180, 182, 186, 222, | cache line (figure) | 207 |
| | 263, 277, 280–282 | cache-line replacements | 214 |
| miss, AHOLD-initiated | 174 | coherency | 222, 227 |
| Instructions | 63 | data cache line (figure) | 207 |
| 3DNow!™ technology | 89–90, 239 | disabling | 46, 211–212 |
| 3DNow!™ technology (table) | 89–90 | flushing | 223 |
| cache | 16 | inquire cycles (table) | 225 |
| decode | 18 | instruction cache line (figure) | 207 |
| digital signal processing | 239 | internal snooping | 223 |
| digital signal processing (table) | 90 | MESI states | 207 |
| EMMS | 21 | organization | 205 |
| FEMMS | 21 | organization (figure) | 206 |
| fetch | 17 | prefetching | 220 |
| floating-point (table) | 82 | sector organization (figure) | 16 |
| integer (table) | 65 | states (table) | 221, 225–226 |
| INVD | 224 | write allocate | 215 |
| MMX technology | 86, 239 | | |
| MMX technology (table) | 86 | | |
| pointer | 31 | | |
| PREFETCH | 17, 220 | | |
| RSM | 241 | | |

| | | | |
|---|--------------------------|--|-----------------------|
| L2 Cache | 112–113, 139, 142 | Memory | |
| access type | 265 | management registers | 54 |
| built-in self test | 251 | or I/O | 121 |
| cache line (figure) | 207 | read and write, misaligned single-transfer | 160 |
| cache sector and line organization (figure) | 265 | read and write, single-transfer | 158 |
| cache-line fills | 214 | reads and writes | 158 |
| cache-line replacements | 214 | type range registers (MTRR) | 49, 231 |
| coherency | 222 | MESI | 1, 168, 172, 206, 227 |
| data location (figure) | 51 | bit | 16, 207, 209 |
| data reads | 266 | states in the data cache | 207 |
| direct access | 50 | Microarchitecture | 3, 11–26 |
| disabling | 211–212 | branch-prediction | 25 |
| disabling for debug | 47 | cache | 16 |
| flushing | 223 | centralized scheduler | 21 |
| inquire cycles (table) | 225 | decoders | 13 |
| least recently used (LRU) algorithm | 215 | enhanced RISC86 | 12 |
| Level-2 Cache Array Access Register (L2AAR) | 50 | execution units | 22 |
| LRU field | 268 | instruction fetching and decode | 17 |
| MESI states | 207 | instruction prefetch | 16 |
| operation | 209 | overview | 11 |
| organization | 205 | predecode | 16 |
| organization (figure) | 206, 265 | Misaligned | |
| predecode bits not stored | 208 | I/O read and write | 167 |
| prefetching | 220 | I/O transfers (table) | 167 |
| sector organization (figure) | 16 | memory transfers (table) | 160 |
| states (table) | 221, 225–226 | single-transfer memory read and write | 160 |
| Super7 platform support | 8 | transfers | 129 |
| T/D bit | 265 | MMX Technology | 19, 21–24, 127 |
| tag array testing | 213, 264 | 3DNow!™ registers | 35 |
| tag information (figure) | 52 | data types | 37 |
| tag location (figure) | 51 | exceptions | 240 |
| tag reads | 267 | INIT state | 203 |
| testing | 264 | instruction compatibility, floating-point and | 240 |
| write allocate | 215 | instructions | 240 |
| L2AAR | 44, 50, 212–213, 264–267 | instructions (table) | 86 |
| L3 Cache | | register operation | 14 |
| debugging | 263 | registers | 35 |
| PCD signal | 264 | RESET state | 199 |
| testing | 263 | Model-Specific Registers (MSR) | 44 |
| Latency, execution (table) | 23 | MSR | 44 |
| Layout and Airflow Considerations | 319 | MTRR | 49, 231 |
| Level-2 Cache Array Access Register (L2AAR) | 264–267 | Multimedia | |
| Literature | iii | and 3DNow!™ execution units | 239 |
| LOCK# Signal | 120 | execution unit | 24, 239 |
| Locked | | functional unit | 23 |
| cycles | 184 | N | |
| operation with BOFF# intervention | 186 | NA# Signal | 122 |
| operation, basic | 184 | Negated signal | 93 |
| Logic | | Next Address | 122 |
| branch-prediction | 15, 23, 25–26 | NMI Signal | 123, 242, 280 |
| external support of floating-point exceptions | 238 | No-Connect Pins | 128, 295 |
| symbol (figure) | 91 | Non-Maskable Interrupt | 123 |
| Low-Power Devices | 4, 288, 291–292, 336 | Non-Pipelined Single-Transfer Memory Read/Write and Write Delayed by EWBE# | 159 |
| M | | Normal State | 277 |
| MIO# Signal | 121 | O | |
| Machine Check Address Register (MCAR) | 44–45, 202 | OBGA Package | 2–3 |
| Machine Check Exception | 45 | pin designations (figure) | 328 |
| Machine Check Type Register (MCTR) | 44–45, 202 | pin designations by function (table) | 330 |
| Maskable Interrupt | 118 | pin differences (table) | 323 |
| MCAR | 44–45, 202 | specification | 334 |
| MCTR | 44–45, 202 | | |

Operating Ranges 287
 OPN 335
 Ordering Information 335
 Ordering Part Number (OPN) 335
 Output
 capacitance 290
 delay timings
 100-MHz bus operation 300
 66-MHz bus operation 304
 leakage current 290
 pin float conditions (table) 141
 signal state after RESET (table) 200

P

Package
 Socket 7 platform 8
 specifications 333
 Super7 platform 8
 thermal specifications 315
 Packed Decimal Data Register 34
 Page
 cache disable 124
 directory entry (PDE) 57–58, 209
 flush/invalidate register (PFIR) 223
 table entry (PTE) 57, 59, 209
 writethrough 126
 Paging 56
 Parity 98, 100, 108, 125, 158
 bit 98, 108, 125
 check 98–99, 108, 125
 error 99, 125, 174, 255
 flags 38
 Part Numbers 335
 PCD Signal 124, 209, 219
 generation (table) 210
 PCHK# Signal 125
 PFIR 49–50, 202, 223
 Pins
 connection requirements 295
 designations 323–331
 float conditions (table) 141
 I/O voltage 289
 input pin types (table) 140
 logic symbol (figure) 91
 no-connect 295
 signal descriptions 93–139
 Pipeline 156–157, 162
 address 156
 control 24–25
 data 157
 register X and Y 24
 Pipelined 23, 122, 157, 162–163, 180, 220
 burst reads 162
 cycles 17, 96, 107
 design 22
 Platform
 Socket 7 8
 Super7 8
 Pointer, Instruction 31
 Power
 and grounding 293
 connections 293
 consumption and thermal resistance (figure) 317
 dissipation 291

isolation region between planes 293
 management 6
 management, enhanced 143
 plane capacitance 294
 sequencing 293
 PowerNow! Technology. See AMD PowerNow!™ Technology.
 Power-on Configuration and Initialization 199
 Precision Real Data Registers 34
 Predecode Bits 16–17, 208
 PREFETCH Instruction 17
 Prefetching 17
 hardware 220
 PREFETCH instruction 220
 software 220
 Processor
 absolute ratings 289
 AMD PowerNow!™ technology 143
 block diagram 13
 bus cycles 153
 cache organization 4, 205
 clock control 277
 configuration 199
 DC characteristics 289
 decoders 13
 electrical data 287
 features 3
 heat dissipation path 318
 logic symbol (figure) 91
 low-power devices 288, 291–292, 336
 microarchitecture overview 11
 multimedia execution unit 239
 operating ranges 287
 ordering information 335
 package specifications 333
 pin connection requirements 295
 power-on initialization 199
 process technology 7
 scheduler 21
 signal descriptions. See 93
 signal switching characteristics 297
 Socket 7 platform 8
 software environment 27
 standard-power devices 288, 336
 state observability register (PSOR) 49, 202
 Super7 platform 8
 System Management Mode (SMM) 241
 test and debug 251
 thermal design 315
 write merge buffer 229
 Processor State Observability Register (PSOR)
 low-power version 148
 Protected Mode
 INIT-Initiated transition 196
 real mode transition 196
 PSOR 49, 148, 202
 PWT Signal 126
 generation (table) 210

R

RDMSR Instruction 46
 RDTSC Instruction 46
 Read and Write
 basic I/O 166
 misaligned I/O 167
 Reads, Burst Reads and Pipelined Burst 162

| | |
|--|----------------------------------|
| Real Mode | |
| INIT-initiated transition | 196 |
| protected mode transition | 196 |
| Register X and Y | |
| functional unit | 24 |
| pipelines | 24 |
| Registers | 14, 27, 240 |
| 3DNow!™ technology | 35 |
| boundary scan (BSR) | 255 |
| bypass (BR) | 259 |
| control | 39 |
| control 0 (CR0) | 40 |
| control 1 (CR1) | 40 |
| control 2 (CR2) | 39 |
| control 3 (CR3) | 39 |
| control 4 (CR4) | 39 |
| data types, floating-point | 34 |
| debug | 41, 270 |
| descriptors and gates | 59 |
| device identification (DIR) | 258 |
| DR3–DR0 | 272 |
| DR5–DR4 | 272 |
| DR6 | 273 |
| DR7 | 273 |
| EAX | 28 |
| EBP | 28 |
| EBX | 28 |
| ECX | 28 |
| EDI | 28 |
| EDX | 28 |
| EFLAGS | 38 |
| enhanced power management (EPMR) | 144 |
| EPMR | 144 |
| ESI | 28 |
| ESP | 28 |
| extended feature enable (EFER) | 44, 47, 202, 229 |
| floating-point | 31 |
| FPU control word | 33 |
| FPU status word | 32 |
| FPU tag word | 33 |
| general-purpose | 28 |
| instruction (IR) | 255 |
| IR | 255 |
| level-2 cache array access (L2AAR) | 264 |
| machine check address (MCAR) | 44–45, 202 |
| machine check type (MCTR) | 44–45, 202 |
| MCAR | 45 |
| memory management | 54 |
| memory type range (MTRR) | 231 |
| MMX technology | 35 |
| model-specific (MSR) | 44 |
| packed decimal data | 34 |
| PFIR | 50 |
| precision real data | 34 |
| processor state observability (PSOR) | 148 |
| processor state observability register (PSOR) | 49, 202 |
| PSOR | 49, 148 |
| reset state | 200 |
| segment | 30 |
| segment (table) | 30 |
| SYSCALL/SYSRET target address (STAR) | 44, 48, 202 |
| System Management Mode (SMM) initial state (table) | 243 |
| test (TR12) | 46 |
| Test Access Port (TAP) | 255 |
| time stamp counter (TSC) | 46 |
| TR12 | 46 |
| UWCCR | 49 |
| X and Y | 21–22, 24 |
| Regulator, Voltage | 319 |
| Reserved (RSVD) Pins | |
| description | 128 |
| pin designations (table) | 327, 331 |
| RESET Signal | 127, 200, 280 |
| signals sampled during reset | 199 |
| state of processor after reset | 200 |
| timing (figure) | 313 |
| timing for 100-MHz bus operation | 308 |
| timing for 66-MHz bus operation | 309 |
| Return Address Stack | 26 |
| RISC86 Microarchitecture | 12 |
| RSM Instruction | 246, 249 |
| RSVD Pins | 128 |
| S | |
| SAMPLE/PRELOAD Instruction | 260 |
| Sampled signal | 93 |
| Scheduler/Instruction Control Unit | 14, 21 |
| SCYC Signal | 129 |
| Sector, Write to a | 216, 220 |
| Segment | |
| descriptor | 30, 59–61 |
| registers | 30 |
| task state | 55 |
| usage | 30 |
| Segment Registers | 30 |
| Serializing Instruction | 94 |
| SGTC Field | 147, 283 |
| Shift-DR state | 262 |
| Shift-IR state | 262 |
| Shutdown Cycle | 192 |
| Signals | |
| A[31:3] | 95 |
| A20M# | 94, 242 |
| ADS# | 96 |
| ADSC# | 96 |
| AHOLD | 97, 280 |
| AP | 98 |
| APCHK# | 99 |
| asserted | 93 |
| BE[7:0]# | 100 |
| BF[2:0] | 101, 285 |
| BOFF# | 102, 182 |
| BRDY# | 103 |
| BRDYC# | 104 |
| BREQ | 104 |
| CACHE# | 105, 210 |
| cache-related | 211 |
| CLK | 105 |
| D/C# | 106 |
| D[63:0] | 107 |
| descriptions | 93–139 |
| DP[7:0] | 108 |
| driven | 93 |
| EADS# | 109 |
| EWBE# | 110, 229, 280 |
| FERR# | 111, 240 |
| floated | 93 |
| FLUSH# | 112, 199, 223, 252, 280, 283–284 |
| HIT# | 113 |

| | | | |
|--|--------------------|---|--------------------|
| HITM# | 113 | cache writeback invalidation | 190 |
| HLDA | 114 | definition (table) | 142 |
| HOLD | 115 | differentiating | 142, 190 |
| IGNNE# | 116, 240 | encoding | 142, 190 |
| INIT | 117, 280, 283 | enhanced power management | 145 |
| INTR | 118, 280, 283 | EPM stop grant | 190 |
| INV | 118 | EWBE# timing | 110 |
| KEN# | 119 | examples | 190 |
| LOCK# | 120 | flush acknowledge | 112, 164, 190, 212 |
| logic symbol (figure) | 91 | halt | 190–191, 280 |
| M/IO# | 121 | shutdown | 190, 192 |
| NA# | 122 | signal states (table) | 142 |
| negated | 93 | stop grant | 132, 190, 193, 281 |
| NML | 123, 280, 283 | System Management Mode (SMM) | 247 |
| output | 200 | Speculative EWBE# Disable (SEWBED) | 230 |
| PCD | 124 | Split Cycle | 129 |
| PCHK# | 125 | Standard-Power Devices | 4, 288, 336 |
| PWT | 126 | STAR | 44, 48, 202 |
| RESET | 127, 280, 283 | State | |
| RSVD | 128 | bus machine (figure) | 155 |
| sampled | 93 | cache | 221 |
| sampled during RESET | 199 | processor | |
| SCYC | 129 | after INIT | 203 |
| SMI# | 130, 241, 280, 283 | after RESET | 200 |
| SMIACT# | 131, 241 | Stepping ID | 202 |
| STPCLK# | 132, 280 | Stop | |
| switching characteristics | 297 | clock | 132 |
| TCK | 133 | power dissipation | 291–292 |
| TDI | 133 | clock state | 193, 285 |
| TDO | 133 | grant | |
| terminology | 93 | inquire state | 277, 280–282 |
| Test Access Port (TAP) | 254 | state | 193, 280, 282 |
| timing (figures) | 159–197, 311–314 | grant state | 283 |
| TMS | 134 | Stop Grant Time-Out Counter Field | 150 |
| TRST# | 134 | Storage Temperature | 289 |
| VCC2DET | 135 | STPCLK# Signal | 132, 280 |
| VCC2H/L# | 136 | Super7 Platform | 1 |
| VID[4:0] | 137, 151 | advantages | 9 |
| W/R# | 138 | initiative | 8 |
| WB/WT# | 139 | Switching Characteristics | 298 |
| SIMD | 15 | 100-MHz bus operation | 298 |
| Single Instruction Multiple Data (SIMD) operations | 15 | 66-MHz bus operation | 299 |
| Single-Transfer Memory Read and Write | 158 | input setup and hold timings for 100-MHz bus | 302 |
| SMI# Signal | 130, 241, 280 | input setup and hold timings for 66-MHz bus | 306 |
| SMIACT# Signal | 131, 241 | output delay timings for 100-MHz bus | 300 |
| SMM | 241 | output delay timings for 66-MHz bus | 304 |
| Snooping | 131, 139, 164 | signal | 297 |
| cache states (table) | 226 | valid delay, float, setup, and hold timings | 300 |
| data cache | 223 | SYSCALL Instruction | 80 |
| instruction cache | 223 | SYSCALL/SYSRET Target Address Register (STAR) | 44, 48, |
| internal cache | 223 | | 202 |
| processor-initiated | 223 | SYSRET Instruction | 80 |
| Software Environment | 27 | System | |
| descriptors | 59 | management interrupt | 130 |
| exceptions (table) | 62 | management interrupt active | 131 |
| gates | 59 | System Design | |
| instructions supported | 63 | airflow management | 321 |
| interrupts (table) | 62 | ATX form factor | 322 |
| memory management registers | 54 | component placement | 294 |
| model-specific registers (MSR) | 44 | decoupling recommendations | 294 |
| paging | 56 | heatsink | 320 |
| registers | 27 | pin connection requirements | 295 |
| Software Prefetching | 220 | power connections | 293 |
| Special Bus Cycles | | voltage regulator | 319 |
| BRDY# timing | 103 | | |
| cache invalidation | 190 | | |

| | | | |
|---|---------------|---|----------------------------|
| System Management Mode (SMM) | 241 | IDCODE | 260 |
| AMD PowerNow!™ features | 145 | SAMPLE/PRELOAD | 260 |
| base address | 246 | registers | 255 |
| debugging in | 250 | boundary scan (BSR) | 255 |
| default register values | 241 | bypass (BR) | 259 |
| enhanced power management | 145 | device identification (DIR) | 258 |
| entering | 241 | instruction register (IR) | 255 |
| exceptions in | 250 | signals | 254 |
| halt restart slot | 246 | states | |
| I/O trap doubleword | 247 | capture-DR | 262 |
| I/O trap restart slot | 248 | capture-IR | 262 |
| INIT | 242 | shift-DR | 262 |
| initial register state (table) | 243 | shift-IR | 262 |
| interrupts in | 250 | state machine | 260 |
| memory (figure) | 242 | test-logic-reset | 262 |
| NMI | 242 | update-DR | 262 |
| operating mode | 241 | update-IR | 262 |
| revision identifier | 245 | Thermal | 317, 320–321 |
| RSM instruction | 241 | design | 315 |
| SMI# interrupt | 241 | extended temperature rating | 315 |
| SMIACT# signal | 241 | heat dissipation path | 318 |
| state-save area (table) | 243 | layout and airflow consideration | 319 |
| system management interrupt active signal | 131 | measuring case temperature (figure) | 319 |
| system management interrupt signal | 130 | model | 317 |
| T | | package specifications | 315 |
| TAP | 253 | Third-Party Support | iv |
| Task State Segment (TSS) | 55 | Three-State Test Mode | 252 |
| TCK Signal | 133 | Time Stamp Counter Register (TSC) | 46 |
| TDI Signal | 133 | Timing | |
| TDO Signal | 133 | bus cycles (figures) | 159–197 |
| Technical Publications | iii | switching characteristics (figures) | 311–314 |
| Technical Support | iii | TLB | 206 |
| Temperature | 288, 315, 317 | TMS Signal | 134 |
| ambient | 315 | TR12 | 44, 46, 202, 210, 218, 263 |
| case | 319 | Transition | 157 |
| case-to-ambient | 316 | Translation Lookaside Buffer (TLB) | 205 |
| extended for low-power devices | 315 | TRST# Signal | 134 |
| storage | 289 | TSC | 44, 46, 202, 280–281, 283 |
| Terminology, Signals | 93 | TSS | 55, 61–62, 244, 273 |
| Test | | U | |
| boundary-scan | 253 | UC Memory Type | 49 |
| built-in self-test (BIST) | 251 | UC/WC Cacheability Control Register (UWCCR) | 200, 202, 210 |
| cache inhibit | 263 | Uncacheable Memory | 49, 230–231 |
| clock signal | 133 | UWCCR | 49, 200, 202, 210, 232 |
| data input signal | 133 | V | |
| data output signal | 133 | VCC2 Pins | |
| debug | 268 | absolute ranges | 289 |
| L2 cache testing | 264 | operating ranges | 288 |
| -logic-reset state | 262 | pin designations (table) | 327, 331 |
| mode select | 134 | power connections | 293 |
| register 12 (TR12) | 46 | processor voltage | 293 |
| registers | 255 | RESET requirements | 200 |
| reset | 134 | VCC2DET Signal | 135 |
| scan chain (table) | 257 | VCC2H/L# Signal | 136 |
| signal timing (figure) | 314 | VCC3 Pins | |
| signal timing (table) | 310 | absolute ranges | 289 |
| signals | 254 | I/O voltage | 293 |
| capacitance | 290 | operating ranges | 288 |
| tag array testing | 264 | pin designations (table) | 327, 331 |
| test access port (TAP) | 253 | instructions | 259 |
| three-state test mode | 252 | BYPASS | 260 |
| Test Access Port (TAP) | | EXTTEST | 260 |
| instructions | 259 | HIGHZ | 260 |
| BYPASS | 260 | | |
| EXTTEST | 260 | | |
| HIGHZ | 260 | | |

| | | | |
|---|---|------------------------|--|
| power connections | 293 | cycles | 94, 96–97, 110, 113, 139, 164, 172, 176, 178, 180, 182, 186, 210, 264, 279, 282 |
| RESET requirements | 200 | L1 cache | 205 |
| unused active low inputs | 295 | L2 cache | 205 |
| VID[4:0] | 137, 151 | memory writes | 227 |
| VID[4:0] Signals | 151 | or writethrough | 139 |
| VIDC Bit | 147 | Write-Combining Memory | 49, 230–231 |
| VIDO Field | 147 | Writethrough | |
| Voltage | | coherency state | 227 |
| active high signals | 154 | memory writes | 227 |
| active low signals | 154 | | |
| bus divisor (table) | 147 | | |
| CLK switching characteristics | 298 | | |
| control | 150 | | |
| dual | 293 | | |
| I/O pin | 289, 293 | | |
| input low | 289 | | |
| plane isolation | 293 | | |
| power connections | 293 | | |
| processor | 293 | | |
| regulator | 319–320 | | |
| supply | 288 | | |
| VCC2 detect signal | 135 | | |
| VCC2 High/Low Signal | 136 | | |
| VCC2DET signal | 135 | | |
| voltage identification control (table) | 147 | | |
| VSS Pins | | | |
| connections | 293 | | |
| pin designations (table) | 327, 331 | | |
| unused active high inputs | 295 | | |
| W | | | |
| W/R# Signal | 138 | | |
| WB/WT# Signal | 139 | | |
| WBINVD Instruction | 224 | | |
| WC Memory Type | 49 | | |
| WHCR | 44, 48, 202, 219 | | |
| Write | | | |
| allocate | 208, 215–216, 219 | | |
| conditions (figure) | 218 | | |
| enabling | 217 | | |
| limit | 216 | | |
| logic mechanisms and conditions (figure) | 219 | | |
| handling control register (WHCR) | 202, 219 | | |
| to a cacheable page | 216 | | |
| to a sector | 216, 220 | | |
| Write Merge Buffer | 229 | | |
| EWBE# control | 229 | | |
| EWBEC settings (table) | 231 | | |
| memory type range registers (MTRRs) | 231 | | |
| memory-range restrictions | 233 | | |
| examples | 235 | | |
| valid masks and range sizes (table) | 234 | | |
| performance (table) | 229 | | |
| UC/WC Cacheability Control Register (UWCCR) | 232 | | |
| uncacheable memory | 231 | | |
| write cycle order | 229 | | |
| write-combining memory | 231 | | |
| Write/Read | 138 | | |
| Writeback | 105, 107–108, 119, 126, 131, 139, 142, 164–165, 190, 278 | | |
| burst | 164 | | |
| cache | 12, 16 | | |
| coherency state | 227 | | |

