Nx586™ Processor and Nx587™ Numerics Processor Databook

July 8, 1993

PROPRIETARY and CONFIDENTIAL
COPYING is FORBIDDEN

NexGen
Microsystems
1623 Buckeye Drive
Milpitas, CA 95035

Information in this manual is subject to change without notice and does not represent a commitment on the part of NexGen Microsystems.

This manual was written for NexGen Microsystems by Warthman Associates, Palo Alto, California.

Contents

Preface ix
Notation ix
Related Publications xi

Nx586 Features and Signals 1
Nx586 Pinouts 4
Nx586 NexBus 10
NexBus Arbitration 10
NexBus Cycle Control 12
NexBus Cache Control 14
NexBus Transceivers 15
NexBus Address and Data 16
Nx586 Level-2 Cache Signals 22
Numerics-Processor Bus Signals (on Nx586) 23
Nx586 System Signals 25
Nx586 Clock 25
Nx586 Interrupts and Reset 26
Nx586 Test 27
Nx586 Alphabetical Signal Summary 29

Nx587 Features and Signals 32
Nx587 Pinouts 34
Numerics-Processor Bus Signals (on Nx587) 38
Nx587 System Signals 40
Nx587 Clock 40
Nx587 Interrupts and Reset 41
Nx587 Test and Reserved Signals 42
Nx587 Alphabetical Signal Summary 43

Hardware Architecture 45
Bus Structure 45
NexBus 45
Level-2 Cache Bus 50
Numerics Processor Bus 50
Operating Frequencies 50

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems.
NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

Internal Architecture.......52
Storage Hierarchy.......54
Transaction Ordering.......57
Cache and Memory Subsystem.......58
Characteristics.......58
Cache Coherency.......59
State Transitions.......61
Invalid.......64
Shared.......65
Exclusive.......65
Modified.......66
Interrupts.......67
Clock Generation.......68
Multiprocessing Support.......69

Bus Operations.......71
Accesses on the Level-2 Cache.......71
NexBus Arbitration and Address Phase.......72
Single-Qword Memory Operations.......73
Burst Memory Operations.......77
I/O Operations.......81
Interrupt-Acknowledge.......81
Halt and Shutdown.......82
Cache-Control Operations.......83
Obtaining Exclusive.......83
Intervenor Operations.......84
Modified Cache-Block Hit During Single-Qword Operations.......85
Modified Cache-Block Hit During Four-Qword (Block) Operations....86

Programming.......89
Operating Modes.......89
Operands.......90
Operand Alignment.......92
Addresses.......93
Memory.......96
Stack Operations.......98
I/O.......99
Interrupts and Exceptions.......101
Application Registers.......102
General-Register Functions.......104
Flags.......106
Instructions.......108
Instruction Groups.......108
Instruction Prefixes.......113
Syntax.......114
Instruction Latency.......117
Performance Optimization.......127
Self-Modifying Code.......127
Software Differences.......128

Application Software 128
System Software 129
Benchmark Software 130

Electrical Data 131

Absolute Maximum Ratings 131
D.C. Characteristics 132
A.C. Characteristics 133
Primary-Signal Timing 134
Clock-Signal Timing 137
Test-Signal Timing 138

Mechanical Data 139

Ordering Information 147

Glossary 149

Index 157

Figures

Figure 1 Nx586 Signal Organization..3
Figure 2 Nx586 Pin List, By Signal Name..4
Figure 3 Nx586 Pin List, By Signal Name (continued)..5
Figure 4 Nx586 Pin List, By Pin Number..6
Figure 5 Nx586 Pin List, By Pin Number (continued)..7
Figure 6 Nx586 Pinout Diagram..8
Figure 7 Nx586 Pinout Diagram (Bottom View)..9
Figure 8 NexBus Address and Status Phase..18
Figure 9 Byte-Enable Usage..19
Figure 10 Bus-Cycle Types..20
Figure 11 Block Transfer Sizes..21
Figure 12 Nx586 Alphabetical Signal Summary..31
Figure 13 Nx587 Signal Organization..33
Figure 14 Nx587 Pin List (By Signal Name)..34
Figure 15 Nx587 Pin List (By Pin Number)..35
Figure 16 Nx587 Pinout Diagram (Top View)..36
Figure 17 Nx587 Pinout Diagram (Bottom View)..37
Figure 18 Nx587 Alphabetical Signal Summary..43
Figure 19 NxPC-Based Single-Processor System Diagram..47
Figure 20 Operating Frequencies (66MHz Processor)..51
Figure 21 Nx586 Internal Architecture..53
Figure 22 Storage Hierarchy (Reads)..55
Figure 23 Storage Hierarchy (Writes)..56
Figure 24 Cache Characteristics..58
Figure 25 Basic Cache-State Transitions..62
Figure 26 Cache State Controls..63
Figure 27 Bus Snooping..64
Figure 28 Clocking Modes..68
Figure 29 Level-2 Cache Read and Write..72
Figure 30 Fastest Single-Qword Read..74
Figure 31 Single-Qword Read With Wait States..75
Figure 32 Fastest Single-Qword Write..76
Figure 33 Single-Qword Write With Wait States..76

Figure 34 Fastest Block Read 79
Figure 35 Fastest Block Write 80
Figure 36 Block Write With One Wait State........ 80
Figure 37 Interrupt-Acknowledge Cycle 82
Figure 38 Halt and Shutdown Encoding 82
Figure 39 Single-Qword Read Hits Modified Cache Block........ 86
Figure 40 Privilege Levels 90
Figure 41 Operand Sizes 91
Figure 42 Pointer Format 92
Figure 43 Examples of Aligned Operands........ 93
Figure 44 Complex Address Calculation (Protected Mode)........ 94
Figure 45 Address Translation (Protected Mode) 95
Figure 46 Address Translation........ 96
Figure 47 Segmented Memory 97
Figure 48 Memory Address Map 98
Figure 49 Stack Mechanism 99
Figure 50 I/O Address Map 100
Figure 51 Interrupts and Exceptions 102
Figure 52 Registers Accessible To Application Programs 103
Figure 53 General-Register Functions 105
Figure 54 EFLAGS Register 106
Figure 55 Instruction Set (By Functional Group)........ 112
Figure 56 instruction Prefixes........ 113
Figure 57 Instruction Format........ 114
Figure 58 Instruction Notation........ 116
Figure 59 instruction Latency and Encoding 126
Figure 60 Application Software Differences........ 128
Figure 61 System Software Differences 129
Figure 62 D.C. Characteristics........ 132
Figure 63 Nx586 Package Diagram........ 140
Figure 64 Nx586 Package Diagram (side)........ 142
Figure 65 Nx586 Package Diagram (bottom)........ 142
Figure 66 Nx587 Package Diagram (top) 143
Figure 67 Nx587 Package Diagram (side)........ 144
Figure 68 Nx587 Package Diagram (bottom)........ 145
Figure 69 Part Numbers and Descriptions 147
Figure 70 Part Number Syntax 147

Preface

This databook covers two chips: the Nx586 processor (called *the processor)*, and the Nx587 numerics processor. The book is written for system designers considering the use of NexGen chips in their designs. We assume an experienced audience, familiar not only with system design conventions but also with the x86 architecture. The *Glossary* at the end of the book defines NexGen's terminology, and the *Index* gives quick access to the subject matter.

NexGen's Applications Engineering Department welcomes your questions and will be glad to provide assistance. In particular, they can recommend system parts that have been tested and proven to work with NexGen products. They can be reached at:

NexGen Applications Engineering: (408) 435-0202

Notation

The following notation and conventions are used in this book:

Chip and Bus Names

- *Processor or CPU*—The Nx586 microprocessor chip described in this book.
- *Numerics Processor or NP*—The Nx587 floating-point unit chip described in this book.
- *NxPC*—The NxPC system controller chip described in the *NxPC System Controller Databook.*
- *NexBus*—The Nx586 processor bus, including its multiplexed address/status and data bus (NxAD<63:0>) and related control signals.

Signals and Timing Diagrams

- *Active-Low Signals*—Signal names that are followed by an asterisk, such as ALE*, indicate active-low signals. They are said to be "asserted" in their low-voltage state and "negated" in their high-voltage state.

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems.

- *Bus Signals*—In signal names, the notation *<n:m>* represents bits *n* through *m* of a bus.
- *Reserved Bits and Signals*—Signals or bus bits marked "reserved" must be driven inactive or left unconnected, as indicated in the signal descriptions. These bits and signals are reserved by NexGen for future implementations. When software reads registers with reserved bits, the reserved bits must be masked. When software writes such registers, it must first read the register and change only the non-reserved bits before writing back to the register.
- *Source*—In timing diagrams, the left-hand column indicates the "Source" of each signal. This is the chip or logic that output the signal. When signals are driven by multiple sources, all sources are shown, in the order in which they drive the signal. In some cases, signals take on different names as outputs are ORed in group-signal logic. In these cases, the signal source is shown with a subscript, where the subscript indicates the device or logic that originally caused the change in the signal.
- *Tri-state®*—In timing diagrams, signal ranges that are high impedance are shown as a straight horizontal line half-way between the high and low level.
- *Invalid and Don't Care*—In timing diagrams, signal ranges that are invalid or don't care are filled with a screen pattern.

Data

- *Quantities*—A *word* is two bytes (16 bits), a *dword* or doubleword is four bytes (32 bits), and a *qword* quadword is eight bytes (64 bits).
- *Addressing*—Memory is addressed as a series of bytes on eight-byte (64-bit) boundaries, in which each byte can be separately enabled.
- *Abbreviations*—The following notation is used for bits and bytes:

Bits	b	as in "64b/qword"
Bytes	B	as in "32B/block"
kilo	k	as in "4kB/page"
Mega	M	as in "1Mb/sec"
Giga	G	as in "4GB of memory space"

- *Little Endian Convention*—The byte with the address xx...xx00 is in the least-significant byte position (little end). In byte diagrams, bit positions are numbered from right to left: the little end is on the right and the big end is on the left. Data structure diagrams in memory show small addresses at the bottom and high addresses at the top. When data items are "aligned," bit notation on a 64-bit data bus maps directly to bit notation in 64-bit-wide memory. Because byte addresses increase from right to left, strings appear in reverse order when illustrated according the little-endian convention.

- *Bit Ranges*—In a range of bits, the highest and lowest bit numbers are separated by a colon, as in <63:0>.
- *Bit Values*—Bits can either be *set* to 1 or *cleared* to 0.
- *Hexadecimal and Binary Numbers*—Unless the context makes interpretation clear, hexadecimal numbers are followed by an *h*, binary numbers are followed by a *b*, and decimal numbers are followed by an *d*.

Related Publications

The following books treat various aspects of computer architecture, hardware design, and programming that may be useful for your understanding of NexGen products:

NexGen Products

- *Nx586 Microprocessor and Nx587 Numerics Processor Databook*, NexGen Microsystems, Milpitas, CA, Tel: (408) 435-0202.
- *NxPC System Controller Databook*, NexGen Microsystems, Milpitas, CA, Tel: (408) 435-0202.
- *NexBus Specification*, NexGen Microsystems, Milpitas, CA, Tel: (408) 435-0202.

x86 Architecture

- *i486™ Microprocessor Hardware Reference Manual*, Intel Corporation, Mt. Prospect, IL, Order # 240552. (800) 548-4725.
- *Intel486™ Microprocessor Family Programmer's Reference Manual*, Intel Corporation, Mt. Prospect, IL, Order # 240486. (800) 548-4725.
- John Crawford and Patrick Gelsinger, *Programming the 80386*, Sybex, San Francisco, 1987.
- Rakesh Agarwal, *80x86 Architecture & Programming*, Volumes I and II, Prentice-Hall, Englewood Cliffs, NJ, 1991.
- Walter A. Triebel, *The 80386DX Microprocessor*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
- Stephen Morse, Eric Isaacson, and Douglas Albert, *The 80386/387 Architecture*, John Wiley & Sons, New York, 1987.

General References

- John L. Hennessy and David A. Patterson, *Computer Architecture*, Morgan Kaufmann Publishers, San Mateo, CA, 1990.

Nx586 Features and Signals

- **x86 Binary Compatible**—All instructions that execute on the Intel i386™ and i387™ processors will execute on the Nx586 processor and Nx587 numerics processor. Full support for all operating modes.
- **Superscalar Pipeline and Control**—Multiple operations are executed simultaneously during each cycle.
- **Integrated Branch Prediction Cache**
- **Multi-Level Storage Hierarchy**—Branch prediction cache, readable write queue, on-chip L1 instruction and data caches, on-chip L2 cache control supporting 256kB or 1MB L2 cache with standard external SRAM.
- **Dual-Access (Superpipelined) On-Chip Caches**—64-bit reads and writes are serviced in parallel in a single clock.
- **Caches Decoupled From Bus**—All on-chip (level 1) and off-chip (level 2) cache is decoupled from the processor bus.
- **On-Chip Cache Control**—MESI modified write-once cache coherency protocol.
- **Burst Transfers**—32-byte cache blocks (lines) are fetched atomically.
- **66MHz to 100MHz Processor Clock Rates**—Based on NexBus clock rates of 33MHz to 50MHz.
- **Two-Phase, Non-Overlapped Clocking**—Integrated phase-locked loop bus-clock doubler. Processor operates at twice the bus frequency. On-chip instruction and data caches operate at twice the processor frequency.
- **Three 64-Bit Synchronous Buses**—NexBus (the processor bus), level-2 SRAM bus, and Nx587 numerics processor bus.
- **Multiprocessor Support**—Fast arbitration and cache coherency for multiple Nx586 and Nx587 processors on NexBus.

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems.

Figure 1 shows the signal organization on the Nx586 processor. The processor supports signals for the NexBus (the processor bus), level-2 cache, and the optional Nx587 numerics processor. Many types of devices can be interfaced to the NexBus, including a backplane, multiple Nx586 processors, shared memory subsystems, high-speed I/O, and industry-standard buses. The signals needed for cache-coherent multiprocessing are supported. All signals are synchronous to the NexBus clock (CLK) and transition at the rising edge of the clock, except four asynchronous signals: INTR*, NMI*, GATEA20, and SLOTID<3:0>. All bidirectional NexBus signals are floated unless they are needed during specific time periods, as specified in the *Bus Operation* chapter. The normal state for all reserved bits is high.

Three types of NexBus signals deserve special mention:

- *Buffered Address And Data Bus*—Address/status and data phases are multiplexed on the NxAD<63:0> bus. In multiprocessor systems, this bus is interfaced to NexBus devices through transceivers, for which control signals are provided on the processor. In single-processor systems using the NxPC chip, these transceivers are integrated into the NxPC chip.
- *Group Signals*—There are several ***group signals*** on the NexBus, typically denoted by signal names beginning with the letter "G." In systems with multiple devices on the NexBus, they provide fast control response. For example, active-low signals such as ALE* are driven by each NexBus device, and the backplane derives an active-high group OR (such as GALE) and distributes it back to each device. When the NxPC chip is used, these group signals are generated within the NxPC chip itself. Backplane logic is not needed.
- *Central Bus Arbitration*—Access to the NexBus is arbitrated by an external NexBus Arbiter. NexBus masters request and are granted access by this Arbiter. For the Nx586 processor, central bus arbitration has the advantage of back-to-back processor access most of the time while supporting fast switching between masters. For single-processor systems, the NxPC chip provides the combined functions of NexBus Arbiter, Alternate-Bus Interface (the system-logic interface to other system buses), and memory controller. The NxPC chip gives the processor back-to-back use of the bus when no device on any other system bus needs access.

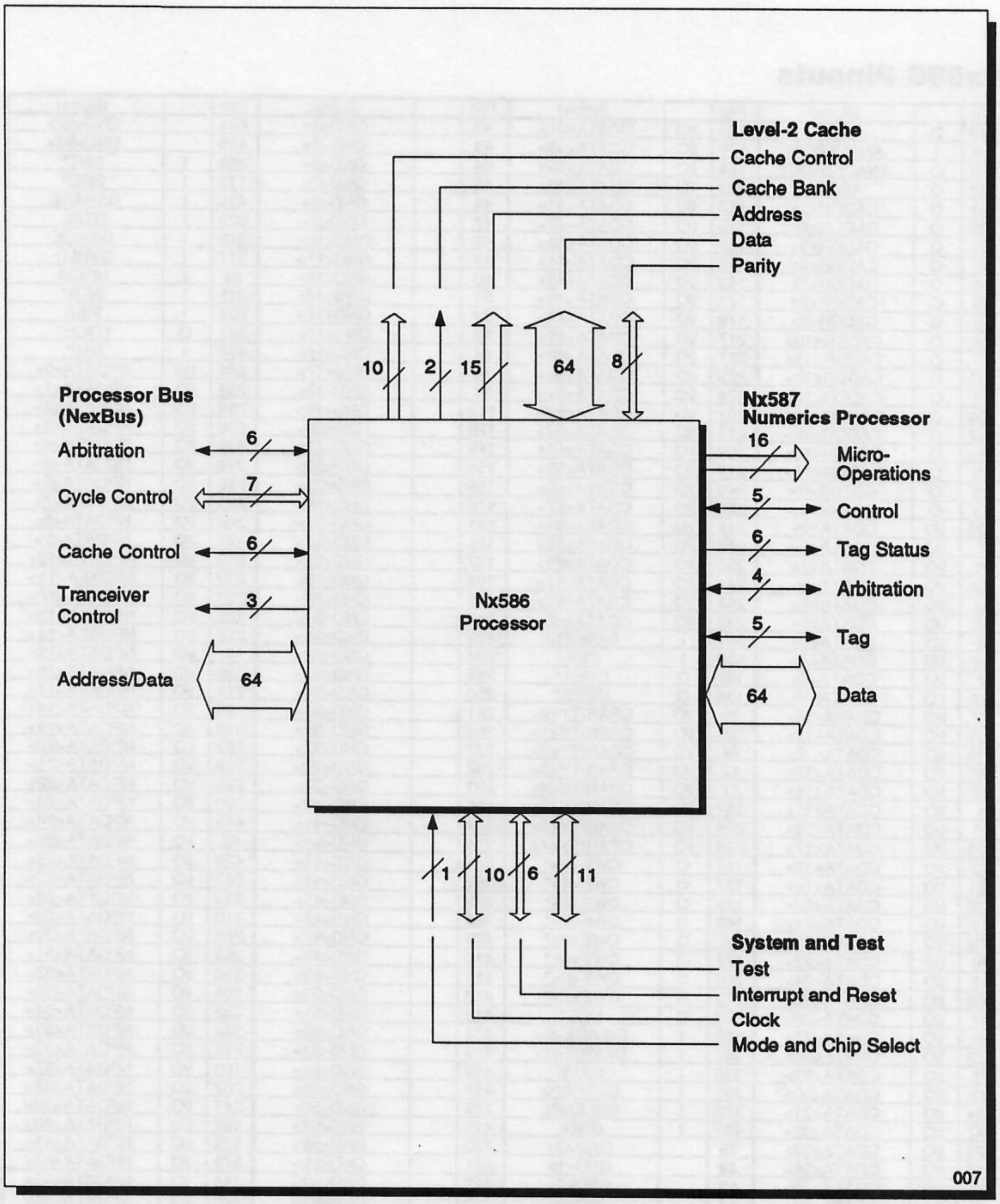


Figure 1 Nx586 Signal Organization

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems. NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

Nx586 Pinouts

Pin		Signal	Pin		Signal	Pin		Signal	Pin		Signal
449	O	ALE*	125	I/O	CDATA<37>	42		GND<4>	424		GND<62>
18	I	ANALYZEIN	176	I/O	CDATA<38>	43		GND<5>	425		GND<63>
168	O	ANALYZEOUT	184	I/O	CDATA<39>	44		GND<6>	368	I	GNT*
340	O	AREQ*	203	I/O	CDATA<40>	45		GND<7>	113	I	GREF
141	O	CADDR<3>	193	I/O	CDATA<41>	46		GND<8>	430	I	GSHARE
123	O	CADDR<4>	195	I/O	CDATA<42>	47		GND<9>	322	I	GTAL
124	O	CADDR<5>	185	I/O	CDATA<43>	48		GND<10>	349	I	GXACK
32	O	CADDR<6>	155	I/O	CDATA<44>	49		GND<11>	377	I	GXHLD
14	O	CADDR<7>	163	I/O	CDATA<45>	50		GND<12>	36	I	HROM
33	O	CADDR<8>	171	I/O	CDATA<46>	51		GND<13>	375	I	INTR*
15	O	CADDR<9>	179	I/O	CDATA<47>	52		GND<14>	323	I	IREF
34	O	CADDR<10>	217	I/O	CDATA<48>	53		GND<15>	341	O	LOCK*
90	O	CADDR<11>	227	I/O	CDATA<49>	54		GND<16>	436	I	NMI*
107	O	CADDR<12>	225	I/O	CDATA<50>	75		GND<17>	244	I/O	NPDATA<0>
88	O	CADDR<13>	224	I/O	CDATA<51>	91		GND<18>	254	I/O	NPDATA<1>
106	O	CADDR<14>	201	I/O	CDATA<52>	112		GND<19>	292	I/O	NPDATA<2>
142	O	CADDR<15>	211	I/O	CDATA<53>	128		GND<20>	408	I/O	NPDATA<3>
169	O	CADDR<16>	209	I/O	CDATA<54>	149		GND<21>	336	I/O	NPDATA<4>
35	O	CADDR<17>	219	I/O	CDATA<55>	154		GND<22>	294	I/O	NPDATA<5>
89	O	CBANK<0>	240	I/O	CDATA<56>	165		GND<23>	286	I/O	NPDATA<6>
16	O	CBANK<1>	251	I/O	CDATA<57>	170		GND<24>	223	I/O	NPDATA<7>
100	I/O	CDATA<0>	249	I/O	CDATA<58>	181		GND<25>	206	I/O	NPDATA<8>
7	I/O	CDATA<1>	248	I/O	CDATA<59>	186		GND<26>	427	I/O	NPDATA<9>
81	I/O	CDATA<2>	232	I/O	CDATA<60>	197		GND<27>	255	I/O	NPDATA<10>
136	I/O	CDATA<3>	241	I/O	CDATA<61>	202		GND<28>	230	I/O	NPDATA<11>
24	I/O	CDATA<4>	243	I/O	CDATA<62>	213		GND<29>	236	I/O	NPDATA<12>
80	I/O	CDATA<5>	233	I/O	CDATA<63>	218		GND<30>	183	I/O	NPDATA<13>
6	I/O	CDATA<6>	361	I	CKMODE	229		GND<31>	212	I/O	NPDATA<14>
99	I/O	CDATA<7>	452	I	CLK	234		GND<32>	191	I/O	NPDATA<15>
9	I/O	CDATA<8>	192	O	COEA*	245		GND<33>	390	I/O	NPDATA<16>
83	I/O	CDATA<9>	138	O	COEB*	250		GND<34>	215	I/O	NPDATA<17>
27	I/O	CDATA<10>	25	I/O	CPARITY<0>	261		GND<35>	199	I/O	NPDATA<18>
119	I/O	CDATA<11>	101	I/O	CPARITY<1>	266		GND<36>	318	I/O	NPDATA<19>
118	I/O	CDATA<12>	84	I/O	CPARITY<2>	277		GND<37>	262	I/O	NPDATA<20>
26	I/O	CDATA<13>	12	I/O	CPARITY<3>	282		GND<38>	228	I/O	NPDATA<21>
82	I/O	CDATA<14>	17	I/O	CPARITY<4>	293		GND<39>	295	I/O	NPDATA<22>
8	I/O	CDATA<15>	187	I/O	CPARITY<5>	298		GND<40>	260	I/O	NPDATA<23>
11	I/O	CDATA<16>	208	I/O	CPARITY<6>	309		GND<41>	445	I/O	NPDATA<24>
103	I/O	CDATA<17>	235	I/O	CPARITY<7>	314		GND<42>	95	I/O	NPDATA<25>
29	I/O	CDATA<18>	117	O	CWE<0>*	335		GND<43>	428	I/O	NPDATA<26>
121	I/O	CDATA<19>	137	O	CWE<1>*	351		GND<44>	220	I/O	NPDATA<27>
139	I/O	CDATA<20>	120	O	CWE<2>*	372		GND<45>	303	I/O	NPDATA<28>
28	I/O	CDATA<21>	140	O	CWE<3>*	388		GND<46>	310	I/O	NPDATA<29>
102	I/O	CDATA<22>	55	O	CWE<4>*	409		GND<47>	268	I/O	NPDATA<30>
10	I/O	CDATA<23>	177	O	CWE<5>*	410		GND<48>	263	I/O	NPDATA<31>
13	I/O	CDATA<24>	200	O	CWE<6>*	411		GND<49>	356	I/O	NPDATA<32>
105	I/O	CDATA<25>	216	O	CWE<7>*	412		GND<50>	196	I/O	NPDATA<33>
31	I/O	CDATA<26>	359	O	DCL*	413		GND<51>	302	I/O	NPDATA<34>
86	I/O	CDATA<27>	160	I	DEVICE<0>	414		GND<52>	300	I/O	NPDATA<35>
122	I/O	CDATA<28>	145	I	DEVICE<1>	415		GND<53>	287	I/O	NPDATA<36>
85	I/O	CDATA<29>	330	I	GALE	416		GND<54>	180	I/O	NPDATA<37>
30	I/O	CDATA<30>	339	I	GATEA20	417		GND<55>	207	I/O	NPDATA<38>
104	I/O	CDATA<31>	378	I	GBLKNBL	418		GND<56>	247	I/O	NPDATA<39>
147	I/O	CDATA<32>	429	I	GDCL	419		GND<57>	198	I/O	NPDATA<40>
129	I/O	CDATA<33>	38		GND<0>	420		GND<58>	308	I/O	NPDATA<41>
110	I/O	CDATA<34>	39		GND<1>	421		GND<59>	373	I/O	NPDATA<42>
92	I/O	CDATA<35>	40		GND<2>	422		GND<60>	246	I/O	NPDATA<43>
87	I/O	CDATA<36>	41		GND<3>	423		GND<61>	278	I/O	NPDATA<44>

Figure 2 Nx586 Pin List, By Signal Name

Pin		*Signal*	*Pin*		*Signal*	*Pin*		*Signal*	*Pin*		*Signal*
190	I/O	NPDATA<45>	114	I	NPTERM<1>	352	I/O	NxAD<51>	269		P4V<34>
338	I/O	NPDATA<46>	355	I	NPWREQ	343	I/O	NxAD<52>	274		P4V<35>
231	I/O	NPDATA<47>	319	I	NPWVAL	344	I/O	NxAD<53>	285		P4V<36>
276	I/O	NPDATA<48>	321	O	NREQ*	326	I/O	NxAD<54>	290		P4V<37>
279	I/O	NPDATA<49>	167	I	NSPARE<0>	457	I/O	NxAD<55>	301		P4V<38>
238	I/O	NPDATA<50>	150	I	NSPARE<1>	329	I/O	NxAD<56>	306		P4V<39>
271	I/O	NPDATA<51>	158	I	NSPARE<2>	328	I/O	NxAD<57>	317		P4V<40>
270	I/O	NPDATA<52>	296	I/O	NxAD<0>	365	I/O	NxAD<58>	332		P4V<41>
316	I/O	NPDATA<53>	267	I/O	NxAD<1>	439	I/O	NxAD<59>	354		P4V<42>
371	I/O	NPDATA<54>	307	I/O	NxAD<2>	364	I/O	NxAD<60>	369		P4V<43>
284	I/O	NPDATA<55>	297	I/O	NxAD<3>	456	I/O	NxAD<61>	391		P4V<44>
188	I/O	NPDATA<56>	443	I/O	NxAD<4>	363	I/O	NxAD<62>	392		P4V<45>
222	I/O	NPDATA<57>	444	I/O	NxAD<5>	381	I/O	NxAD<63>	393		P4V<46>
311	I/O	NPDATA<58>	463	I/O	NxAD<6>	73	O	NxADINUSE	394		P4V<47>
334	I/O	NPDATA<59>	312	I/O	NxAD<7>	433	I/O	NxADP<0>	395		P4V<48>
239	I/O	NPDATA<60>	313	I/O	NxAD<8>	451	I/O	NxADP<1>	396		P4V<49>
252	I/O	NPDATA<61>	315	I/O	NxAD<9>	432	I/O	NxADP<2>	397		P4V<50>
204	I/O	NPDATA<62>	281	I/O	NxAD<10>	376	I/O	NxADP<3>	398		P4V<51>
353	I/O	NPDATA<63>	283	I/O	NxAD<11>	450	I/O	NxADP<4>	399		P4V<52>
446	I	NPIRQ*	459	I/O	NxAD<12>	357	I/O	NxADP<5>	400		P4V<53>
337	I	NPNOERR	460	I/O	NxAD<13>	431	I/O	NxADP<6>	401		P4V<54>
172	O	NPOUTFTYP<0>	441	I/O	NxAD<14>	320	I/O	NxADP<7>	402		P4V<55>
98	O	NPOUTFTYP<1>	348	I/O	NxAD<15>	447	I	OWNABL	403		P4V<56>
79	O	NPPOPBUS<0>	387	I/O	NxAD<16>	76	I	P4REF	404		P4V<57>
116	O	NPPOPBUS<1>	370	I/O	NxAD<17>	57		P4V<0>	405		P4V<58>
3	O	NPPOPBUS<2>	331	I/O	NxAD<18>	58		P4V<1>	406		P4V<59>
93	O	NPPOPBUS<3>	333	I/O	NxAD<19>	59		P4V<2>	380	O	PARERR*
164	O	NPPOPBUS<4>	325	I/O	NxAD<20>	60		P4V<3>	453	I	PHE1
135	O	NPPOPBUS<5>	345	I/O	NxAD<21>	61		P4V<4>	379	I	PHE2
134	O	NPPOPBUS<6>	327	I/O	NxAD<22>	62		P4V<5>	153	I	POPHOLD
21	O	NPPOPBUS<7>	383	I/O	NxAD<23>	63		P4V<6>	214	I	RESET*
2	O	NPPOPBUS<8>	347	I/O	NxAD<24>	64		P4V<7>	362	I	RESETCPU*
97	O	NPPOPBUS<9>	384	I/O	NxAD<25>	65		P4V<8>	144	I	SERIALIN
148	O	NPPOPBUS<10>	458	I/O	NxAD<26>	66		P4V<9>	280	O	SERIALOUT
74	O	NPPOPBUS<11>	346	I/O	NxAD<27>	67		P4V<10>	448	O	SHARE*
22	O	NPPOPBUS<12>	438	I/O	NxAD<28>	68		P4V<11>	130	I	SLOTID<0>
156	O	NPPOPBUS<13>	382	I/O	NxAD<29>	69		P4V<12>	161	I	SLOTID<1>
23	O	NPPOPBUS<14>	437	I/O	NxAD<30>	70		P4V<13>	152	I	SLOTID<2>
96	O	NPPOPBUS<15>	455	I/O	NxAD<31>	71		P4V<14>	127	I	SLOTID<3>
37	O	NPPOPTAG<0>	259	I/O	NxAD<32>	72		P4V<15>	264	I/O	SSPARE<0>
159	O	NPPOPTAG<1>	257	I/O	NxAD<33>	94		P4V<16>	272	I/O	SSPARE<1>
56	O	NPPOPTAG<2>	265	I/O	NxAD<34>	109		P4V<17>	288	I/O	SSPARE<2>
132	O	NPPOPTAG<3>	275	I/O	NxAD<35>	131		P4V<18>	256	I/O	SSPARE<3>
151	O	NPPOPTAG<4>	273	I/O	NxAD<36>	146		P4V<19>	143	I/O	SSPARE<4>
182	O	NPRREQ	462	I/O	NxAD<37>	157		P4V<20>	374	I	TESTPWR*
174	O	NPRVAL	304	I/O	NxAD<38>	162		P4V<21>	108	I	TPH1
5	I/O	NPTAG<0>	426	I/O	NxAD<39>	173		P4V<22>	126	I	TPH2
77	I/O	NPTAG<1>	299	I/O	NxAD<40>	178		P4V<23>	324	I	VDDA
20	I/O	NPTAG<2>	289	I/O	NxAD<41>	189		P4V<24>	358	O	XACK*
111	I/O	NPTAG<3>	291	I/O	NxAD<42>	194		P4V<25>	386	O	XBCKE*
115	I/O	NPTAG<4>	305	I/O	NxAD<43>	205		P4V<26>	461	O	XBOE*
19	O	NPTAGSTAT<0>	440	I/O	NxAD<44>	210		P4V<27>	454	O	XHLD*
1	O	NPTAGSTAT<1>	366	I/O	NxAD<45>	221		P4V<28>	442	O	XNOE*
175	O	NPTAGSTAT<2>	367	I/O	NxAD<46>	226		P4V<29>	360	O	XPH1
78	O	NPTAGSTAT<3>	385	I/O	NxAD<47>	237		P4V<30>	342	O	XPH2
4	O	NPTAGSTAT<4>	407	I/O	NxAD<48>	242		P4V<31>	434	O	XREF
166	O	NPTAGSTAT<5>	389	I/O	NxAD<49>	253		P4V<32>	435	I	XSEL
133	I	NPTERM<0>	350	I/O	NxAD<50>	258		P4V<33>			

Figure 3 Nx586 Pin List, By Signal Name (continued)

Pin		*Signal*	*Pin*		*Signal*	*Pin*		*Signal*	*Pin*		*Signal*
1	O	NPTAGSTAT<1>	59		P4V<2>	117	O	CWE<0>*	175	O	NPTAGSTAT<2>
2	O	NPPOPBUS<8>	60		P4V<3>	118	I/O	CDATA<12>	176	I/O	CDATA<38>
3	O	NPPOPBUS<2>	61		P4V<4>	119	I/O	CDATA<11>	177	O	CWE<5>*
4	O	NPTAGSTAT<4>	62		P4V<5>	120	O	CWE<2>*	178		P4V<23>
5	I/O	NPTAG<0>	63		P4V<6>	121	I/O	CDATA<19>	179	I/O	CDATA<47>
6	I/O	CDATA<6>	64		P4V<7>	122	I/O	CDATA<28>	180	I/O	NPDATA<37>
7	I/O	CDATA<1>	65		P4V<8>	123	O	CADDR<4>	181		GND<25>
8	I/O	CDATA<15>	66		P4V<9>	124	O	CADDR<5>	182	O	NPRREQ
9	I/O	CDATA<8>	67		P4V<10>	125	I/O	CDATA<37>	183	I/O	NPDATA<13>
10	I/O	CDATA<23>	68		P4V<11>	126	I	TPH2	184	I/O	CDATA<39>
11	I/O	CDATA<16>	69		P4V<12>	127	I	SLOTID<3>	185	I/O	CDATA<43>
12	I/O	CPARITY<3>	70		P4V<13>	128		GND<20>	186		GND<26>
13	I/O	CDATA<24>	71		P4V<14>	129	I/O	CDATA<33>	187	I/O	CPARITY<5>
14	O	CADDR<7>	72		P4V<15>	130	I	SLOTID<0>	188	I/O	NPDATA<56>
15	O	CADDR<9>	73	O	NxADINUSE	131		P4V<18>	189		P4V<24>
16	O	CBANK<1>	74	O	NPPOPBUS<11>	132	O	NPPOPTAG<3>	190	I/O	NPDATA<45>
17	I/O	CPARITY<4>	75		GND<17>	133	I	NPTERM<0>	191	I/O	NPDATA<15>
18	I	ANALYZEIN	76	I	P4REF	134	O	NPPOPBUS<6>	192	O	COEA*
19	O	NPTAGSTAT<0>	77	I/O	NPTAG<1>	135	O	NPPOPBUS<5>	193	I/O	CDATA<41>
20	I/O	NPTAG<2>	78	O	NPTAGSTAT<3>	136	I/O	CDATA<3>	194		P4V<25>
21	O	NPPOPBUS<7>	79	O	NPPOPBUS<0>	137	O	CWE<1>*	195	I/O	CDATA<42>
22	O	NPPOPBUS<12>	80	I/O	CDATA<5>	138	O	COEB*	196	I/O	NPDATA<33>
23	O	NPPOPBUS<14>	81	I/O	CDATA<2>	139	I/O	CDATA<20>	197		GND<27>
24	I/O	CDATA<4>	82	I/O	CDATA<14>	140	O	CWE<3>*	198	I/O	NPDATA<40>
25	I/O	CPARITY<0>	83	I/O	CDATA<9>	141	O	CADDR<3>	199	I/O	NPDATA<18>
26	I/O	CDATA<13>	84	I/O	CPARITY<2>	142	O	CADDR<15>	200	O	CWE<6>*
27	I/O	CDATA<10>	85	I/O	CDATA<29>	143	I/O	SSPARE<4>	201	I/O	CDATA<52>
28	I/O	CDATA<21>	86	I/O	CDATA<27>	144	I	SERIALIN	202		GND<28>
29	I/O	CDATA<18>	87	I/O	CDATA<36>	145	I	DEVICE<1>	203	I/O	CDATA<40>
30	I/O	CDATA<30>	88	O	CADDR<13>	146		P4V<19>	204	I/O	NPDATA<62>
31	I/O	CDATA<26>	89	O	CBANK<0>	147	I/O	CDATA<32>	205		P4V<26>
32	O	CADDR<6>	90	O	CADDR<11>	148	O	NPPOPBUS<10>	206	I/O	NPDATA<8>
33	O	CADDR<8>	91		GND<18>	149		GND<21>	207	I/O	NPDATA<38>
34	O	CADDR<10>	92	I/O	CDATA<35>	150	I	NSPARE<1>	208	I/O	CPARITY<6>
35	O	CADDR<17>	93	O	NPPOPBUS<3>	151	O	NPPOPTAG<4>	209	I/O	CDATA<54>
36	I	HROM	94		P4V<16>	152	I	SLOTID<2>	210		P4V<27>
37	O	NPPOPTAG<0>	95	I/O	NPDATA<25>	153	I	POPHOLD	211	I/O	CDATA<53>
38		GND<0>	96	O	NPPOPBUS<15>	154		GND<22>	212	I/O	NPDATA<14>
39		GND<1>	97	O	NPPOPBUS<9>	155	I/O	CDATA<44>	213		GND<29>
40		GND<2>	98	O	NPOUTFTYP<1>	156	O	NPPOPBUS<13>	214	I	RESET*
41		GND<3>	99	I/O	CDATA<7>	157		P4V<20>	215	I/O	NPDATA<17>
42		GND<4>	100	I/O	CDATA<0>	158	I	NSPARE<2>	216	O	CWE<7>*
43		GND<5>	101	I/O	CPARITY<1>	159	O	NPPOPTAG<1>	217	I/O	CDATA<48>
44		GND<6>	102	I/O	CDATA<22>	160	I	DEVICE<0>	218		GND<30>
45		GND<7>	103	I/O	CDATA<17>	161	I	SLOTID<1>	219	I/O	CDATA<55>
46		GND<8>	104	I/O	CDATA<31>	162		P4V<21>	220	I/O	NPDATA<27>
47		GND<9>	105	I/O	CDATA<25>	163	I/O	CDATA<45>	221		P4V<28>
48		GND<10>	106	O	CADDR<14>	164	O	NPPOPBUS<4>	222	I/O	NPDATA<57>
49		GND<11>	107	O	CADDR<12>	165		GND<23>	223	I/O	NPDATA<7>
50		GND<12>	108	I	TPH1	166	O	NPTAGSTAT<5>	224	I/O	CDATA<51>
51		GND<13>	109		P4V<17>	167	I	NSPARE<0>	225	I/O	CDATA<50>
52		GND<14>	110	I/O	CDATA<34>	168	O	ANALYZEOUT	226		P4V<29>
53		GND<15>	111	I/O	NPTAG<3>	169	O	CADDR<16>	227	I/O	CDATA<49>
54		GND<16>	112		GND<19>	170		GND<24>	228	I/O	NPDATA<21>
55	O	CWE<4>*	113	I	GREF	171	I/O	CDATA<46>	229		GND<31>
56	O	NPPOPTAG<2>	114	I	NPTERM<1>	172	O	NPOUTFTYP<0>	230	I/O	NPDATA<11>
57		P4V<0>	115	I/O	NPTAG<4>	173		P4V<22>	231	I/O	NPDATA<47>
58		P4V<1>	116	O	NPPOPBUS<1>	174	O	NPRVAL	232	I/O	CDATA<60>

Figure 4 Nx586 Pin List, By Pin Number

Pin		Signal	Pin		Signal	Pin		Signal	Pin		Signal
233	I/O	CDATA<63>	291	I/O	NxAD<42>	349	I	GXACK	407	I/O	NxAD<48>
234		GND<32>	292	I/O	NPDATA<2>	350	I/O	NxAD<50>	408	I/O	NPDATA<3>
235	I/O	CPARITY<7>	293		GND<39>	351		GND<44>	409		GND<47>
236	I/O	NPDATA<12>	294	I/O	NPDATA<5>	352	I/O	NxAD<51>	410		GND<48>
237		P4V<30>	295	I/O	NPDATA<22>	353	I/O	NPDATA<63>	411		GND<49>
238	I/O	NPDATA<50>	296	I/O	NxAD<0>	354		P4V<42>	412		GND<50>
239	I/O	NPDATA<60>	297	I/O	NxAD<3>	355	I	NPWREQ	413		GND<51>
240	I/O	CDATA<56>	298		GND<40>	356	I/O	NPDATA<32>	414		GND<52>
241	I/O	CDATA<61>	299	I/O	NxAD<40>	357	I/O	NxADP<5>	415		GND<53>
242		P4V<31>	300	I/O	NPDATA<35>	358	O	XACK*	416		GND<54>
243	I/O	CDATA<62>	301		P4V<38>	359	O	DCL*	417		GND<55>
244	I/O	NPDATA<0>	302	I/O	NPDATA<34>	360	O	XPH1	418		GND<56>
245		GND<33>	303	I/O	NPDATA<28>	361	I	CKMODE	419		GND<57>
246	I/O	NPDATA<43>	304	I/O	NxAD<38>	362	I	RESETCPU*	420		GND<58>
247	I/O	NPDATA<39>	305	I/O	NxAD<43>	363	I/O	NxAD<62>	421		GND<59>
248	I/O	CDATA<59>	306		P4V<39>	364	I/O	NxAD<60>	422		GND<60>
249	I/O	CDATA<58>	307	I/O	NxAD<2>	365	I/O	NxAD<58>	423		GND<61>
250		GND<34>	308	I/O	NPDATA<41>	366	I/O	NxAD<45>	424		GND<62>
251	I/O	CDATA<57>	309		GND<41>	367	I/O	NxAD<46>	425		GND<63>
252	I/O	NPDATA<61>	310	I/O	NPDATA<29>	368	I	GNT*	426	I/O	NxAD<39>
253		P4V<32>	311	I/O	NPDATA<58>	369		P4V<43>	427	I/O	NPDATA<9>
254	I/O	NPDATA<1>	312	I/O	NxAD<7>	370	I/O	NxAD<17>	428	I/O	NPDATA<26>
255	I/O	NPDATA<10>	313	I/O	NxAD<8>	371	I/O	NPDATA<54>	429	I	GDCL
256	I/O	SSPARE<3>	314		GND<42>	372		GND<45>	430	I	GSHARE
257	I/O	NxAD<33>	315	I/O	NxAD<9>	373	I/O	NPDATA<42>	431	I/O	NxADP<6>
258		P4V<33>	316	I/O	NPDATA<53>	374	I	TESTPWR*	432	I/O	NxADP<2>
259	I/O	NxAD<32>	317		P4V<40>	375	I	INTR*	433	I/O	NxADP<0>
260	I/O	NPDATA<23>	318	I/O	NPDATA<19>	376	I/O	NxADP<3>	434	O	XREF
261		GND<35>	319	I	NPWVAL	377	I	GXHLD	435	I	XSEL
262	I/O	NPDATA<20>	320	I/O	NxADP<7>	378	I	GBLKNBL	436	I	NMI*
263	I/O	NPDATA<31>	321	O	NREQ*	379	I	PHE2	437	I/O	NxAD<30>
264	I/O	SSPARE<0>	322	I	GTAL	380	O	PARERR*	438	I/O	NxAD<28>
265	I/O	NxAD<34>	323	I	IREF	381	I/O	NxAD<63>	439	I/O	NxAD<59>
266		GND<36>	324	I	VDDA	382	I/O	NxAD<29>	440	I/O	NxAD<44>
267	I/O	NxAD<1>	325	I/O	NxAD<20>	383	I/O	NxAD<23>	441	I/O	NxAD<14>
268	I/O	NPDATA<30>	326	I/O	NxAD<54>	384	I/O	NxAD<25>	442	O	XNOE*
269		P4V<34>	327	I/O	NxAD<22>	385	I/O	NxAD<47>	443	I/O	NxAD<4>
270	I/O	NPDATA<52>	328	I/O	NxAD<57>	386	O	XBCKE*	444	I/O	NxAD<5>
271	I/O	NPDATA<51>	329	I/O	NxAD<56>	387	I/O	NxAD<16>	445	I/O	NPDATA<24>
272	I/O	SSPARE<1>	330	I	GALE	388		GND<46>	446	I	NPIRQ*
273	I/O	NxAD<36>	331	I/O	NxAD<18>	389	I/O	NxAD<49>	447	I	OWNABL
274		P4V<35>	332		P4V<41>	390	I/O	NPDATA<16>	448	O	SHARE*
275	I/O	NxAD<35>	333	I/O	NxAD<19>	391		P4V<44>	449	O	ALE*
276	I/O	NPDATA<48>	334	I/O	NPDATA<59>	392		P4V<45>	450	I/O	NxADP<4>
277		GND<37>	335		GND<43>	393		P4V<46>	451	I/O	NxADP<1>
278	I/O	NPDATA<44>	336	I/O	NPDATA<4>	394		P4V<47>	452	I	CLK
279	I/O	NPDATA<49>	337	I	NPNOERR	395		P4V<48>	453	I	PHE1
280	O	SERIALOUT	338	I/O	NPDATA<46>	396		P4V<49>	454	O	XHLD*
281	I/O	NxAD<10>	339	I	GATEA20	397		P4V<50>	455	I/O	NxAD<31>
282		GND<38>	340	O	AREQ*	398		P4V<51>	456	I/O	NxAD<61>
283	I/O	NxAD<11>	341	O	LOCK*	399		P4V<52>	457	I/O	NxAD<55>
284	I/O	NPDATA<55>	342	O	XPH2	400		P4V<53>	458	I/O	NxAD<26>
285		P4V<36>	343	I/O	NxAD<52>	401		P4V<54>	459	I/O	NxAD<12>
286	I/O	NPDATA<6>	344	I/O	NxAD<53>	402		P4V<55>	460	I/O	NxAD<13>
287	I/O	NPDATA<36>	345	I/O	NxAD<21>	403		P4V<56>	461	O	XBOE*
288	I/O	SSPARE<2>	346	I/O	NxAD<27>	404		P4V<57>	462	I/O	NxAD<37>
289	I/O	NxAD<41>	347	I/O	NxAD<24>	405		P4V<58>	463	I/O	NxAD<6>
290		P4V<37>	348	I/O	NxAD<15>	406		P4V<59>			

Figure 5 Nx586 Pin List, By Pin Number (continued)

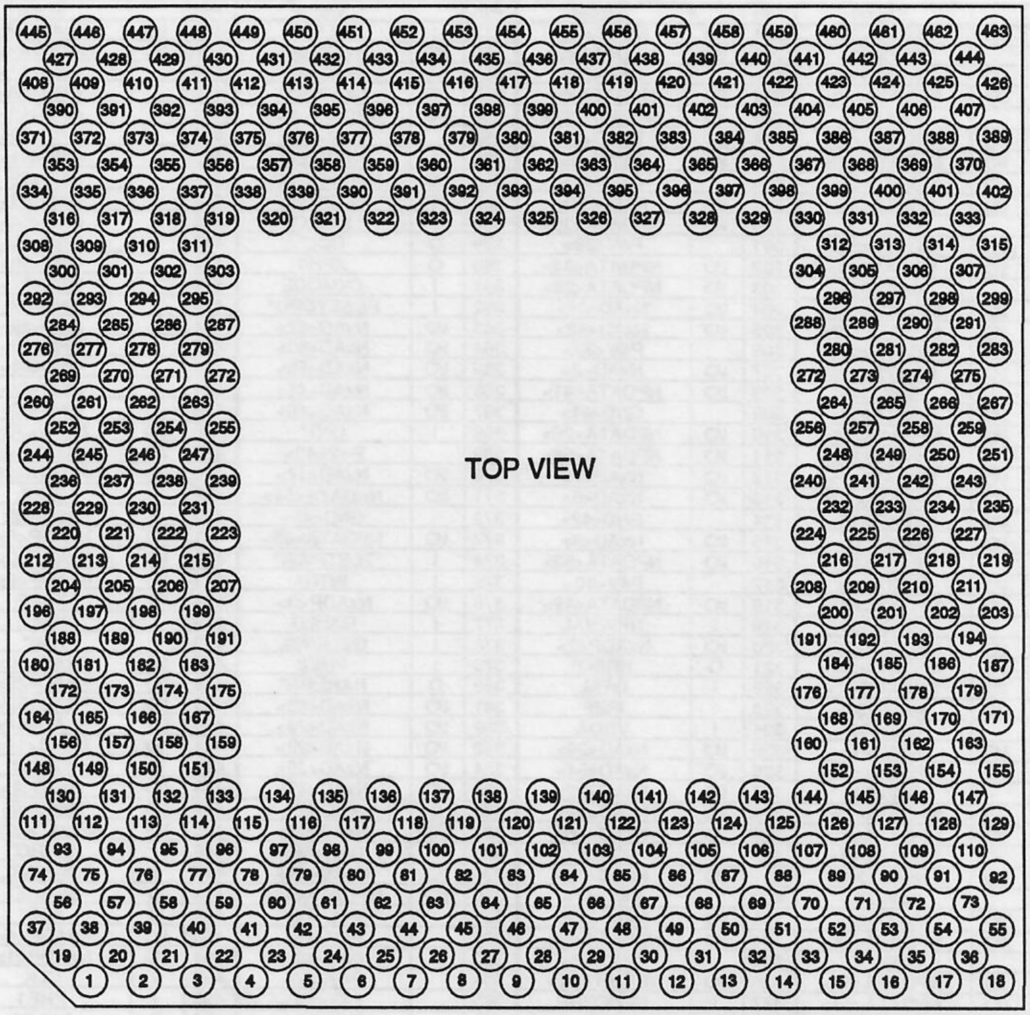


Figure 6 Nx586 Pinout Diagram (Top View)

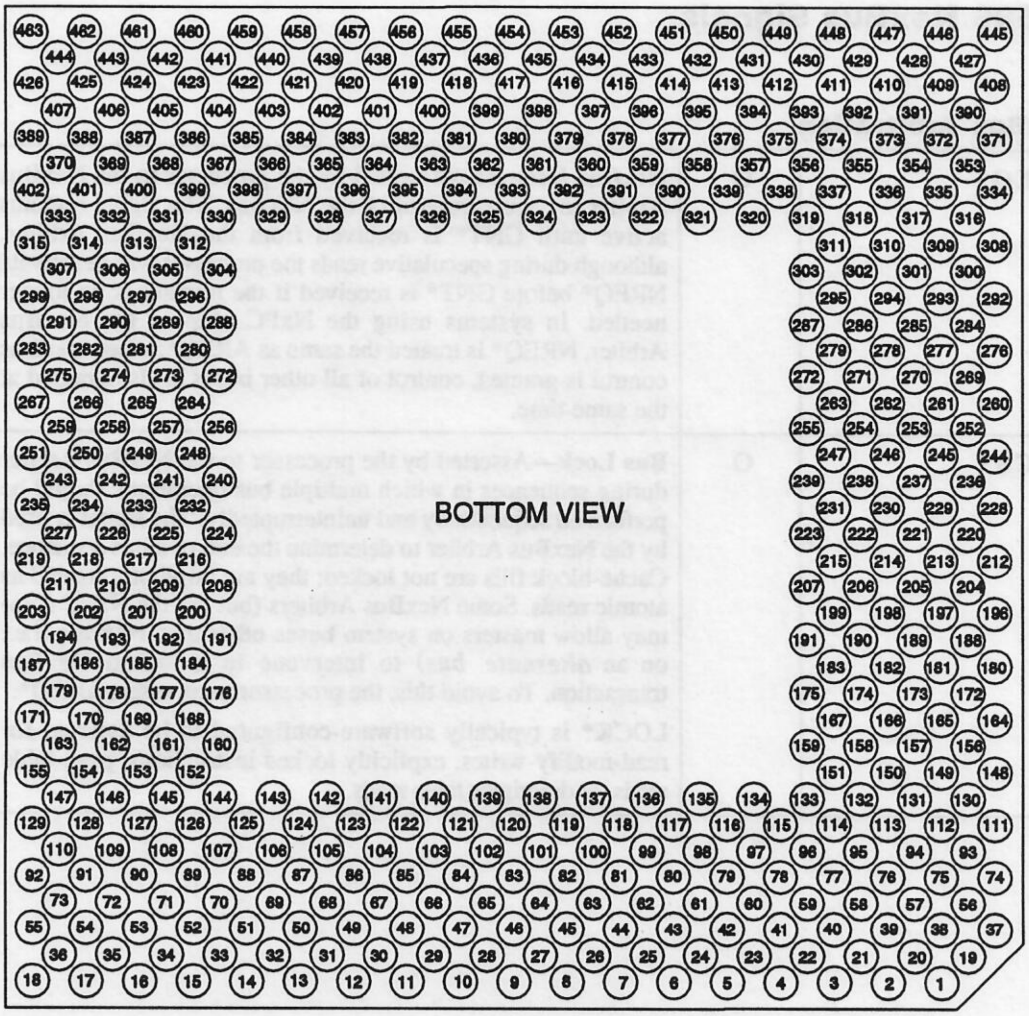


Figure 7 Nx586 Pinout Diagram (Bottom View)

Nx586 NexBus Signals

NexBus Arbitration

NREQ*	O	**NexBus Request**—Asserted by the processor to the NexBus Arbiter to secure control of the NexBus. The signal remains active until GNT* is received from the NexBus Arbiter, although during speculative reads the processor will deactivate NREQ* before GNT* is received if the transfer is no longer needed. In systems using the NxPC chip as the NexBus Arbiter, NREQ* is treated the same as AREQ*; when NexBus control is granted, control of all other buses is also granted at the same time.
LOCK*	O	**Bus Lock**—Asserted by the processor to the NexBus Arbiter during sequences in which multiple bus operations should be performed sequentially and uninterruptedly. The signal is used by the NexBus Arbiter to determine the end of a bus sequence. Cache-block fills are not locked; they are implicitly treated as atomic reads. Some NexBus Arbiters (but not the NxPC chip) may allow masters on system buses other than NexBus (i.e., on an *alternate bus)* to intervene in a locked NexBus transaction. To avoid this, the processor must assert AREQ*. LOCK* is typically software-configured to be asserted for read-modify-writes, explicitly locked instructions, page-table reads, or descriptor-table reads.

AREQ*	O	**Alternate-Bus Request**—Asserted by the processor to the NexBus Arbiter to secure control of the NexBus and of any other buses (called *alternate buses)* supported by the system. The signal remains active until GNT* is received from the NexBus Arbiter; unlike NREQ*, the processor does not make speculative requests with AREQ*. The NexBus Arbiter does not issue GNT* until the other system buses are available. If the processor does not know which bus its intended resource is on, it asserts NREQ*. If a GTAL is subsequently returned, the processor assumes the resources are on another system bus and it retries the transfer by asserting AREQ*. In systems using the NxPC chip as the NexBus Arbiter (shown in Figure 19), AREQ* and NREQ* have the same effect: either one causes the NxPC global bus arbiter to grant all buses to the winning requester at the end of the current bus cycle. In multiprocessor systems (shown in Figure 014), the assertion of AREQ* is usually the only method of ensuring uninterrupted access to a resource on the NexBus. It is more secure than asserting NREQ* and LOCK*, because NexBus masters have lower priority than masters on other system buses, which may interrupt even locked transfers by NexBus masters. In typical multiprocessor configurations, AREQ* is software-configured to be asserted for explicitly locked instructions, for page-table reads, or for descriptor-table reads. If AREQ* is not enabled to be asserted during references needing continuous bus access, NREQ* will be asserted instead.
GNT*	I	**Grant NexBus**—Asserted by the NexBus Arbiter to indicate that the processor has been granted control of the NexBus.
SLOTID<3:0>	I	**NexBus Slot ID**—These bits identify NexBus backplane slots. SLOTID 1111 (0Fh) is reserved for the system's primary processor. Normally, only the primary processor receives PC-compatible signals such as RESET*, RESETCPU*, INTR*, NMI*, and GATEA20, and this processor is responsible for initializing any secondary processors. SLOTID 0000 is reserved for the system logic that interfaces the NexBus to any other system buses (called the *alternate-bus interface).* (The NxPC chip acts as an Alternate-Bus Interface.) The signal is asynchronous to the NexBus clock.
DEVICE<1:0>	I	**Devices Per Slot**—These bits identify up to four devices to be connected to a single NexBus slot. In systems with only one device per slot, all bits must be tied high.

NexBus Cycle Control

ALE*	O	**Address Latch Enable**—Asserted by the processor to backplane logic or to the system-logic interface between the NexBus and any other system buses (called the *alternate-bus interface)* when the processor is driving valid address and status information on the NxAD<63:0> bus.
GALE	I	**Group Address Latch Enable**—Asserted by a backplane NAND of all ALE* signals, to indicate that the NexBus address and status can be latched. In single-processor systems using the NxPC chip, GALE is generated by the NxPC chip.
GTAL	I	**Group Try Again Later**—Asserted by the system-logic interface between the NexBus and other system buses (called the *alternate-bus interface)* to indicate that the attempted bus-crossing operation cannot be completed, because the system-logic bus interface is busy or cannot access the other system bus. In response, the processor aborts its current operation and attempts to re-try it by asserting AREQ*, thereby assuring that the processor will not receive a GNT* until the desired system bus is available. A bus-crossing operation can happen without the system-logic bus interface asserting GTAL and without the processor asserting AREQ*, if the other system bus and its system-logic interface are both available when the processor asserts NREQ*. The GTAL and AREQ* protocol is only used when NREQ* is asserted while either the other system bus or its system-logic interface is unavailable. The protocol prevents deadlocks and prevents the processor from staying on the NexBus until the other system bus becomes available. Unlike other group signals, which are the logical OR of a set of active-low signals generated by each participating device in the group, GTAL does not have such a corresponding active-low signal.
XACK*	O	**Transfer Acknowledge**—Asserted by the processor, as slave, to backplane logic or to the system-logic interface between the NexBus and other system buses (called the *alternate-bus interface)* to indicate to another NexBus master that the processor is prepared to respond as a slave to the current operation.

GXACK	I	**Group Transfer Acknowledge**—Asserted by a backplane NAND of all XACK* signals, to indicate that a NexBus device is prepared to respond as a slave to the processor's current operation. The system-logic interface between the NexBus and other system buses (called the *alternate-bus interface)* monitors the XACK* responses from all adapters. In systems using the NxPC chip as the Alternate-Bus Interface, when no XACK* response is forthcoming within three clocks, the NxPC chip asserts GXACK and initiates a *bus-crossing operation*. GXACK must be asserted for the transaction to continue. In general, since the system-logic interface to other system buses may take a variable number of cycles to respond to a GALE, the maximum time between assertion of GALE and the responding assertion of GXACK is not specified.
XHLD*	O	**Transfer Hold**—Asserted by the processor, as slave, to backplane logic or to the system-logic interface between the NexBus and other system buses (called the *alternate-bus interface)* in response to another NexBus master's request for data, when the processor is unable to respond on the next clock after GXACK.
GXHLD	I	**Group Transfer Hold**—Asserted by a backplane NAND of all XHLD* signals, to indicate that a slave cannot respond to the processor's request. GXHLD causes wait states to be inserted into the current operation. Both the master and the slave must monitor GXHLD to synchronize data transfers. During a bus-crossing read by the processor, the simultaneous assertion of GXACK and negation of GXHLD indicates that valid data is available on the bus. During a bus-crossing write, the same signal states indicate that data has been accepted by the slave.

NexBus Cache Control

DCL*	O	**Dirty Cache Line**—During reads by another NexBus master, this signal is asserted by the processor to indicate that the location being accessed is cached by the processor's level-2 cache in a *modified* (dirty) state. The requesting master's cycle is then aborted so that the processor, as an intervenor, can preemptively gain control of the NexBus and write back its modified data to main memory. While the data is being written to memory, the requesting master reads it off the NexBus. The assertion of DCL* is the only way in which atomic 32-byte cache-block fills by another NexBus master can be preempted by the processor for the purpose of writing back dirty data. During writes by another NexBus master, this signal is likewise asserted by the processor to indicate that it has a *modified* copy of the data. But in this case, the initiating master is allowed to finish its write to memory. The NexBus Arbiter must then guarantee that the processor asserting DCL* gains access to the bus in the very next arbitration grant, so that the processor can write back all of its modified data *except* the bytes written by the initiating master. (In this case, the initiating master's data is more recent than the data cached by the processor asserting DCL*.) The processor's level-2 cache can be software-configured for write-through rather than write-back operation. When this is done, DCL* is never asserted.
GDCL	I	**Group Dirty Cache Line**—Asserted by a backplane NAND of all DCL* signals, to indicate that a NexBus device has, in its cache, a *modified* copy of the data being accessed. During reads, when the processor is the bus master, the processor aborts its cycle so that the other caching device can write back its data; the processor reads the data on the fly. During writes, when the processor is the bus master, the processor finishes its write before the device asserting DCL* writes back all bytes *other than* those written by the processor.
GBLKNBL	I	**Group Block (Burst) Enable**—Asserted by a memory slave to enable burst transfers, and to indicate that the addressed space may be cached. Paged devices (such as video adapters) and any other devices that cannot support burst transfers or whose data is non-cacheable should negate this signal.

OWNABL	I	**Ownable**—Asserted by the system logic during accesses by the processor to locations that may be cached in the *exclusive* state. Negated during accesses that may only be cached in the *shared* state, such as bus-crossing accesses to an address space that cannot support the MESI cache-coherency protocol. All NexBus addresses are assumed to be cacheable in the *exclusive* state. The OWNABL signal is provided in case system logic needs to restrict caching to certain locations. In single-processor systems using the NxPC chip, that chip does not have an OWNABL signal and the processor's OWNABL input is typically tied high for write-back configurations to allow caching in the *exclusive* state on all reads.
SHARE*	O	**Shared Data**—Asserted by the processor during block reads by another NexBus master to indicate to the other master that its read hit in a block cached by the processor.
GSHARE	I	**Group Shared Data**—Asserted by a backplane NAND of all SHARE* signals, to indicate that the data being read must be cached in the *shared* state, if OWN* (NxAD<49>) is negated. However, if GSHARE and OWN* are both negated during the read, the data will be promoted to the *exclusive* state, since no other NexBus device has declared via SHARE* that it has cached a copy.

NexBus Transceivers

XBCKE*	O	**Transceiver BAD-Bus Clock Enable**—Asserted by the processor to enable the optional 29FCT52B NexBus transceivers to latch addresses and data onto the NxAD<63:0> bus for subsequent driving onto the AD<63:0> bus (see Figure 014). There is no comparable clock-enable for the NexBus side of these transceivers; they are always enabled on the NexBus side. In systems using the NxPC chip as the interface to other system buses, these NexBus transceivers are emulated within the NxPC chip, and this signal is tied to the same-named input on the NxPC chip.

XBOE*	O	**Transceiver-to-BAD-Bus Output Enable**—Asserted by the processor to enable the optional 29FCT52B NexBus transceivers to drive addresses and data onto the NxAD<63:0> bus from the AD<63:0> bus (see Figure 014). In systems using the NxPC chip as the interface to other system buses, these transceivers are emulated within the NxPC chip, and this signal is tied to the same-named input on the NxPC chip.
XNOE*	O	**Transceiver-to-NexBus Output Enable**—Asserted by the processor to enable the optional 29FCT52B NexBus transceivers to drive addresses and data onto the AD<63:0> bus from the NxAD<63:0> bus (see Figure 014).

NexBus Address and Data

NxAD<63:0>	I/O	**NexBus Address and Status, or Data**—This bus multiplexes address and status information during one phase, called the "address phase" or the "address and status phase" (see Figure 8), with up to 64 bits of data during a subsequent "data phase". The address and status phase occurs when GALE is asserted. At that time, NxAD<31:0> carries the address and NxAD<63:32> carries the status for a bus cycle. The meanings of these fields are detailed immediately below. The data phase occurs on the cycle after GXACK is asserted and GXHLD is simultaneously negated. To avoid contention, the two phases are separated by a guaranteed dead cycle (a minimum of one clock) which occurs between the assertion of GALE and the assertion of GXACK.
NxAD<1:0> *address phase*	I/O	**Reserved**—These bits must be driven high by the bus master.
NxAD<2> *address phase*	I/O	**ADRS<2> (Dword Address)**—For I/O cycles, this bit selects between the four-byte doublewords (dwords) in an eight-byte quadword (qword). For memory cycles, the bit is driven but the information is not normally used.

NxAD<31:3> *address phase*	I/O	**ADRS<31:3> (Qword Address)**—For memory cycles, these bits address an eight-byte quadword *(qword)* within the 4GB memory address space. For I/O cycles, NxAD<15:3> specifies a qword within the 64kB I/O address space and NxAD<31:16> are driven low by the processor. In either case, the addressed data may be further restricted by the BE<7:0>* bits on NxAD<39:32>. Memory cycles (but not I/O cycles) may be expanded to additional consecutive qwords by the BLKSIZ<1:0>* bits on NxAD<51:50>.
NxAD<39:32> *address phase*	I/O	**BE<7:0>* (Byte Enables)**—Byte-enable bits for the data phase of the NxAD<63:0> bus. BE<0>* corresponds to the byte on NxAD<7:0>, and BE<7>* corresponds to the byte on NxAD<63:56>. The meaning of these bytes is shown in Figure 9. For I/O cycles, BE<3:0>* specify the bytes to be transferred on NxAD<31:0> and BE<7:4>* are driven high by the processor. For memory cycles, all eight bits are used to specify the bytes to be transferred on NxAD<63:0>. For burst transfers, these bits have meaning only for the first qword of the transfer, and only for four-qword block transfers (see Figure 7). In four-qword write transfers, the rest of the qwords have implicit byte-enable bits of all 0's, so that all bytes are transferred in each of the subsequent qwords. In four-qword read transfers (used for cache block fills), the byte-enable bits specify the bytes that the processor needs immediately; a non-cacheable slave may force a single-qword transfer and return only those bytes for which byte-enables are asserted.

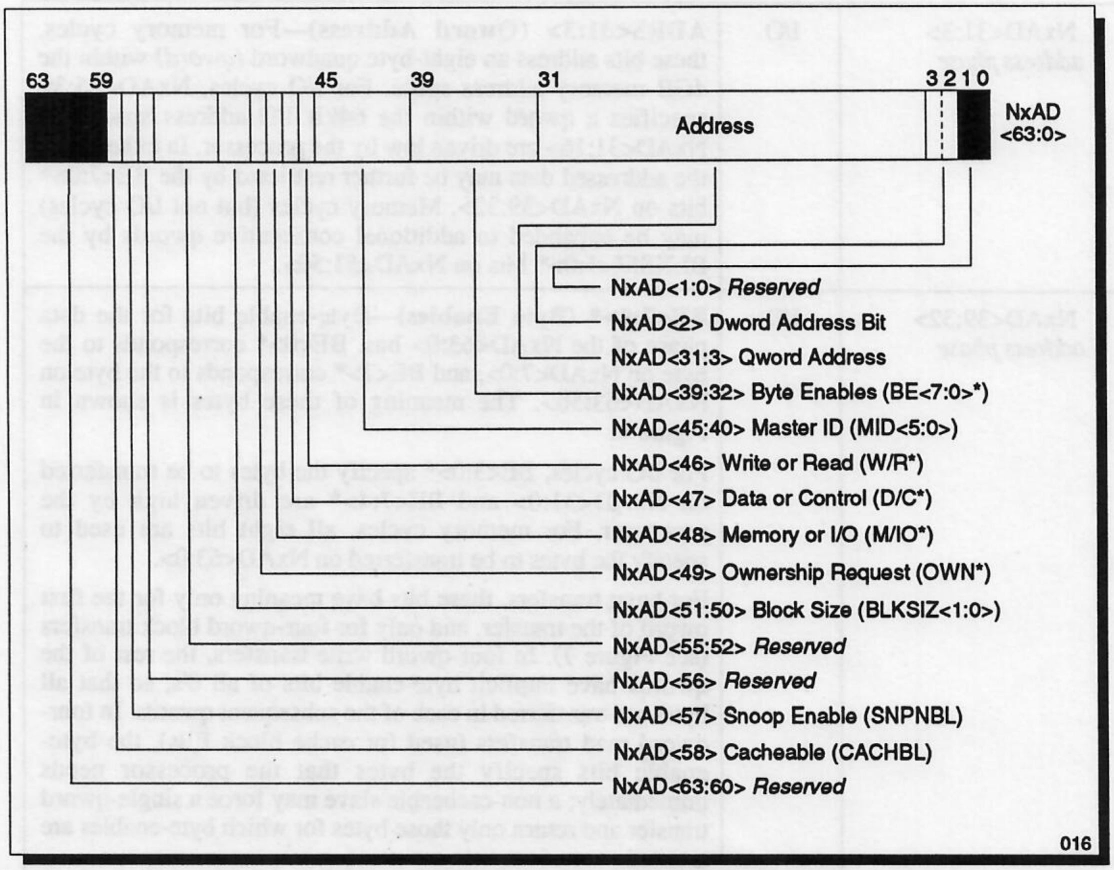

63
59
45
39
31
3 2 1 0
Address
NxAD
<63:0>
NxAD<1:0> Reserved
NxAD<2> Dword Address Bit
NxAD<31:3> Qword Address
NxAD<39:32> Byte Enables (BE<7:0>*)
NxAD<45:40> Master ID (MID<5:0>)
NxAD<46> Write or Read (W/R*)
NxAD<47> Data or Control (D/C*)
NxAD<48> Memory or I/O (M/IO*)
NxAD<49> Ownership Request (OWN*)
NxAD<51:50> Block Size (BLKSIZ<1:0>)
NxAD<55:52> Reserved
NxAD<56> Reserved
NxAD<57> Snoop Enable (SNPNBL)
NxAD<58> Cacheable (CACHBL)
NxAD<63:60> Reserved
016

Figure 8 NexBus Address and Status Phase

Transfer Type		*Meaning of BE<7:0>**
I/O		BE<3:0>* specify the bytes to transfer on NxAD<31:0>. BE<7:4>* are driven high by the processor.
Memory	Single Qword Read or Write	BE<7:0>* specify the bytes to transfer on NxAD<63:0>.
	Four-Qword Block Write	BE<7:0>* specify the bytes to transfer on NxAD<63:0> for first qword only. For all other qwords, BE<3:0>* are implicit zeros. and all bytes are transferred.
	Four-Qword Block Read (Cache-Block Fill)	BE<7:0>* specify the bytes that are to be fetched immediately.
Compare Figure 11.		

Figure 9 Byte-Enable Usage

NxAD<45:40> *address phase*	I/O	**MID<5:0> (Master ID)**—These bits indicate to a slave, and to the system-logic interface between the NexBus and other system buses (called the *alternate-bus interface)* during bus-crossing cycles, the identity of the NexBus master that initiated the cycle. The most-significant four bits are the device's SLOTID<3:0> bits. The least-significant two bits are the device's DEVICE<1:0> bits. In systems using the NxPC chip as the interface to other system buses, MID 000000 is reserved for the NxPC chip.
NxAD<46> *address phase*	I/O	**W/R* (Write or Read*)**—This bit usually distinguishes between read and write operations on the NexBus. See Figure 10. The bus-cycle encoding is the same as the Intel i486 processor's encoding.
NxAD<47> *address phase*	I/O	**D/C* (Data or Code*)**—This bit usually distinguishes between data and code operations on the NexBus. See Figure 10. The bus-cycle encoding is the same as the Intel i486 processor's encoding.
NxAD<48> *address phase*	I/O	**M/IO* (Memory or I/O*)**—This bit usually distinguishes between memory and I/O operations on the NexBus. See Figure 10. The bus-cycle encoding is the same as the i486 processor's encoding.

NxAD<48> *M/IO**	*NxAD<47>* *D/C**	*NxAD<46>* *W/R**	*Type of Bus Cycle*
0	0	0	Interrupt Acknowledge
0	0	1	Halt or Shutdown wr
0	1	0	I/O Data Read
0	1	1	I/O Data Write
1	0	0	Memory Code Read
1	0	1	*(reserved)*
1	1	0	Memory Data Read
1	1	1	Memory Data Write

Figure 10 Bus-Cycle Types

NxAD<49> *address phase*	I/O	**OWN* (Ownership Request)**—Asserted by a master when it intends to cache data in the *exclusive* state. The bit is asserted during write-misses or read-modify-write misses. If such an operation hits in the cache of another master, that master writes back (if copy is modified) and changes the state of its copy to *invalid*. If OWN* is negated during a read or write, this implies that SHARE* is asserted by the same master, so other masters can change their copies to *shared.*
NxAD<51:50> *address phase*	I/O	**BLKSIZ<1:0>* (Block Size)**—For memory operations, these bits define the number of qwords to be transferred. For I/O operations, these bits are driven high by the processor. Figure 11 shows the block sizes. For single-qword transfers and burst writes, the bytes to be transferred in the first qword are specified by the byte-enable bits, BE<7:0>* on NxAD<39:32>. If the slave is incapable of transferring more than a single qword, it or the system-logic interface between the NexBus and other system buses (called the *alternate-bus interface)* may deny a request for subsequent qwords by negating the GXACK or GBLKNBL inputs to the processor after a single-qword transfer, or after returning all bytes specified by BE<7:0>* in the first qword.

Block Length	*BLKSIZ<1>** *NxAD<51>*	*BLKSIZ<0>** *NxAD<50>*	*Cacheable?*
Single Qword **(0 to 8 bytes)**	**1**	**1**	**no**
Four-Qword Block	**0**	**1**	**yes**
Reserved	**1**	**0**	***reserved***
Reserved	**0**	**0**	***reserved***

Figure 11 Block Transfer Sizes

NxAD<55:52> *address phase*	I/O	**Reserved**—These bits must be driven high.
NxAD<56> *address phase*	I/O	**Reserved**—These bits must be driven high.
NxAD<57> *address phase*	I/O	**SNPNBL (Snoop Enable)**—Asserted to indicate that the current operation affects memory that may be present in other caches. When this signal is negated, snooping devices need not look up the addressed data in their cache tags.
NxAD<58> *address phase*	I/O	**CACHBL (Cacheable)**—Asserted by the bus master to indicate that it may cache a copy of the addressed data. The master typically decides what it will cache, based on software-configured address ranges. The bit supports higher-performance designs by letting the NexBus interface know what the master intends to do with the data, thereby allowing other devices to sometimes prevent unnecessary invalidations or write-backs.
NxAD<63:59> *address phase*	I/O	**Reserved**—These bits must be driven high by the bus master.

Nx586 Level-2 Cache Signals

COEA*	O	**L2 Cache Output Enable A**—Enables reading from four second-level cache SRAMs to drive the CDATA<63:0> bus. Standard asynchronous static RAMs are used for this cache.
COEB*	O	**L2 Cache Output Enable B**—Enables reading from the other four second-level cache SRAMs to drive the CDATA<63:0> bus.
CWE<7:0>*	O	**L2 Cache Write Enable**—Enables writing to the second-level cache SRAMs. The CWE<0>* bit enables writing the byte on CDATA<7:0>. The CWE<7>* bit enables writing the byte on CDATA<63:56>.
CBANK<1:0>	O	**L2 Cache Bank**—Selects one of four banks (sets) in the four-way set associative second-level cache. Each bank is either 64kB or 256kB. These signals are connected to the two least-significant address bits of the SRAM chips.
CADDR<17:3>	O	**L2 Cache Address**—The address of an eight-byte quantity in the second-level cache bank selected by CBANK<1:0>. Bits 17:16 are not used for a 256kB L2 cache; they are only used for a 1MB cache.
CDATA<63:0>	I/O	**L2 Cache Data**—Carries either one to eight bytes of second-level cache data, or the tags and state bits for one to four second-level cache banks (sets). Transfers on this bus occur at the peak rate of eight bytes every two processor clocks, but the transfers can begin on any processor clock.

Numerics-Processor Bus Signals (on Nx586)

NPPOPBUS<15:0>	O	**Numerics Processor Micro-Operations Bus**—Driven by the Nx586 processor to the Nx587 numerics processor to provide a floating-point micro-operation at the peak rate of one per processor clock. The NPPOPBUS<15:0> bus carries both micro-operations and their associated tags, both of which are issued by the Nx586 processor's Decode Unit.
NPNOERR	I	**Numerics Processor No Error**—Asserted by the Nx587 numerics processor to the Nx586 processor for handshaking to implement the IBM-compatible mode of interrupt handling. This signal is enabled and disabled in software. The signal must be pulled up.
NPOUTFTYP<1:0>	O	**Numerics Processor Output Type**—Asserted by the Nx586 processor to the Nx587 numerics processor for handshaking to implement the IBM-compatible mode of interrupt handling. These signals are enabled and disabled in software.
NPTERM<1:0>	I	**Numerics Processor Termination**—Asserted by the Nx587 numerics processor to the Nx586 processor to indicate completion of floating-point operations. The signal must be pulled up.
NPTAGSTAT<5:0>	O	**Numerics Processor Tag Status**—Driven by the Nx586 processor to the Nx587 numerics processor to synchronize the issuing, retiring, and aborting of instructions.
NPRREQ	O	**Numerics Processor Read Request**—Asserted by the Nx586 processor to the Nx587 numerics processor, to request use of the NPDATA<63:0> and NPTAG<4:0> buses to transfer data on the next clock. The NPRREQ signal has priority over the NPWREQ signal. When neither is requesting, the processor drives the bus. The processor sometimes makes speculative requests, such as when it concurrently does cache lookups for the data to be transferred. If the processor finds that it cannot use the bus after requesting it, it negates NPRVAL when the bus is granted, otherwise it asserts NPRVAL and transfers the data in the same clock.

NPRVAL	O	**Numerics Processor Read Valid**—Asserted by the Nx586 processor to the Nx587 numerics processor in the clock following a successful request, to indicate that the data being transferred on the NPDATA<63:0> bus in the current clock is valid.
NPWREQ	I	**Numerics Processor Write Request**—Asserted by the Nx587 numerics processor to the Nx586 processor, to request control of the NPDATA<63:0> and NPTAG<4:0> buses to transfer data on the next clock. The NPRREQ signal has priority over the NPWREQ signal. The signal must be pulled down. The numerics processor makes speculative requests concurrently with its first pass at formatting the output. If it discovers that more formatting is needed, it negates NPWVAL when the NPDATA<63:0> bus is granted, otherwise it asserts NPWVAL and transfers the data in the same clock.
NPWVAL	I	**Numerics Processor Write Valid**—Asserted by the Nx587 numerics processor to the Nx586 processor in the clock following a successful request, to indicate that the data being transferred on the NPDATA<63:0> bus in the current clock is valid. The signal must be pulled down.
NPTAG<4:0>	I/O	**Numerics Processor Tag Bus**—On each processor clock, this bus carries the five-bit micro-operation tag between the Nx586 processor and the Nx587 numerics processor. The tag identifies the instruction from which the micro-operation was decoded, and it corresponds to the data being transferred on the NPDATA<63:0> bus.
NPDATA<63:0>	I/O	**Numerics Processor Data**—On each processor clock, this bus carries up to 64 bits of read or write data between the Nx586 processor and the Nx587 numerics processor. The Nx586 processor uses it to provide read data to the Nx587 numerics processor, and the Nx587 numerics processor uses it to write results. The bus's bi-directionality is implemented with arbitration among the NPRREQ and NPWREQ signals. Arbitration priority is given to the processor, hence reads prevail over writes. The winner gets the bus on the next clock. The arbitration and the bus transfer are pipelined one clock apart at the processor-clock frequency. Thus, in every clock, both a request and a transfer are made.

Nx586 System Signals

Nx586 Clock

CLK	I	**NexBus Clock**—A TTL-level clock running at a frequency between 33MHz and 50MHz. The duty cycle is roughly 45% to 55%. All signals on NexBus transition on the rising edge of CLK, except the asynchronous signals, INTR*, NMI*, GATEA20, and SLOTID<3:0>. The processor's internal phase-locked loop (PLL) synchronizes internal processor clocks at twice the frequency of the CLK reference. Static operation (0MHz) is possible in non-PLL clock modes; for details, contact NexGen Applications Engineering.
PHE1	I	**Clock Phase 1**—For normal clocking operation, this signal should be tied high.
PHE2	I	**Clock Phase 2**—For normal clocking operation, this signal should be tied low.
CKMODE	I	**Clock Mode**—For normal clocking operation, this signal should be tied high.
XSEL	I	**Clock Mode Select**—For normal clocking operation, this signal should be tied low.
XPH1	O	**Processor Clock Phase 1**—For normal clocking operation, this signal must be left unconnected.
XPH2	O	**Processor Clock Phase 2**—For normal clocking operation, this signal must be left unconnected.
IREF	I	**Clock Input Reference**—For normal clocking operation, this signal must be tied low.
XREF	O	**Clock Output Reference**—For normal clocking operation, this signal must be left unconnected.
VDDA	I	**PLL Analog Power**—See the *Electrical Data* chapter for decoupling requirements.

Nx586 Interrupts and Reset

INTR*	I	**Maskable Interrupt**—If not masked by software, this signal is asserted by an interrupt controller. The processor responds by stopping its current flow of instructions at the next instruction boundary, abort earlier instructions that have been partially executed, and perform an interrupt acknowledge sequence, as described in the *Bus Operations* chapter. The signal is asynchronous to the processor and to the NexBus clock.
NMI*	I	**Non-Maskable Interrupt**—Asserted by system logic. The effect of this signal is similar to INTR*, except that NMI* cannot be masked by software, the interrupt acknowledge sequence is not performed, and the handler is always located by interrupt vector 2 in the interrupt descriptor table. The signal is asynchronous to the processor and to the NexBus clock.
RESET*	I	**Global Reset (Power-Up Reset)**—Asserted by system logic. The processor responds by resetting its internal state machines and loading default values in its registers. At power-up it must remain asserted for a minimum of several milliseconds to stabilize the phase-locked loop. See the *Electrical Data* chapter.
RESETCPU*	I	**Reset CPU (Soft Reset)**—Asserted by the system-logic interface between the NexBus and other system buses (called the *alternate-bus interface)* to reset the processor without changing the state of memory or the processor's caches. The signal is normally routed only to the primary processor in SLOTID 0Fh; on other slots, the signal is normally tied high.
GATEA20	I	**Gate Address 20**—When asserted by the system controller or keyboard controller, the processor drives bit 20 of the physical address at its current value. When negated, address bit 20 is cleared to zero, causing the address to wrap around into a 20-bit address space. GATEA20 is asynchronous to the NexBus clock. The method replicates the 8086 processor's handling of address wraparound. All physical addresses are affected by the ANDing of GATEA20 with address bit 20, including cached addresses. The signal is asynchronous to the processor's internal clock and to the NexBus clock (CLK).

Nx586 Test and Reserved Signals

ANALYZEIN	I	**Reserved**—This signal must be grounded for normal operation.
ANALYZEOUT	O	**Reserved**—This signal must be left unconnected for normal operation.
CPARITY<7:0>	I/O	**Reserved**—These signals must be left unconnected.
GREF	O	**Ground Reference**—This analog reference must be left unconnected for normal operation.
HROM	I	**Reserved**—This signal must be tied low.
NPIRQ*	O	**Reserved**—This signal must be connected to the same-named signal on the Nx587 numerics processor, if the latter chip is used. Otherwise, the signal must be pulled up.
NPPOPTAG<4:0>	I/O	**Reserved**—These signals must be connected to the same-named signals on the Nx587 numerics processor, if the latter chip is used. Otherwise, the signals must be left unconnected.
NSPARE<2:0>	I	**Reserved**—These signals must be left unconnected.
NxADINUSE	O	**Reserved**—This signal must be left unconnected for normal operation.
NxADP<7:0>	I/O	**Reserved**—These signals must be left unconnected.
P4REF	O	**Power Reference**—This analog reference must be left unconnected for normal operation.
PARERR*	O	**Reserved**—These signals must be left unconnected.
POPHOLD	I	**Reserved**—This signal must be grounded for normal operation.
SERIALIN	I	**Serial In**—The input of the scan-test chain. This signal must be tied low for normal operation.
SERIALOUT	O	**Serial Out**—The output of the scan-test chain. This signal must be left unconnected for normal operation.
SSPARE<4:0>	I/O	**Reserved**—These signals must be connected between the Nx586 processor and the NxPC system controller, if the latter chip is used. Otherwise, the signals must be grounded.

TESTPWR*	I	**Test Power**—Powers-down circuits that use static power during scan tests. This signal must be tied high for normal operation.
TPH1	I	**Test Phase 1 Clock**—For scan test support. This signal must be tied low for normal operation.
TPH2	I	**Test Phase 2 Clock**—For scan test support. This signal must be tied low for normal operation.

Nx586 Alphabetical Signal Summary

ALE*	O	Address Latch Enable
ANALYZEIN	I	Analyze In
ANALYZEOUT	O	Analyze Out
AREQ*	O	Alternate-Bus Request
CADDR<17:3>	O	L2 Cache Address
CBANK<1:0>	O	L2 Cache Bank
CDATA<63:0>	I/O	L2 Cache Data
CKMODE	I	Clock Mode
CLK	I	NexBus Clock
COEA*	O	L2 Cache Output Enable A
COEB*	O	L2 Cache Output Enable B
CPARITY<7:0>	I/O	*Reserved*
CWE<7:0>*	O	L2 Cache Write Enable
DCL*	O	Dirty Cache Line
DEVICE<1:0>	I	Devices Per Slot
GALE	I	Group Address Latch Enable
GATEA20	I	Gate Address 20
GBLKNBL	I	Group Block (Burst) Enable
GDCL	I	Group Dirty Cache Line
GNT*	I	Grant NexBus
GREF	I	Ground Reference
GSHARE	I	Group Shared Data
GTAL	I	Group Try Again Later
GXACK	I	Group Transfer Acknowledge
GXHLD	I	Group Transfer Hold
HROM	I	*Reserved*
INTR*	I	Maskable Interrupt
IREF	I	Clock Input Reference
LOCK*	O	Bus Lock
NMI*	I	Non-Maskable Interrupt
NPDATA<63:0>	I/O	Numerics Processor Data
NPIRQ*	O	*Reserved*
NPNOERR	I	Numerics Processor No Error
NPOUTFTYP<1:0>	O	Numerics Processor Output Type
NPPOPBUS<15:0>	O	Numerics Processor Micro-Operations Bus

NPPOPTAG<4:0>	I/O	*Reserved*
NPRREQ	O	Numerics Processor Read Request
NPRVAL	O	Numerics Processor Read Valid
NPTAG<4:0>	I/O	Numerics Processor Tag Bus
NPTAGSTAT<5:0>	O	Numerics Processor Tag Status
NPTERM<1:0>	I	Numerics Processor Termination
NPWREQ	I	Numerics Processor Write Request
NPWVAL	I	Numerics Processor Write Valid
NREQ*	O	NexBus Request
NSPARE<2:0>	I	*Reserved*
NxAD<63:0>	I/O	Bus Address/Status, or Bus Data
NxAD<1:0> address phase	I/O	*Reserved*
NxAD<2> address phase	I/O	ADRS<2> (Dword Address Bit)
NxAD<31:3> address phase	I/O	ADRS<31:3> (Qword Address Bits)
NxAD<39:32> address phase	I/O	BE<7:0>* (Byte-Enable Bits)
NxAD<45:40> address phase	I/O	MID<5:0> (Master ID)
NxAD<46> address phase	I/O	W/R* (Write or Read*)
NxAD<47> address phase	I/O	D/C* (Data or Code*)
NxAD<48> address phase	I/O	M/IO* (Memory or I/O*)
NxAD<49> address phase	I/O	OWN* (Ownership Request)
NxAD<51:50> address phase	I/O	BLKSIZ<1:0>* (Block Size)
NxAD<55:52> address phase	I/O	*Reserved*
NxAD<56> address phase	I/O	*Reserved*
NxAD<57> address phase	I/O	SNPNBL (Snoop Enable)
NxAD<58> address phase	I/O	CACHBL (Cacheable)
NxAD<63:59> address phase	I/O	*Reserved*

NxADINUSE	O	*Reserved*
NxADP<7:0>	I/O	*Reserved*
OWNABL	I	Ownable
P4REF	O	Power Reference
PARERR*	O	*Reserved*
PHE1	I	Clock Phase 1
PHE2	I	Clock Phase 2
POPHOLD	I	Processor-Operation Hold
RESET*	I	Global Reset (Power-Up Reset)
RESETCPU*	I	Reset CPU (Soft Reset)
SERIALIN	I	Serial In
SERIALOUT	O	Serial Out
SHARE*	O	Shared Data
SLOTID<3:0>	I	NexBus Slot ID
SSPARE<4:0>	I/O	*Reserved*
TESTPWR*	I	Test Power
TPH1	I	Test Phase 1 Clock
TPH2	I	Test Phase 2 Clock
VDDA	I	PLL Analog Power
XACK*	O	Transfer Acknowledge
XBCKE*	O	NexBus-Transceiver Clock Enable
XBOE*	O	NexBus-Transceiver Output Enable
XHLD*	O	Transfer Hold
XNOE*	O	NexBus-Transceiver Output Enable
XPH1	O	Processor Clock Phase 1
XPH2	O	Processor Clock Phase 2
XREF	O	Clock Output Reference
XSEL	I	Clock Mode Select

Figure 12 Nx586 Alphabetical Signal Summary

Nx587 Features and Signals

- **x87 Compatible**—Runs all x87-architecture floating-point binary code. Supports all IEEE 754 formats.
- **Dedicated 64-Bit Processor Bus**—Fast, synchronous, non-multiplexed interface to Nx586 processor.
- **High Bus Bandwidth**—Speculative requests and simple arbitration on the Nx586-Nx587 bus maximize bandwidth. Arbitration and data transfers occur in parallel, one clock apart.
- **Fully Integrated Into Nx586 Pipeline**—Works in parallel with the Nx586 Decode, Address, and Integer Units.

Figure 13 shows the signal organization on the Nx587 numerics processor. These include signals shared with the Nx586 processor, system signals (including an interrupt request signal, NPIRQ*, to an external interrupt controller), and test signals. The signals shared with the Nx586 processor operate at the processor-clock frequency and have the same functionality as those on the processor, but with reverse directionality. The normal state for all reserved bits is high.

The bi-directionality of the bus between the processor and numerics processor is implemented with a simple arbitration method involving the NPRREQ and NPWREQ signals. Arbitration priority is given to the processor, hence reads prevail over writes. The winner gets the bus on the next clock. The arbitration and data transfers are pipelined one clock apart at the processor-clock frequency. Thus, in every clock, both a bus request and a data transfer are made.

Both the processor and the numerics processor sometimes make speculative requests for the bus. For example, the processor will make a speculative bus request when it concurrently does cache lookups for the data to be transferred. If the processor finds that it cannot use the bus after requesting it, it negates its read-valid signal, NPRVAL, when the bus is granted. The numerics processor makes speculative requests concurrently with its first pass at formatting the output. If it discovers that more formatting is needed, it negates its write-valid signal, NPWVAL, when the bus is granted.

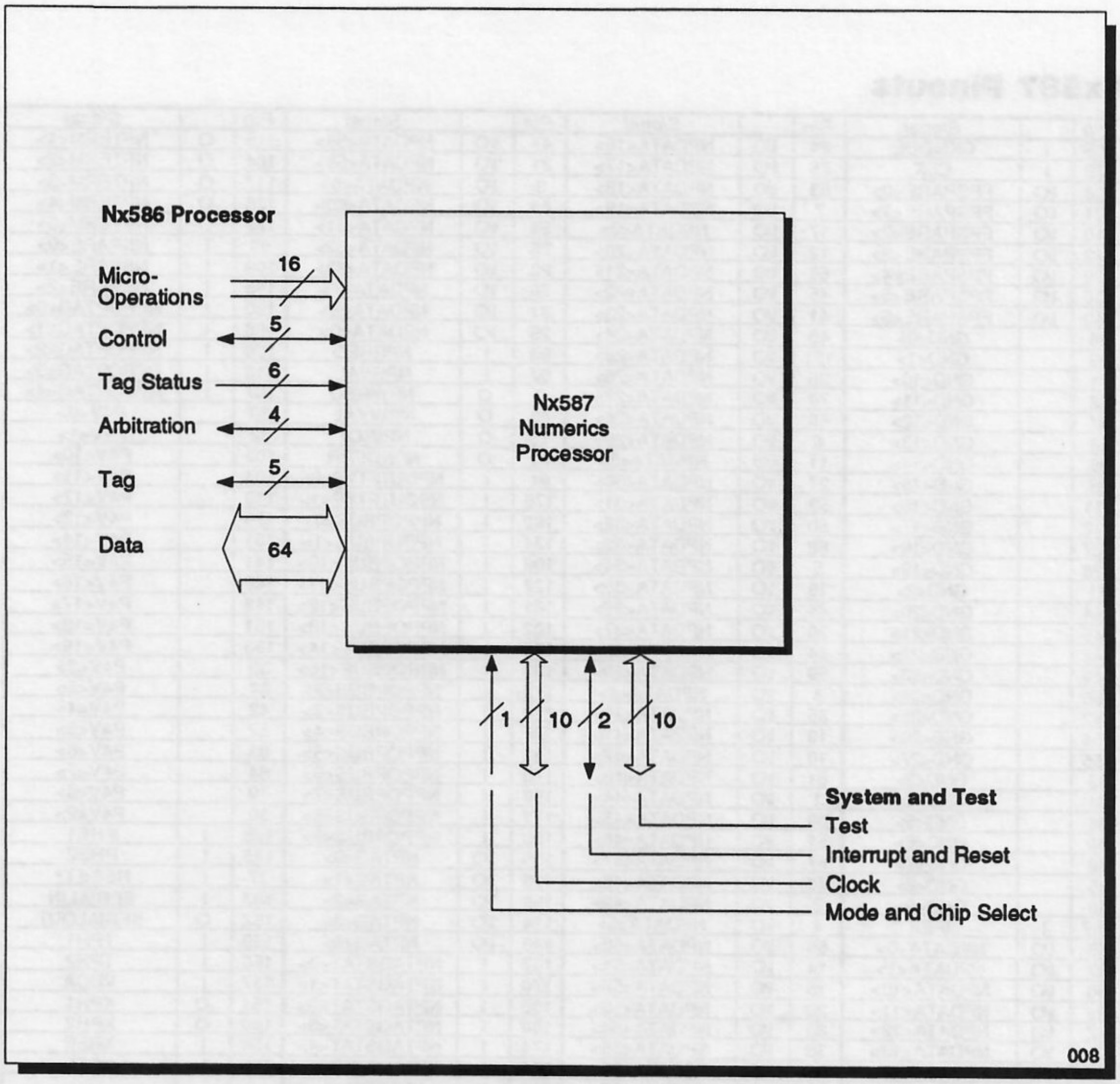


Figure 13 Nx587 Signal Organization

Nx587 Pinouts

Pin		*Signal*	*Pin*		*Signal*	*Pin*		*Signal*	*Pin*		*Signal*
165	I	CKMODE	24	I/O	NPDATA<16>	47	I/O	NPDATA<58>	115	O	NPTERM<1>
179	I	CLK	75	I/O	NPDATA<17>	21	I/O	NPDATA<59>	164	O	NPTERM<2>
168	I/O	FPSPARE<0>	83	I/O	NPDATA<18>	3	I/O	NPDATA<6>	117	O	NPTERM<3>
124	I/O	FPSPARE<1>	7	I/O	NPDATA<19>	68	I/O	NPDATA<60>	125	O	NPTERM<4>
110	I/O	FPSPARE<2>	17	I/O	NPDATA<2>	54	I/O	NPDATA<61>	177	O	NPTERM<5>
142	I/O	FPSPARE<3>	12	I/O	NPDATA<20>	78	I/O	NPDATA<62>	99	I	NSPARE<0>
129	I/O	FPSPARE<4>	66	I/O	NPDATA<21>	22	I/O	NPDATA<63>	109	I	NSPARE<1>
136	I/O	FPSPARE<5>	45	I/O	NPDATA<22>	76	I/O	NPDATA<7>	105	I	NSPARE<2>
140	I/O	FPSPARE<6>	41	I/O	NPDATA<23>	81	I/O	NPDATA<8>	130	I	NPPOPTAG<0>
28		GND<0>	40	I/O	NPDATA<24>	25	I/O	NPDATA<9>	108	I	NPPOPTAG<1>
29		GND<1>	121	I/O	NPDATA<25>	93	I	NPRREQ	126	I	NPPOPTAG<2>
71		GND<10>	26	I/O	NPDATA<26>	97	I	NPRVAL	113	I	NPPOPTAG<3>
72		GND<11>	70	I/O	NPDATA<27>	9	O	NPWREQ	107	I	NPPOPTAG<4>
87		GND<12>	46	I/O	NPDATA<28>	48	O	NPWVAL	30		P4V<0>
88		GND<13>	6	I/O	NPDATA<29>	13	O	NPIRQ*	32		P4V<1>
95		GND<14>	11	I/O	NPDATA<3>	49	O	NPNOERR	103		P4V<10>
96		GND<15>	27	I/O	NPDATA<30>	94	I	NPOUTFTYP<0>	104		P4V<11>
111		GND<16>	53	I/O	NPDATA<31>	176	I	NPOUTFTYP<1>	119		P4V<12>
112		GND<17>	50	I/O	NPDATA<32>	162	I	NPPOPBUS<0>	120		P4V<13>
127		GND<18>	82	I/O	NPDATA<33>	134	I	NPPOPBUS<1>	131		P4V<14>
128		GND<19>	5	I/O	NPDATA<34>	106	I	NPPOPBUS<10>	141		P4V<15>
31		GND<2>	18	I/O	NPDATA<35>	122	I	NPPOPBUS<11>	146		P4V<16>
144		GND<20>	44	I/O	NPDATA<36>	161	I	NPPOPBUS<12>	148		P4V<17>
145		GND<21>	90	I/O	NPDATA<37>	102	I	NPPOPBUS<13>	151		P4V<18>
147		GND<22>	84	I/O	NPDATA<38>	175	I	NPPOPBUS<14>	153		P4V<19>
149		GND<23>	59	I/O	NPDATA<39>	132	I	NPPOPBUS<15>	35		P4V<2>
150		GND<24>	8	I/O	NPDATA<4>	172	I	NPPOPBUS<2>	37		P4V<3>
152		GND<25>	85	I/O	NPDATA<40>	118	I	NPPOPBUS<3>	42		P4V<4>
154		GND<26>	19	I/O	NPDATA<41>	98	I	NPPOPBUS<4>	52		P4V<5>
155		GND<27>	10	I/O	NPDATA<42>	163	I	NPPOPBUS<5>	63		P4V<6>
33		GND<3>	61	I/O	NPDATA<43>	173	I	NPPOPBUS<6>	64		P4V<7>
34		GND<4>	1	I/O	NPDATA<44>	159	I	NPPOPBUS<7>	79		P4V<8>
36		GND<5>	89	I/O	NPDATA<45>	171	I	NPPOPBUS<8>	80		P4V<9>
38		GND<6>	51	I/O	NPDATA<46>	133	I	NPPOPBUS<9>	166	I	PHE1
39		GND<7>	67	I/O	NPDATA<47>	135	I/O	NPTAG<0>	138	I	PHE2
55		GND<8>	43	I/O	NPDATA<48>	123	I/O	NPTAG<1>	77	I	RESET*
56		GND<9>	2	I/O	NPDATA<49>	158	I/O	NPTAG<2>	183	I	SERIALIN
167	I	IREF	4	I/O	NPDATA<5>	114	I/O	NPTAG<3>	182	O	SERIALOUT
58	I/O	NPDATA<0>	65	I/O	NPDATA<50>	143	I/O	NPTAG<4>	169	I	TPH1
57	I/O	NPDATA<1>	14	I/O	NPDATA<51>	157	I	NPTAGSTAT<0>	156	I	TPH2
60	I/O	NPDATA<10>	15	I/O	NPDATA<52>	170	I	NPTAGSTAT<1>	137	I	VDDA
69	I/O	NPDATA<11>	20	I/O	NPDATA<53>	100	I	NPTAGSTAT<2>	181	O	XPH1
62	I/O	NPDATA<12>	23	I/O	NPDATA<54>	160	I	NPTAGSTAT<3>	180	O	XPH2
91	I/O	NPDATA<13>	16	I/O	NPDATA<55>	174	I	NPTAGSTAT<4>	139	I	XREF
74	I/O	NPDATA<14>	86	I/O	NPDATA<56>	101	I	NPTAGSTAT<5>	178	I	XSEL
92	I/O	NPDATA<15>	73	I/O	NPDATA<57>	116	O	NPTERM<0>			

Figure 14 Nx587 Pin List (By Signal Name)

Pin		*Signal*	*Pin*		*Signal*	*Pin*		*Signal*	*Pin*		*Signal*
1	I/O	NPDATA<44>	47	I/O	NPDATA<58>	93	I	NPRREQ	139	I	XREF
2	I/O	NPDATA<49>	48	O	NPWVAL	94	I	NPOUTFTYP<0>	140	I/O	FPSPARE<6>
3	I/O	NPDATA<6>	49	O	NPNOERR	95		GND<14>	141		P4V<15>
4	I/O	NPDATA<5>	50	I/O	NPDATA<32>	96		GND<15>	142	I/O	FPSPARE<3>
5	I/O	NPDATA<34>	51	I/O	NPDATA<46>	97	I	NPRVAL	143	I/O	NPTAG<4>
6	I/O	NPDATA<29>	52		P4V<5>	98	I	NPPOPBUS<4>	144		GND<20>
7	I/O	NPDATA<19>	53	I/O	NPDATA<31>	99	I	NSPARE<0>	145		GND<21>
8	I/O	NPDATA<4>	54	I/O	NPDATA<61>	100	I	NPTAGSTAT<2>	146		P4V<16>
9	O	NPWREQ	55		GND<8>	101	I	NPTAGSTAT<5>	147		GND<22>
10	I/O	NPDATA<42>	56		GND<9>	102	I	NPPOPBUS<13>	148		P4V<17>
11	I/O	NPDATA<3>	57	I/O	NPDATA<1>	103		P4V<10>	149		GND<23>
12	I/O	NPDATA<20>	58	I/O	NPDATA<0>	104		P4V<11>	150		GND<24>
13	O	NPIRQ*	59	I/O	NPDATA<39>	105	I	NSPARE<2>	151		P4V<18>
14	I/O	NPDATA<51>	60	I/O	NPDATA<10>	106	I	NPPOPBUS<10>	152		GND<25>
15	I/O	NPDATA<52>	61	I/O	NPDATA<43>	107	I	NPPOPTAG<4>	153		P4V<19>
16	I/O	NPDATA<55>	62	I/O	NPDATA<12>	108	I	NPPOPTAG<1>	154		GND<26>
17	I/O	NPDATA<2>	63		P4V<6>	109	I	NSPARE<1>	155		GND<27>
18	I/O	NPDATA<35>	64		P4V<7>	110	I/O	FPSPARE<2>	156	I	TPH2
19	I/O	NPDATA<41>	65	I/O	NPDATA<50>	111		GND<16>	157	I	NPTAGSTAT<0>
20	I/O	NPDATA<53>	66	I/O	NPDATA<21>	112		GND<17>	158	I/O	NPTAG<2>
21	I/O	NPDATA<59>	67	I/O	NPDATA<47>	113	I	NPPOPTAG<3>	159	I	NPPOPBUS<7>
22	I/O	NPDATA<63>	68	I/O	NPDATA<60>	114	I/O	NPTAG<3>	160	I	NPTAGSTAT<3>
23	I/O	NPDATA<54>	69	I/O	NPDATA<11>	115	O	NPTERM<1>	161	I	NPPOPBUS<12>
24	I/O	NPDATA<16>	70	I/O	NPDATA<27>	116	O	NPTERM<0>	162	I	NPPOPBUS<0>
25	I/O	NPDATA<9>	71		GND<10>	117	O	NPTERM<3>	163	I	NPPOPBUS<5>
26	I/O	NPDATA<26>	72		GND<11>	118	I	NPPOPBUS<3>	164	O	NPTERM<2>
27	I/O	NPDATA<30>	73	I/O	NPDATA<57>	119		P4V<12>	165	I	CKMODE
28		GND<0>	74	I/O	NPDATA<14>	120		P4V<13>	166	I	PHE1
29		GND<1>	75	I/O	NPDATA<17>	121	I/O	NPDATA<25>	167	I	IREF
30		P4V<0>	76	I/O	NPDATA<7>	122	I	NPPOPBUS<11>	168	I/O	FPSPARE<0>
31		GND<2>	77	I	RESET*	123	I/O	NPTAG<1>	169	I	TPH1
32		P4V<1>	78	I/O	NPDATA<62>	124	I/O	FPSPARE<1>	170	I	NPTAGSTAT<1>
33		GND<3>	79		P4V<8>	125	O	NPTERM<4>	171	I	NPPOPBUS<8>
34		GND<4>	80		P4V<9>	126	I	NPPOPTAG<2>	172	I	NPPOPBUS<2>
35		P4V<2>	81	I/O	NPDATA<8>	127		GND<18>	173	I	NPPOPBUS<6>
36		GND<5>	82	I/O	NPDATA<33>	128		GND<19>	174	I	NPTAGSTAT<4>
37		P4V<3>	83	I/O	NPDATA<18>	129	I/O	FPSPARE<4>	175	I	NPPOPBUS<14>
38		GND<6>	84	I/O	NPDATA<38>	130	I	NPPOPTAG<0>	176	I	NPOUTFTYP<1>
39		GND<7>	85	I/O	NPDATA<40>	131		P4V<14>	177	O	NPTERM<5>
40	I/O	NPDATA<24>	86	I/O	NPDATA<56>	132	I	NPPOPBUS<15>	178	I	XSEL
41	I/O	NPDATA<23>	87		GND<12>	133	I	NPPOPBUS<9>	179	I	CLK
42		P4V<4>	88		GND<13>	134	I	NPPOPBUS<1>	180	O	XPH2
43	I/O	NPDATA<48>	89	I/O	NPDATA<45>	135	I/O	NPTAG<0>	181	O	XPH1
44	I/O	NPDATA<36>	90	I/O	NPDATA<37>	136	I/O	FPSPARE<5>	182	O	SERIALOUT
45	I/O	NPDATA<22>	91	I/O	NPDATA<13>	137	I	VDDA	183	I	SERIALIN
46	I/O	NPDATA<28>	92	I/O	NPDATA<15>	138	I	PHE2			

Figure 15 Nx587 Pin List (By Pin Number)

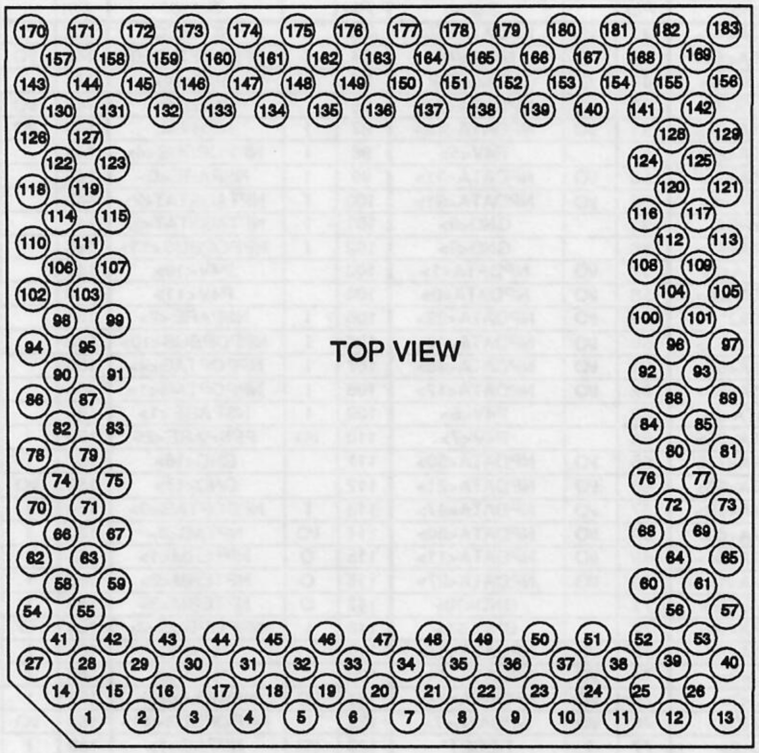


Figure 16 Nx587 Pinout Diagram (Top View)

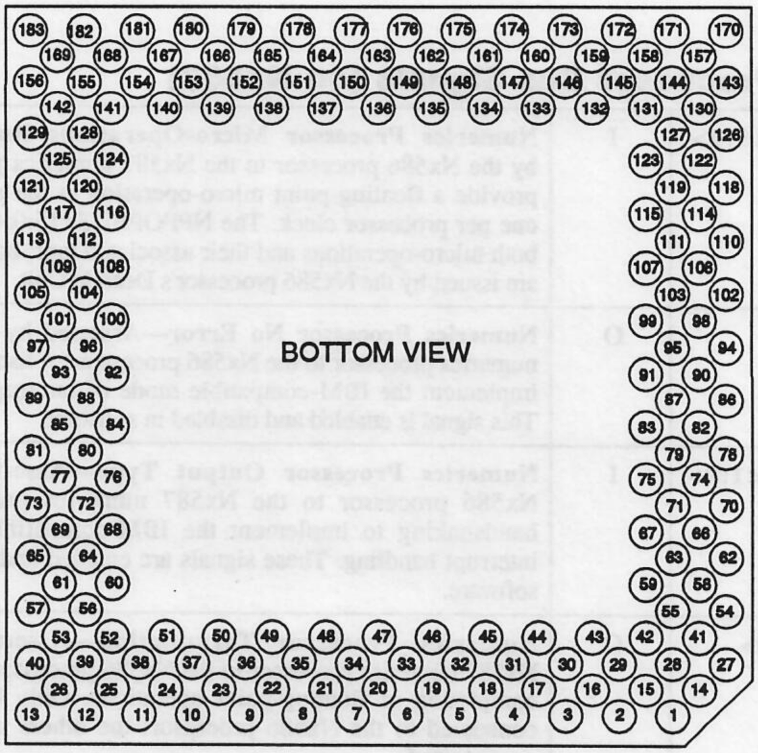


Figure 17 Nx587 Pinout Diagram (Bottom View)

Numerics-Processor Bus Signals (on Nx587)

NPPOPBUS<15:0>	I	**Numerics Processor Micro-Operations Bus**—Driven by the Nx586 processor to the Nx587 numerics processor to provide a floating-point micro-operation at the peak rate of one per processor clock. The NPPOPBUS<15:0> bus carries both micro-operations and their associated tags, both of which are issued by the Nx586 processor's Decode Unit.
NPNOERR	O	**Numerics Processor No Error**—Asserted by the Nx587 numerics processor to the Nx586 processor for handshaking to implement the IBM-compatible mode of interrupt handling. This signal is enabled and disabled in software.
NPOUTFTYP<1:0>	I	**Numerics Processor Output Type**—Asserted by the Nx586 processor to the Nx587 numerics processor for handshaking to implement the IBM-compatible mode of interrupt handling. These signals are enabled and disabled in software.
NPTERM<5:0>	O	**Numerics Processor Termination**—Asserted by the Nx587 numerics processor to the Nx586 processor to indicate completion of floating-point operations. Only bits 1:0 are connected to the Nx586 processor; the others must be left unconnected.
NPTAGSTAT<5:0>	I	**Numerics Processor Tag Status**—Driven by the Nx586 processor to the Nx587 numerics processor to synchronize the issuing, retiring, and aborting of instructions.
NPRREQ	I	**Numerics Processor Read Request**—Asserted by the Nx586 processor to the Nx587 numerics processor, to request use of the NPDATA<63:0> and NPTAG<4:0> buses to transfer data on the next clock. The NPRREQ signal has priority over the NPWREQ signal. When neither is requesting, the processor drives the bus. The processor sometimes makes speculative requests, such as when it concurrently does cache lookups for the data to be transferred. If the processor finds that it cannot use the bus after requesting it, it negates NPRVAL when the bus is granted, otherwise it asserts NPRVAL and transfers the data in the same clock.

NPRVAL	I	**Numerics Processor Read Valid**—Asserted by the Nx586 processor to the Nx587 numerics processor in the clock following a successful request, to indicate that the data being transferred on the NPDATA<63:0> bus in the current clock is valid.
NPWREQ	O	**Numerics Processor Write Request**—Asserted by the Nx587 numerics processor to the Nx586 processor, to request control of the NPDATA<63:0> and NPTAG<4:0> buses to transfer data on the next clock. The NPRREQ signal has priority over the NPWREQ signal. The numerics processor makes speculative requests concurrently with its first pass at formatting the output. If it discovers that more formatting is needed, it negates NPWVAL when the NPDATA<63:0> bus is granted, otherwise it asserts NPWVAL and transfers the data in the same clock.
NPWVAL	O	**Numerics Processor Write Valid**—Asserted by the Nx587 numerics processor to the Nx586 processor in the clock following a successful request, to indicate that the data being transferred on the NPDATA<63:0> bus in the current clock is valid.
NPTAG<4:0>	I/O	**Numerics Processor Tag Bus**—On each processor clock, this bus carries the five-bit micro-operation tag between the Nx586 processor and the Nx587 numerics processor. The tag identifies the instruction from which the micro-operation was decoded, and it corresponds to the data being transferred on the NPDATA<63:0> bus.
NPDATA<63:0>	I/O	**Numerics Processor Data**—On each processor clock, this bus carries up to 64 bits of read or write data between the Nx586 processor and the Nx587 numerics processor. The Nx586 processor uses it to provide read data to the Nx587 numerics processor, and the Nx587 numerics processor uses it to write results. The bus's bi-directionality is implemented with arbitration among the NPRREQ and NPWREQ signals. Arbitration priority is given to the processor, hence reads prevail over writes. The winner gets the bus on the next clock. The arbitration and the bus transfer are pipelined one clock apart at the processor-clock frequency. Thus, in every clock, both a request and a transfer are made.

Nx587 System Signals

Nx587 Clock

CLK	I	**NexBus Clock**—A TTL-level clock running at a frequency between 33MHz and 50MHz. The duty cycle is roughly 45% to 55%. All signals on NexBus transition on the rising edge of CLK. The processor's internal phase-locked loop (PLL) synchronizes internal processor clocks at twice the frequency of the CLK reference. Static operation (0MHz) is possible in non-PLL clock modes; for details, contact NexGen Applications Engineering.
PHE1	I	**Clock Phase 1**—For normal clocking operation, this signal should be tied high.
PHE2	I	**Clock Phase 2**—For normal clocking operation, this signal should be tied low.
CKMODE	I	**Clock Mode**—For normal clocking operation, this signal should be tied high.
XSEL	I	**Clock Mode Select**—For normal clocking operation, this signal should be tied low.
XPH1	O	**Processor Clock Phase 1**—For normal clocking operation, this signal must be left unconnected.
XPH2	O	**Processor Clock Phase 2**—For normal clocking operation, this signal must be left unconnected.
IREF	I	**Clock Input Reference**—For normal clocking operation, this signal must be tied low.
XREF	O	**Clock Output Reference**—For normal clocking operation, this signal must be left unconnected.
VDDA	I	**PLL Analog Power**—See the *Electrical Data* chapter for decoupling requirements.

Nx587 Interrupts and Reset

NPIRQ*	O	**Numerics Processor Interrupt Request**—Asserted by the Nx587 numerics processor to the interrupt controller that services the NexBus during unmasked floating-point exceptions. The same-named signal from the Nx586 must also be connected to this signal. The Nx587 numerics processor supports two modes of floating-point error handling: (1) *IBM-compatible mode*, in which IRQ* requests are made to an interrupt controller, and (2) *Intel-compatible mode*, in which coprocessor exceptions are vectored through location 16 in the interrupt descriptor table.
RESET*	I	**Global Reset (Power-Up Reset)**—Asserted by system logic. The processor responds by resetting its internal state machines and loading default values in its registers. At power-up it must remain asserted for a minimum of several milliseconds to stabilize the phase-locked loop. See the *Electrical Data* chapter.

Nx587 Test and Reserved Signals

FSPARE<6:0>	I/O	**Reserved**—For normal operation, these signals must be left unconnected.
NPPOPTAG<4:0>	I/O	**Reserved**—These signals must be connected to the same-named signals on the Nx586 processor.
NSPARE<2:0>	I	**Reserved**—These signals must be left unconnected.
SERIALIN	I	**Serial In**—The input of the scan-test chain. This signal must be tied low for normal operation.
SERIALOUT	O	**Serial Out**—The output of the scan-test chain. This signal must be left unconnected for normal operation.
TPH1	I	**Test Phase 1 Clock**—For scan test support. This signal must be tied low for normal operation.
TPH2	I	**Test Phase 2 Clock**—For scan test support. This signal must be tied low for normal operation.

Nx587 Alphabetical Signal Summary

CKMODE	I	Clock Mode
CLK	I	NexBus Clock
FSPARE<6:0>	I/O	Reserved
IREF	I	Clock Input Reference
NPDATA<63:0>	I/O	Numerics Processor Data
NPIRQ*	O	Numerics Processor Interrupt Request
NPNOERR	O	Numerics Processor No Error
NPOUTFTYP<1:0>	I	Numerics Processor Output Type
NPPOPBUS<15:0>	I	Numerics Processor Micro-Operations Bus
NPPOPTAG<4:0>	I/O	Reserved
NPRREQ	I	Numerics Processor Read Request
NPRVAL	I	Numerics Processor Read Valid
NPTAG<4:0>	I/O	Numerics Processor Tag Bus
NPTAGSTAT<5:0>	I	Numerics Processor Tag Status
NPTERM<5:0>	I	Numerics Processor Termination
NPWREQ	O	Numerics Processor Write Request
NPWVAL	O	Numerics Processor Write Valid
NSPARE<2:0>	I	Reserved
PHE1	I	Clock Phase 1
PHE2	I	Clock Phase 2
RESET*	I	Global Reset (Power-Up Reset)
SERIALIN	I	Serial In
SERIALOUT	O	Serial Out
TPH1	I	Test Phase 1 Clock
TPH2	I	Test Phase 2 Clock
VDDA	I	PLL Analog Power
XPH1	O	Processor Clock Phase 1
XPH2	O	Processor Clock Phase 2
XREF	O	Clock Output Reference
XSEL	I	Clock Mode Select

Figure 18 Nx587 Alphabetical Signal Summary

Hardware Architecture

The Nx586 processor and Nx587 numerics processor are closely coupled into a parallel architecture with distributed pipeline, distributed control, and rich hierarchy of storage elements. While the features of the two chips are sometimes listed separately elsewhere in this book, they are treated as an integrated architecture in this chapter. The Nx587 numerics processor is optional, but if used, each Nx587 chip requires a companion Nx586 processor. Alternatively, the Nx586 processor can be used by itself, without the numerics processor.

Bus Structure

The Nx586 processor supports three 64-bit buses: NexBus (the processor bus), the level-2 cache SRAM bus, and the numerics processor bus that is shared with the optional Nx587 chip. All buses are synchronous to the NexBus clock, although the numerics processor bus runs at twice the frequency of the other two buses.

NexBus

The NexBus is a 64-bit synchronous, multiplexed bus that supports all signals and bus protocols needed for cache-coherent processing using one or more Nx586 processors. Many types of devices can be interfaced to it, including a backplane, one or more Nx586 processors (with optional Nx587 numerics processors), one or more memory subsystems shared between processors, high-speed I/O devices, a central NexBus Arbiter, and an interface between the NexBus and other system buses. A modified write-once MESI protocol is used for cache coherency. The processor continually monitors the NexBus to guarantee cache coherency.

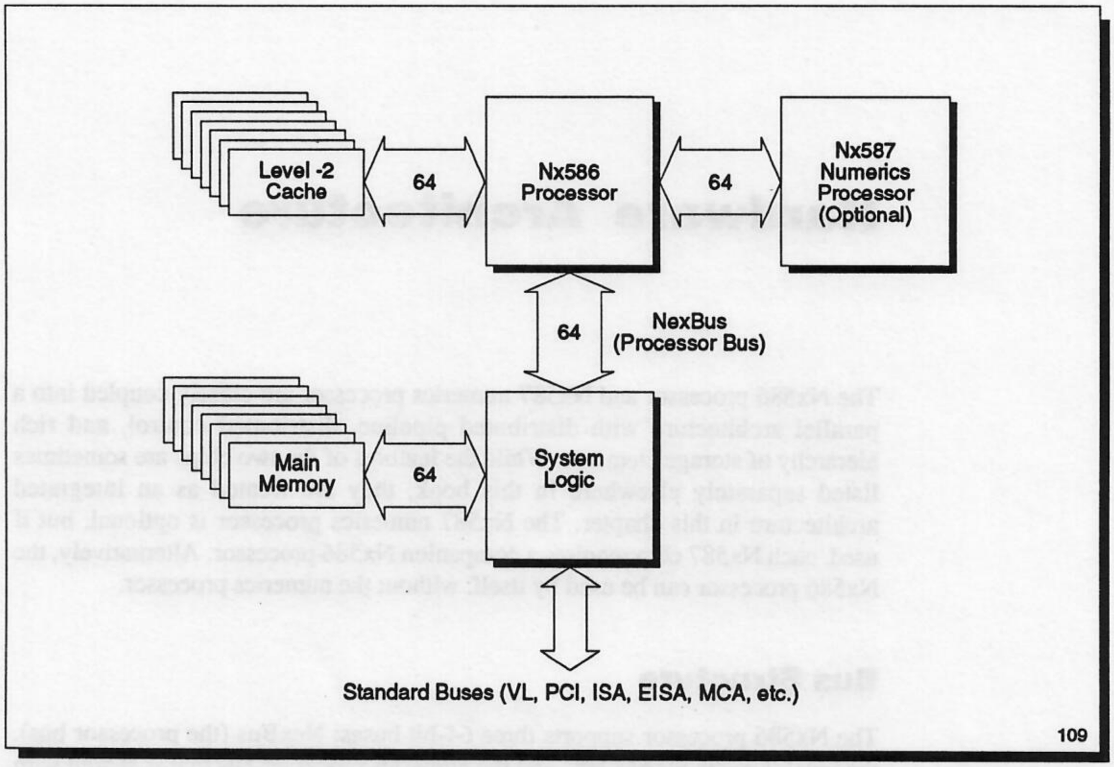


Figure 109 Single-Processor System Diagram

Figure 109 shows the general organization of a single-processor system. The system logic on the NexBus includes the following functions:

- **NexBus arbitration**
- **NexBus interface to standard buses (such as VL, PCI, ISA, EISA, MCA)**
- **NexBus interface to main memory and peripherals**
- **Main-memory control and arbitration**
- **Peripheral control**
- **System ROM**

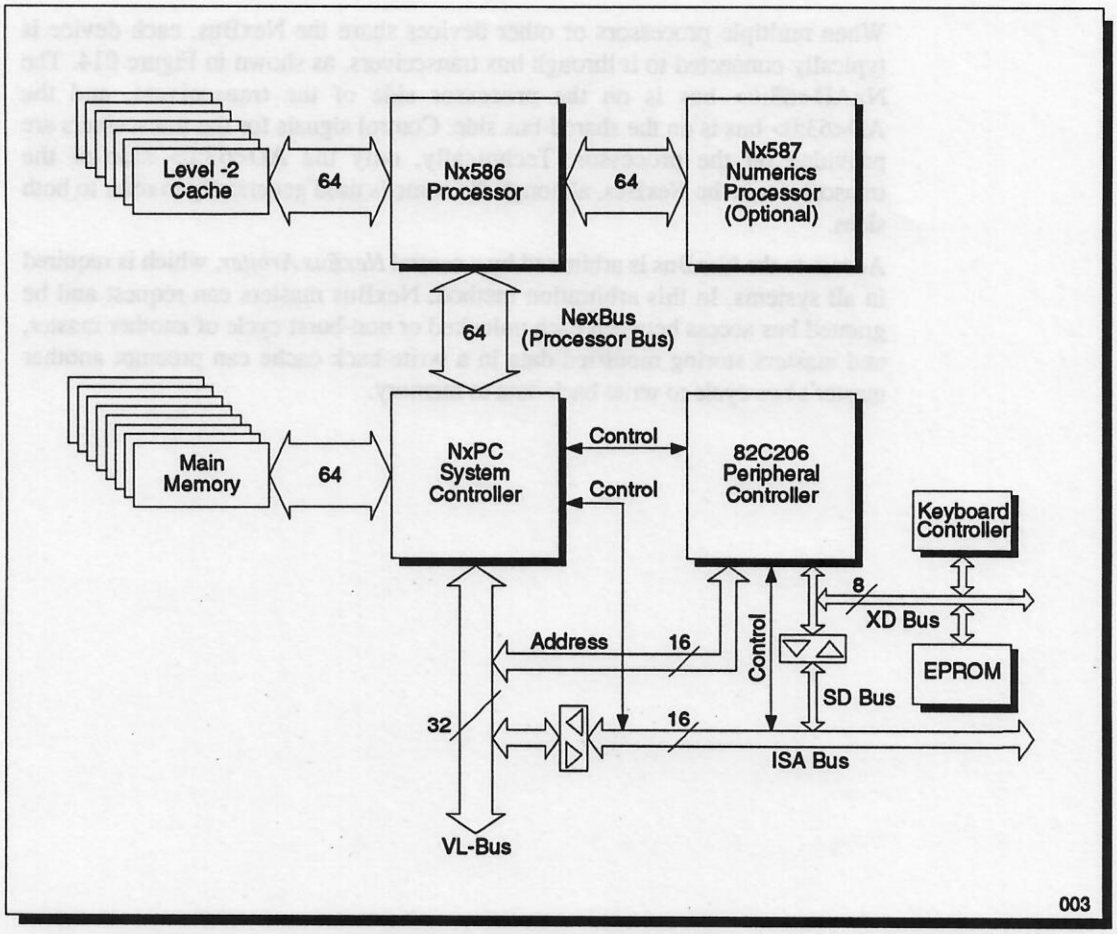


Figure 19 NxPC-Based Single-Processor System Diagram

Figure 19 shows a specific implementation of a single-processor system—one that uses NexGen's NxPC system controller chip. While the NexBus is a buffered bus, single-processor systems supported by the NxPC chip do not normally use transceivers on the NexBus because the NxPC chip can simulate the functions of the transceivers.

When multiple processors or other devices share the NexBus, each device is typically connected to it through bus transceivers, as shown in Figure 014. The NxAD<63:0> bus is on the processor side of the transceivers, and the AD<63:0> bus is on the shared-bus side. Control signals for the transceivers are provided on the processor. Technically, only the AD<63:0> side of the transceivers is the NexBus, although the name is used generically to refer to both sides.

Access to the NexBus is arbitrated by a central *NexBus Arbiter,* which is required in all systems. In this arbitration method, NexBus masters can request and be granted bus access between each unlocked or non-burst cycle of another master, and masters storing modified data in a write-back cache can preempt another master's bus cycle to write back data to memory.

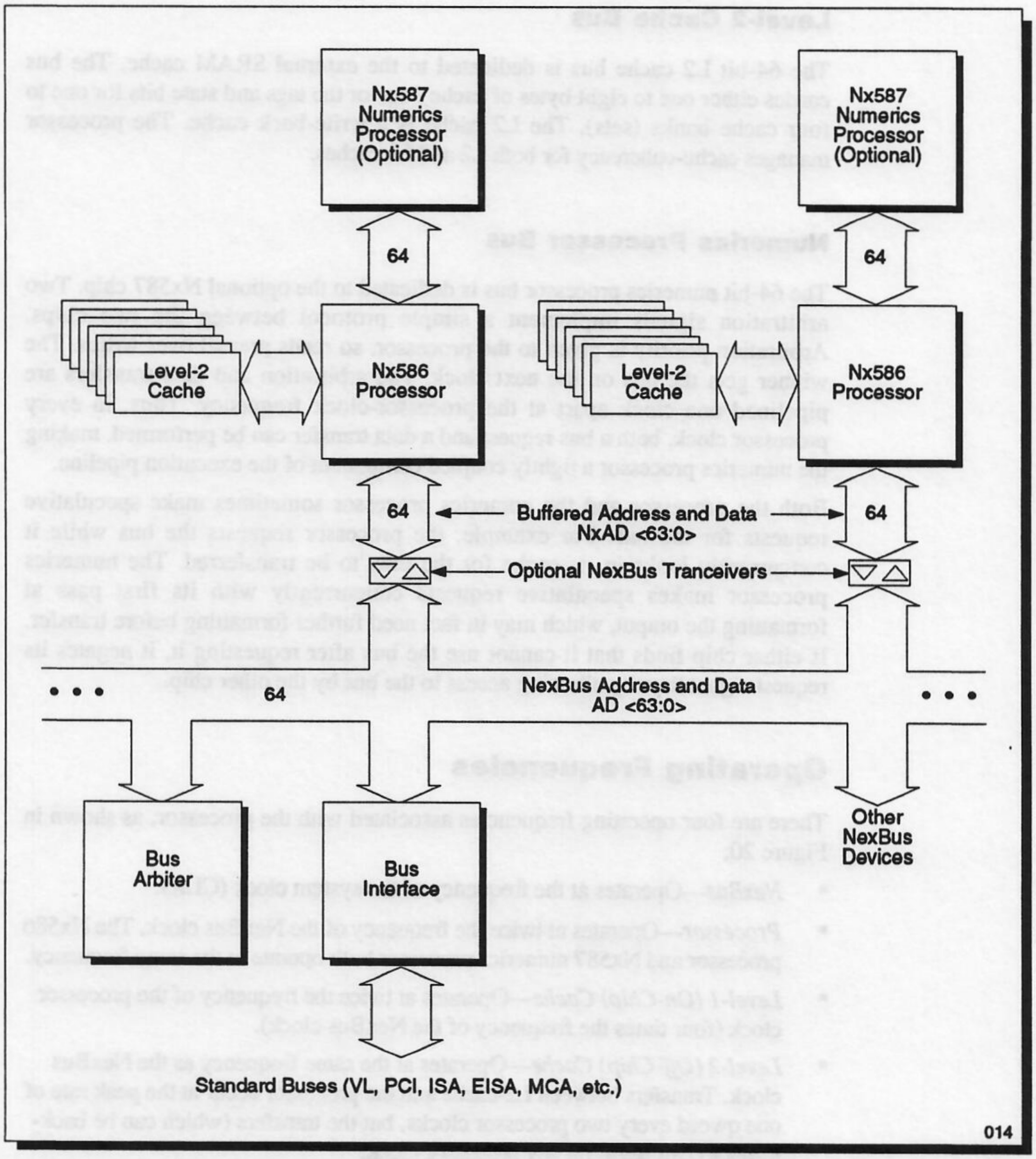


Figure 014 Multiprocessor System Diagram

Level-2 Cache Bus

The 64-bit L2 cache bus is dedicated to the external SRAM cache. The bus carries either one to eight bytes of cache data, or the tags and state bits for one to four cache banks (sets). The L2 cache is a write-back cache. The processor manages cache-coherency for both L2 and L1 caches.

Numerics Processor Bus

The 64-bit numerics processor bus is dedicated to the optional Nx587 chip. Two arbitration signals implement a simple protocol between the two chips. Arbitration priority is given to the processor, so reads prevail over writes. The winner gets the bus on the next clock. The arbitration and data transfers are pipelined one clock apart at the processor-clock frequency. Thus, in every processor clock, both a bus request and a data transfer can be performed, making the numerics processor a tightly coupled component of the execution pipeline.

Both the processor and the numerics processor sometimes make speculative requests for the bus. For example, the processor requests the bus while it concurrently looks in its cache for the data to be transferred. The numerics processor makes speculative requests concurrently with its first pass at formatting the output, which may in fact need further formatting before transfer. If either chip finds that it cannot use the bus after requesting it, it negates its request signal thereby allowing access to the bus by the other chip.

Operating Frequencies

There are four operating frequencies associated with the processor, as shown in Figure 20:

- *NexBus*—Operates at the frequency of the system clock (CLK).
- *Processor*—Operates at twice the frequency of the NexBus clock. The Nx586 processor and Nx587 numerics processor both operate at the same frequency.
- *Level-1 (On-Chip) Cache*—Operates at twice the frequency of the processor clock (four times the frequency of the NexBus clock).
- *Level-2 (Off-Chip) Cache*—Operates at the same frequency as the NexBus clock. Transfers between L2-cache and the processor occur at the peak rate of one qword every two processor clocks, but the transfers (which can be back-to-back) can begin on any processor clock.

Unless otherwise specified in this book, a *clock cycle* means the Nx586 processor's clock cycle. However, the timing diagrams in the *Bus Operations* chapter are relative to the NexBus clock, not the processor clock.

Figure 20 shows the relative frequencies for a 66 MHz processor (actually 66.666...MHz). If the NexBus clock runs at 33 MHz (actually 33.333... MHz), the processor and numerics processor run at 66.666...MHz, the on-chip L1 caches run at 133.333... MHz, and the L2-cache bus runs at 33.333... MHz.

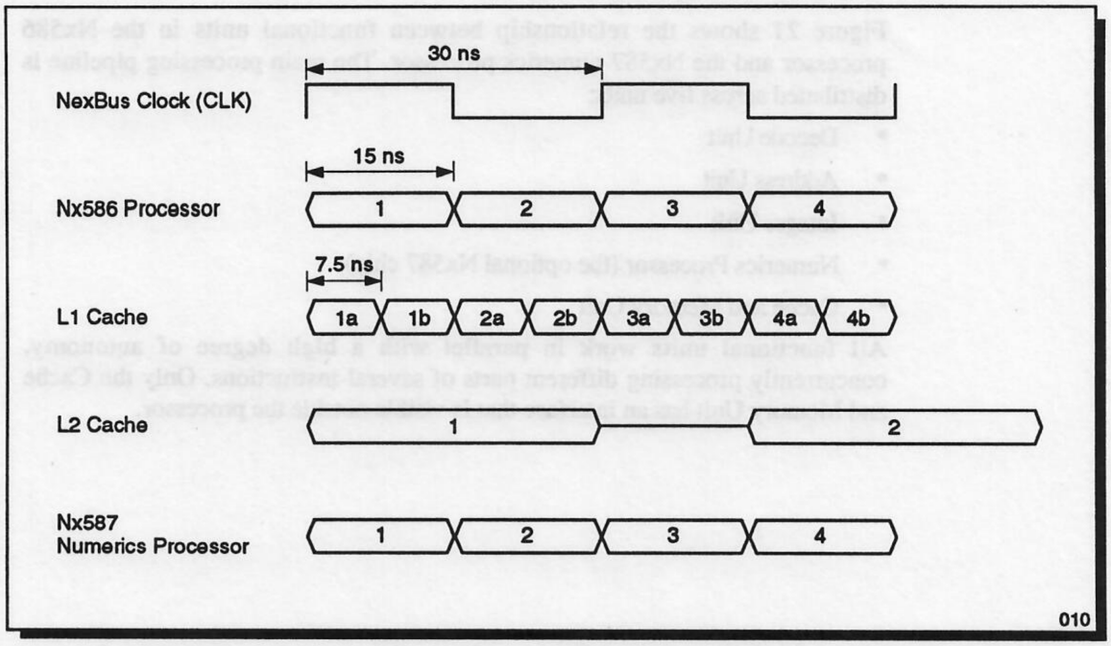


Figure 20 Operating Frequencies (66MHz Processor)

The processor uses an on-chip phase-locked loop and the NexBus clock to internally generate two non-overlapped phases of its own clock, shown in Figure 20 as the 7.5ns phases that drive the L1 cache. Most of the processor's pipeline stages operate on these phases. For example, a register-file access, an adder cycle, a lookup in the translation lookaside buffer (TLB), and an on-chip cache read or write all take a single phase of the processor clock.

The processor supports an average sustainable read and write bandwidth on NexBus of 152 MBytes per second for the 66MHz Nx586 processor (229 MB/sec for the 100MHz Nx586 processor), and a peak transfer rate of 267 MBytes per second for the 66MHz Nx586 processor (400 MB/sec for the 100MHz Nx586 processor).

While the minimum bus-clock reference frequency is specified at 33MHz, both the Nx586 and Nx587 chips are built from static-logic CMOS. With a special bus-clock reference scheme that does not use the on-chip phase-locked loop, the chips can operate at any clock frequency between zero and the specified maximum. There are no dynamic circuits that force a minimum frequency, so the chips can be brought to zero frequency without losing data. For application details, contact the NexGen Applications Engineering group.

Internal Architecture

Figure 21 shows the relationship between functional units in the Nx586 processor and the Nx587 numerics processor. The main processing pipeline is distributed across five units:

- Decode Unit
- Address Unit
- Integer Unit
- Numerics Processor (the optional Nx587 chip)
- Cache and Memory Unit

All functional units work in parallel with a high degree of autonomy, concurrently processing different parts of several instructions. Only the Cache and Memory Unit has an interface that is visible outside the processor.

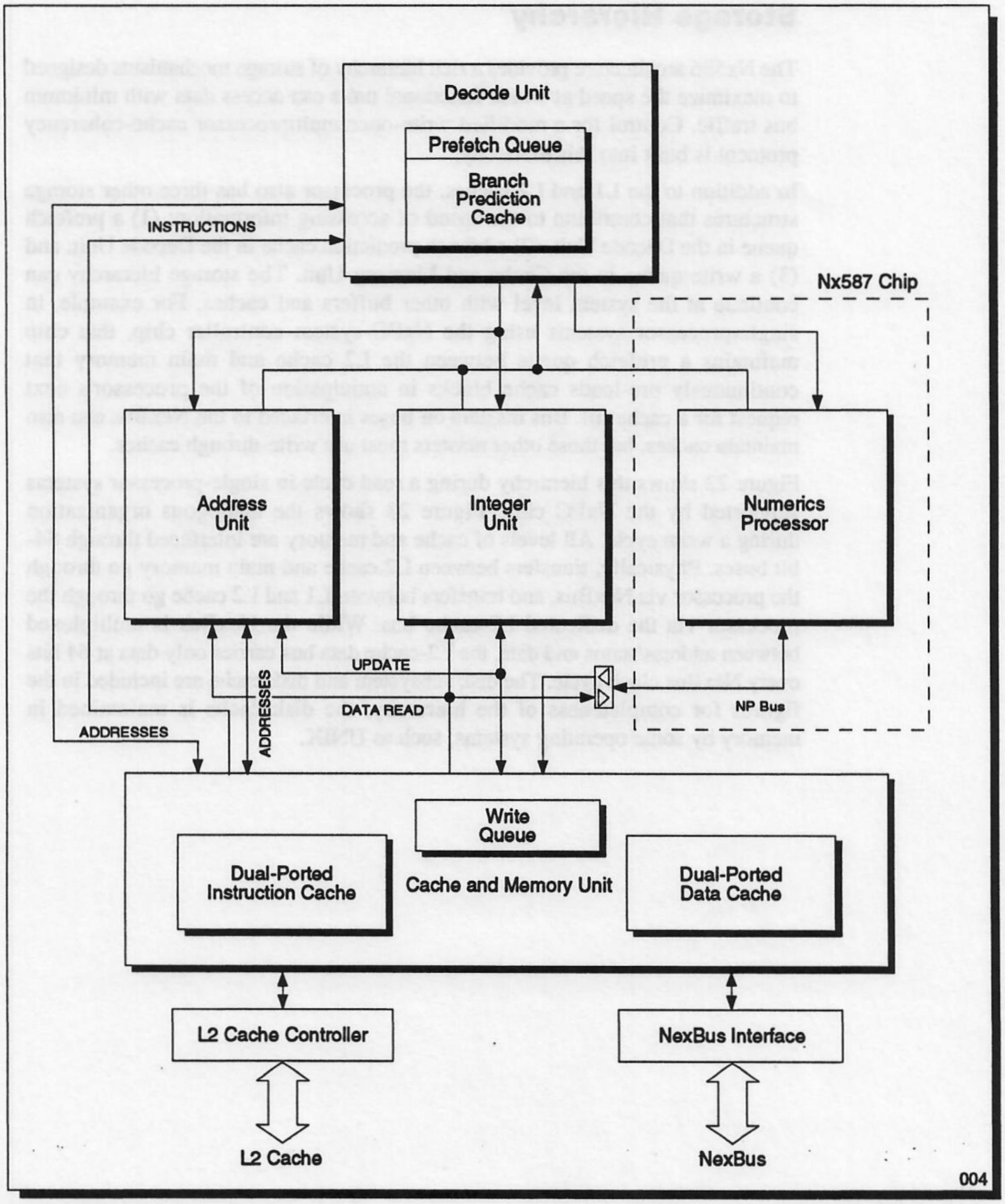


Figure 21 Nx586 Internal Architecture

Storage Hierarchy

The Nx586 architecture provides a rich hierarchy of storage mechanisms designed to maximize the speed at which functional units can access data with minimum bus traffic. Control for a modified write-once multiprocessor cache-coherency protocol is built into this hierarchy.

In addition to the L1 and L2 caches, the processor also has three other storage structures that contribute to the speed of accessing information: (1) a prefetch queue in the Decode Unit, (2) a branch prediction cache in the Decode Unit, and (3) a write queue in the Cache and Memory Unit. The storage hierarchy can continue at the system level with other buffers and caches. For example, in single-processor systems using the NxPC system controller chip, that chip maintains a prefetch queue between the L2 cache and main memory that continuously pre-loads cache blocks in anticipation of the processor's next request for a cache fill. Bus masters on buses interfaced to the NexBus can also maintain caches, but those other masters must use write-through caches.

Figure 22 shows this hierarchy during a read cycle in single-processor systems supported by the NxPC chip. Figure 23 shows the analogous organization during a write cycle. All levels of cache and memory are interfaced through 64-bit buses. Physically, transfers between L2 cache and main memory go through the processor via NexBus, and transfers between L1 and L2 cache go through the processor via the dedicated L2-cache bus. While the NexBus is multiplexed between address/status and data, the L2-cache data bus carries only data at 64 bits every NexBus clock cycle. The disk subsystem and disk cache are included in the figures for completeness of the hierarchy; the disk cache is maintained in memory by some operating systems, such as UNIX.

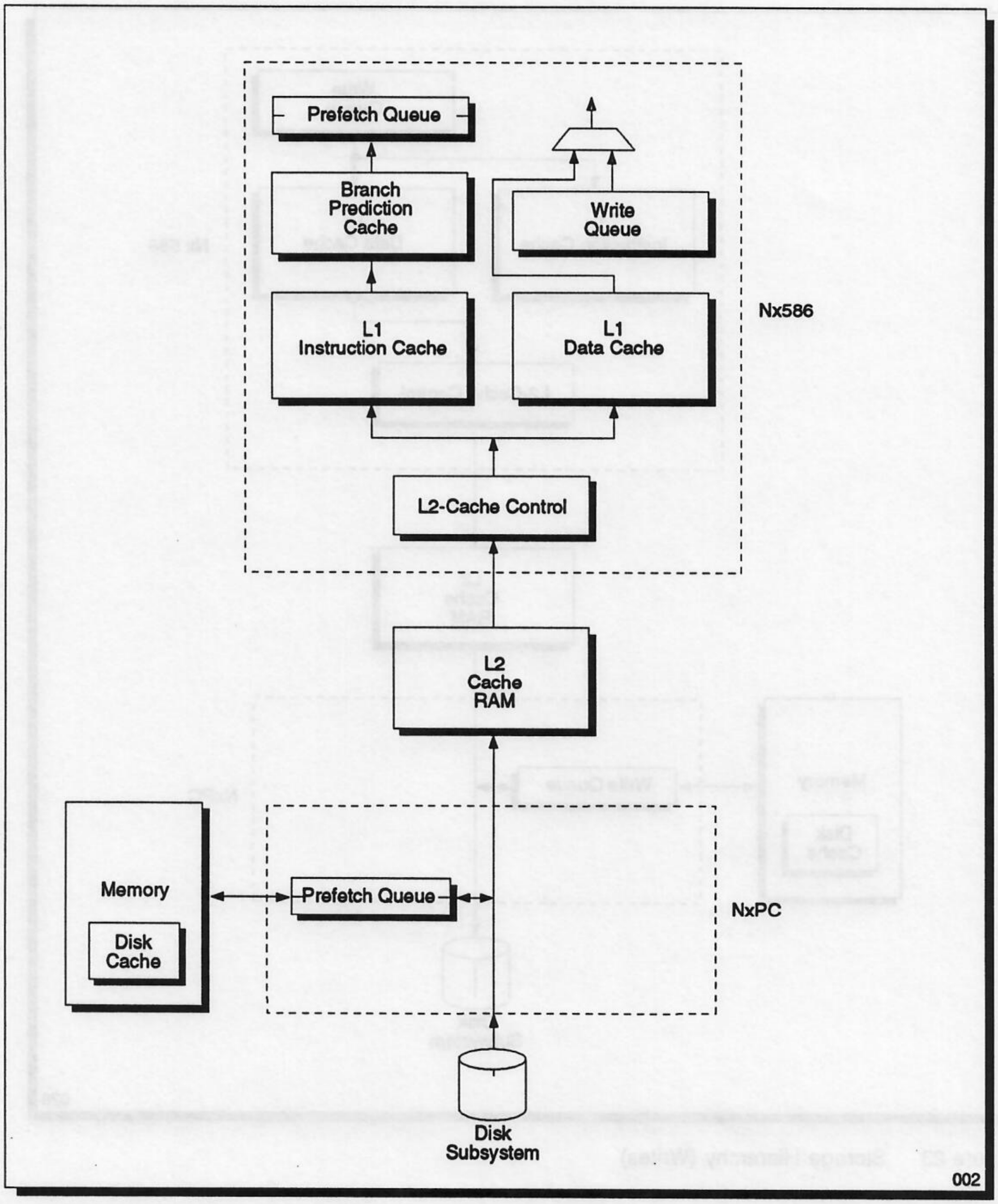


Figure 22 Storage Hierarchy (Reads)

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems.
NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

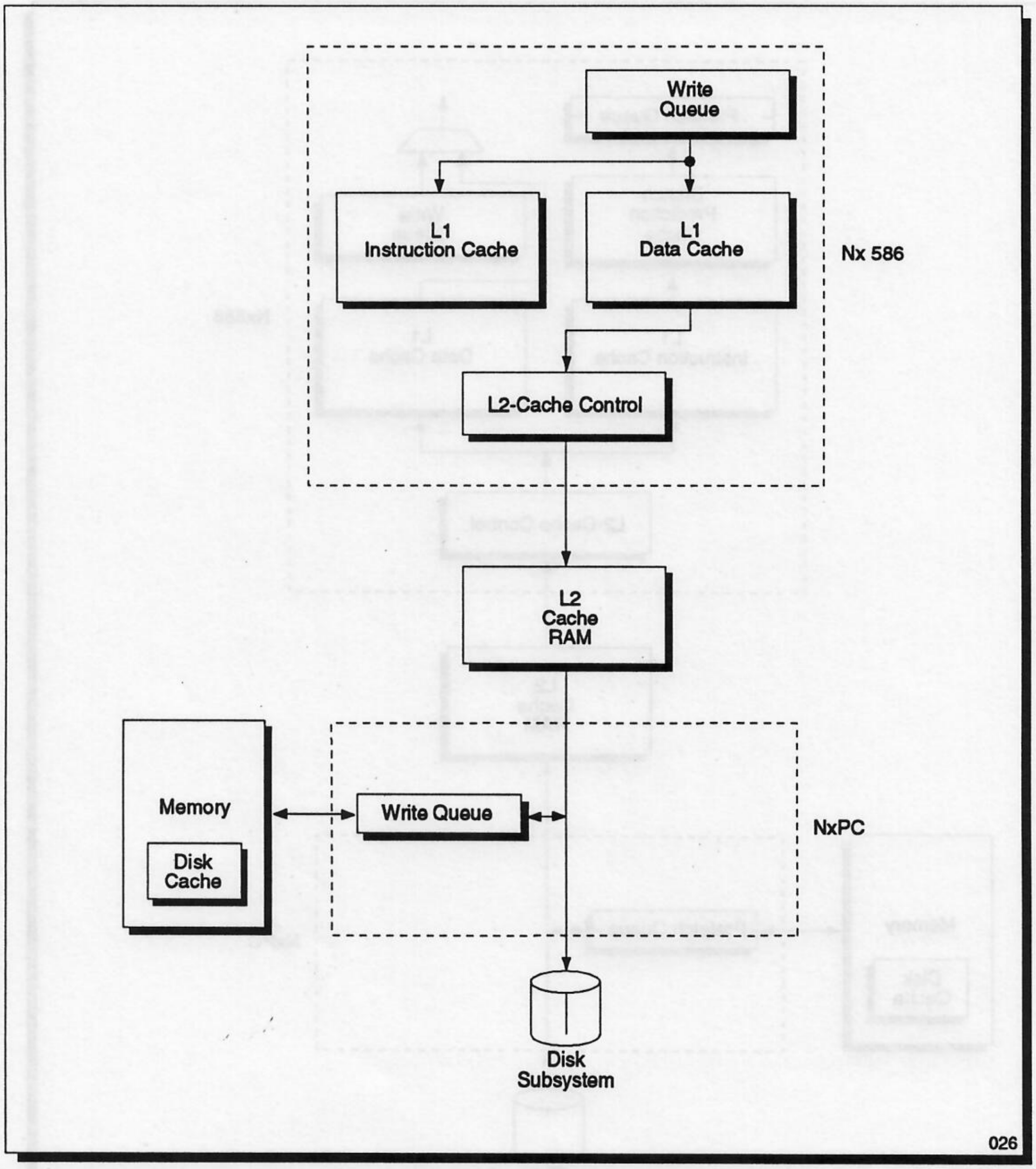


Figure 23 Storage Hierarchy (Writes)

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems. NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

Transaction Ordering

Interlocks enforce transaction ordering in a manner that optimizes read accesses. With the exceptions detailed below, the *general rules* for transaction ordering are:

- *Memory Reads*—Memory reads (whether cache hits or reads on the NexBus) are re-ordered ahead of writes, are performed out of order with respect to other reads, and are done speculatively. Reads are always looked up in the Cache and Memory Unit's write queue simultaneously with the L1 cache lookup. With respect to the most recent copy of data, the write queue takes priority over the cache. A hit in the write queue is serviced directly from that queue.
- *I/O Reads*—I/O reads are not done speculatively because they can have side effects in memory that may cause the I/O read to be done improperly. I/O reads have higher priority than memory reads, but all pending writes are completed first.
- *All Writes*—Writes are performed in order with respect to other writes, and they are never performed speculatively. Writes are always held in the write queue until the processor knows the outcome of all older instructions.
- *Semaphores*—Locked read-modify-writes are stalled until the write queue is emptied.

Now for the *exceptions:*

- *Cache-Hit Reads*—The processor holds reads that hit in the cache if any of the following conditions exist:
  - — The cache entry depends upon pending writes that have not yet received their data, are mapped as non-cacheable, are mapped as write-protected.
  - — The read is locked (hence, the rules below for Memory Reads on NexBus are followed).
- *Memory Reads on NexBus*—The processor holds memory reads on the NexBus (cache misses) if any of the following conditions exist:
  - — The write queue has pending writes to I/O or to memory that are mapped as non-cacheable I/O.
  - — The read is locked, and the write portion of a previous locked read-modify-write has not yet been performed.
  - — The read is locked, is a HLT instruction, is part of an interrupt-acknowledge cycle, results in a shutdown, or is a read to memory that is mapped as non-cacheable I/O, *AND* there are older outstanding instructions or pending writes.

Cache and Memory Subsystem

Characteristics

The cache and memory subsystem is a key element in the processor's performance. Each of the two on-chip L1 caches (instruction and data) are 8kB in size and dual-ported, as shown in Figure 24. The L2 cache is either 256kB or 1MB and single-ported. It is built from a single array of eight SRAM chips. The L2 cache stores instructions and data in 32-byte cache blocks (lines), each of which has an associated tag and cache-coherency state. Separate tag RAMs are not used. Instead, tag data is stored in a small part of the L2 cache. L2 is a random-access cache, with SRAM very closely coupled to the processor. Memory references of any kind can be interleaved without compromising performance. It responds to random accesses just as quickly as to block transfers. Eight bytes is the minimum unit of transfer between memory and cache.

	Level 1 (L1)		*Level 2 (L2)*
Contents	Instructions (I Cache)	Data (D Cache)	Instructions and Data
Location	processor	processor	controller is on processor; SRAM accessed from 64-bit SRAM bus
Cache Size	8kB	8kB	256kB or 1MB
Ports	2	2	1
Clock Frequency, Relative to Processor Clock	2x	2x	0.5x

Figure 24 Cache Characteristics

If a write needs to go to the NexBus for cache-coherency purposes, it does so before it goes to a cache. Whether the write is needed on the NexBus depends on the caching state of the data: if the data is *shared* (as described later in the *Cache Coherency* section), all other NexBus caching devices need to know about the imminent write so that they can take appropriate action. The processor's caches can be configured so that specified locations in the memory space can be cacheable or non-cacheable and read/write or read only (write-protected).

The Cache and Memory Unit contains a write queue that stores partially and fully assembled writes. The queue serves several functions. First, it buffers writes that are waiting for bus access, and it reorders writes with respect to reads or other more important actions. Second, it assembles the pieces of a write as they become available. (Addresses and data arrive at the queue separately as they come out of the distributed pipelines of other functional units.) Third, the queue is used to back out of instructions when necessary. All writes remain in the queue until signaled by the Decode Unit that the instruction associated with the write is retired—*i.e.*, that there is no possibility of an instruction backout due to a branch not taken or to an exception or interrupt during execution.

Reads are looked up in the write queue simultaneously with the L1 cache lookup. A hit in the write queue is serviced directly from that queue, and write locations pending in the queue take priority over any L1-cache copy of the same location. Reads coming into the unit from NexBus are routed in a pipeline first to the processor, then to L2 cache, then to the L1 caches. Reads coming in from the L2 cache are routed first to the processor, then to the L1 caches, then (for write-backs) to NexBus. Pending writes in the queue go first to the L1 caches (both the instruction and data caches can be written), then to L2 if necessary, then to NexBus if necessary. The unit remembers which instructions from the L1 cache have been copied by the Decode Unit into the branch-prediction cache (BPC).

NexBus references by other masters are not looked up in the write queue when the processor snoops the bus. Instead, coherency mechanisms other than snooping are provided to ensure that such things as semaphore operations work correctly.

The dual ports on the L1 instruction and data caches protect the processor from stalls. In a single clock, the processor can read from port A on each cache while it reads or writes port B on each cache, such as for cache lookups, cache fills, and other cache housekeeping overhead. The dual ports allow both L1 caches to contain identical data, as when a 32-byte cache block contains both instructions and data and is loaded into both L1 caches in different cache-block reads.

Cache Coherency

The processor continually monitors (snoops) NexBus operations by other bus masters to guarantee coherency with data cached in the processor's L2 cache, L1 caches, and branch prediction cache (BPC). A type of write-invalidate cache-coherency protocol called *modified write-once* (MWO) or *modified, exclusive, shared, or invalid* (MESI) is used. In this protocol, each 32-byte block in the L2 cache is in one of four states:

- *Exclusive*—Data copied into a single bus-master's cache. The master then has the exclusive right (not yet exercised) to modify the cached data. Also called *owned clean* data.
- *Modified*—Data copied into a single bus-master's cache (originally in the exclusive state) but that has subsequently been written to. Also called *dirty, owned dirty,* or *stale* data.

- *Shared*—Data that may be copied into multiple bus-masters' caches and can therefore only be read, not written.
- *Invalid*—Cache locations in which the data is not correctly associated with the tag for that cache block. Also called *absent* or *not present* data.

The protocol allows any NexBus caching device to gain exclusive ownership of cache blocks, and to modify them, without writing the updated values back to main memory. It also allows caching devices to share read-only versions of data. To implement the protocol, the processor:

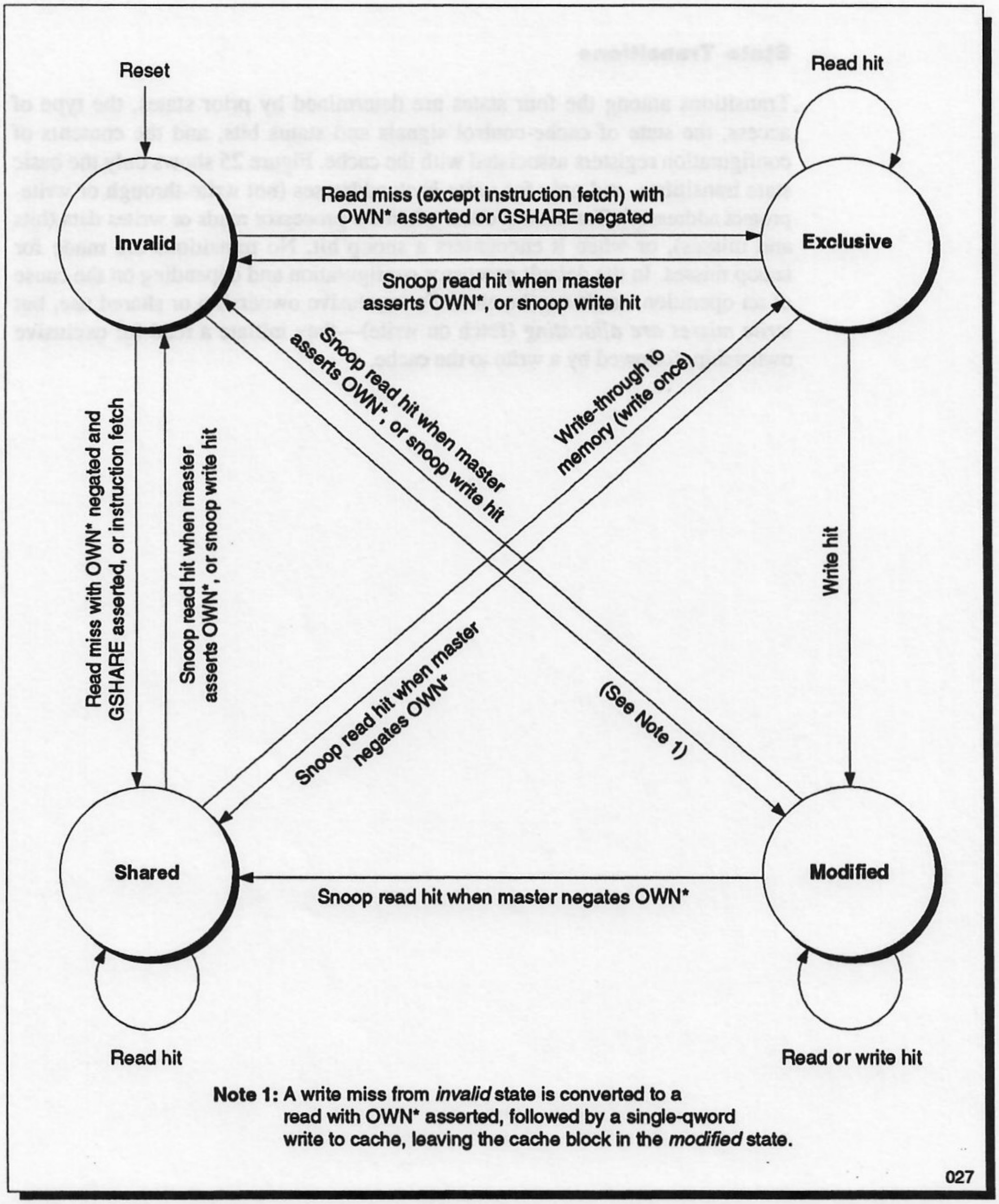
- *Requests data* in a specific state by asserting or negating NexBus cache-control bits and signals.
- *Caches data* in a specific state by watching NexBus cache-control input signals from system logic and the slave being accessed.
- *Snoops* the NexBus to detect operations by other masters that hit in the processor's caches.
- *Intervenes* in the operations of other NexBus masters to write back modified data to main memory if a hit occurs during a bus snoop.
- *Updates the state* of cached blocks if a hit occurs during a bus snoop.

The protocol name, *write-once*, reflects the processor's ability to obtain exclusive ownership of certain types of data by writing once to memory. If the processor caches data in the shared state and subsequently writes to that location, a write-through to memory occurs. During the write-through, all other caching devices with shared copies invalidate their copies (hence the name, write-invalidate). After the write, the processor owns the data in the exclusive state, since the processor has the only valid copy and it matches the copy in memory. Any additional writes are local—they change the state of the cached data to modified, although the changes are not written back to memory until a snoop cycle by another bus master forces the write-back. Write-once protocols maximize the processor's opportunities to cache data in the exclusive (owned) state even when the processor has not specifically requested exclusive use of data, thereby maximizing the number of transactions that can be performed from the cache.

There are also other means of obtaining ownership of data besides writing to memory, and write operations can be performed in a way that does not modify ownership. The protocol is compatible with caching devices that employ write-through caching policies, if the devices implement bus snooping and support cache-block invalidation. Caching devices that use a cache-block (line) size other than four-qwords must use a write-through policy.

State Transitions

Transitions among the four states are determined by prior states, the type of access, the state of cache-control signals and status bits, and the contents of configuration registers associated with the cache. Figure 25 shows only the basic state transitions, and only for write-back addresses (not write-through or write-protect addresses). Transitions occur when the processor reads or writes data (hits and misses), or when it encounters a snoop hit. No transitions are made for snoop misses. In the default processor configuration and depending on the cause of an operation, reads can be either for exclusive ownership or shared use, but *write misses are allocating* (fetch on write)—they initiate a read for exclusive ownership, followed by a write to the cache.


Reset
Read hit
Read miss (except instruction fetch) with OWN* asserted or GSHARE negated
Invalid
Exclusive
Snoop read hit when master asserts OWN*, or snoop write hit
Snoop read hit when master asserts OWN*, or snoop write hit
Write-through to memory (write once)
Read miss with OWN* negated and GSHARE asserted, or instruction fetch
Snoop read hit when master asserts OWN*, or snoop write hit
Write hit
Snoop read hit when master negates OWN*
(See Note 1)
Shared
Modified
Snoop read hit when master negates OWN*
Read hit
Read or write hit
027

Note 1: A write miss from *invalid* state is converted to a read with OWN* asserted, followed by a single-qword write to cache, leaving the cache block in the *modified* state.

Figure 25 Basic Cache-State Transitions

Figure 26 describes the primary signals and status bits that affect the state transitions shown in Figure 25. The OWN* and SHARE* signals control many transitions. The assertion of OWN* implies that the data is both snoopable (SNPNBL) and cacheable (CACHBL). Figure 27 describes the signals and status bits that affect processor responses during bus snooping. The four sections following these tables describe the characteristics of the states in more detail.

OWN* NxAD<49> *address phase*	I/O	*Ownership Request*—Asserted by a master when it intends to cache data in the *exclusive* state. The bit is asserted during write-misses or read-modify-write misses. If such an operation hits in the cache of another master, that master writes back (if copy is modified) and changes the state of its copy to *invalid.* If OWN* is negated during a read or write, this implies that SHARE* is asserted by the same master, so other masters can change their copies to *shared.*
OWNABL	I	*Ownable*—Asserted by the system logic during accesses by the processor to locations that may be cached in the *exclusive* state. Negated during accesses that may only be cached in the *shared* state, such as bus-crossing accesses to an address space that cannot support the MESI cache-coherency protocol. All NexBus addresses are assumed to be cacheable in the *exclusive* state. The OWNABL signal is provided in case system logic needs to restrict caching to certain locations. In single-processor systems using the NxPC chip, that chip does not have an OWNABL signal and the processor's OWNABL input is typically tied high for write-back configurations to allow caching in the *exclusive* state on all reads.
SHARE* GSHARE	O I	*Shared Data*—SHARE* is asserted by any NexBus master during block reads by another NexBus master to indicate to the other master that its read hit in a block cached by the asserting master, and that the data being read can only be cached in the *shared* state, if OWN* is negated. GSHARE is the backplane NAND of all SHARE* signals. If GSHARE and OWN* are both negated during the read, the data will be promoted to the *exclusive* state because no other NexBus device declared via SHARE* that it has cached a copy.

Figure 26 Cache State Controls

SNPNBL NxAD<57>	I/O	*Snoop Enable*—Asserted to indicate that the current operation affects memory that may be valid in other caches. When this signal is negated, snooping devices need not look up the addressed data in their cache tags.
DCL* GDCL	O I	*Dirty Cache Line*—Asserted during operations by another master to indicate that the processor has cached the location being accessed in a *modified* (dirty) state. During reads, the requesting master's cycle is aborted so that the processor, as an intervenor, can preemptively gain control of the NexBus and write back its modified data to main memory. While the data is being written to memory, the requesting master reads it off the NexBus. The assertion of DCL* is the only way in which atomic 32-byte cache-block fills by another NexBus master can be preempted by the processor for the purpose of writing back dirty data. During writes, the initiating master is allowed to finish its write. The NexBus Arbiter must then guarantee that the processor asserting DCL* gains access to the bus in the very next arbitration grant, so that the processor can write back all of its modified data *except* the bytes written by the initiating master. (In this case, the initiating master's data is more recent than the data cached by the processor asserting DCL*.)

Figure 27 Bus Snooping Controls

Invalid State

After reset, all cache locations are invalid. This state implies that the block being accessed is not correctly associated with its tag. Such an access produces a *cache miss*. A read-miss causes the processor to fetch the block from memory on the NexBus and place a copy in the cache. If OWN* is negated and GSHARE is asserted, the block changes state from invalid to shared, provided that the memory slave asserts the GBLKNBL signal when each qword is transferred. If the processor asserts OWN* when OWNABL is asserted, or if no other caching device shares the block (GSHARE negated), the processor will change the state of the block from invalid to exclusive. If GBLKNBL is negated, the data may be used by the processor but it will not be cached, and the cache block will remain invalid.

The processor will invalidate a block if another master performs any operation with OWN* asserted that addresses that block, and OWNABL and GXACK are simultaneously asserted. If the block's previous state was modified, the processor will also intervene in the other master's operation to write back the modified data.

Shared State

When the processor performs a read with OWN* negated and GSHARE asserted, and the read misses the cache, the block will be cached in the shared state. The shared state indicates that the cache block may be shared with other caching devices. A block in this state mirrors the contents of main memory. When the processor has cached data in the shared state, it snoops NexBus memory operations by other masters, ignoring only operations for which SNPNBL is negated. When the processor performs block reads that hit in a block shared with another master, that master asserts SHARE*.

When the processor performs a write with OWN* negated—or when it performs a write with OWN* asserted, OWNABL negated, and GXACK asserted—other masters may either invalidate their copy or update it and retain it in the shared state (as determined by configuration software).

When the processor performs a write to a shared block, the processor (1) writes the data through to main memory while asserting OWN* so as to cause other caching masters to invalidate their copies, (2) updates its cache to reflect the write, and (3) if OWNABL and GXACK are both asserted during the write, the processor changes the state of the block to exclusive, otherwise the state remains shared.

If the processor performs a read or write in which OWN*, OWNABL, and GXACK are all asserted, other masters invalidate their copy of such blocks.

Exclusive State

When the processor performs a read with OWN* asserted or GSHARE negated, and the read misses the cache, the block will be cached in the exclusive (owned clean) state. In the exclusive state, as in the shared state, the contents of a cache block mirrors that of main memory. However, the processor is assured that it contains the only copy of the data in the system. Thus, any subsequent write can be performed directly to cache and need not be immediately written back to memory. The cache block so modified will then be in the modified state. Just as with shared cache blocks, the processor snoops NexBus memory operations when it has cached data in the exclusive state, except when SNPNBL is negated.

If another master asserts OWN* while hitting in an exclusive block in the processor, the processor invalidates its copy. A read by another master with OWN* negated that hits in an exclusive block forces the processor to assert SHARE* and change the block to the shared state, if CACHBL is asserted. If CACHBL is negated, the block remains in the exclusive state.

If a write by another master hits in an exclusive block, the processor invalidates the block (if OWN* is asserted), or updates the block (if OWN* is negated) and changes it to the shared state (if CACHBL is asserted) or to the exclusive state (if CACHBL is negated). OWNABL has no effect on snooping the exclusive and modified states, since a cache block could not have been cached in these states if the block were not ownable.

Modified State

The modified (owned stale or dirty) state implies that a cache block previously fetched in the exclusive state has been subsequently written to and no longer matches main memory. As in the exclusive state, the processor is assured that no other master has cached a copy so the processor can perform writes to the cache without writing them to memory.

Reads and single-qword writes by other masters that address a modified block cause the processor to assert DCL* and perform an *intervenor operation*. The processor writes back its cached data to memory and the other master simultaneously reads it from the NexBus.

During external non-OWN* reads, the processor changes its copy of the block to the shared state if CACHBL is asserted, or to the exclusive state otherwise. If an external non-OWN* single-qword write with CACHBL asserted hits in a modified block, the processor asserts DCL* and intervenes in the operation. The processor then either asserts SHARE* during the operation, records the data in its cache, and changes the state to shared, or it simply invalidates the block, depending on software configuration. If an external non-OWN* single-qword write with CACHBL negated hits in a modified block, the processor either treats it like the preceding case (non-OWN* CACHBL) or keeps DCL* negated, records the data in its cache, and leaves the block in the modified state.

During external block writes (unlike the single-qword writes described above) the processor does not perform an intervenor operation with write-back because the other master overwrites the entire cache block(s). If an external block write hits a modified processor block, the processor does asserts SHARE* instead of DCL*. If OWN* is negated by the writing master, the processor updates the block and, if CACHBL is negated, changes the state of the block to exclusive, or otherwise changes the state to shared.

Internal reads or writes do not change the state of a modified block. However, if another master attempts to write to a block that has been modified by the processor, the modified data (or portions thereof) is written back to memory. During the write-back, the processor negates SNPNBL to relieve other caching devices of the obligation to look the address up in their caches, since a modified block can never be in another cache.

Interrupts

The processor supports maskable interrupts on its INTR* input, non-maskable interrupts on its NMI* input, and software interrupts through the INT instruction. Hardware interrupts (INTR* and NMI*) are asynchronous to the NexBus clock. They are asserted by external interrupt control logic when that logic receives an interrupt request from an I/O device, system timer, or other source. When an active non-maskable interrupt request is sensed by the interrupt controller, the request is passed to the processor which then performs an interrupt acknowledge sequence, as defined in the *Bus Operations* chapter. Maskable interrupt requests must be asserted until cleared by the interrupt service routine.

In single-processor systems supported by the NxPC chip, an 82C206 peripheral controller handles I/O interrupts. The NxPC chip generates the non-maskable interrupt (NMI*) input to the processor, and it passes along the processor's non-maskable interrupt acknowledge to the 82C206 via the NxPC chip's INTA* output. For a description of these interrupts, see the *NxPC System Controller Databook.*

In multiprocessor environments, the NexBus architecture supports 15 interrupt signals, IRQ<14:0>*. Each processor has its own interrupt control logic on an alternate bus. Only the appropriate controller must respond to a given interrupt request. To do this, all interrupt controllers must examine the MID<5:2> bits that are transmitted on NxAD<45:42> during the address phase of each bus operation. For a description of these interrupts, see the *NexBus Specification.*

Clock Generation

Five signals determine the manner in which the processor's internal clock phases (PH1 and PH2) are derived or provided. These signals include CKMODE, XSEL, CLK, PHE1, and PHE2. These signals determine one of three modes: Phase-Locked Loop (the normal operating mode), External Phase Inputs, or External Processor Clock, as shown in Figure 28 and described in the sections below.

Mode	*CKMODE*	*XSEL*	*CLK*	*PHE1*	*PHE2*
Phase-Locked Loop (normal operating mode)	1	0	33, 40, or 50 MHz	1	0
External Phase Inputs	x	1	33, 40, or 50 MHz	Externally supplied at 2x the CLK frequency	Externally supplied at 2x the CLK frequency
External Processor Clock	0	0	33, 40, or 50 MHz	Double-rail differential input at 2x the CLK frequency	

Figure 28 Clocking Modes

In the normal *phase-locked loop* mode, the internal clock phases are derived from the external NexBus clock (CLK) via a phase-locked loop (PLL). In all modes, the CLK input must be driven at one-half the processor's internal operating frequency so as to provide the bus-interface logic with a signal that defines the external clock cycle. For TTL compatibility, the rising edge of CLK is its significant edge. The Phase-Locked Loop mode is recommended for most system designs. In the other two modes, the internal phases are derived from input signals that bypass the PLL.

In the *external phase inputs* mode, the internal clock phases are controlled by the two external phase inputs, PHE1 and PHE2. These inputs are buffered to drive the internal clock distribution system. In the scan-test submode, both phases are stopped in an off (low) state, which is necessary to employ scan logic.

In the *external processor clock* mode, the internal clock phases are derived from a double-rail differential signal applied at the two phase inputs, PHE1 and PHE2. The differential signal operates at twice the frequency of CLK. The falling edge of PHE2 must occur before the rising edge of CLK. The signal is run through an external processor clock and buffer to produce the internal phases at the same frequency as the input. This mode allows bypassing the PLL for test purposes or to change the clock frequency, as when entering or leaving a low-power mode.

Unlike the Phase-Locked Loop mode, the other two modes operate the internal phases at the externally supplied frequency, rather than twice the external frequency. In order to allow these modes to be generated and controlled by an external phase-locked loop, both internal clock phases are output via buffers on the XPH1 and XPH2 signals, and an additional signal (XREF) that uses the same type of output buffer is provided for measuring the delay of the output buffers: the XREF output can be connected to the IREF input to measure the buffer's delay, so as to compensate for this delay in the external PLL.

Multiprocessing Support

The NexBus supports all signals and bus protocols needed for multiprocessing. Many types of devices can be interfaced to the NexBus, including a backplane, one or more processors (with optional numerics processors), one or more memory subsystems shared between processors, high-speed I/O devices, a NexBus Arbiter, and a system-logic interface between the NexBus and other system buses (called an *alternate-bus interface).* The multiprocessing hooks ensure cache coherency and guarantee that stale data is either invalidated in time or updated immediately. NexBus caching masters watch the address phase on the bus at all times to implement a Modified Write Once (MESI) protocol for cache coherency.

external phase-locked loop, both internal clock phases are output via buffers on the XPH1 and XPH2 signals, and an additional signal (XREF) that uses the same type of output buffer is provided for measuring the delay of the output buffers; the XREF output can be connected to the IREF input to measure the buffer's delay, so as to compensate for this delay in the external PLL.

Multiprocessing Support

The NexBus supports all signals and bus protocols needed for multiprocessing. Many types of devices can be interfaced to the NexBus, including a backplane, one or more processors (with optional numerics processors), one or more memory subsystems shared between processors, high-speed I/O devices, a NexBus Arbiter, and a system-logic interface between the NexBus and other system buses (called an *alternate-bus interface*). The multiprocessing hooks ensure cache coherency and guarantee that stale data is either invalidated in time or updated immediately. NexBus caching masters watch the address phase on the bus at all times, to implement a Modified Write Once (MESI) protocol for cache coherency.

Bus Operations

This chapter covers bus cycles and cache-coherency operations. The bus cycles are conducted primarily on NexBus although their effects can also be seen on the SRAM bus. In this chapter, the term "clock" refers to the *NexBus clock* not to the processor clock, as is meant elsewhere throughout this book. The NexBus clock, shown in the timing diagrams accompanying this text, runs at half the frequency of the processor clock.

Operations between the processor and the L2-cache SRAM, as well as operations between the processor and the Nx587 numerics processor on the NP bus are not described here, since these operations are not intended for system logic interfacing. Instead, a single-processor design example is provided near the end of the *Hardware Architecture* chapter in which the processor-to-SRAM and processor-to-587 connections are illustrated.

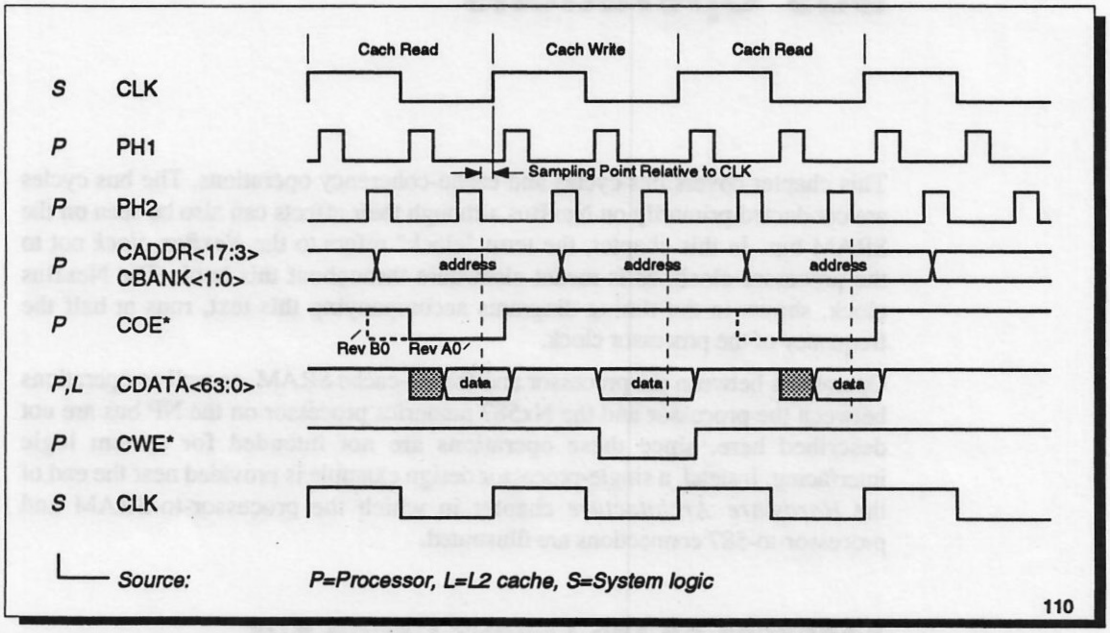
Accesses on the Level-2 Cache Bus

Figure 20 in the *Nx586 Features and Signals* chapter compares the basic timing for the processor, its L1 caches, and the L2 cache. An L1-cache miss may cause an access to the L2 cache, which resides off-chip on a dedicated 64-bit bus. Figure 29 shows a read, write, and read to the L2 cache. Transfers can begin on any processor clock and occur at the peak rate of eight bytes every two processor clocks.

The notation regarding *Source* in the left-hand column of Figure 29 indicates the chip or logic that output the signal. When signals are driven by multiple sources, all sources are shown, in the order in which they drive the signal. In some cases, signals take on different names as outputs are ORed in group-signal logic. In these cases, the signal source is shown with a subscript, where the subscript indicates the device or logic that originally caused the change in the signal.

While Figure 29 shows interleaved reads and writes, reads (or writes) can be back-to-back without dead cycles. The processor clock, which runs at twice the rate of the NexBus clock (CLK), is represented here by its two phases, PH1 and

PH2. These phases are not visible at the pins except through the delayed outputs, XPH1 and XPH2. The data-sampling point is shown as the falling edge of PH2, which is relative to the rising edge of CLK. Two versions of COE* are shown, one for Revision A of the Nx586 processor and another for Revision B. In both versions, transitions on COE* and CWE* occur on the rising edge of PH1.



Source: P=Processor, L=L2 cache, S=System logic

Figure 29 Level-2 Cache Read and Write

NexBus Arbitration and Address Phase

Processor operations on the NexBus may or may not begin with arbitration for the bus. To obtain the bus, the processor asserts NREQ*, LOCK*, and/or AREQ* to the NexBus Arbiter, which responds to the arbitration winner with GNT*. Automatic re-grant occurs when the NexBus Arbiter holds GNT* asserted at the time the processor samples it, in which case the processor need not assert NREQ*, LOCK*, or AREQ* and can immediately begin its operation.

NREQ*, when asserted, remains active until GNT* is received from the NexBus Arbiter, although during speculative reads the processor will deactivate NREQ* before GNT* is received if the transfer is no longer needed. In systems using the NxPC chip as the NexBus Arbiter, NREQ* is treated the same as AREQ*; when NexBus control is granted, control of all other buses is also granted at the same time.

LOCK* is asserted during sequences in which multiple bus operations should be performed sequentially and uninterruptedly. The signal is used by the NexBus Arbiter to determine the end of such a sequence. Cache-block fills are not locked; they are implicitly treated as atomic reads. Some NexBus Arbiters (but not the NxPC chip) may allow a master on another system bus to intervene in a locked NexBus transaction. To avoid this, the processor asserts AREQ*. LOCK* is typically software-configured to be asserted for read-modify-writes, explicitly locked instructions, page-table reads, or descriptor-table reads.

AREQ* is asserted to secure control not only of the NexBus but also of any other buses supported by the system. The signal always remains active until GNT* is received; unlike NREQ*, the processor does not make speculative requests for other system buses with AREQ*. If the processor does not know which bus its intended resource is on, it asserts NREQ*. If a GTAL is subsequently returned, the processor assumes the that resources are on another system bus and retries its transfer by asserting AREQ*.

When GNT* is received, the processor places the address of a qword (for memory operations) on NxAD<31:3> or the address of a dword (for I/O operations) on NxAD<15:2>. It drives status bits on NxAD<63:32> and asserts its ALE* signal to assume bus mastership and to indicate that there is valid address on the bus. The processor asserts ALE* for only one bus clock. The slave uses the GALE signal generated by system logic to enable the latching of address and status from the NexBus.

Single-Qword Memory Operations

Figure 30 shows the fastest possible single-qword read. The notation regarding *Source* indicates the logic that originated the signal as an output. In this figure and others to follow, the source of group-ORed signals (such as GXACK) is shown subscripted with a symbol indicating the device or logic that output the originally activating signal. For example, the source of the GXACK signal is shown as "S_P", which means that system logic (S) generated GXACK but that the processor (P) caused this by generating XACK*.

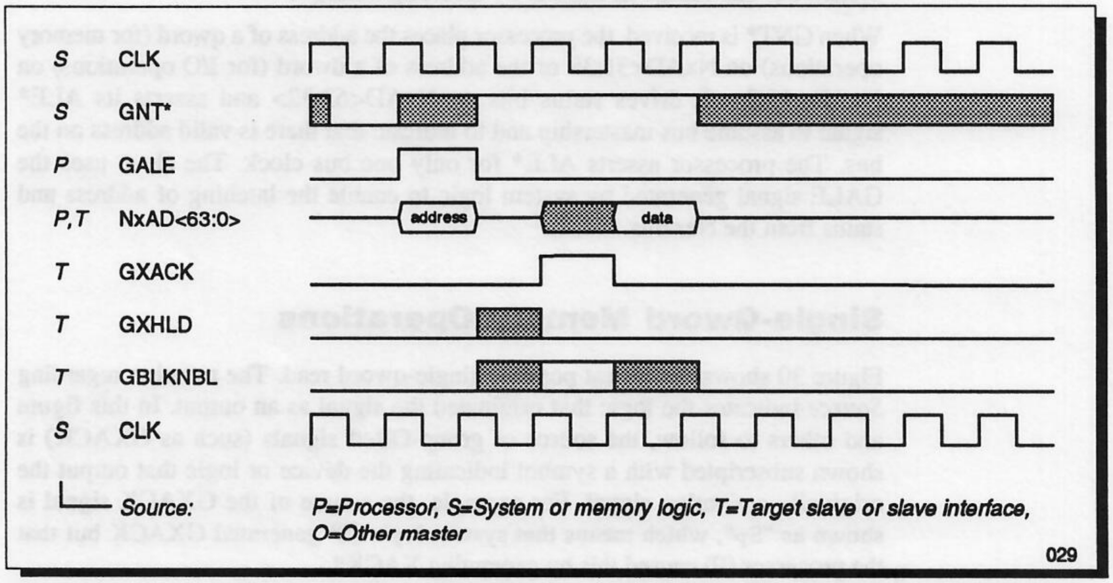
In some timing diagrams later in this section, bus signals take on different names as outputs cross buses through transceivers or are ORed in group-signal logic; in these cases, the source of the signals is shown subscripted with a symbol indicating the logic that originally output the activating signals.

The data phase of a fast single-qword read starts when the slave responds to the processor's request by asserting its XACK* signal. The processor samples the GXACK and GXHLD signals from system logic to determine when data is placed on the bus. The processor then samples the data at the end of the bus clock after GXACK is asserted and GXHLD is negated. The operation finishes with an idle phase of at least one clock.

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems.
NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

This protocol guarantees the processor and other caching devices enough time to recognize a modified cache block and to assert GDCL in time to cancel a data transfer. A slave may not assert XACK* until the second clock following GALE. However, the slave must always assert XACK* during or before the third clock following GALE, since otherwise the absence of an active GXACK indicates to the system-logic interface between the NexBus and other system buses (called the *alternate-bus interface)* that the address must reside on the other system bus. In that case, the system-logic interface to that other bus assumes the role of slave and asserts GXACK.

Figure 30 shows when GBLKNBL may be asserted. If appropriate, the slave must assert GBLKNBL no later than it asserts XACK*, and it must keep GBLKNBL asserted until it negates XACK*. It must negate GBLKNBL at or before it stops placing data on the bus. Although not shown, OWNABL must also be valid (either asserted or negated) whenever GXACK is asserted.



Source: *P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master*

Figure 30 Fastest Single-Qword Read

If the slave is unable to supply data during the next clock after asserting XACK*, the slave must assert its XHLD* signal at the same time. Similarly, if the processor is not ready to accept data in the next clock it asserts its XHLD* signal. The slave supplies data in the clock following the first clock during which GXACK is asserted and GXHLD is negated. The processor strobes the data at the end of that clock. A single-qword read with wait states is shown in Figure 31. For such an operation, the slave must negate XACK* after a single clock during which GXACK is asserted and GXHLD is negated, and it must stop driving data onto the bus one clock thereafter. The processor does not assert

XHLD* while GALE is asserted, nor may either party to the transaction assert XHLD* after the slave negates GXACK. In the case shown in Figure 31, the slave asserts GXACK at the latest allowable time, thereby inserting one wait state, and GXHLD is asserted for one clock to insert an additional wait state. The slave may or may not drive the NxAD<63:0> signals during the wait states. The processor will not drive them during the data phase of a read operation.

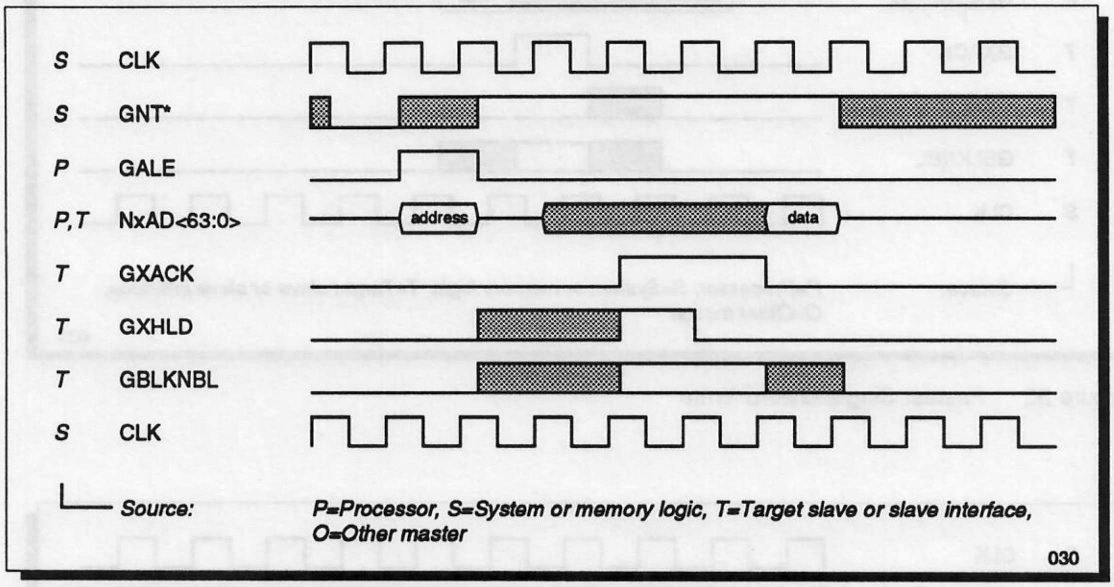
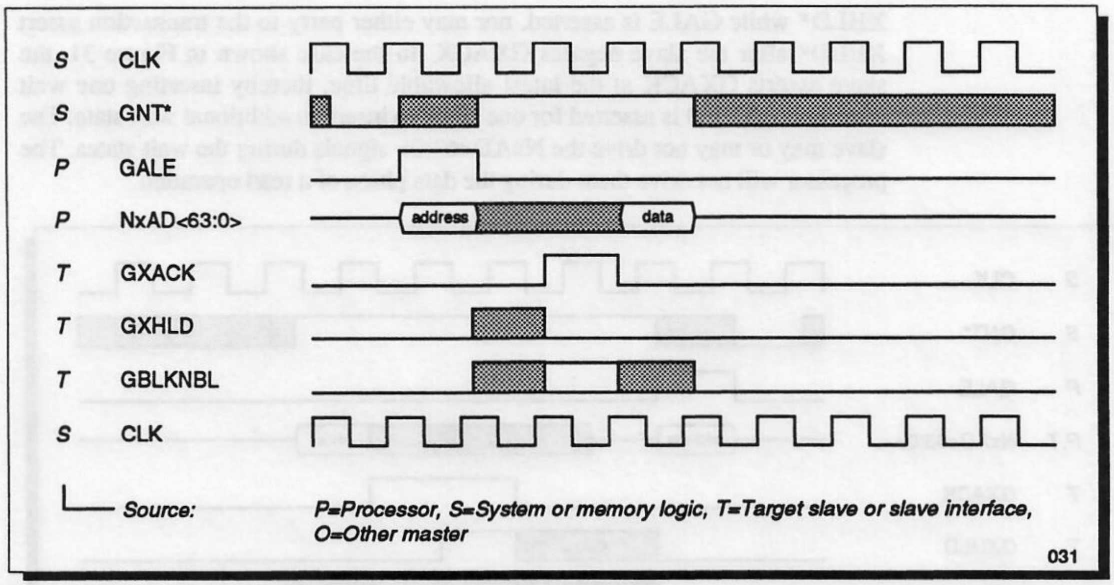

S CLK
S GNT*
P GALE
P,T NxAD<63:0>
address
data
T GXACK
T GXHLD
T GBLKNBL
S CLK
Source:
P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master
030

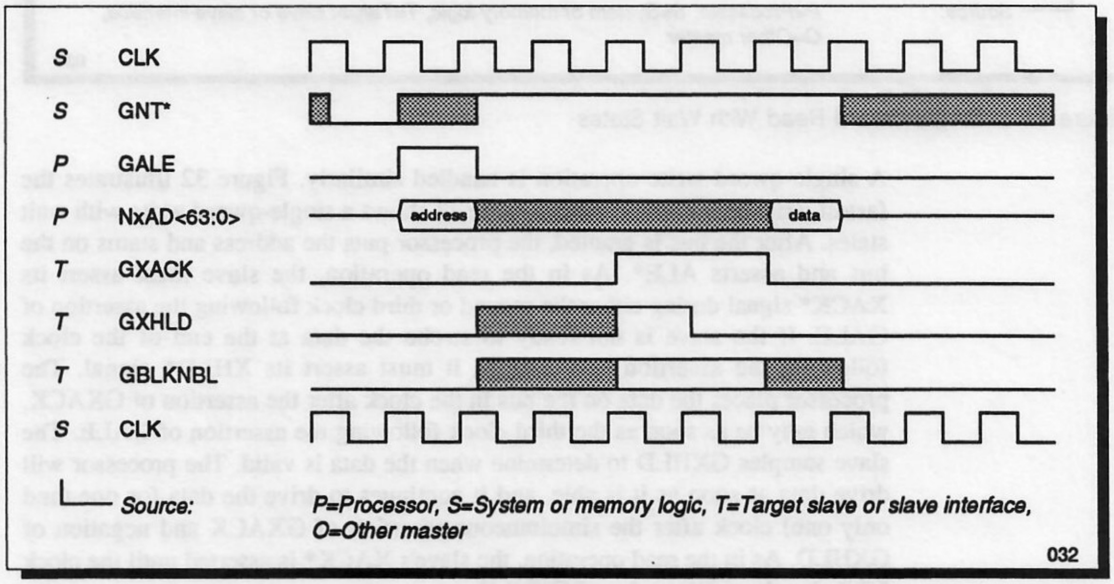
Figure 31 Single-Qword Read With Wait States

A single-qword write operation is handled similarly. Figure 32 illustrates the fastest write operation possible. Figure 33 shows a single-qword write with wait states. After the bus is granted, the processor puts the address and status on the bus and asserts ALE*. As in the read operation, the slave must assert its XACK* signal during either the second or third clock following the assertion of GALE. If the slave is not ready to strobe the data at the end of the clock following the assertion of GXACK, it must assert its XHLD* signal. The processor places the data on the bus in the clock after the assertion of GXACK, which may be as soon as the third clock following the assertion of GALE. The slave samples GXHLD to determine when the data is valid. The processor will drive data as soon as it is able, and it continues to drive the data for one (and only one) clock after the simultaneous assertion of GXACK and negation of GXHLD. As in the read operation, the slave's XACK* is asserted until the clock following the trailing edge of GXHLD.



Source: P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master

Figure 32 Fastest Single-Qword Write



Source: P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master

Figure 33 Single-Qword Write With Wait States

Burst Memory Operations

The processor performs burst operations with memory at a much higher bandwidth than the single-qword operations described in the previous section. Bursts, both reads and writes, are done only in four-qword increments. All burst reads are cache fills.

Burst reads and writes are indicated by the assertion of one or both of the BLKSIZ<1:0>* bits during the address/status phase of the bus operations, as previously defined for single-qword operations. For example, a four-qword burst read or write is indicated by BLKSIZ<1:0>* = 01 during the address phase.

A burst operation consists of a single address phase followed by a multi-transfer data phase. The data transfer may begin with *any* qword in the block, as indicated by the address bits, but it then proceeds through the other qwords of the specified contiguous data in an order that is defined by the expression

qword-address XOR count

where "XOR" represents bit-wise exclusive-OR, and "count" is initialized to 0 and incremented by one for each qword transferred. If NxAD<31:3> is 6 in a block operation, qword 6 is first transferred then qword 7, then qword 4, and finally qword 5, while if NxAD<31:3> is 7, qword 7 is first addressed, then qword 6, then qword 5, and then qword 4. Thus, the data are transferred in an order that always wraps around to transfer the entire burst containing the starting address, even though the processor will request the most urgently needed qword first.

Figure 34 shows the fastest possible block read (no wait states). A block read proceeds as follows: When it is granted the NexBus, the processor drives the address and status on the bus, indicating a burst operation by setting NxAD<51:50> (BLKSIZ<1:0>*) to 01. After accessing the first qword, the slave must step the address to sequentially access the subsequent qwords of the burst, using the algorithm defined above. This algorithm implements *read-forward* cache loading by causing all the data to be accessed, regardless of the size of burst specified. For example, a block operation addresses in sequence all combinations of address bits <4:3>, the bits specifying qwords within a burst of four qwords.

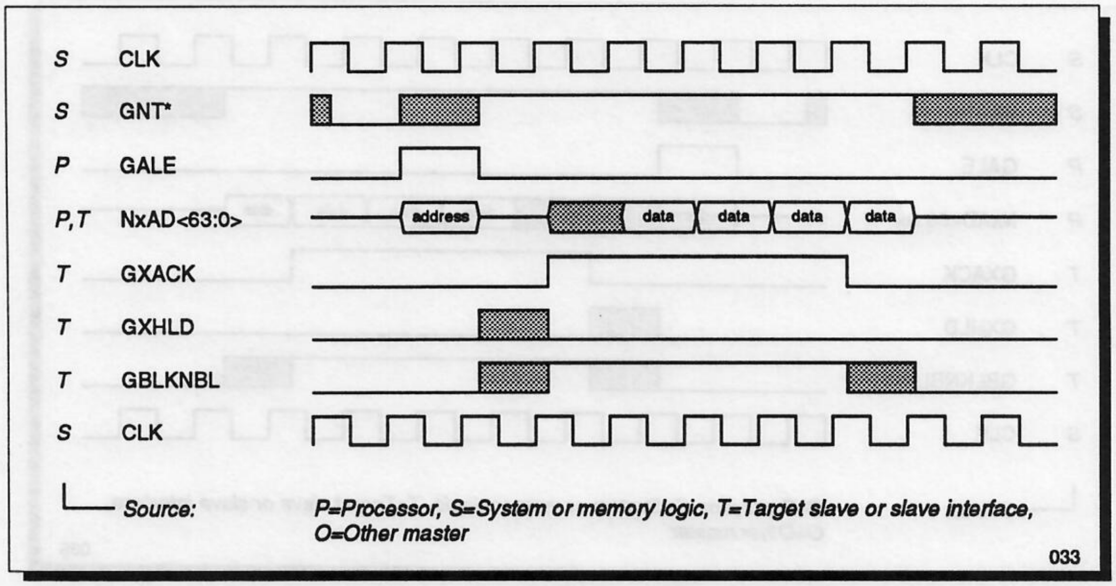
Following the address phase, the slave recognizes its address and so asserts its XACK* in either the second or the third clock following GALE. Devices that are capable of burst operations always assert GBLKNBL, at or before the assertion of XACK*, and keep it asserted throughout the data transfer phase of the operation. If the slave is located on another system bus, the system-logic interface to that bus must either negate GBLKNBL or it must simultaneously assert GBLKNBL and SHARE* and negate OWNABL whenever it asserts XACK*.

When the slave asserts XACK*, it must keep it active through the last qword transfer. The continued assertion of GXACK by system logic indicates that the operation is not finished, and so prevents re-arbitration of the bus. While idle clocks are not normally inserted between the qwords of a burst transfers, they can be if necessary. Either the processor or the slave can assert its XHLD* to cause one wait state to be inserted into the transfer during the following clock.

For single-qword reads, the transfer of data takes place at the end of each clock following the simultaneous assertion of GXACK and negation of GXHLD. In the clock following the assertion of GXACK and negation of GXHLD, the processor starts sampling data from the bus. The slave must also recognize the assertion of GXACK and negation of GXHLD and supply a qword of data during the following clock, then remove the transferred qword and prepare to supply the next qword. The slave must negate its XACK* signal in the same clock that the last qword is sampled.

Figure 34 shows the GBLKNBL (cacheable) signal being asserted. If the slave is a cacheable main-memory device, it must assert GBLKNBL whenever it asserts XACK* and it must support block reads and writes.

If the slave is a non-main-memory device that does not assert its GBLKNBL signal, such as a memory-mapped I/O device or a video RAM, the processor will not cache the data. Instead the processor simply samples a single qword and does not update any cache locations. The non-cacheable slave may ignore the BLKSIZ<1:0>* signals and always perform a single-qword read or write. Thus, the slave will not assert its XACK* for more than a single qword transfer. In effect, the burst read or write initiated by the processor is turned into a single-qword operation. In any single-qword read, the byte-enable bits, which are ignored for a burst read, determine which data bytes must be returned to the processor. A non-cacheable slave may drive any bytes for which the byte-enable bits are negated, but they will not be sampled by the processor.



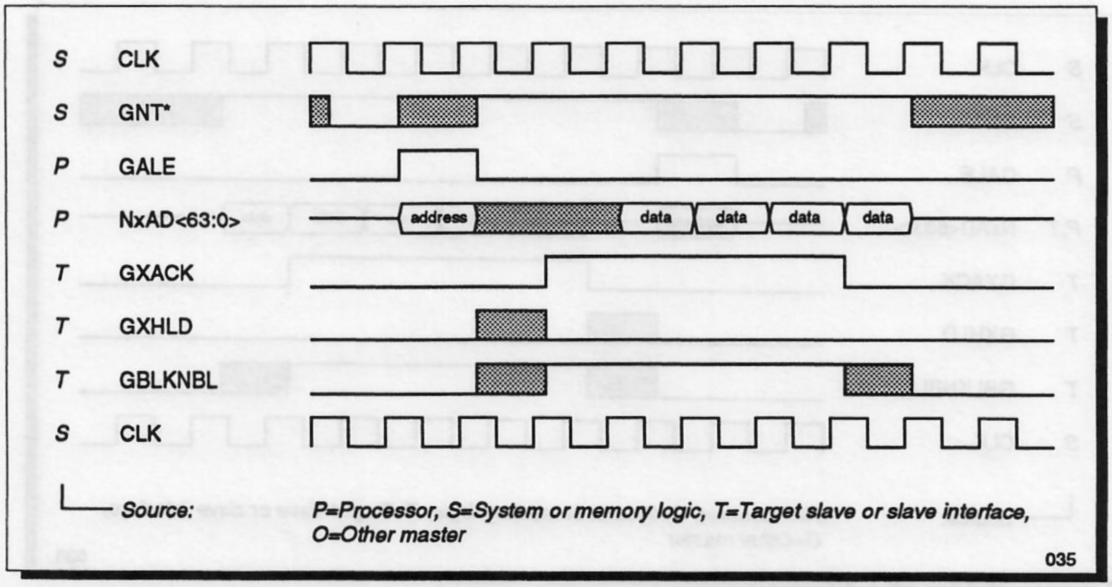
P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master

Figure 34 Fastest Block Read

Burst writes are similar to a burst reads. Figure 35 shows the timing for the fastest possible block write (no wait states).

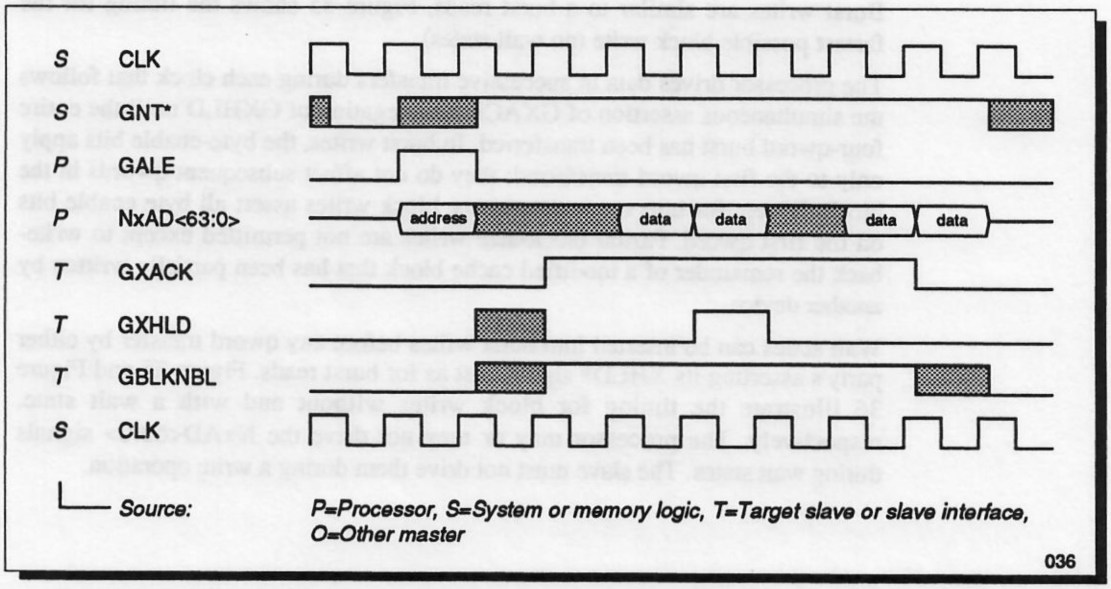
The processor drives data in successive transfers during each clock that follows the simultaneous assertion of GXACK and negation of GXHLD until the entire four-qword burst has been transferred. In burst writes, the byte-enable bits apply only to the first qword transferred; they do not affect subsequent qwords in the block. Except for intervenor operations, block writes assert all byte enable bits on the first qword. Partial block-size writes are not permitted except to write-back the remainder of a modified cache block that has been partially written by another device.

Wait states can be inserted into burst writes before any qword transfer by either party's asserting its XHLD* signal, just as for burst reads. Figure 35 and Figure 36 illustrate the timing for block writes without and with a wait state, respectively. The processor may or may not drive the NxAD<63:0> signals during wait states. The slave must not drive them during a write operation.



Source: P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master

Figure 35 Fastest Block Write



Source: P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master

Figure 36 Block Write With One Wait State

I/O Operations

I/O operations on the NexBus are performed exactly like single-qword reads and writes, with three exceptions. First, the I/O address space is limited to 64K bytes. Second, the 16-bit I/O address is broken into two fields: fourteen address bits and four byte-enable bits. I/O addresses do not use BE<7:4>* (which must be set to all 1's) but instead specify a quad address on NxAD<2>. Third, data is always transferred on NxAD<31:0>, and NxAD<63:32> is undefined during the data transfer phase of an I/O operation.

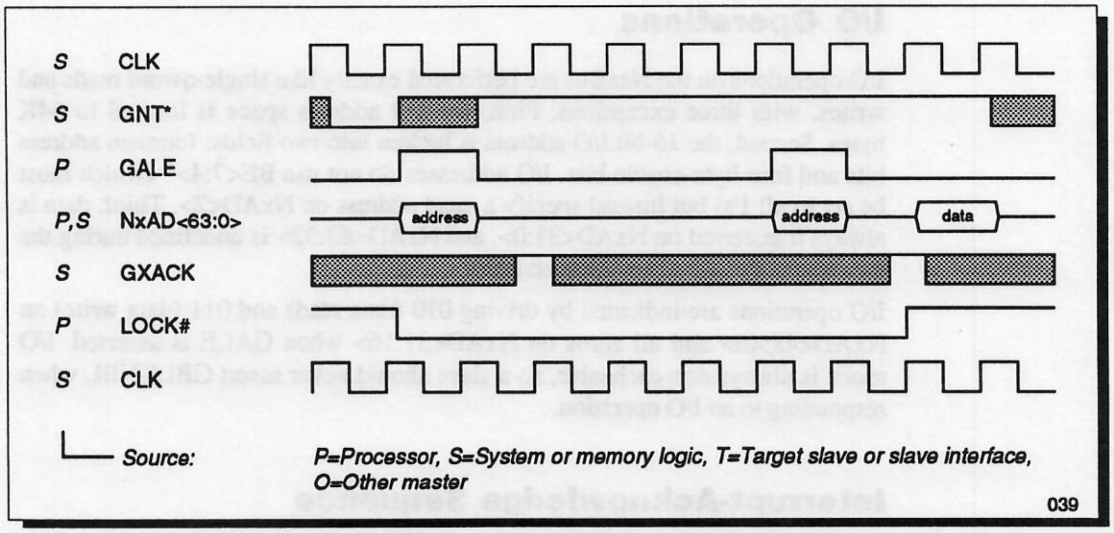
I/O operations are indicated by driving 010 (data read) and 011 (data write) on NxAD<48:46> and all zeros on NxAD<31:16> when GALE is asserted. I/O space is always non-cacheable, so a slave should never assert GBLKNBL when responding to an I/O operation.

Interrupt-Acknowledge Sequence

When an interrupt request is sensed by external interrupt-control logic, the request is signaled to the processor by the control logic, the processor acknowledges the interrupt request (during which sequence the controller passes the interrupt vector), and the processor services the interrupt as specified by the vector. The hardware mechanism is described above in the *Hardware Architecture* chapter.

An interrupt-acknowledge sequence, shown in Figure 37, consists of two back-to-back locked reads on NexBus, where the operation type (NxAD<48:46>) is 000 and the byte enable bits BE<7:0>* = 11111110. The first (synchronizing) read is used latch the state of the interrupt controller. It is indicated by NxAD<2> = 1 (I/O-byte address 4). The second read is used to transfer the 8-bit interrupt vector on NxAD<7:0> to the processor, which uses it as an index to the interrupt service routine. This read is indicated by NxAD<2> = 0 (I/O-byte address 0). During these two reads only the least significant bit of the address field is driven to a valid state. The most significant bits are undefined. After the interrupt is serviced, the request is cleared and normal processing resumes.

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems.
NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.



Source: P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master

Figure 37 Interrupt-Acknowledge Cycle

Halt and Shutdown Operations

Halt and shutdown operations are signaled on the NexBus by driving 001 on NxAD<48:46> during the address/status phase, as shown in Figure 38. The halt and shutdown conditions are distinguished from one another by the address that is simultaneously signaled on the byte-enable bits, BE<7:0>* on NxAD<39:32>. The processor does not generate a data phase for these operations.

Type of Bus Cycle	*NxAD<48> M/IO**	*NxAD<47> D/C**	*NxAD<46> W/R**	*NxAD<39:32> BE<7:0>**	*NxAD<31:3>*	*NxAD<2>*
Halt	0	0	1	11111011	all zeros	0
Shutdown	0	0	1	11111110	all zeros	0

Figure 38 Halt and Shutdown Encoding

For the halt operation, the processor places an address of 2 on the bus, signified by BE<7:0>* bits (NxAD<39:32>) = 11111011. NxAD<2> = 0 and NxAD<31:3> are undefined. After this, the processor remains in halted state until NMI*, RESETCPU*, or RESET* becomes active.

For the shutdown operation, the processor places an address of 0 on the bus, signified by BE<7:0>* bits (NxAD<39:32>) = 11111110. NxAD<2> = 0 and NxAD<31:3> are undefined. After this, the processor performs a soft reset equivalent to the assertion of RESETCPU*; that is, the processor is reset, but the memory contents, including modified cache blocks, are retained.

The encoding shown in Figure 38 matches that of the Intel i486 processor, but since the Nx586 processor has a 64-bit data bus rather than the i486 processor's 32-bit data bus, four additional byte-enable bits (BE<7:4>*) are specified for the most-significant dword of the bus. These four additional bits are all high.

Cache-Control OperationsControl

Single-qword operations with BE<7:0>* all negated are *no-ops*, since they address no bytes. The processor does not generate such operations. However, the single-qword memory-reference codes with BE<7:0>* = 11111111 (all negated) are defined for broadcast cache-control operations in systems with multiple NexBus caching devices. For such systems, backplane logic can generate these operations to sync, write-back and invalidate, or invalidate cache blocks. Implementation of these codes and conventions in NexBus caching devices is optional but strongly recommended:

- *Sync*—A single-qword read with OWN* and BE<7:0>* negated (a 0-byte non-OWN* read) forces all caches that have the addressed block in a modified state to write it back to memory. Thereafter, the caching device can change the state of the block to shared and retain the copy.
- *Write-Back and Invalidate*—A single-qword read with OWN* asserted and BE<7:0>* negated (a 0-byte OWN read) forces all caches that have the addressed block in a modified state to intervene in the current bus operation and write the block back to memory. Thereafter, all caching devices with that block invalidate the block.
- *Invalidate*—A single-qword write with OWN* asserted and BE<7:0>* negated (a 0-byte OWN write) forces all caches that have the addressed block, no matter what the state, to invalidate it without writing a modified version back to memory.

Obtaining Exclusive Use Of Cache Blocks

The processor can obtain ownership of a cache block either *preemptively* or *passively*. Preemptive ownership is gained by asserting OWN* during the address/status phase of a read or write operation. Whenever the processor needs to write a cache block that is either cached in the shared or invalid state, it performs a preemptive read-to-own operation by asserting OWN* during a single-qword or four-qword block read.

Passive ownership is normally gained when the processor performs a block read, because other NexBus caching devices must snoop block reads. If any part of a block addressed by the processor's read operation resides in another NexBus device's cache, regardless of state, that device asserts SHARE* after the assertion of GALE but not later than the clock during which the first qword of the block is transferred. SHARE* remains asserted through the entire data transfer. If the processor sees GSHARE negated during a block read when it samples the first qword of the block, it knows that it has the only copy. It can therefore cache the block in the exclusive state rather than the shared state, if and only if OWNABL is asserted by system logic.

If another NexBus caching device is unable to meet this timing in the fastest possible case, it must assert XHLD* to delay the operation until it is able to perform the cache check. While it is possible to put a caching device on NexBus that is unable to check its cache and report SHARE* correctly, but instead always asserts SHARE*, this has a very negative effect on system efficiency. It is also possible to design a device that invalidates its cache block during any block read hit, in which case only the efficiency of that one device is impaired.

If the processor addresses a non-cacheable block on a system bus other than NexBus, the system-logic interface between the NexBus and the other system bus (called the *alternate-bus interface)* must indicate this by negating GBLKNBL, and it may not perform block reads or writes to such a block. If the block on the other bus is cacheable, it can only be cached in the shared state, since standard system buses (such as VL bus and ISA bus) do not support the MESI caching protocol, and it is not possible to cache their memory addresses in the exclusive state.

The OWNABL signal from system logic is used to indicate cacheability of locations on other system buses. Whenever OWNABL is negated during a bus operation, the processor will not cache the block in the exclusive state even if the processor asserted OWN*; instead, it may cache the block in the shared state if other conditions permit it.

GBLKNBL and GSHARE must be asserted by system logic at the same time that OWNABL is negated. The timing of these three signals is identical: they should be valid whenever GXACK is asserted. They may be (but need not be) asserted ahead of XACK*, and may (but, except for GSHARE, need not) be held one clock after the negation of XACK*. This timing differs from that of GSHARE, since when OWNABL is asserted GSHARE is not required to be valid until the clock following the negation of GXHLD—i.e., coincident with the data transfer.

Intervenor Operations

The examples given above assume that the addressed data does not reside in a modified cache block. When an operation by another NexBus master results in a cache hit to a modified block in the processor, the processor intervenes in the operation by asserting DCL*. The timing for DCL* is the same as that for SHARE*: the NexBus master samples GDCL on the same clock in which it samples NexBus data. An asserted GDCL indicates to the master that data cached by the processor is modified. To meet the fastest timing requirements, the processor asserts DCL* no later than the third clock following the assertion of GALE. If a MESI write-back caching device is unable to determine in a timely manner whether a transaction hits in its cache, it must assert XHLD* to delay the transfer.

If a block write operation by another master hits a modified cache block in the processor, the processor does not assert DCL*, since such a block write replaces all of a cache block. Instead, the processor either invalidates the block or, if either the master negates OWN* or system logic negates OWNABL when it asserts GXACK, the processor asserts SHARE*, updates its cache block with the write data on NexBus, and changes the cache block to shared state.

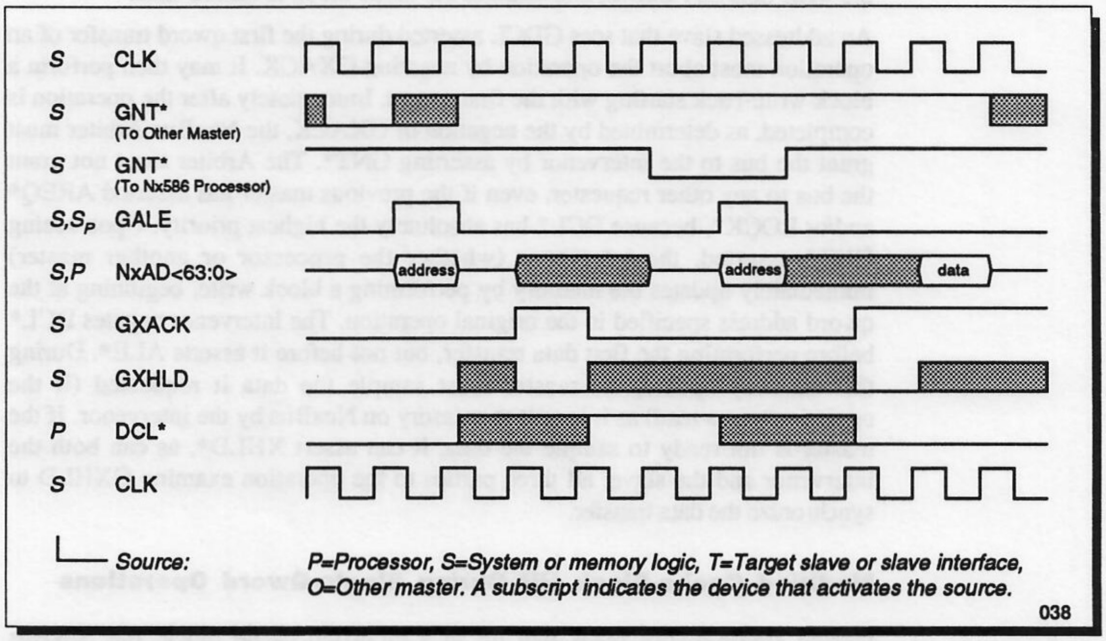
An addressed slave that sees GDCL asserted during the first qword transfer of an operation must abort the operation by negating GXACK. It may then perform a block write-back starting with the first qword. Immediately after the operation is completed, as determined by the negation of GXACK, the NexBus Arbiter must grant the bus to the intervenor by asserting GNT*. The Arbiter must not grant the bus to any other requester, even if the previous master has asserted AREQ* and/or LOCK*, because DCL* has absolutely the highest priority. Upon seeing GNT* asserted, the intervenor (whether the processor or another master) immediately updates the memory by performing a block write, beginning at the qword address specified in the original operation. The intervenor negates DCL* before performing the first data transfer, but not before it asserts ALE*. During this memory update, the master must sample the data it requested (if the operation was a read) as it is sent to memory on NexBus by the intervenor. If the master is not ready to sample the data, it can assert XHLD*, as can both the intervenor and the slave; all three parties to the operation examine GXHLD to synchronize the data transfer.

Modified Cache-Block Hit During Single-Qword Operations

During single-qword reads that hit in a modified cache block, the NexBus sequence looks like a normal single-qword read from the memory followed by a block write by the intervenor. Figure 39 illustrates the timing. The fastest time is shown for the operation, while both the fastest and slowest possible times are shown for the leading edge of GDCL. For a slow device intervening in a fast operation, GDCL is available to be sampled on the same clock as the first qword of data is available.

In Figure 39, two sources are shown for GALE and NxAD<63:0>, and one source (Sp) has a subscript. The source is the chip or logic that outputs the signal. The subscript for the source indicates the chip or logic that originally caused the change in the signal. In systems that use the NxPC chip for system and memory control, the source labeled "S" is the NxPC chip or other system logic.

During single-qword writes, the master with the modified cache block asserts DCL* to indicate that the single write will be followed by a block write. If the single write included only some of the bytes of the qword, the intervenor records this fact, and during the subsequent block write it outputs byte-enable bits indicating the other bytes of the qword. For example, if the byte-enable bits of the single write were 00000111, the intervenor outputs 11111000. In other words, the intervenor updates only those bytes that were not written by the master. Except for such intervening write-back operations, block writes must have all byte-enable bits asserted (00000000). During block write-backs, byte-enable bits apply only to the first qword, so all bytes of the final three qwords are written.



Source: P=Processor, S=System or memory logic, T=Target slave or slave interface, O=Other master. A subscript indicates the device that activates the source.

Figure 39 Single-Qword Read Hits Modified Cache Block

Modified Cache-Block Hit During Four-Qword (Block) Operations

As described above for single-qword operations, a block read by another NexBus master may hit a modified cache block in the processor. When this happens, the processor responds exactly as for a single-qword operation: it asserts DCL*, waits for the assertion of GNT* following the negation of GXACK, and proceeds with a block write-back. It writes the entire four-qword block back to memory. The original bus master must sample the data in this second block operation while it is transferred to memory. The master may insert wait states by asserting XHLD*. Since the processor, as intervenor, begins its write-back with the address requested by the master, if the original block read is a four-qword operation, the master can intercept the data as it is transferred to memory and find it in the expected order.

Block writes can hit in a modified or exclusive cache block only if the operation was initiated by the DMA action of a disk controller, not by the processor. Since only complete block writes are permitted, no write-back is required and the processor invalidates its cache block.

Programming

The following sections are intended as a quick guide to the basic characteristics of the x86 programmer's model. Full descriptions of the programmer's model, system data structures, and the instruction set can be found in the commercial publications listed in the *Preface* of this book.

Operating Modes and Privilege Levels

System software can select one of three execution modes that accommodate mixtures of 16-bit and 32-bit code. Each mode has one or more privilege levels associated with its code and data:

- *Protected Virtual-Address Mode (Protected Mode)*—In this full-capability mode, the processor supports a linear memory space of 4 GB, a virtual-memory space up to 64 terabytes, and operand sizes of 16 or 32 bits. All segmentation, paging, multitasking, and protection functions are available. Virtually all binary programs that run on the Intel 80286, i386, and i486 processors will run in protected mode on the Nx586 processor. In Protected Mode, programs run at privilege level 0, 1, 2, or 3. Typically, application programs run at privilege level 3, the operating system runs at privilege levels 0 and 1, and privilege level 2 is available to system software for other uses. In multitasking, access to tasks is managed on the basis of privilege level.
- *Virtual-8086 Mode*—In this mode, programs written for the Intel 8086, 8088, 80186 or 80188 processor can run as a task under Protected Mode, with the support of the processor's paging (but not segmentation) functions. The processor supports a linear memory space of 1 MB and operand sizes of 16 (default) or 32 bits (with instruction prefixes) and generates 8086 real-mode addresses. Programs enter and leave Virtual-8086 Mode from Protected Mode, under which they run as protected tasks at privilege level 3.
- *8086 Real-Address Mode (Real Mode)*—In this mode, which is entered after reset or powered up, the processor supports a linear memory space of 1 MB and operand sizes of 16 (default) or 32 bits (with instruction prefixes).

Interrupt handling and address generation are nearly identical to the Intel 80286 processor's real mode. Segmentation and paging are not supported, so all programs run at privilege level 0.

The privilege levels maintained by system software implement protections for special instructions (such as those that change control registers), for memory accesses to certain segments, and use of I/O instructions or port accesses. The four privilege levels are shown in Figure 40.

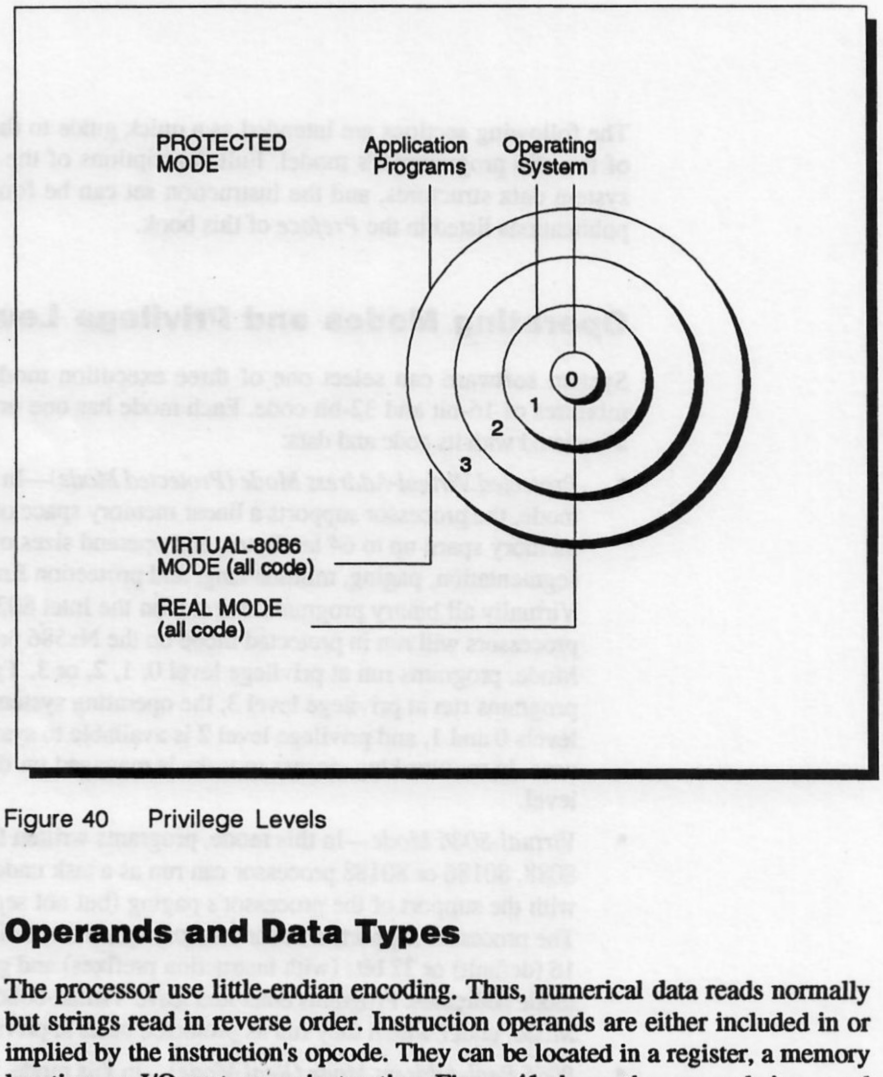


Figure 40 Privilege Levels

Operands and Data Types

The processor use little-endian encoding. Thus, numerical data reads normally but strings read in reverse order. Instruction operands are either included in or implied by the instruction's opcode. They can be located in a register, a memory location, an I/O port, or an instruction. Figure 41 shows the operand sizes used in the Nx586 processor.

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems. NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

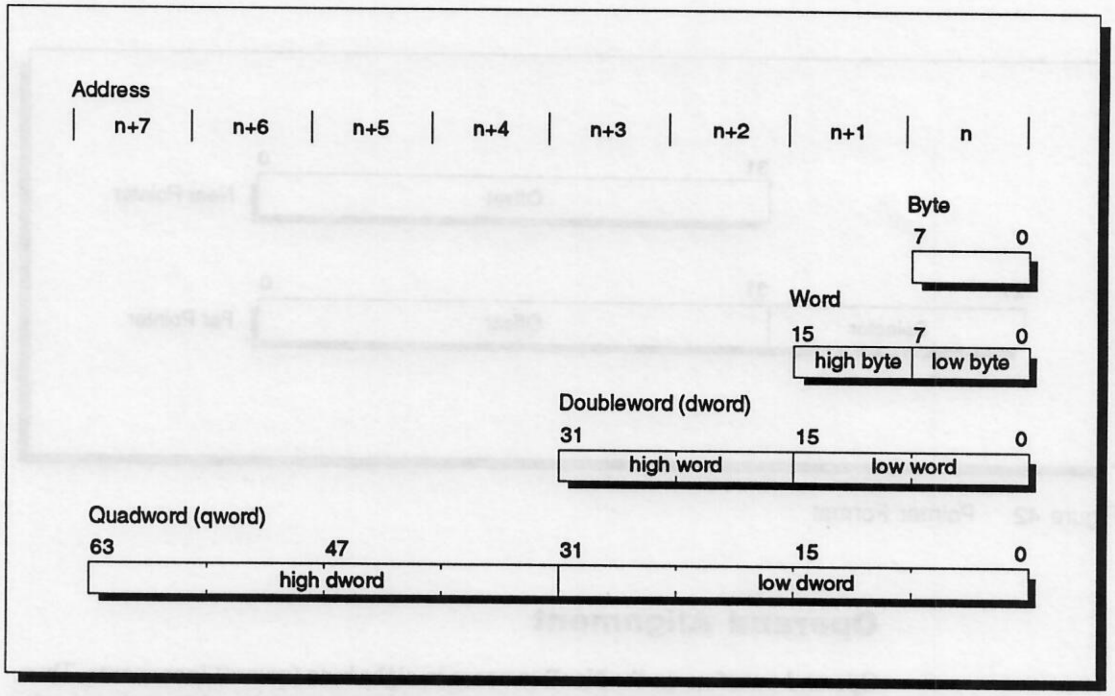


Figure 41 Operand Sizes

Operand size is a function of the instruction and/or the processor's execution mode. Short operands are always one byte in length. In Real Mode and Virtual-8086 Mode, the default size for long operands is one word (two bytes). In Protected Mode, the default size for long operands is either one word or one dword (four bytes), as determined by a bit in the code-segment descriptor. Qwords (four-byte quadwords) are not used in instructions, although the Nx586 processor operates on qwords as a means of increasing performance.

The data types that can be manipulated include:

- Unsigned integers
- Signed (two's-complement) integers
- Binary-coded decimal (BCD) numbers
- ASCII Strings
- Bit Strings
- Pointers

Figure 42 shows the formats for near and far pointers.

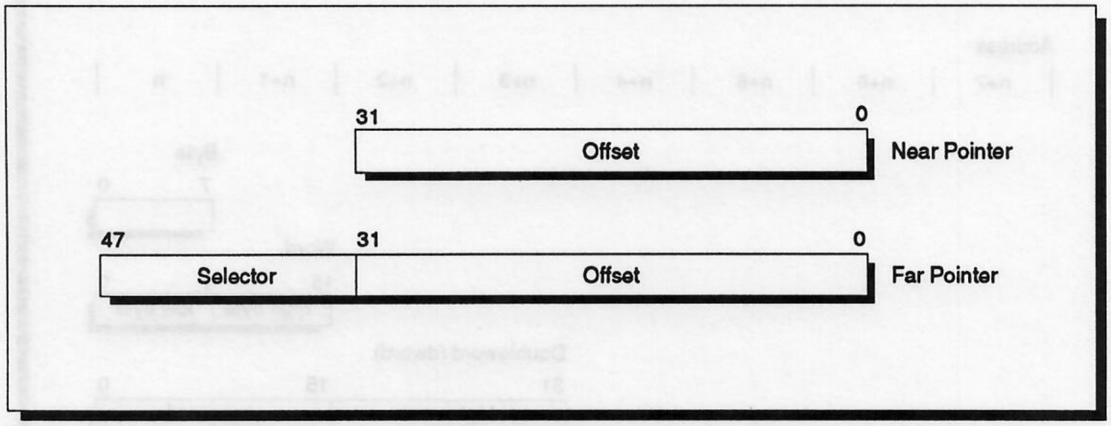


Figure 42 Pointer Format

Operand Alignment

Operand transfers on the NexBus occur in eight-byte (qword) increments. Thus, operands are aligned if all of their bytes fall within a qword boundary in the memory space. Figure 43 shows a few examples of aligned operands. Floating-point operands can be up to ten bytes in length, so those that exceed eight bytes cannot be aligned and will require two bus transfers.

For the highest performance from systems software, it is important that segment descriptors (which are eight bytes long) be aligned to qword boundaries since these descriptors are loaded frequently during certain common operations such as interrupts.

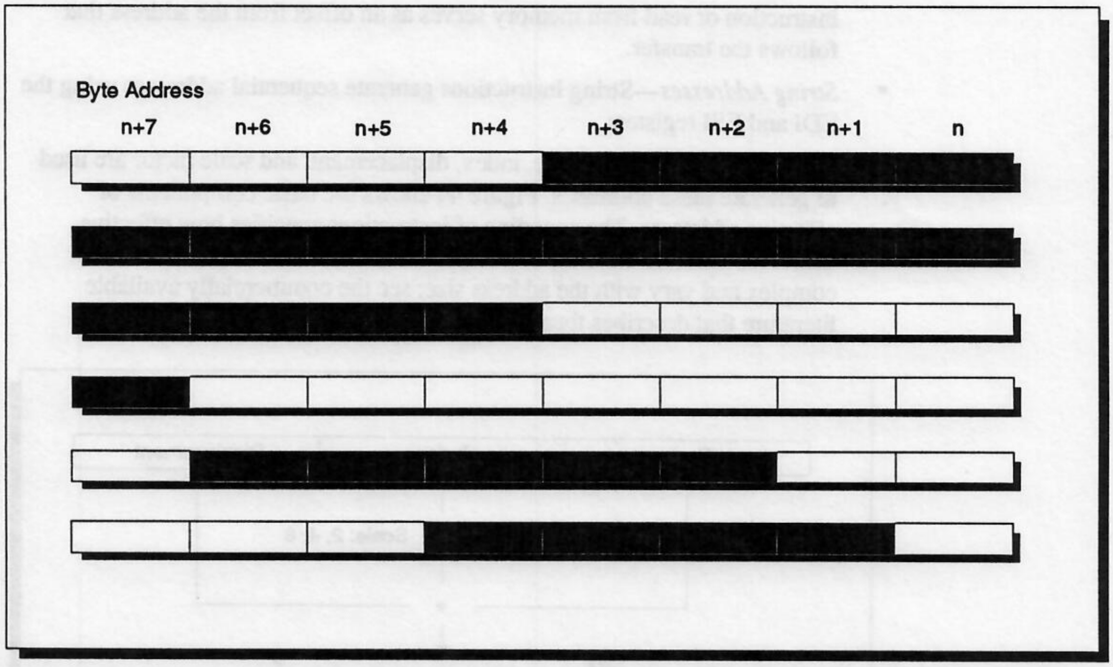


Figure 43 Examples of Aligned Operands

Addresses

The size (16-bit or 32-bit) of addresses generated by the processor is determined by the same bit in the code-segment descriptor that determines operand size. In Real Mode and Virtual-8086 Mode, the default size is 16 bits. In Protected Mode, the default size can be either 16 or 32 bits. The address-size instruction prefix size can be used to override these defaults. For instructions that transfer control, the size of the target address and displacement field are determined by the code-segment descriptor and the operand-size prefix (rather than address-size prefix).

Effective addresses are calculated by the processor prior to segmentation and paging translations. There are six methods for generating these effective addresses:

- *Absolute Addresses*—Some instructions contain a displacement into the data segment (DS) that points to a memory location.
- *Stack Addresses*—PUSH, POP, CALL, RET, and INT instructions generate addresses on the stack.
- *Instruction-Relative Addresses*—These addresses are generated by control-transfer instructions so as to reach their target. A displacement in the

instruction or read from memory serves as an offset from the address that follows the transfer.

- *String Addresses*—String instructions generate sequential addresses using the EDI and ESI registers.
- *Complex Addresses*—A base, index, displacement, and scale factor are used to generate these addresses. Figure 44 shows the basic components of effective addresses. The encoding of instructions specifies how effective addresses are to be calculated. The details of this addressing modes are complex and vary with the address size; see the commercially available literature that describes them.

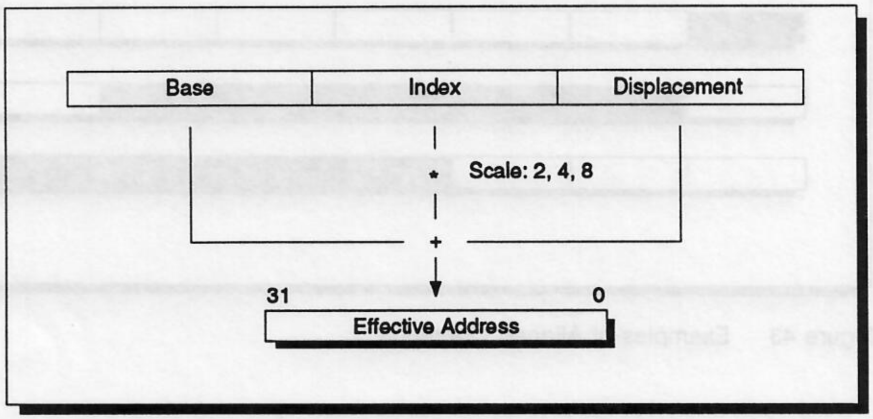


Figure 44 Complex Address Calculation (Protected Mode)

After the effective address is calculated by one of the above methods, it is used along with the segment selector in the segmentation translation process. This produces a linear address. If paging is enabled, the page translation uses page directories and page tables created and maintained by the operating system to arrive at a physical address; if paging is disabled, the linear address is used as the physical address. Figure 45 shows the flow.

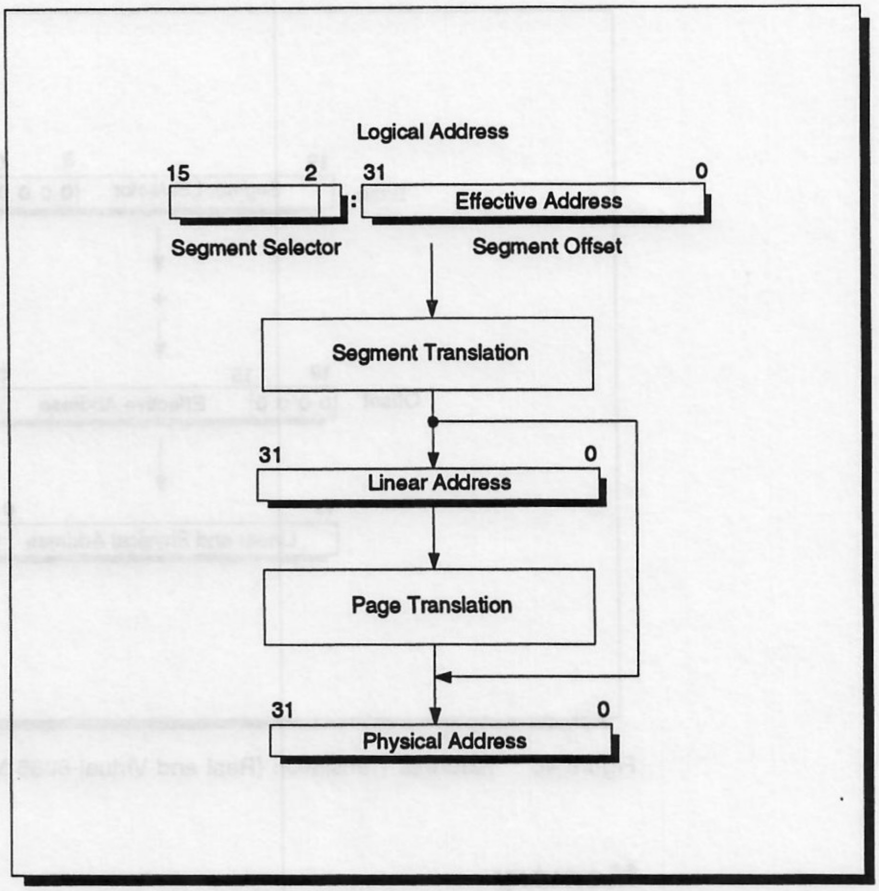


Figure 45 Address Translation (Protected Mode)

In Real Mode and Virtual-8086 Mode, the segment selector is simply multiplied by 16 to obtain the base address of a segment, and the effective address is added to obtain the linear and physical address, as shown in Figure 46. The memory space is thus divided into 64kB segments with no segment-level protection.

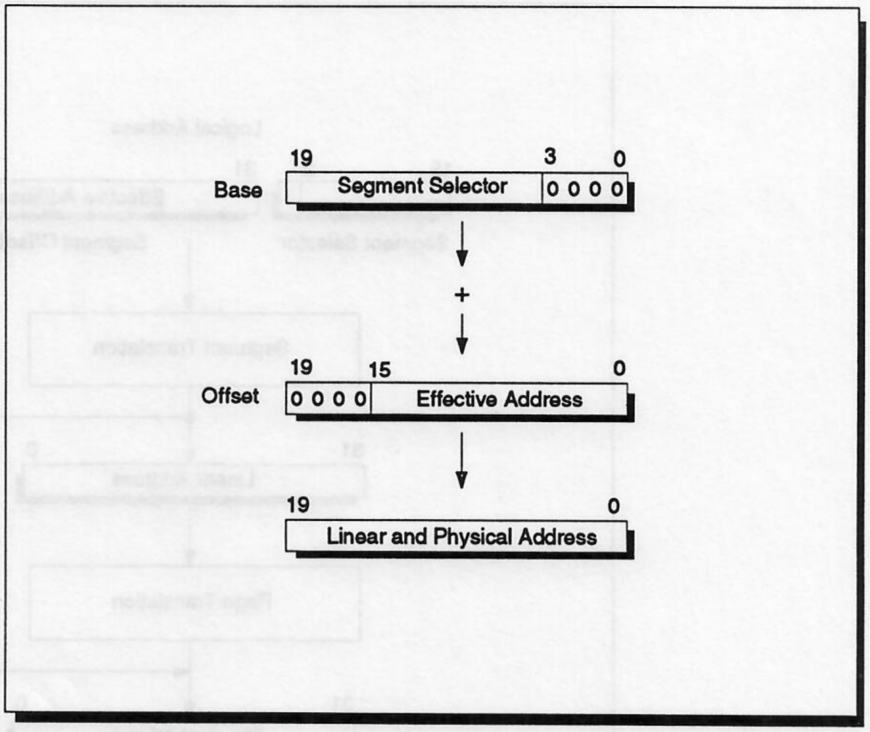


Figure 46 Address Translation (Real and Virtual-8086 Mode)

Memory

In Protected Mode, the 4-gigabyte physical memory space is divided into segments (which can overlap) by the operating system so as to create up to 64 terabytes of virtual memory space. Each program can have up to 16,383 segments of up to six types: code, data, stack, and three extra segments. The logical addresses in instructions identify a segment. The base (starting) address of the segment is obtained indirectly (via a system-software descriptor) from the value (called a *selector)* in the segment register for the segment. The remainder of the logical address (called the *effective address)* serves as an offset into that segment. The segment descriptors maintained by system software specify the required privilege level for access to the segment. Figure 47 shows the arrangement.

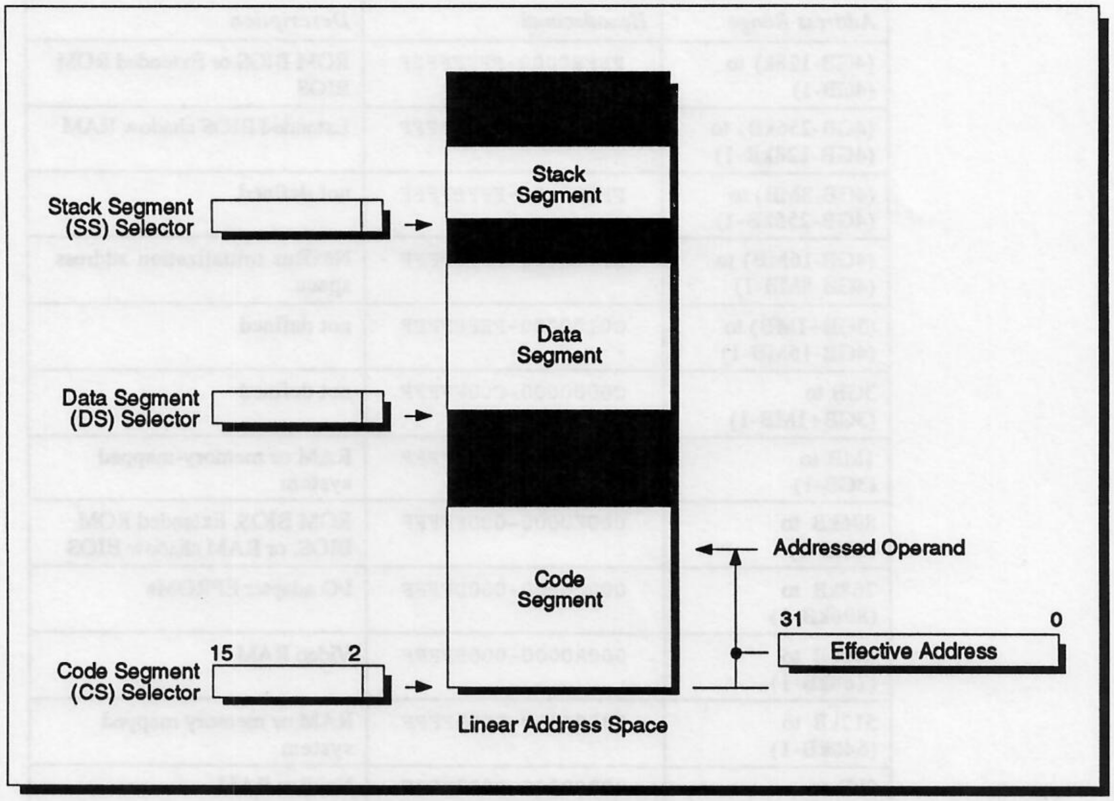


Figure 47 Segmented Memory

The address map for NexBus-compliant physical memory is shown in Figure 48. This map is more detailed than that specified by standard x86 architecture but it nevertheless conforms to the standard architecture.

Address Range	*Hexadecimal*	*Description*
(4GB-128k) to (4GB-1)	FFFE0000-FFFFFFFF	ROM BIOS or Extended ROM BIOS
(4GB-256kB) to (4GB-128kB-1)	FFFC0000-FFFDFFFF	Extended BIOS shadow RAM
(4GB-8MB) to (4GB-256kB-1)	FF800000-FFFBFFFF	not defined
(4GB-16MB) to (4GB-8MB-1)	FF000000-FF7FFFFF	NexBus initialization address space
(3GB+1MB) to (4GB-16MB-1)	C0100000-FEFFFFFF	not defined
3GB to (3GB+1MB-1)	C0000000-C00FFFFF	not defined
1MB to (3GB-1)	00100000-BFFFFFFF	RAM or memory-mapped system
896kB to (1MB-1)	000E0000-000FFFFF	ROM BIOS, Extended ROM BIOS, or RAM shadow BIOS
768kB to (896kB-1)	000C0000-000DFFFF	I/O adapter EPROMs
640kB to (768kB-1)	000A0000-000BFFFF	Video RAM
512kB to (640kB-1)	00080000-0009FFFF	RAM or memory mapped system
0kB to (512kB-1)	00000000-0007FFFF	NexBus RAM

Figure 48 Memory Address Map

Stack Operations

In Protected Mode, multiple stacks can be maintained in separate segments. The stack-segment selector points to the current stack. The *stack frame* consists of a stack-frame base pointer (stored in the EBP general register) and a stack point (stored in the ESP general register). The stack-frame base pointer points to the last entry in a passed data structure. The stack pointer points to the top of the stack—the last entry in the stack after passing a data structure or PUSHing or POPing data. Immediately after a data structure is passed to a subroutine, the stack-frame base pointer and the stack pointer are coincident. Thereafter, if local variables are PUSHed onto or POPed from the stack, the stack-frame base pointer remains the same but the stack pointer changes. Figure 49 shows the mechanism.

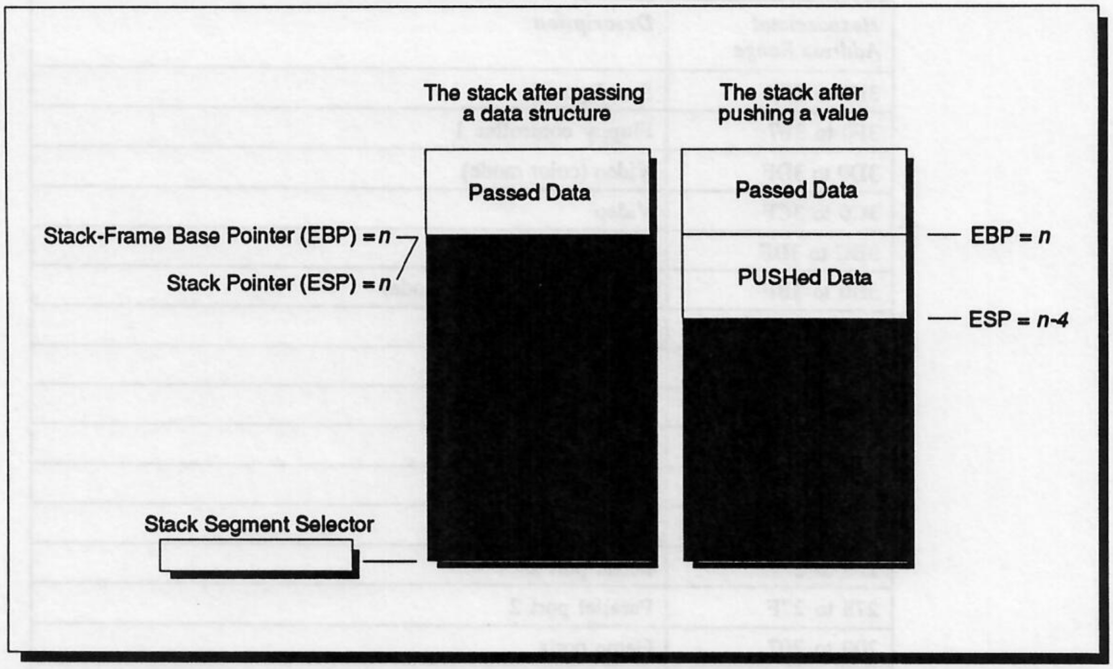

The stack after passing a data structure
The stack after pushing a value
Passed Data
Passed Data
Stack-Frame Base Pointer (EBP) = n
Stack Pointer (ESP) = n
PUSHed Data
EBP = n
ESP = n-4
Stack Segment Selector

Figure 49 Stack Mechanism

I/O

I/O devices can be addressed in the 64k-address I/O space with INx and OUTx instructions, or in the much larger memory space using most of the instruction set. The I/O address map for I/O space on the NexBus is shown in Figure 50. It is identical to that used in the IBM PC/AT.

In Protected Mode, the use of I/O instructions can be controlled by system software with the I/O privilege-level (IOPL) flag in the EFLAGS register, and in both Protected and Virtual-8086 Modes, access to specific ports can be controlled by system software with the I/O permissions bitmap (IOPB) in the task-state segments. Neither mechanism is typically accessible to applications programs.

Hexadecimal Address Range	*Description*
3F8 to 3FF	Serial port 1
3F0 to 3F7	Floppy controller 1
3D0 to 3DF	Video (color mode)
3C0 to 3CF	Video
3BC to 3BF	Parallel port 3
3B0 to 3BF	Video (monochrome mode)
3A0 to 3AF	SDLC controller 1
380 to 387	SDLC controller 2
378 to 37F	Parallel port 1
372 to 377	Floppy controller 2
360 to 36F	Reserved
300 to 3F1	Prototype card
2F8 to 2FF	Serial port 2
278 to 27F	Parallel port 2
200 to 207	Game ports
1F0 to 1F7	Hard disk 2
170 to 177	Hard disk 1
100 to 3FF	Expansion bus
F8 to FF	Coprocessor
F1	Reset coprocessor
F0	Clear coprocessor busy
C0 to DF	DMA controller 2
A0 to BF	Interrupt controller 2
80 to 9F	DMA page register
70 to 7F	Real time clock
60 to 6F	Keyboard controller
40 to 5F	Timers
20 to 3F	Interrupt controller 1
0 to 1F	DMA controller 1

Figure 50 I/O Address Map

Interrupts and Exceptions

Interrupts are caused either by hardware or software. System hardware can assert a maskable or non-maskable interrupt, and software can issue an INT, INTO or BOUND instruction. Hardware-initiated interrupts occur asynchronously with respect to the NexBus clock. Interrupts are serviced either when the currently executing instruction is completed or when the instruction (such as a string instruction) comes to a well-defined stopping point. Application programs request interrupts by using the INT *n* instruction, where *n* is the interrupt vector.

Exceptions result from certain normal or abnormal conditions during instruction decoding or execution. Exceptions and software-initiated interrupts occur synchronously with respect to the NexBus clock. There are three types of exceptions:

- *Faults*—The machine state prior to that instruction is restored before the fault handler is invoked, and then the instruction is retried. Page faults are normal encountered in Protected Mode operation. Other types of faults are undesirable.
- *Traps*—The instruction causing the exception finishes before the trap handler is invoked. Software interrupts and certain breakpoint exceptions used in debugging work like traps.
- *Aborts*—The instruction causing the exception, and possibly an indeterminate additional number of instructions, complete before the handler is invoked. The processor shuts down if it encounters a double-fault abort (an exception while another exception is being processed) followed by another exception.

Figure 51 shows the vectors, types, and causes of all interrupts and exceptions that can occur.

Vector	*Description*	*Type*
0	Divide By Zero	fault
1	Debug Exception	fault/trap
2	NMI* Interrupt	interrupt
3	Breakpoint (INT 3)	trap
4	Overflow (INTO instruction)	trap
5	Bound Range Exceeded	fault
6	Invalid Opcode	fault
7	Coprocessor Not Available	fault
8	Double Fault	abort
9	Coprocessor Segment Overrun	abort
10	Invalid Task State Segment	fault
11	Segment Not Present	fault
12	Stack Fault	fault
13	General Protection	fault
14	Page Fault	fault
15	(reserved)	fault
16	Coprocessor Error	fault
0-255	Interrupt Instructions	trap
	Hardware Maskable Interrupts	interrupt

Figure 51 Interrupts and Exceptions

Application Registers

Figure 52 shows the registers that are typically accessible in Protected Mode to application programs. The include the general registers, status and control registers, and segment registers.

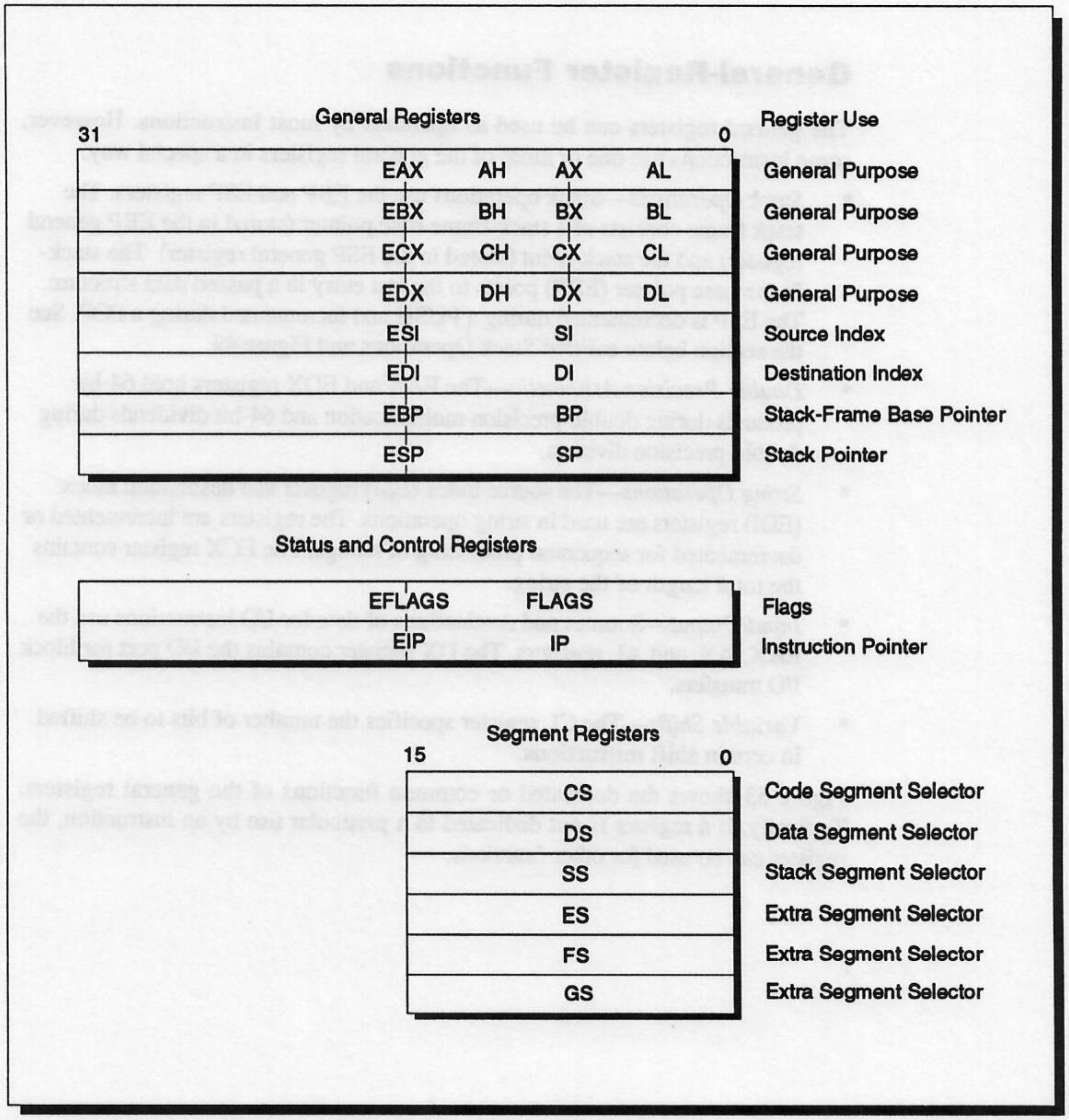


Figure 52 Registers Accessible To Application Programs

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems.
NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

General-Register Functions

The general registers can be used as operands by most instructions. However, some instructions use one or more of the general registers in a special way:

- *Stack Operations*—Stack operations use the EBP and ESP registers. The stack frame consists of a stack-frame base pointer (stored in the EBP general register) and the stack point (stored in the ESP general register). The stack-frame base pointer (EBP) points to the last entry in a passed data structure. The ESP is decremented during a PUSH and incremented during a POP. See the section below entitled Stack Operations and Figure 49.
- *Double-Precision Arithmetic*—The EAX and EDX registers hold 64-bit products during double-precision multiplication and 64-bit dividends during double-precision division.
- *String Operations*—The source index (ESI) register and destination index (EDI) registers are used in string operations. The registers are incremented or decremented for sequential processing of strings. The ECX register contains the total length of the string.
- *Input/Output*—Sources and destinations of data for I/O instructions use the EAX, AX, and AL registers. The DX register contains the I/O port for block I/O transfers.
- *Variable Shifts*—The CL register specifies the number of bits to be shifted in certain shift instructions.

Figure 53 shows the dedicated or common functions of the general registers. Typically, if a register is not dedicated to a particular use by an instruction, the register can be used for other functions.

Register	*Function*
EAX	**Operand for decimal arithmetic, multiply, divide, and I/O instructions. Special encoding for ADD, XOR, and MOV instructions.**
EBX	**Address generation in 16-bit code.**
ECX	**Bit index for shift instructions. Iteration count for LOOP and repeated string instructions.**
EDX	**Operand for multiply and divide instructions. Port number for I/O instructions.**
ESI	**Memory address of source operand for string instructions. Memory index for 16-bit addresses.**
EDI	**Memory address of destination operand for string instructions. Memory index for 16-bit addresses.**
EBP	**Stack frame base pointer.**
ESP	**Stack pointer (top of stack).**

Figure 53 General-Register Functions

Flags

The FLAGS register is the low-order 16 bits of the EFLAGS register, shown in Figure 54. It contains the following status and control flags:

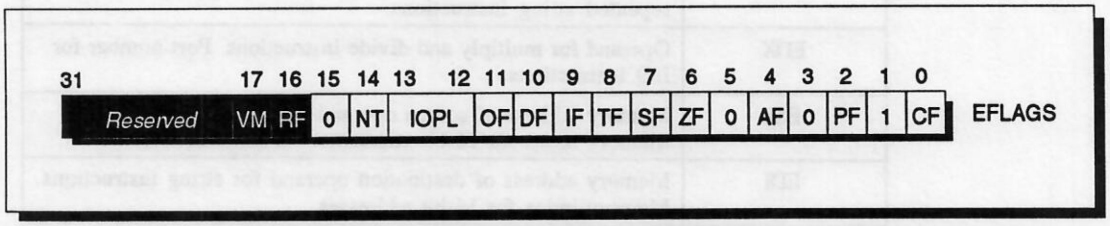


Figure 54 EFLAGS Register

Bit	*Flag*	
11	OF	*Overflow Flag*—When set to 1, it indicates that the uppermost bit (the sign bit) of an operand has changed. The flag is undefined after a shift operation of more than one bit.
10	DF	*Direction Flag*—When set to 1, it indicates that a source or destination address pointer of a string instruction (the contents of the ESI and/or EDI register) should be *incremented* after each iteration of the instruction execution. When cleared to 0, it indicates the pointer should be decremented. The flag can be set and cleared with the STD and CLD instructions.
7	SF	*Sign Flag*—When set to 1, it indicates that an arithmetic operation had a negative result, as indicated by the high-order bit (the sign bit). When cleared to 0, it indicates that the operation had a positive result.
6	ZF	*Zero Flag*—When set to 1, it indicates that an arithmetic operation resulted in zero. When cleared to 0, the result was non-zero.
4	AF	*Auxiliary Flag*—When set to 1, it indicates that a BCD arithmetic operation resulted in a carry out (addition) or borrow (subtraction) from bit 3 of the least significant byte, regardless of the operand size. When cleared to 0, the result had no carry out or a borrow.
2	PF	*Parity Flag*—When set to 1, it indicates that the number of 1s in the low-order operand byte after an arithmetic operation is *even*. When cleared to 0, the number of 1s is *odd*.

0 CF *Carry Flag*—When set to 1, it indicates that an arithmetic operation resulted in a carry out (addition) or borrow (subtraction) into the high-order bit (bit 6, 14, and 30 for a signed integers; bit 7, 15, and 31 for unsigned integers). The STC and CLC instructions set or clear the flag. The CMC instruction complements it.

Instructions

The Nx586 processor's instruction set is identical to that of the Intel i386 processor, and the Nx587 numerics processor instruction set is identical to that of the Intel i387 math coprocessor. Most instructions can be used in application programs. Some can only have operands of a particular type or can only operate on data in a certain register. Only a few instructions are reserved for operating-system programs (those operating at privilege level 0).

Instruction Groups

Figure 55 lists the instruction set by type of operation and shows which instructions are useful in, and (due to privilege-level protection constraints) can actually be used in application programs.

In Real Mode, the privilege level of all code is 0 so application programs can use all instructions except those that access protection, segmentation, and paging resources. In Virtual-8086 Mode, the privilege level of all code is 3 and only a subset of those instructions usable in Protected-Mode application programs can be used, except those that access segmentation resources, which are not available.

Mnemonic	Instruction	Useful To and Usable In Protected-Mode Applications
	Arithmetic Operations	
AAA	ASCII adjust after addition	yes
AAD	ASCII adjust before division	yes
AAM	ASCII adjust after multiplication	yes
AAS	ASCII adjust after subtraction	yes
ADC	Add with carry	yes
ADD	Add	yes
BOUND	Verify array bounds	yes
CMP	Compare	yes
DAA	Decimal adjust after addition	yes
DAS	Decimal adjust after subtraction	yes
DEC	Decrement	yes
DIV	Divide (unsigned)	yes
IDIV	Divide (signed)	yes
IMUL	Multiply (signed)	yes
INC	Increment	yes
MUL	Multiply (unsigned)	yes
NEG	Negate (two's-complement)	yes
SBB	Subtract with borrow	yes
SUB	Subtract	yes
	Logical Operations	
AND	Logical AND	yes
NEG	Negate (two's-complement)	yes
NOT	Logical NOT	yes
OR	OR (inclusive)	yes
TEST	Test bits (AND)	yes
XOR	OR (exclusive)	yes
	Data Manipulation	
BSF	Bit scan (forward)	yes
BSR	Bit scan (reverse)	yes
BT	Bit test	yes
BTC	Bit test and complement	yes
BTR	Bit test and reset	yes

BTS	Bit test and set	yes
RCL	Rotate and carry left	yes
RCR	Rotate and carry right	yes
ROL	Rotate left	yes
ROR	Rotate right	yes
SAR	Shift right	yes
SHL	Shift left	yes
SHLD	Shift left (double)	yes
SHRD	Shift right (double)	yes
Data Conversion		
CBW	Convert byte to word	yes
CDQ	Convert dword to qword	yes
CWDx	Convert word to dword	yes
XLATB	Translate byte	yes
Data Transfer		
INx	Input from port	yes
LEA	Load effective address	yes
MOVx	Copy, load, or store	yes, except for control, debug, or test registers
OUTx	Output to port	yes
POPx	Pop stack	yes
PUSHx	Push stack	yes
XCHG	Exchange	yes
String Operations		
CMPSx	Compare string	yes
LODSx	Load string	yes
MOVSx	Move string	yes
SCASx	String compare	yes
STOSx	Store string	yes
Control Operations		
CALL	Call a procedure	yes
ENTER	Enter a procedure	yes
HLT	Halt	no
INT	Interrupt	yes
INTO	Interrupt	yes

IRETx	Interrupt return	yes
Jcond	Jump conditional	yes
JMP	Jump unconditional	yes
LEAVE	Leave a procedure	yes
LOOPx	Loop	yes
NOP	No operation	yes
RET	Return from procedure	yes
SETx	Set conditional	yes
WAIT	Wait for interrupt	yes
	Segment Operations	
LDS	Load DS with pointer	yes
LES	Load ES with pointer	yes
LFS	Load FS with pointer	yes
LGS	Load GS with pointer	yes
LSS	Load SS with pointer	yes
MOV	Load segment registers	yes
POPx	Pop segment registers	yes
PUSHx	Push segment registers	yes
VERRx	Verify segment	yes
	Flag Operations	
CLC	Clear CF flag	yes
CLD	Clear DF flag	yes
CLI	Clear IF flag	varies
CLTS	Clear TS flag	no
LAHF	Load AH with flags	yes
POPFx	Pop flags	yes
PUSHFx	Push flags	yes
SAHF	Store AH to flags register	yes
STC	Set CF flag	yes
STD	Set DF flag	yes
STI	Set IF flag	varies
	Stack Operations	
LSS	Load SS register	yes
POPFx	Pop flags	yes
POPx	Pop operand	yes

PUSHFx	Push flags	yes
PUSHx	Push operand	yes
	Privilege Levels	
ARPL	Adjust RPL	no
LAR	Load access rights	yes
VERRx	Verify segment	yes
	System Administration	
LGDT	Load GDT	no
LIDT	Load IDT	no
LLDT	Load LDT	no
LMSW	Load machine status word	no
LSL	Load segment limit	yes
LTR	Load task register	no
SGDT	Store GDT	no
SIDT	Store IDT	no
SLDT	Store LDT	no
SMSW*	Store machine status word	no
STR	Store task register	yes

Figure 55 Instruction Set (By Functional Group)

Instruction Prefixes

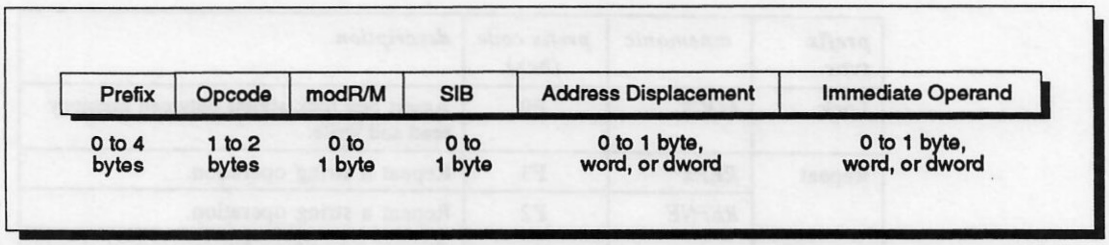
The instruction prefixes and their conditions are shown in Figure 56. Only a single memory operand can ever be overridden in a given instruction. Address-size prefixes apply only to memory operands, but operand-size prefixes can be used for any 16-bit or 32-bit operand.

prefix type	*mnemonic*	*prefix code (hex)*	*description*
Lock	*LOCK*	F0	Assert bus lock signal between memory read and write.
Repeat	*REPx*	F3	Repeat a string operation.
	REPNE	F2	Repeat a string operation.
Operand Size		66	Change a 16-bit or 32-bit operand size to the opposite of its default.
Address Size		67	Change a 16-bit or 32-bit address size to the opposite of its default.
Segment Override	*CS*	2E	Use CS segment for memory operands.
	DS	3E	Use DS segment for memory operands.
	ES	26	Use ES segment for memory operands.
	FS	64	Use FS segment for memory operands.
	GS	65	Use GS segment for memory operands.
	SS	36	Use SS segment for memory operands.

Figure 56 instruction Prefixes

Syntax and Notation

Instructions encode an operation, a location or value of source data (if any), and a location where a result (if any) is to be stored. Figure 57 shows the syntax of instructions, which can be up to length to 15 bytes in length. The section below entitled *Instruction Encoding* details the encoding of each instruction.



Prefix	Opcode	modR/M	SIB	Address Displacement	Immediate Operand
0 to 4 bytes	1 to 2 bytes	0 to 1 byte	0 to 1 byte	0 to 1 byte, word, or dword	0 to 1 byte, word, or dword

Figure 57 Instruction Format

The following table, Figure 58, defines the notations used in the instruction descriptions that follow.

Notation	*Meaning*
<16,32>	A 16-bit or 32-bit operand, as determined by the default [D bit] and an instruction prefix.
<8>	An 8-bit operand.
/n	A number from 0 to 7 in the reg field of the modR/M byte that is used to specify an extension to the opcode.
AF	The value of the auxiliary flag.
AH	The operand in the upper byte of AX register.
AL	The operand in the lower byte of AX register.
BH	The operand in the upper byte of BX register.
BL	The operand in the lower byte of BX register.
CF	The value of the carry flag.
CH	The operand in the upper byte of CX register.
CL	The operand in the lower byte of CX register.
cr	The value in the control register CR0, CR2, or CR3.
CS	The value in the code segment register.
DF	The value of the direction flag.
DH	The operand in the upper byte of DX register.
DL	The operand in the lower byte of DX register.

dr	The value in the debug register, DR0, DR1, DR2, DR3, DR6, or DR7.
DS	The value in the data segment register.
dst	Destination operand.
[E]AX	The operand in the AX or EAX register.
[E]BP	The operand in the BP or EBP register.
[E]BX	The operand in the BX or EBX register.
[E]CX	The operand in the CX or ECX register.
[E]DI	The operand in the DI or EDI register.
([E]DI)	The address in the DI or EDI register.
[E]DX	The operand in the DX or EDX register.
[E]IP	The operand in the IP or EIP register.
ES	The value in the ES segment register.
[E]SI	The operand in the SI or ESI register.
([E]SI)	The address in the SI or ESI register.
[E]SP	The operand in the SP or ESP register.
FS	The value in the FS segment register.
GDT	Global descriptor table.
GS	The value in the GS segment register.
IDT	Interrupt descriptor table.
IF	The value of the interrupt flag.
imm	An immediate value in an instruction.
imm8	An 8-bit immediate value in an instruction.
imm16	A 16-bit immediate value in an instruction.
LDT	Local descriptor table.
m	A memory operand in the r/m field of the modR/M byte.
(m)	A memory address in the r/m field of the modR/M byte.
m16	A 16-bit memory operand.
m32	A 32-bit memory operand.
m64	A 64-bit memory operand.
m80	An 80-bit memory operand.
moffs	An offset in memory relative to a segment base.
NT	The value of the nested task flag.
OF	The value of the overflow flag.
off	Offset value.
PF	The value of the parity flag.
r	A general register encoded in the reg field of the modR/M byte.

r8	An 8-bit general register.
r16	A 16-bit general register.
r32	A 32-bit general register.
reg	A 16-bit or 32-bit general register encoded in opcode bits 2:0.
rel	A 16-bit or 32-bit value added to an address.
rel8	An 8-bit value added to an address.
RF	The value of the resume flag.
r/m	An operand in a general register or memory location.
(r/m)	An address or offset in a general register or memory location.
r/m8	An 8-bit *r/m* value.
r/m16	A 16-bit *r/m* value.
r/m32	A 32-bit *r/m* value.
sel	The value in a segment selector.
SF	The value of the sign flag.
sr	The value in the segment register CS, SS, DS, ES, FS, or GS.
src	Source operand.
SS	The value in the stack segment register.
TF	The value of the trap flag.
tr	The value in the test register TR6 or TR7.
ZF	The value of the zero flag.

Figure 58 Instruction Notation

Instruction Latency and Encoding

The basic assumptions for the minimum latency given in this section are:

- Memory accesses hit in the L1 cache.
- Page translations hit in the TLB.
- Operands are located within qword boundaries.
- Branch prediction is correct.
- Immediate and displacement values are 32 bits or less.
- Instruction prefixes are not used (these add one clock).
- The immediately preceding instruction does not modify a register that is used in the address calculation for the current instruction.
- Exceptions or interrupts are not encountered.

The following table, Figure 59, gives the instruction clock counts. The counts are in processor clocks, which operate at twice the frequency of NexBus clock cycles. A delta-shape (Δ) in the "Processor Clocks" column indicates a clock count that can vary considerably, depending on events within the processor. For more information, contact your NexGen Application Engineer.

Mnemonic	*Description*	*Processor Clocks*
AAA	ASCII adjust after addition	3
AAD	ASCII adjust before division	3
AAM	ASCII adjust after multiplication	9
AAS	ASCII adjust after subtraction	3
ADC	Add with carry	1
ADD	Add	1
AND	Logical AND	1
ARPL	Adjust RPL	1
BOUND	Verify array bounds	2
BSF	Bit scan (forward)	2
BSR	Bit scan (reverse)	2
BT	Bit test	2
BTC	Bit test and complement	2
BTR	Bit test and reset	2
BTS r/m, imm8	Bit test and set	2
BTS r/m, r	Bit test and set	1
CALL rel	Call a procedure (near)	Δ
CALL sel:off	Call a procedure (direct far)	Δ
CALL (m)	Call a procedure (indirect near)	Δ
CALL (r/m)	Call a procedure (indirect far)	Δ
CBW	Convert	1
CDQ	Convert	1
CLC	Clear CF flag	1
CLD	Clear DF flag	1
CLI	Clear IF flag	1
CLTS	Clear TS flag	2
CMC	Complement CF flag	1
CMP	Compare	1
CMPSB	Compare string	2
CMPSD	Compare string	2
CMPSW	Compare string	2
CWD	Convert (word to dword)	1
CWDE	Convert (word to dword extended)	1
DAA	Decimal adjust after addition	3

DAS	Decimal adjust after subtraction	3
DEC	Decrement	1
DIV r/m16	Divide (unsigned)	18
DIV r/m32	Divide (unsigned)	34
DIV r/m8	Divide (unsigned)	8
ENTER imm16, 0	Enter a procedure	3
ENTER imm16, 1	Enter a procedure	4
ENTER imm16, imm8	Enter a procedure	6 + 2*n
ESC ecode, m	Escape to coprocessor	Δ
ESC ecode, r	Escape to coprocessor	Δ
F2XM1	Compute $2^{ST(0)-1}$	Δ
FABS	Absolute value	2
FADD	Add	2
FADDP	Add and pop	2
FBLD	Load binary coded decimal	Δ
FBSTP	Store binary coded decimal and pop	Δ
FCHS	Change sign	2
FCLEX	Clear exceptions	2
FCOM	Compare real	4
FCOMP	Compare real and pop	4
FCOMPP	Compare real and pop twice	4
FCOS	Cosine of ST(0)	Δ
FDECSTP	Decrement stack-top pointer	2
FDIV	Divide	39
FDIVP	Divide and pop	39
FDIVR	Reverse divide	39
FDIVRP	Reverse divide and pop	39
FFREE	Free ST(i) register	2
FIADD	Add integer	5
FICOM	Compare integer	7
FICOMP	Compare integer and pop	7
FIDIV	Divide	4
FIDIVR	Reverse divide	4
FILD	Load integer	3
FIMUL	Muliply	5
FINCSTP	Increment stack pointer	2

FINIT	Initialize floating-point unit	2
FIST	Store integer	8
FISTP	Store integer and pop	8
FISUB	Subtract	5
FISUBR	Reverse substract	5
FLD	Load real	2
FLD1	Load + 1.0 into ST(0)	2
FLDCW	Load control word	2
FLDENV	Load FPU environment	6
FLDL2E	Load $\log_2(\varepsilon)$ into ST(0)	2
FLDL2T	Load $\log_2(10)$ into ST(0)	2
FLDLG2	Load $\log_{10}(2)$ into ST(0)	2
FLDPI	Load π into ST(0)	2
FLDZ	Load +0.0 into ST(0)	2
FMUL	Multiply	2
FMULP	Multiply	2
FNOP	No operation	2
FPATAN	Partial arctangent	Δ
FPREM	Partial reminder	
FPREM1	Partial reminder (IEEE)	
FPTAN	Partial tangent	Δ
FRNDINT	Round to integer	2
FRSTOR	Restore FPU state	22
FSAVE	Store FPU state	38
FSCALE	Scale	Δ
FSIN	Sine	Δ
FSINCOS	Sine and cosine	Δ
FSQRT	Square root	39
FST	Store real	2
FSTCW	Store control word	2
FSTENV	Store FPU environment	22
Mnemonic	Description	Clocks
FSTP	Store real and pop	2
FSTSW	Store status word into AX	2
FSTSW	Store status word into memory	2

FSUB	Subtract	2
FSUBP	Subtract and pop	2
FSUBR	Reverse subtract	2
FSUBRP	Reverse subtract and pop	2
FTST	Test	4
FUCOM	Unordered compare real	4
FUCOMP	Unordered compare and pop	4
FUCOMPP	Unordered compare and pop twice	4
FXAM	Examine	3
FXCH	Exchange ST(0) and ST(i)	4
FXTRACT	Extract exponent and significand	Δ
FYL2X	ST(1) x log$_2$(ST(0))	Δ
FYL2XP1	ST(1) x log$_2$(ST(0) + 1.0)	Δ
FWAIT	Wait until FPU ready	2
HLT	Halt	1
IDIV r/m16	Divide (signed)	26
IDIV r/m32	Divide (signed)	42
IDIV r/m8	Divide (signed)	16
IMUL AL, r/m8	Multiply (signed)	2
IMUL AX, r/m16	Multiply (signed)	3
IMUL EAX, r/m32	Multiply (signed)	5
IMUL r, r/m	Multiply (signed)	3, 5
IMUL r, r/m, imm	Multiply (signed)	3, 5
IMUL r, r/m, imm8	Multiply (signed)	3
IN	Input from port	real: 1 prot: Δ V86: Δ
INC	Increment	1
INSB	Input from port (string)	real: 4 prot: Δ V86: Δ
INSD	Input from port (string)	real: 4 prot: Δ V86: Δ

INSW	Input from port (string)	real: 4 prot: Δ V86: Δ
INT 3	Interrupt	Δ
INT imm8	Interrupt	Δ
INTO	Interrupt	Δ
IRET	Interrupt return	Δ
IRETD	Interrupt return	Δ
JA/JNBE rel	Jump conditional	2
JA/JNBE rel8	Jump conditional	1
JAE/JNB rel	Jump conditional	2
JAE/JNB rel8	Jump conditional	1
JB/JNAE rel	Jump conditional	2
JB/JNAE rel8	Jump conditional	1
JBE/JNA rel	Jump conditional	2
JBE/JNA rel8	Jump conditional	1
JCXZ rel8	Jump conditional	1
JE/JZ rel	Jump conditional	2
JE/JZ rel8	Jump conditional	1
JECXZ rel8	Jump conditional	1
JG/JNLE rel	Jump conditional	2
JG/JNLE rel8	Jump conditional	1
JGE/JNL rel	Jump conditional	2
JGE/JNL rel8	Jump conditional	1
JL/JNGE rel	Jump conditional	2
JL/JNGE rel8	Jump conditional	1
JLE/JNG rel	Jump conditional	2
JLE/JNG rel8	Jump conditional	1
JNE/JNZ rel	Jump conditional	2
JNE/JNZ rel8	Jump conditional	1
JNO rel	Jump conditional	2
JNO rel8	Jump conditional	1
JNP rel	Jump conditional	2
JNP rel8	Jump conditional	1
JNS rel	Jump conditional	2
JNS rel8	Jump conditional	1

JO rel	Jump conditional	2
JO rel8	Jump conditional	1
JP rel	Jump conditional	2
JP rel8	Jump conditional	1
JS rel	Jump conditional	2
JS rel8	Jump conditional	1
JMP rel	Jump unconditional (near)	1
JMP rel8	Jump unconditional (near)	1
JMP (r/m)	Jump unconditional (indirect near)	1
JMP sel:off	Jump unconditional (direct far)	Δ
JMP (m)	Jump unconditional (indirect far)	Δ
LAHF	Load AH with flags	1
LAR	Load access rights	Δ
LDS	Load DS with pointer	Δ
LEA	Load effective address	Δ
LEAVE	Leave procedure	3
LES	Load ES with pointer	Δ
LFS	Load FS with pointer	Δ
LGDT	Load GDT descriptor	Δ
LGS	Load GS with pointer	Δ
LIDT	Load IDT descriptor	Δ
LLDT	Load LDT descriptor	Δ
LMSW	Load machine status word	Δ
LOCK	Lock memory bus (prefix)	1
LODSx	Load string	2
LOOPx	Loop	1
LSL	Load segment limit	Δ
LSS	Load SS with pointer	Δ
LTR	Load task register	Δ
MOV reg, moffs	Copy general register	1
MOV reg, moffs	Copy general register	1
MOV cr, r32	Load control register	2
MOV dr, r32	Load debug register	2
MOV sr, r/m	Load segment register	Δ
MOV moffs, reg	Copy general registers	1
MOV r/m, r/m	Copy general registers	1

MOV r/m, sr	Store segment register	1
MOV r/m, imm	Copy general registers	1
MOV r32, cr	Store control register	Δ
MOV r32, dr	Store debug register	Δ
MOV r32, tr	Store test register	Δ
MOV reg, imm	Copy general registers	1
MOV tr, r32	Load test register	2
MOVSx	Copy string	2
MOVSX	Copy (sign-extend)	2
MOVZX	Copy (sign-extend)	2
MUL AL, r/m8	Multiply (unsigned)	3
MUL AX, r/m16	Multiply (unsigned)	4
MUL EAX, r/m32	Multiply (unsigned)	6
NEG	Negate (two's-complement)	1
NOP	No operation	1
NOT	Logical NOT	1
OR	OR (inclusive)	1
OUT	Output to port	real: 1 prot: Δ V86: Δ
OUTSx	Output to port (string)	real: 1 prot: Δ V86: Δ
POP sr	Pop segment register	Δ
POP r/m	Pop operand	2
POPAx	Pop general registers	9
POPFx	Pop flags	1
PUSH sr	Push segment register	Δ
PUSH imm	Push operand	1
PUSH imm8	Push operand	1
PUSH r/m	Push operand	2
PUSH reg	Push operand	1
PUSHAx	Push general registers	8
PUSHFx	Push flags	2
RCL	Rotate and carry left	1
RCR	Rotate and carry right	1

REP INSx	Repeat input from port (prefix)	real: 1 + 5*n prot: Δ V86: Δ
REP LODSx	Repeat load (prefix)	1 + 3*n
REP MOVSx	Repeat copy (prefix)	1 + 3*n
REP OUTSx	Repeat output to port (prefix)	real: 1 + 3*n prot: Δ V86: Δ
REP STOSx	Repeat store (prefix)	1 + 2*n
REPE CMPSx	Repeat compare (prefix)	1 + 3*n
REPE SCASx	Repeat search (prefix)	1 + 2*n
REPNE CMPSx	Repeat compare (prefix)	1 + 3*n
REPNE SCASx	Repeat search (prefix)	1 + 2*n
RET/RETF	Return from procedure (far)	Δ
RET/RETN	Return from procedure (near)	Δ
ROL	Rotate left	1
ROR	Rotate right	1
SAHF	Store AH to flags register	1
SAL/SHL	Shift left	1
SAR	Shift right	1
SBB	Subtract with borrow	1
SCASx	String compare	1
SETx	Set conditional	2
SGDT	Store GDT register	Δ
SHLD	Shift left (double)	3
SHR	Shift right	1
SHRD	Shift right (double)	3
SIDT	Store IDT register	Δ
SLDT	Store LDT register	Δ
SMSW	Store machine status word	Δ
STC	Set CF flag	1
STD	Set DF flag	1
STI	Set IF flag	1
STOSx	Store string	1
STR	Store task register	1
SUB	Subtract	1

TEST	Test bits (AND)	1
VERR	Verify segment (read)	Δ
VERW	Verify segment (write)	Δ
WAIT	Wait for interrupt	1
XCHG	Exchange	2
XLATB	Translate byte	1
XOR	OR (exclusive)	1

Figure 59 instruction Latency and Encoding

Performance Optimization

The following optional guidelines will usually result in faster program execution on the Nx586 processor:

- Avoid instruction prefixes. They add one clock to execution time.
- For multiplication or division by a power of 2, use shift instructions instead of multiply and divide instructions.
- In conditional jumps, make the not-taken path the more common one.
- For address calculations that use memory operands, load the memory operand as early as possible before using it.
- Keep all bytes of an operand within qword boundaries. See the section above entitled *Operand Alignment*.
- Separate writable data from code by at least four 32-byte cache blocks. See the section above entitled *Operand Alignment*.
- Avoid sequential crossing of 4kB page boundaries, even in Real Mode. Instead, jump across such boundaries.

Self-Modifying Code

The standard x86 caveat for self-modifying code applies to the Nx586 processor: this type of code will only be guaranteed to execute correctly if there is a taken jump between the modification and the execution of the modified instruction.

For example, if a code location is written to, and this write is followed by a taken jump to any location at any time before the modified instruction itself is executed, the modified instruction will execute correctly.

Software Differences from Intel Processors

Application Software

Figure 60 shows the differences between application software written for the Intel i386/i387 or i486 and the NexGen Nx586/587 processors.

Intel Processor	*Nx586/587 Differences*		
	Real Mode	*Virtual-86 Mode*	*Protected Mode*
i386/i387	None	None	None
i486	None	None	▪ The BSWAP (byte swap) instruction not supported. ▪ The XADD (exchange and add) instruction not supported. ▪ The CMPXCHG (compare and exchange) instruction not supported.

Figure 60 Application Software Differences

System Software

Figure 61 shows the differences between system software (typically implemented in the BIOS) written for the Intel i386/i387 or i486 and the NexGen Nx586/587 processors. For details on these system software differences, contact NexGen Applications Engineering.

Intel Processor	*Nx586/587 Differences*		
	Real Mode	*Virtual-86 Mode*	*Protected Mode*
i386/i387	None	None	▪ Test registers work differently. ▪ Debug registers work differently. ▪ I/O permission bitmap (IOPB) works differently.
i486	None	None	The i486 differences include the i386 differences, plus: ▪ The INVD (invalidate cache) instruction is not supported. ▪ The WBINVD (write-back and invalidate cache) instruction is not supported. ▪ The INVLPG (invalidate TLB entry) instruction is not supported. ▪ The MOV TR<7:3>,r32 and MOV r32,TR<7:3> instructions are not supported. ▪ In EFLAGS, the AC flag (alignment check, bit 18) is not supported. ▪ In CR0, the NE bit (numeric error, bit 5), the WP bit (write protect, bit 16), the WP bit (write protect, bit 16), the AM bit (alignment mask, bit 18), the NW bit (not write-through, bit 29), and the CD bit (cache disable, bit 30) are not supported. ▪ In CR3, the PWT bit (page-level write transparent, bit 3) and the PCD bit (page-level cache disable, bit 4) are not supported. ▪ In page-table entries, the PWT and PCD bits (3 and 4) are not supported. ▪ Exception 17 (alignment checking) is not supported.

Figure 61 System Software Differences

Benchmark Software

Benchmark software implemented for the Intel processors will see the following differences when run on the Nx586/Nx587 processors:

- The *qaplus* timer test fails, due to the speed at which multiplies are executed on the Nx586/Nx587 processors.

Electrical Data

Absolute Maximum Ratings

DC supply voltage	6.5V
Voltage on any pin (with respect to ground)	V_{SS} -0.5V to V_{CC} +0.5V
Case temperature under bias	0°C to +70°C
Power dissipation at 50MHz	TBD
Storage temperature	65°C to +150°C

Operation of the device beyond the maximum ratings may result in permanent damage to the device. Operation should be limited to the conditions specified under D.C. Characteristics.

D.C. Characteristics

The Nx586 processor and Nx587 numerics processor both operate on a single 4V power supply. The Nx586 processor can be interfaced directly to the Nx587 numerics processor and to the NxPC system controller.

Conditions: T_A = 0°C to +60°C, V_{cc} = 4V ± 5%, V_{ss} = 0V, unless otherwise specified. Input-signal minimum rise time (t_r) = 2ns. Input-signal minimum fall time (t_f) = 2ns. Contact NexGen for test information.

Symbol	*Parameter*	*Condition*	*Min*	*Typ*	*Max*	*Units*	*Notes*
V_{ih}	Input High Voltage		2.2		V_{cc} + 0.4	V	
V_{il}	Input Low Voltage		-0.5		0.8	V	
V_{oh}	Output High Voltage	I_{oh} = -2mA	2.4			V	1
V_{ol}	Output Low Voltage	I_{ol} = 8mA			0.4	V	1
V_{oh}	Output High Voltage	I_{oh} = -2mA	2.4			V	2
V_{ol}	Output Low Voltage	I_{ol} = 3.2mA			0.4	V	2
I_{in}	Input Leakage Current	V_{in} = Vcc or Vss			TBD	A	
I_{cc}	Supply Current	Inputs Active			TBD	mA	
I_{cc1}	Quiescent Current					mA	
C_{clk}	CLK Input Capacitance	f_{in} at 1MHz (not 100% tested)			15	pF	
C_{in}	Input Capacitance	f_{in} at 1MHz (not 100% tested)			10	pF	
C_{out}	Output Capacitance				10	pF	

1. Output Type 1: I_{oh} = -2mA @ V_{oh} = 2.4V; I_{ol} = 8mA @ V_{ol} = 0.4mV
2. Output Type 2: I_{oh} = -2mA @ V_{oh} = 2.4V; I_{ol} = 3.2mA @ V_{ol} = 0.4mV

Figure 62 D.C. Characteristics

A.C. Characteristics

Conditions: V_{CC} = 4V ± 5%, T_{case} = 0°C to +80°C, C_{Load} = 75pF, unless otherwise specified. Propagation delay is measured between the times that an input crosses 1.5V (from 0V or 3V) and an output either falls below 0.8V or rises above 2.4V. All numbers are in nanoseconds with respect to the processor clock, unless otherwise specified. Contact NexGen for test information.

Primary-Signal Timing

No.	Signal	Type	Parameter	66MHz		80MHz		100MHz		Notes
				min	max	min	max	min	max	
	ALE*	O	Prop		10		8			2
	AREQ*	O	Prop		12		9			2
	NxADINUSE	O	Prop		12		9			2
	NxADP<7:0>	I/O	Setup	3		2				2
			Hold	0		0				
			Prop		12		10			
	CADDR<17:3>	O	Prop		8		7			1,3
	CBANK<1:0>	O	Prop		8		7			1,3
	CDATA<63:0>	I/O	Setup	3		2				2
			Hold	0		0				
			Prop		9		7			
	t_{AA} SRAM access	—		13		11				
	CLK period	I		33	100	25	100			
	low time			14	19	10	15			
	high time			14	19	10	15			
	rise time				3		2			
	fall time				3		2			
	COEA*, COEB*	O	Prop		7		5			
	t_{OE} SRAM access	—	Prop							Step A0
										Step B0
	CPARITY<7:0>	I/O	Setup	3		2				2
			Hold	0		0				
			Prop		9		8			
	CWE<7:0>*	O	Prop		9		8			2
	t_{DS} RAM data setup	—	Setup							
	DCL*	O	Prop		12		10			2
	DEVICE<1:0>	I	Setup	3		2				
			Hold	0		0				
	GALE	I	Setup	3		2				
			Hold	0		0				
	GATEA20	I	Setup	3		2				
			Hold	0		0				

	GBLKNBL	I	Setup	3		2				
			Hold	0		0				
	GDCL	I	Setup	3		2				
			Hold	0		0				
	GNT*	I	Setup	3		2				
			Hold	0		0				
	GSHARE	I	Setup	3		2				
			Hold	0		0				
	GTAL	I	Setup	3		2				
			Hold	0		0				
	GXACK	I	Setup	3		2				
			Hold	0		0				
	GXHLD	I	Setup	3		2				
			Hold	0		0				
	HROM	I	Setup	3		2				
			Hold	0		0				
	INTR*	I	Setup	3		2				
			Hold	0		0				
	LOCK*	O	Prop		12		10			2
	NMI*	I	Setup	3		2				
			Hold	0		0				
	NPDATA<63:0>	I/O	Setup	3		2				2
			Hold	0		0				
			Prop		12		10			
	NPRREQ prop delay	O	Prop		12		10			2
	NPRVAL	O	Prop		12		10			2
	NPWREQ	I	Setup	3		2				
			Hold	0		0				
	NPWVAL	I	Setup	3		2				
			Hold	0		0				
	NPIRQ*	I	Setup	3		2				
			Hold	0		0				
	NPNOERR	I	Setup	3		2				
			Hold	0		0				
	NPOUTFTYP0	O	Prop		12		10			2
	NPOUTFTYP1	O	Prop		12		10			2
	NPPOPBUS<15:0>	O	Prop		12		10			2

	NPTAG<4:0>	I/O	Setup	3		2				2
			Hold	0		0				
			Prop		12		10			
	NPTAGSTAT<5:0>	O	Prop		12		10			2
	NPTERM<1:0>	I	Setup	3		2				
			Hold	0		0				
	NREQ*	O	Prop		12		10			2
	NSPARE<4:0>	I/O	Setup	3		2				2
			Hold	0		0				
			Prop		12		10			
	NxAD<63:0>	I/O	Setup	3		2				2
			Hold	0		0				
			Prop		12		10			
	OWNABL	I	Setup	3		2				
			Hold	0		0				
	PARERR*	O	Prop		12		10			2
	RESET*	I	Setup	100		100				
			Hold	3		2				
			Prop	0		0				
	RESETCPU*	I	Setup	3		2				
			Hold	0		0				
	SHARE*	O	Prop		12		10			2
	SLOTID<4:0>	I	Setup	3		2				
			Hold	0		0				
	XACK*	O	Prop		9		8			2
	XBCKE*	O	Prop		9		8			1
	XBOE*	O	Prop		9		8			1
	XHLD*	O	Prop		9		8			2
	XNOE*	O	Prop		9		8			1
	NPPOPTAG<4:0>	O	Prop		12		10			2

Clock-Signal Timing

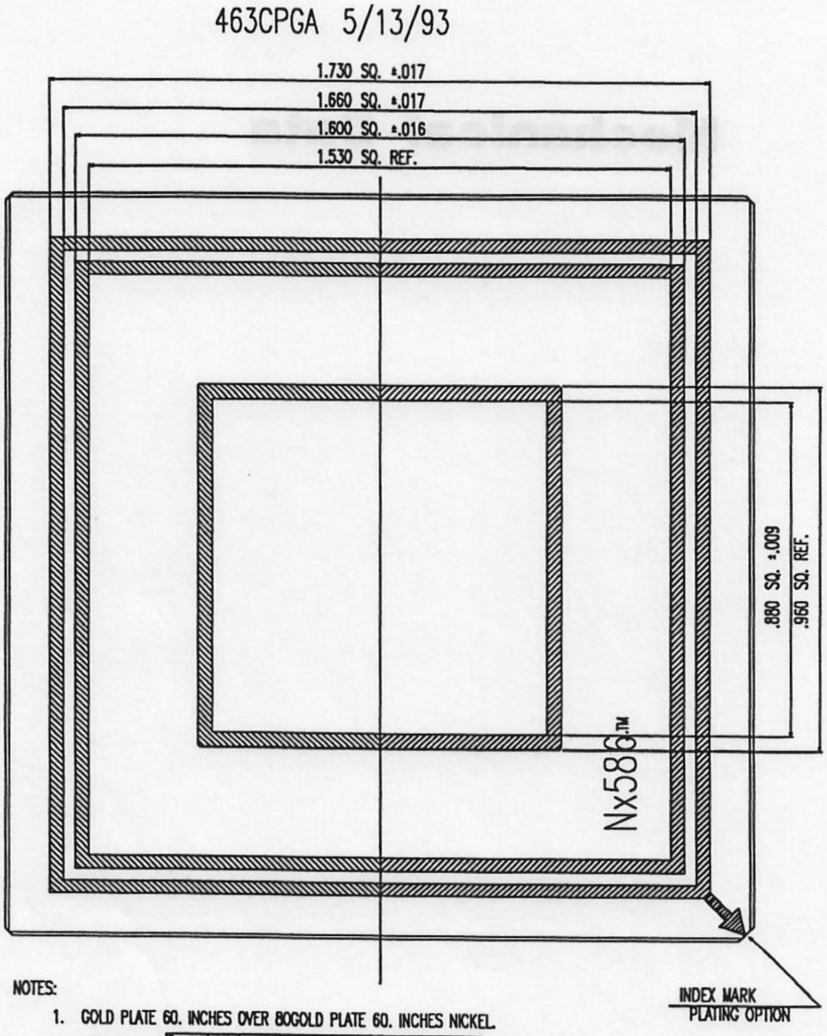
				66MHz		80MHz		100MHz		
No.	*Signal*	*Type*	*Parameter*	*min*	*max*	*min*	*max*	*min*	*max*	*Notes*
	CKMODE	I	Setup	3		2				
			Hold	2		2				
	PHE1	I	Setup	3		2				
			Hold	2		2				
	PHE2	I	Setup	3		2				
			Hold	2		2				
	XPH1	O	Prop		8		7			CL=100pF
	XPH2	O	Prop		8		7			CL=100pF
	XREF	O	Prop		8		7			CL=100pF
	XSEL	I	Setup	3		2				
			Hold	2		2				
	IREF	I								

Test-Signal Timing

				66MHz		*80MHz*		*100MHz*		
No.	*Signal*	*Type*	*Parameter*	*min*	*max*	*min*	*max*	*min*	*max*	*Notes*
	ANALYZEIN	Setup	I	3		2				
		Hold		0		0				
	ANALYZEOUT	Prop	O		12		10			2
	POPHOLD	Setup	I	3		2				
		Hold		0		0				
	SERIALIN	Setup	I	3		2				
		Hold		0		0				
	SERIALOUT	Prop	O		12		10			1
	TPH1	Setup	I	3		2				
		Hold		0		0				
	TPH2	Setup	I	3		2				
		Hold		0		0				
	TESTPWR*	Setup	I	3		2				
		Hold		0		0				
	NPSPARE<2:0>	Setup	I/O							
		Hold								
		Prop								

Mechanical Data

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems. NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.



NOTES:

1. GOLD PLATE 60. INCHES OVER 80GOLD PLATE 60. INCHES NICKEL.
2. OPTION:

	h	
01	.165	PROTOTYPES
02	.155	PRODUCTION

3. LID AND HEATSINK ARE TIED TO GROUND
4. IF NOT INDICATED DIM TOLERANCE IS +/- 1%

Figure 63 Nx586 Package Diagram (top)

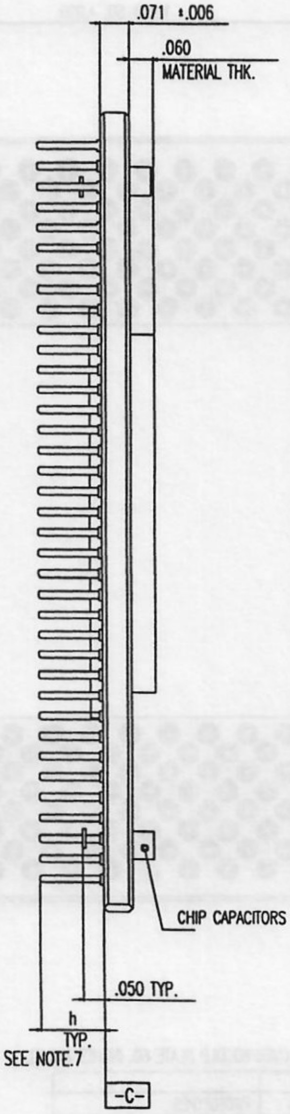
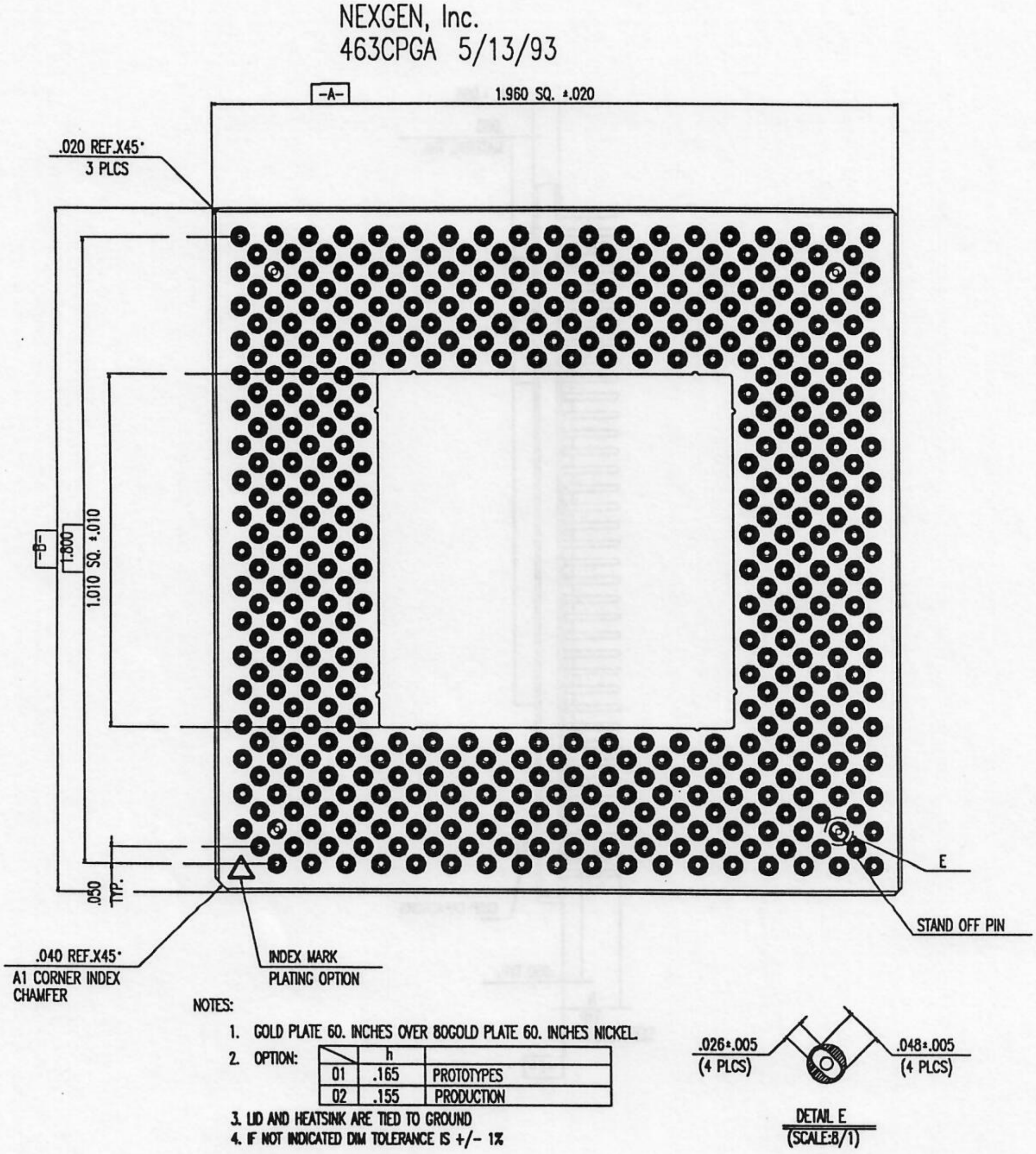


Figure 64 Nx586 Package Diagram (side)



NEXGEN, Inc.
463CPGA 5/13/93

NOTES:

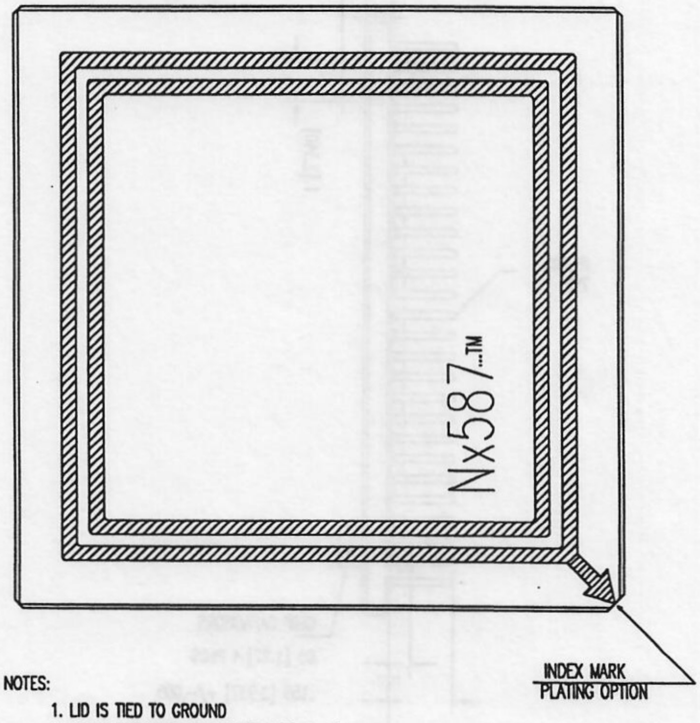
1. GOLD PLATE 60. INCHES OVER 80GOLD PLATE 60. INCHES NICKEL.
2. OPTION:

	h	
01	.165	PROTOTYPES
02	.155	PRODUCTION

3. LID AND HEATSINK ARE TIED TO GROUND
4. IF NOT INDICATED DIM TOLERANCE IS +/- 1%

Figure 65 Nx586 Package Diagram (bottom)

183 CPGA 5/19/93



NOTES:

1. LID IS TIED TO GROUND
2. IF NOT INDICATED DIM TOLERANCE IS +/- 1%

Figure 66 Nx587 Package Diagram (top)

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems. NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ***ONLY****. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.*

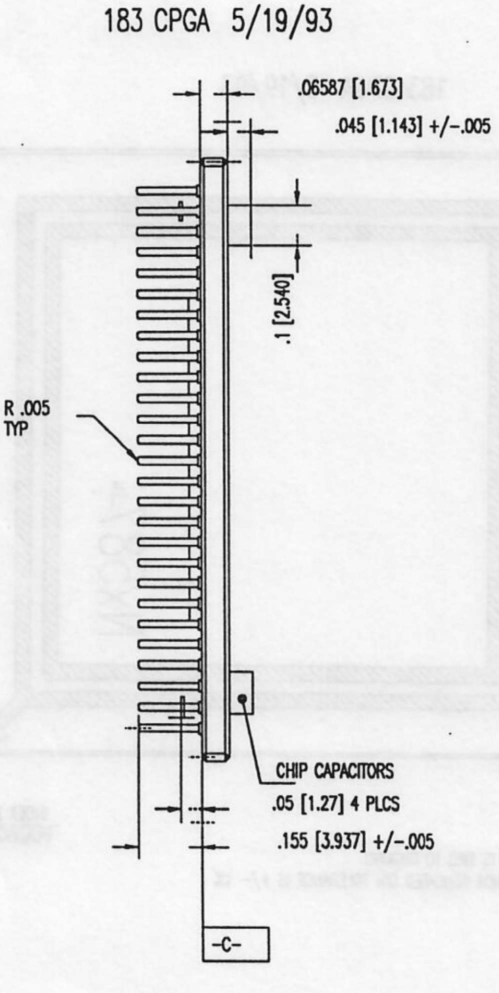


Figure 67 Nx587 Package Diagram (side)

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems. NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

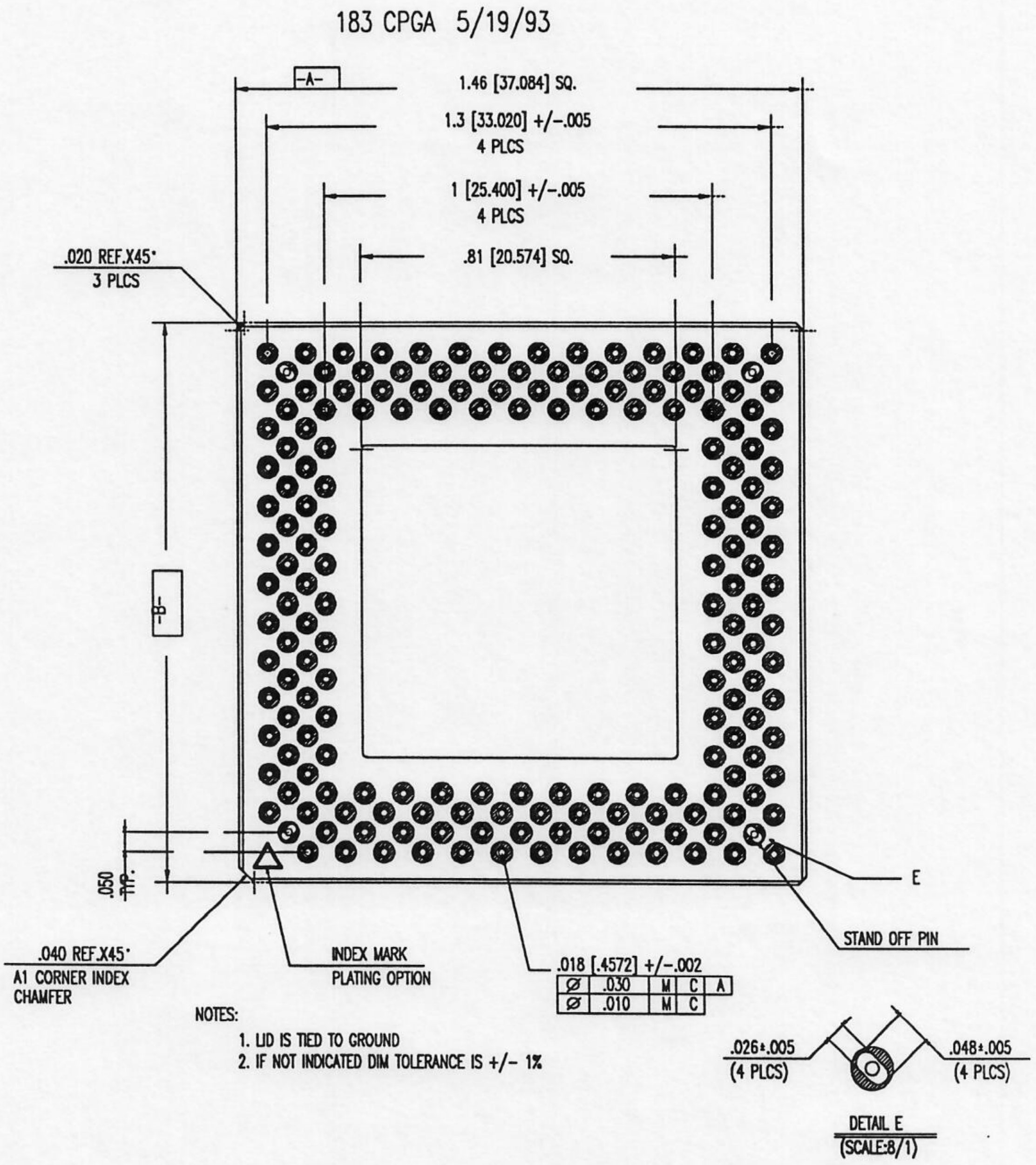


Figure 68 Nx587 Package Diagram (bottom)

Ordering Information

Figure 69 lists the currently available Nx586 and Nx587 products.

Part Number	*Description*
Nx586-66PC	Nx586 processor, 66MHz, I-PGA 463 package
Nx586-80PC	Nx586 processor, 80MHz, I-PGA 463 package
Nx587-66PC	Nx587 numerics processor, 66MHz, I-PGA 183 package
Nx587-80PC	Nx587 numerics processor, 80MHz, I-PGA 183 package
Nx586HS	Heat sink for Nx586 processor
Nx587HS	Heat sink for Nx587 numerics processor

Figure 69 Part Numbers and Descriptions

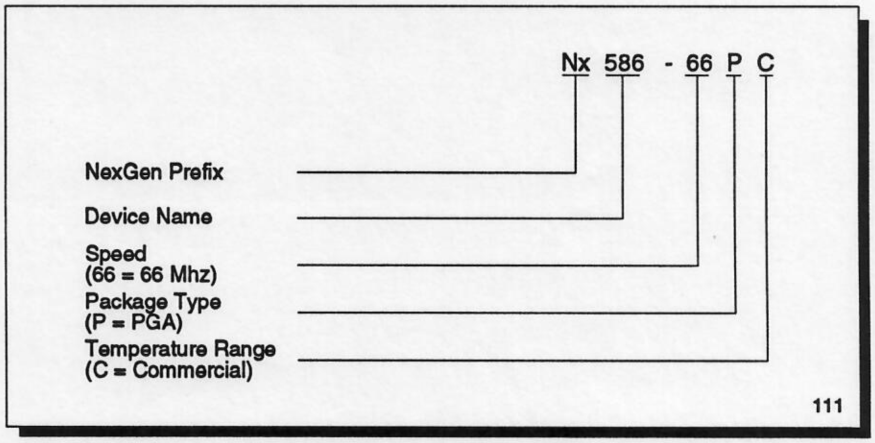


Figure 70 Part Number Syntax

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems.

Glossary

Access—A bus master is said to "have access to a bus" when it can initiate a *bus cycle* on that bus. Compare *bus ownership*.

Adapter—A central processor, memory subsystem, I/O device, or other device that is attached to a slot on the NexBus, VL-Bus, or ISA bus. Also called a *slot*.

Address Unit—A logic block in the processor that contains the memory-segmentation logic and segment registers used in translating logical addresses to linear addresses, and the paging logic and translation lookaside buffer (TLB) used in translating linear addresses to physical addresses. It also implements the x86 privilege-level protection mechanisms.

Aligned—Data or instructions that have been rotated until the relevant bytes begin in the least-significant byte position.

Allocating Write—A read-to-own (read for exclusive ownership of cacheable data) followed by a write to the cache.

Arbiter—A resource-conflict resolver, such as the NexBus arbiter. The NxPC chip includes a NexBus arbiter.

b—Bit.

B—Byte.

Bank—In a cache, same as *set* and *way*. In main memory, a qword-wide group of addressable locations.

Block—See *cache block*.

Block Operation—Burst transfers of four qwords (32 bytes). Compare *cache block*.

BPC—Same as *branch prediction cache*.

Branch Prediction Cache—A cache in the processor's Decode Unit that stores, for several branch instructions, the instruction's address, the address of its last target, its branch history, and several instructions for that target stream.

Burst Transfer—Same as *block operation*.

NexGen, Nx586, Nx587, NexBus, and NxPC are trademarks of NexGen Microsystems. NOTICE: THESE MATERIALS ARE PROPRIETARY TO NEXGEN AND ARE PROVIDED PURSUANT TO A CONFIDENTIALITY AGREEMENT FOR YOUR EVALUATION ONLY. ANY VIOLATION IS SUBJECT TO LEGAL ACTION.

Bus-Crossing Cycle—A bus cycle that originates on one bus and accesses a resource on another bus. In single-processor systems supported by the NxPC chip, if the addressed location of a cycle initiated by the NexBus processor, 82C206 DMA, or an ISA-bus master is not in main memory, the cycle is converted to a VL-Bus cycle and thereby becomes a bus-crossing cycle. If a VL-Bus device does not respond in a specified time period, the cycle falls through to an ISA-bus cycle.

Bus Cycle—A complete transaction between a bus master and a slave. For the Nx586 processor, a bus cycle is typically composed of an address and status phase, a data phase, and any necessary idle phases. Also called a *bus operation*, or simply *operation*.

Bus Operation—Same as *bus cycle*.

Bus Ownership—A bus is said to be owned by a master when the master can initiate cycles on the bus. In single-processor systems supported by the NxPC chip, the NxPC chip arbitrates access to all buses. The master to which bus ownership is granted controls only its own interface with the NxPC chip. The NxPC chip, on behalf of that master, acts as a master on the other buses in the system. It does this so as to support the master in the event that a bus-crossing operation is requested. Compare *access*.

Bus Phase—Part of bus cycle that lasts one or more bus clocks. For example, it may be a transfer of address and status, a transfer of data, or idle clocks.

Bus Sequence—A sequence of bus cycles (or operations) that must occur sequentially due to their being explicitly locked by the continuous assertion of the master's AREQ* and/or LOCK* signals, or implicitly locked by the GDCL signal.

Cache and Memory Unit—A logic block in the processor that contains the L1 instruction and data caches, the L2 cache control and interface, the NexBus interface, and the NexBus cache coherency logic. It maintains a three-stage read pipeline on the incoming address and data buses, a write reservation queue, read-after-write forwarding (by-pass) paths, and pipeline interlocks. It also remembers which instructions have been copied by the Decode Unit into the Branch-Prediction Cache (BPC).

Cache Block—A 32-byte unit of data in a cache. The Nx586 processor's caches are organized around such blocks. Each cache block has an associated tag and MESI-protocol state. Cache blocks can be fetched atomically as a contiguous group of 32-bytes or in eight-byte subblock units. Compare *cache line*.

Cache-Block Tag—The high-order address bits of a cache block that identifies the area of memory from which it was copied. During a cache lookup, the high-order address bits of the processor's operand is compared with the tags of all blocks stored in the cache.

Cache Hit—An access to a cache block whose state is *modified, exclusive,* or *shared* (i.e., not *invalid)*. Compare *cache miss*.

Cache Line—If a *cache block* can be fetched atomically (rather than in subblock units), the concepts of cache block and cache line are identical. However, in the Nx586 processor, cache blocks are often fetched in eight-byte subblock units, leaving only parts of the cache block valid. Compare *cache block*.

Cache Lookup—Comparison between a processor address and the cache tags and state bits in all four sets (ways) of a cache.

Cache Miss—An access to a cache block whose state is *invalid*. Compare *cache hit*.

Cache Subblock—An eight-byte (qword) sector of a 32-byte cache block, with state bits. Cache blocks can be fetched atomically (as a unit) or in eight-byte (qword) subblocks. See *cache block*. A cache subblock is sometimes called a *sector*.

Caching Master—A bus master that internally caches data originated elsewhere. The caching master must continually monitor the bus to guarantee cache coherency. Masters on buses other than the NexBus can maintain caches, but they must be write-through (not write-back) caches.

Clean—Same as *exclusive*.

Clock Cycle—Unless otherwise stated, this a *processor-clock cycle* rather than a bus-clock cycle. The Nx586 processor's clock runs at twice the frequency of the NexBus clock (CLK). The level-1 cache runs at twice the frequency of the processor clock. The level-2 cache runs at the same frequency as the NexBus clock (CLK).

Clock Phase—One-half of a processor clock cycle.

Crossing Operation—Same as *bus-crossing operation*.

Cycle—See *bus cycle*, *clock cycle*, *bus phase*, and *clock phase*.

D Cache—The level-1 (L1) data cache.

Decode Unit—A logic block in the processor that issues decoded instructions and performs other functions.

Device—Same as *adapter*.

Dirty—Same as *modified*.

Dword—A doubleword. A four-byte (32-bit) unit of data that is addressed on an four-byte boundary. Also called a *dword* (doubleword). Same as *quad*.

Exclusive—One of the four states that a 32-byte cache block can have in the MESI cache-coherency protocol. *Exclusive* data is owned by a single caching device and is the only known-correct copy of data in the system. Also called *clean* data. When exclusive data is written over, it is called *modified* (or *dirty)* data.

Flush—(1) To write back a cache block to memory and invalidate the cache location, also called *write-back and invalidate*, or (2) to invalidate a storage location such as a register without writing the contents to any other location. This is an ambiguous term that is best not used.

Functional Unit—The Decode Unit, Address Unit, Integer Unit, Numerics Processor, or Cache and Memory Unit.

Group Signal—A NexBus control signal that represents the logical OR of several inputs. These signals typically have signal names that begin with the letter "G". They provide fast control response for systems with multiple devices on the NexBus. An active-low signal (such as ALE*) is driven by each NexBus device to a NAND gate on the backplane to produce an active-high group signal (such as GALE). This group signal—in effect an OR of all its inputs—is then distributed back to each NexBus device. This type of signaling works faster than open-collector buses.

I Cache—The level-1 (L1) instruction cache.

IDT—Interrupt descriptor table.

Integer Unit—A logic block in the processor that contains the ALUs and the rest of the datapath.

Interlocks—Mechanisms that enforce ordering of transactions in the processor.

Intervenor—A caching device on the NexBus that intervenes in the NexBus operation of another master to provide data from one of its modified cache blocks. To maintain cache coherency, the intervenor must update memory before other NexBus masters can access that location in memory.

Invalid—One of the four states that a 32-byte cache block can have in the MESI cache-coherency protocol. *Invalid* data is not correctly associated with the tag for its cache block.

Invalidate—To change the state of an cache block to *invalid.*

L1—The level-1 cache located on the Nx586 processor chip.

L2—The level-2 cache located in SRAM connected to the processor's SRAM bus and controlled by logic on the Nx586 processor.

Line—See *cache block.*

Main Memory—See *memory.*

Memory—A RAM or ROM subsystem located on any bus, including the *main memory* most directly accessible to a processor. In single-processor systems using the NxPC chip, main memory is the DRAM on the NxPC chip's memory bus. Also called *main memory.*

MESI—The cache-coherency protocol used in the Nx586 processor. In the protocol, cached blocks in the L2 write-back cache can have four states (modified, exclusive, shared, invalid), hence the acronym MESI. See *modified, exclusive, shared,* and *invalid.*

Modified Write-Once Protocol—The cache-coherency protocol used in the Nx586 processor. See *MESI.*

Modified—One of the four states that a 32-byte cache block can have in the MESI cache-coherency protocol. *Modified* data is *exclusive* data that has been written to after being read from lower-level memory, and is therefore the only valid copy of that data. Also called *dirty or stale.*

MWO—See *modified write-once protocol.*

NB—Same as *NexBus.*

NexBus—A 64-bit synchronous, multiplexed, multiprocessor bus defined by NexGen. The Nx586 processor and NxPC system controller are interfaced on this bus. When bus transceivers are used between the Nx586 processor and other system devices, the NexBus on the processor (unbuffered) side of the transceivers is called NxAD<63:0> and on the buffered side is called AD<63:0>. When no bus transceivers are used, as in systems using the NxPC system controller (which can emulate the bus transceivers), the NexBus is called NxAD<63:0>.

No-Op—A single-qword operation with BE<7:0>* all negated. No-ops address no bytes and do nothing except consume processor cycles.

NP—Same as *Nx587* and *numerics processor.*

Numerics Processor—The Nx587 numerics processor (NP) chip. The logic in the numerics processor is integrated into the parallel pipeline of the Nx586 processor. All of the 32-, 64-, and 80-bit formats in the IEEE Standard 754 are supported, and the chip is binary-compatible with all x87 and i486 floating-point code.

Nx586—The Nx586 processor (CPU) chip.

Nx587—The Nx587 numerics processor (NP) chip. See *numerics processor.*

NxPC—A NexBus system controller chip that supports a single Nx586 processor or Nx586/587 pair, main memory, 82C206 peripheral controller, VL-Bus, and ISA bus.

Octet—Same as *qword.*

Operation—See *bus operation* and *micro-operation.*

Owned—A cache block whose state is *exclusive* (owned clean) or *modified* (owned dirty). See also *bus ownership.*

Ownership—See *bus ownership.*

Page—(1) a DRAM page of 256kb (single-sided SIMMs) or 512kb (double-sided SIMMs), or (2) a 4kB block of adjacent memory locations, used by the paging hardware.

Peripheral Controller—A chip that supports interrupts, DMA, timer/counters, and a real-time clock. The NxPC chip is designed to interface to an 82C206 peripheral controller.

Phase—See *bus phase* and *clock phase.*

PLL—Phase-locked loop.

Present—Same as *valid.*

Processor—Unless otherwise specified, an Nx586 processor. When the NxPC chip is used in a system design, there can be only one processor on the NexBus so "the processor" refers to this single Nx586 processor.

Processor Clock—The Nx586 processor clock. See *clock cycle.*

Qword—A quadword. An eight-byte unit of data that is addressed on an eight-byte boundary. Also called an *octet.*

Sector—Same as *cache subblock.*

Set—In a cache, one of the degrees of associativity. The group of cache blocks in such a set. Same as *bank* and *way.*

Shared—One of the four states that a 32-byte cache block can have in the MESI cache-coherency protocol. *Shared* data is valid data that can only be read, not written.

Snoop—To compare an address on a bus with a tag in a cache, so as to detect operations that are inconsistent with cache coherency.

Snoop Hit—A snoop in which the compared data is found to be in a *modified* state. Compare *snoop miss.*

Snoop Miss—A snoop in which the compared data is not found, or is found to be in a *shared* state. Compare *snoop hit.*

Source—In timing diagrams, the left-hand column of the diagram indicates the "source" of each signal. This is the chip that originated the signal as an output. When signals are driven by multiple sources, all sources are shown, in the order in which they drive the signal. The source of a signal that takes on a different name as it crosses buses through transceivers is shown as the transceivers overwhich the signals cross, subscripted with a symbol indicating the logic that originally output the signals. The source of group-ORed signals (such as GXACK) is likewise subscripted with a symbol indicating the logic that originally output the activating signal (such as XACK*).

Stale—Same as *modified.*

System Bus—A bus to which the NexBus interfaces. The NxPC chip supports two system buses, VL-Bus and ISA bus.

System Controller—The device or logic that provides NexBus arbitration and interfacing to main memory and any other buses in the system. The NxPC chip is a system controller.

T-Byte—An 80-bit floating-point number.

TLB—Translation lookaside buffer. An on-chip cache that contains references to the most recently used page directory and page table entries. By using the TLB, linear addresses can be translated to physical addresses without having to access a page directory, page table, and/or page from disk.

Unmasked Floating-Point Exception—An exception detected by a floating-point unit that is handled by the processor rather than handled in a default way by the floating-point unit.

Word—An two-byte (16-bit) unit of data.

Write Queue—A buffer in the Cache and Memory Unit.

Index

A.C. Characteristics 133
Aborts 101
Absolute Maximum Ratings 131
Access 149
Active-Low Signals ix
AD bus 15
Adapter 149
address and status phase 16, 18
address calculations 127
Address Latch Enable 12
Address Map
  I/O 100
  Memory 98
address phase 16, 72, 77
Address Translation 96
Address Unit 52, 149
Addresses 93
Addressing x
ADRS 16, 17
ALE* 2, 12, 85
Aligned 149
Alignment 92, 93, 127
Allocating Write 149
Alternate bus 10
Alternate-Bus Request 11
ANALYZEIN 27
ANALYZEOUT 27
Applications Engineering ix
Arbiter 11, 149
Arbitration 32, 46, 48, 72
Architecture 45
AREQ* 10, 11, 72, 85
Arithmetic Operations 109
ASCII Strings 91
asterisk ix
b x, 149
Bank 149
BE 17, 19, 81, 82, 83
Binary compatibility 1
Binary-coded decimal (BCD) numbers 91
Bit Strings 91
BLKSIZ 20, 21, 77
Block 149
Block Length 21
Block Operation 149
Block Size 20
Block Transfer Sizes 21
Block Write With One Wait State 80
BPC 59, 149
Branch Prediction Cache 149
branch-prediction cache 59
Buffered Address And Data Bus 2
Burst Memory Operations 77
Burst Transfer 149
Bus Arbitration 2
Bus Cycle 150
Bus Lock 10
Bus Operation 150
Bus Operations 71
  Cache 83
  Fastest Block Read 79
  Fastest Block Write 80
  Fastest Single-Qword Read 74
  Fastest Single-Qword Write 76
  Halt and Shutdown 82
  Halt and Shutdown Encoding 82
  I/O 81
  Intervenor 84
  nterrupt-Acknowledge Cycle 82
  Single-Qword Read Hits Modified Cache Block 86
Bus Ownership 150

Bus Phase 150
Bus Sequence 150
Bus Signals x
Bus Structure 45
Bus-Crossing Cycle 150
Bus-Cycle Types 20
Buses
  AD 15, 48
  Alternate 10
  Cycles 71
  Level-2 Cache 50
  NexBus 45
  Numerics Processor 50
  NxAD 2, 16, 48, 73
  Operations 71
  Snooping 60
  Structure 45
  VL, PCI, ISA, EISA, MCA 46
Byte Enables 17
byte-enable bits 86
Byte-Enable Usage 19
cachable 81
CACHBL 21
Cache
  Cache and Memory Subsystem 58
  Cache and Memory Unit 52
  Characteristics 58
  Coherency 59
  Control 14
  Data 58
  Instruction 58
  Level-2 22, 50
  Level-2 Cache Accesses 71
  Ports 59
  States 61
Cache and Memory Subsystem 58
Cache and Memory Unit 52, 150
Cache Block 150
Cache Control 14
cache fills 77
Cache Hit 150
Cache Line 151
Cache Lookup 151
Cache Miss 151
Cache Subblock 151
Cache-Block Tag 150
Cache-Control Operations 83
Cache-Hit Reads 57
Cacheable 21, 78
Caching Master 151
CADDR 22, 72
Capacitance 132, 133
Case temperature under bias 131
CBANK 22, 72
Cclk 132
CDATA 22, 72
Cin 132
CKMODE 25, 40, 68
Clean 151
CLK 25, 40, 50, 68, 71
Clock 25, 40
Clock Cycle 151
Clock Input Reference 25, 40
Clock Mode 25, 40
Clock Mode Select 25, 40
Clock Output Reference 25, 40
Clock Phase 151
Clock Phase 1 25, 40
Clock Phase 2 25, 40
Clock Rates 1
Clock-Signal Timing 137
Clocking Modes 68
Clocks 50
  Cycles 151
  Generation 68
  L1-cache 50
  L2-cache 50
  Modes 68
  NexBus 50
  processor 50
  Timing 137
COE* 72
COEA* 22
COEB* 22
Coherency 59
Compatibility 1, 32, 89
conditional jumps 127
Control Operations 110
Cout 132
CPARITY 27
Crossing Operation 151
CS 113
CWE 22
CWE* 72
Cycle 151
Cycle Control 12
Cycle Types 20
D Cache 58, 151

D.C. Characteristics 132
D/C* 19
Data x
Data Conversion 110
Data Manipulation 109
Data or Code* 19
data phase 16, 73, 77
Data Transfer 110
Data Types 90
DC supply voltage 131
DCL* 14, 64, 66, 84, 85, 86
Decode Unit 52, 151
DEVICE 11, 19, 151
Devices Per Slot 11
dirty 66, 151
Dirty Cache Line 14, 64
DMA 87
Double-Precision Arithmetic 104
doubleword x, 16
DS 113
dword x, 151
Dword Address 16
EFLAGS register 106
Electrical Data 131
Endian Convention x
ES 113
Exceptions 101
Exclusive 59, 65, 83, 151
External Phase Inputs 68
External Processor Clock 68
Fastest Block Read 79
Fastest Block Write 80
Fastest Single-Qword Read 74
Fastest Single-Qword Write 76
Faults 101
Flag Operations 111
Flags 106
FLAGS register 106
Flush 151
Four-Qword Block 21
Four-Qword Block Read (Cache-Block Fill) 19
Four-Qword Block Write 19
FS 113
FSPARE 42
Functional Unit 152
G x
GALE 2, 12, 16, 74, 77, 83, 85
Gate Address 20 26
GATEA20 2, 11, 26
GBLKNBL 14, 20, 64, 74, 77, 81
GDCL 14, 64, 84
Global Reset (Power-Up Reset) 26, 41
GNT* 10, 11, 72, 85, 86
Grant NexBus 11
GREF 27
Ground Reference 27
Group Address Latch Enable 12
Group Block (Burst) Enable 14
Group Dirty Cache Line 14
Group Shared Data 15
Group Signal 152
Group Signals 2
Group Transfer Acknowledge 13
Group Transfer Hold 13
Group Try Again Later 12
GS 113
GSHARE 15, 63, 64, 65, 84
GTAL 12, 73
GXACK 13, 16, 20, 64, 65, 73, 74, 77, 85, 86
GXHLD 16, 73, 74, 78
Halt 20, 82
Halt and Shutdown 82
Halt and Shutdown Encoding 82
HROM 27
I Cache 58, 152
I/O 19, 99
I/O Address Map 100
I/O Data Read 20
I/O Data Write 20
I/O operations 73, 81
I/O Reads 57
I/O space 81
Icc 132
Icc1 132
IDT 152
Iin 132
Input Capacitance 132
Input High Voltage 132
Input Leakage Current 132
Input Low Voltage 132
Input/Output 104
Instruction Encoding 117
Instruction Format 114
Instruction Latency 117
Instruction Prefixes 113, 127
Instructions 108
INT instruction 67
Integer Unit 52, 152

Interlocks 152
Internal Architecture 52
Interrupt 26
Interrupt Acknowledge 20
interrupt handling 23
interrupt vector 81
Interrupt-Acknowledge 81
Interrupts 67
Interrupts and Exceptions 101
Intervenor 152
intervenor operation 66
Intervenor Operations 84
INTR* 2, 11, 26, 67
Invalid 60, 64, 152
Invalidate 83, 152
IREF 25, 40
k x
L1 152
L1-cache clock 50
L2 152
L2 Cache Address 22
L2 Cache Bank 22
L2 Cache Data 22
L2 Cache Output Enable A 22
L2 Cache Output Enable B 22
L2 Cache Write Enable 22
L2-cache clock 50
latency 117
Level-2 Cache 71
Level-2 Cache Bus 50
Level-2 Cache Read and Write 72
Level-2 Cache Signals 22
Line 152
LOCK 113
LOCK* 10, 72, 85
Logical Operations 109
M x
M/IO* 19
Main Memory 152
Main-memory control and arbitration 46
Maskable Interrupt 26
Master ID 19
Mechanical Data 139
Memory 19, 96, 152
Memory Address Map 98
Memory Code Read 20
Memory Data Read 20
Memory Data Write 20
memory operations 73
  Burst 77
  Single-Qword 73
Memory or I/O* 19
Memory Reads 57
Memory Reads on NexBus 57
MESI 152
MESI cache-coherency protocol 59
MID 19
minimum fall time (tf) 132
minimum rise time (tr) 132
Modified 59, 66, 153
Modified Cache-Block Hit During Four-Qword (Block) Operations 86
Modified Cache-Block Hit During Single-Qword Operations 85
modified write-once 59
Modified Write-Once Protocol 152
modified, exclusive, shared, or invalid (MESI) 59
multiple processors 48
multiplication or division 127
Multiprocessing Support 69
Multiprocessor System Diagram 49
multitasking 89
MWO 59, 153
Names ix
NB 153
NexBus ix, 10, 45, 153
NexBus Address and Status, or Data 16
NexBus Arbiter 11, 48, 72
NexBus Arbitration and Address Phase 72
NexBus Clock 25, 40, 50
NexBus Request 10
NexBus Slot ID 11
NMI* 2, 11, 26, 67
No-Op 153
no-ops 83
Non-Maskable Interrupt 26
Notation ix, 114
NP 153
NPDATA 24, 39
NPIRQ* 27, 41
NPNOERR 23, 38
NPOUTFTYP 23, 38
NPPOPBUS 23, 38
NPPOPTAG 27, 42
NPRREQ 23, 38
NPRVAL 24, 39
NPTAG 24, 39
NPTAGSTAT 23, 38

NPTERM 23, 38
NPWREQ 24, 39
NPWVAL 24, 39
NREQ* 10, 11, 72
NSPARE 27, 42
nterrupt-Acknowledge Cycle 82
Numerics Processor 52, 153
Numerics Processor Bus 50
Numerics Processor Data 24, 39
Numerics Processor Interrupt Request 41
Numerics Processor Micro-Operations Bus 23, 38
Numerics Processor No Error 23, 38
Numerics Processor Output Type 23, 38
Numerics Processor Read Request 23, 38
Numerics Processor Read Valid 24, 39
Numerics Processor Tag Bus 24, 39
Numerics Processor Tag Status 23, 38
Numerics Processor Termination 23, 38
Numerics Processor Write Request 24, 39
Numerics Processor Write Valid 24, 39
Numerics-Processor Bus Signals (on Nx586) 23
Numerics-Processor Bus Signals (on Nx587) 38
Nx586 153
Nx586 Features and Signals 1
Nx587 153
Nx587 Features and Signals 32
NxAD 16
NxAD bus 2
NxADP 27
NxPC ix, 153
NxPC chip 2, 10, 67
Octet 153
Operand Alignment 92
Operand Sizes 91
Operands 90
Operating Frequencies 50
Operating Modes 89
operation 82, 153
Order of Transactions 57
Ordering Information 147
Output Capacitance 132
Output High Voltage 132
Output Low Voltage 132
OWN* 15, 20, 63, 64, 65, 83, 85
OWNABL 15, 63, 64, 65, 74, 77, 84, 85
Ownable 15, 63
Owned 153
Ownership 153
Ownership Request 20, 63
P4REF 27
Package Diagram 140
Page 153
page boundaries 127
Paged devices 14
paging 89
PARERR* 27
Part Numbers 147
passive exclusive use 83
Performance Optimization.i.Optimization 127
Peripheral control 46
Peripheral Controller 153
PH1 71
PH2 72
Phase 153
Phase-Locked Loop 68
PHE1 25, 40, 68
PHE2 25, 40, 68
Pinout Diagram 8
Pinouts
  Nx586 4
  Nx587 34
PLL 68, 153
PLL Analog Power 25, 40
Pointer Format 92
Pointers 91
POPHOLD 27
Power dissipation 131
Power Reference 27
preemptive exclusive use 83
Prefixes 113
Present 154
Primary-Signal Timing 134
Privilege Levels 89, 90, 112
Processor 154
Processor clock 50, 154
Processor Clock Phase 1 25, 40
Processor Clock Phase 2 25, 40
Programming 89
Protected Mode 89
Publications xi
quadword x, 17
Quiescent Current 132
qword x, 154
Qword Address 17
Read Order 57
read-modify-writes 57
References xi

Registers 102, 104
REPNE 113
REPx 113
Reserved 16, 21, 27, 42
reserved bits 32
Reserved Bits and Signals x
Reset 26
Reset CPU (Soft Reset) 26
RESET* 11, 26, 41
RESETCPU* 11, 26, 82
Sector 154
Segment Operations 111
segmentation 89
Segmented Memory 97
Self-Modifying Code 127
Semaphores 57
Serial In 27, 42
Serial Out 27, 42
SERIALIN 27, 42
SERIALOUT 27, 42
Set 154
SHARE* 15, 20, 63, 65, 77, 83, 84
Shared 60, 65, 154
Shared Data 15, 63
Shutdown 20, 82
signal organization 2, 33
Signals ix
  Arbitration 10
  Cache Control 14
  Clock 25, 40
  Cycle Control 12
  Interrupt 26
  Level-2 Cache 22
  NexBus 10
  NexBus Address and Data 16
  Numerics-Processor (on 586) 23
  Numerics-Processor Bus (on Nx587) 38
  Reserved 27, 42
  Reset 26
  Test 27, 42
Signed (two's-complement) integers 91
Single Qword 21
Single Qword Read or Write 19
Single-Qword Memory Operations 73
Single-Qword Read Hits Modified Cache Block 86
Single-Qword Read With Wait States 75
Single-Qword Write With Wait States 76
slot 11
SLOTID 2, 11, 19
SLOTID 0000 11
Snoop 154
Snoop Enable 21, 64
Snoop Hit 154
Snoop Miss 154
Snooping 14, 60, 64
SNPNBL 21, 64
Software Differences 128
Source x, 154
SRAM 50
SS 113
SSPARE 27
Stack Operations 98, 104, 111
stale 66, 154
State Transitions 61
Static operation 25
Storage Hierarchy 54
Storage temperature 131
String Operations 104, 110
subscript x, 73
Supply Current 132
Sync 83
synchronous signals 2
Syntax 114
System Administration 112
System Bus 154
System Controller 154
System ROM 46
T-Byte 154
Temperature 131, 132, 133
Test 27
Test Phase 1 Clock 28, 42
Test Phase 2 Clock 28, 42
Test Power 28
Test-Signal Timing 138
TESTPWR* 28
Timing Diagrams ix
TLB 154
TPH1 28, 42
TPH2 28, 42
Transaction Ordering 57
Transceiver BAD-Bus Clock Enable 15
Transceiver-to-BAD-Bus Output Enable 16
Transceiver-to-NexBus Output Enable 16
transceivers 15
Transfer Acknowledge 12
Transfer Hold 13
Transfer Type 19

Traps 101
Try Again Later 12
Unmasked Floating-Point Exception 155
Unsigned integers 91
Variable Shifts 104
VDDA 25, 40
video adapters 14
Vih 132
Vil 132
Virtual-8086 Mode 89
Voh 132
Vol 132
Voltage 132, 133
Voltage on any pin (with respect to ground) 131
W/R* 19
word x, 155
writable data 127
Write or Read* 19
Write Order 57
write queue 58, 155
Write-Back and Invalidate 83
Writes 57
x86 Architecture xi
XACK* 73, 74, 77
XBCKE* 15
XBOE* 16
XHLD* 13, 74, 78, 84, 85, 86
XNOE* 16
XPH1 25, 40
XPH2 25, 40
XREF 25, 40
XSEL 25, 40, 68