Hardware Architecture

The Nx586 processor and the optional integrated floating-point execution unit are tightly coupled into a parallel architecture with a distributed pipeline, distributed control, and rich hierarchy of storage elements. While the features of the two devices are sometimes listed separately elsewhere in this book, they are treated as an integrated architecture in this chapter. Both the Nx586 and Nx586 with the floating point have the identical system bus architecture. Therefore, the two devices are interchangeable within the processor socket.

Bus Structure

The Nx586 processor supports two external 64-bit buses: the processor bus, the L2 cache bus, and one internal 64-bit bus (for the integrated floating-point unit). All buses are synchronous to the NxCLK clock. The internal floating-point unit bus operates at the same speed as the processor core or twice the frequency of the local bus.

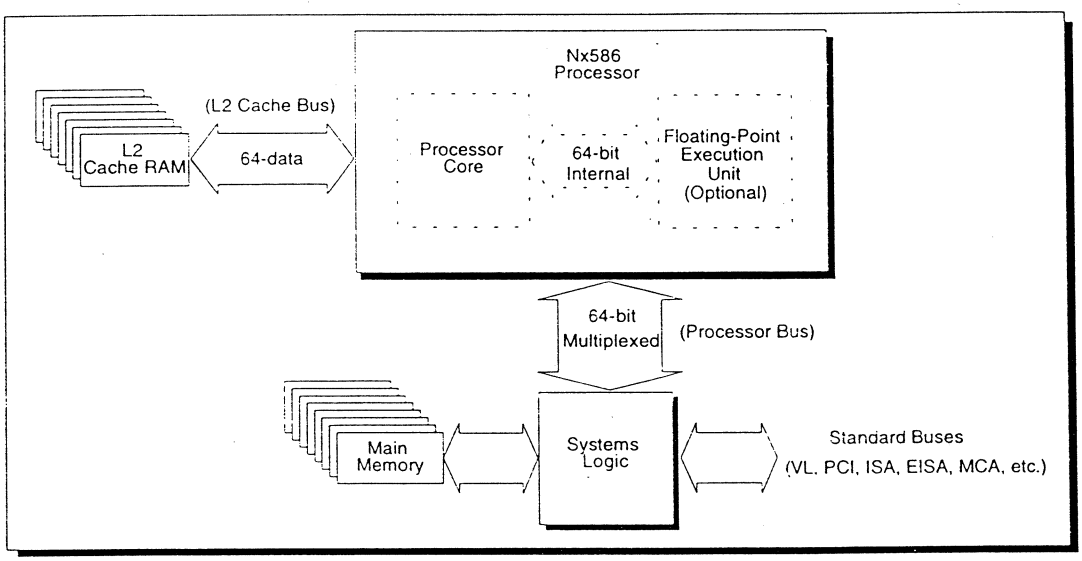

Figure 14 Nx586 Bus Structure Diagram

Processor Bus

The Nx586 supports two local bus interfaces, NexBus and NexBus^5. NexBus is considered a true CPU local bus. Where as, NexBus^5 is a NexGen proprietary system bus. During RESET* active, the XCVERE* pin is sampled for the local bus mode. XCVERE* determines what type of bus is generated by the processor. When pulled high, the Nx586 will generate the NexBus standard which requires external transceivers to connect the processor to the NexBus^5 system bus. Figure 15 is a system block diagram showing the Nx586 configured with a NexBus interface (XCVERE* = 1). The NexBus transceivers are high speed non-inverting registered transceivers controlled by signals provided by the Nx586.

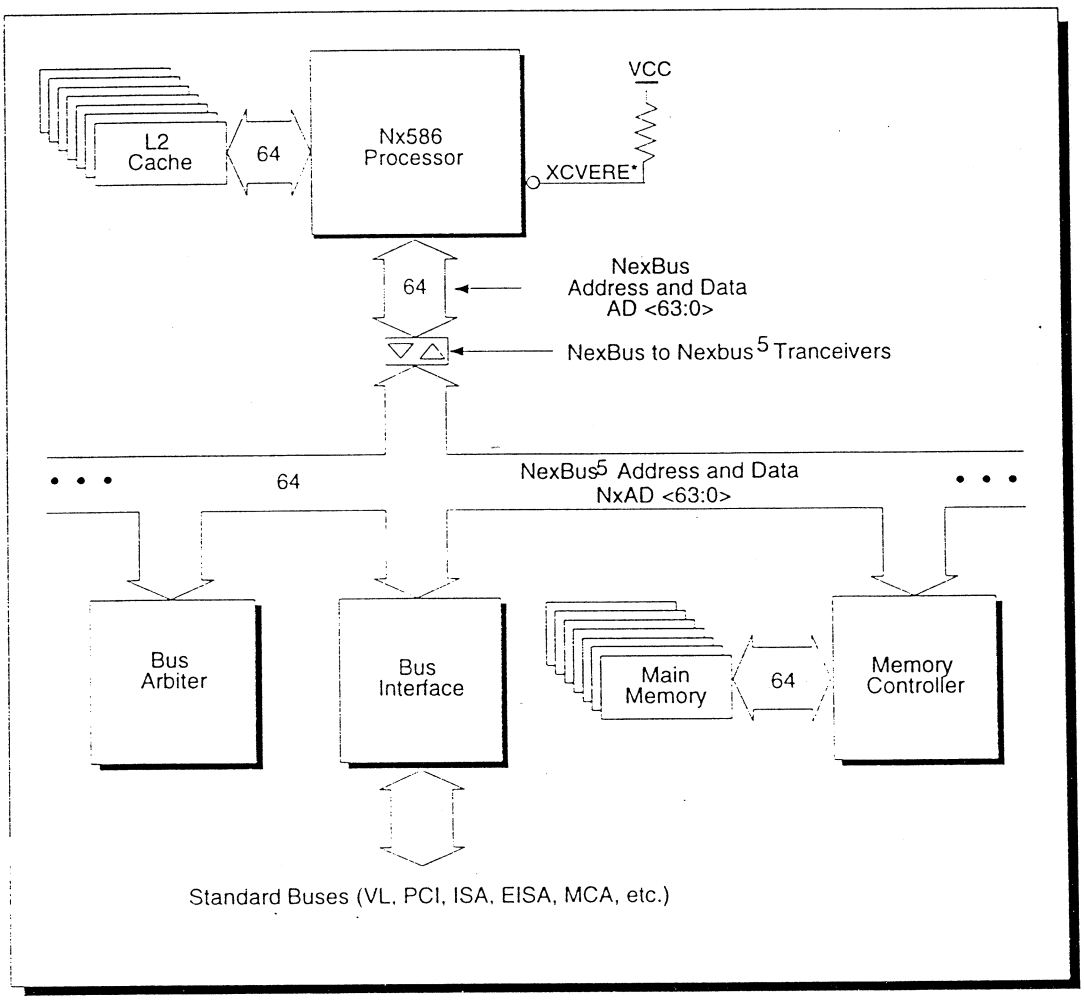

Figure 15 Nx586 Basic System Diagram

Another approach for processors configured with external NexBus transceivers is to include the transceiver within the systems logic. This however forces the systems logic to provide complete arbitration and signal routing to the processor via NexBus. As shown in figure 16, the example PC-AT compatible systems controller contains the system arbiter, the memory controller, the VL-Bus controller, and the ISA-Bus controller. The example systems logic is completely responsible for bus interconnections between NexBus, the memory bus, VL-Bus and ISA bus. The Integrated Peripheral Controller contains the interrupt controller, DMA controller, CMOS memory, timers and counters.

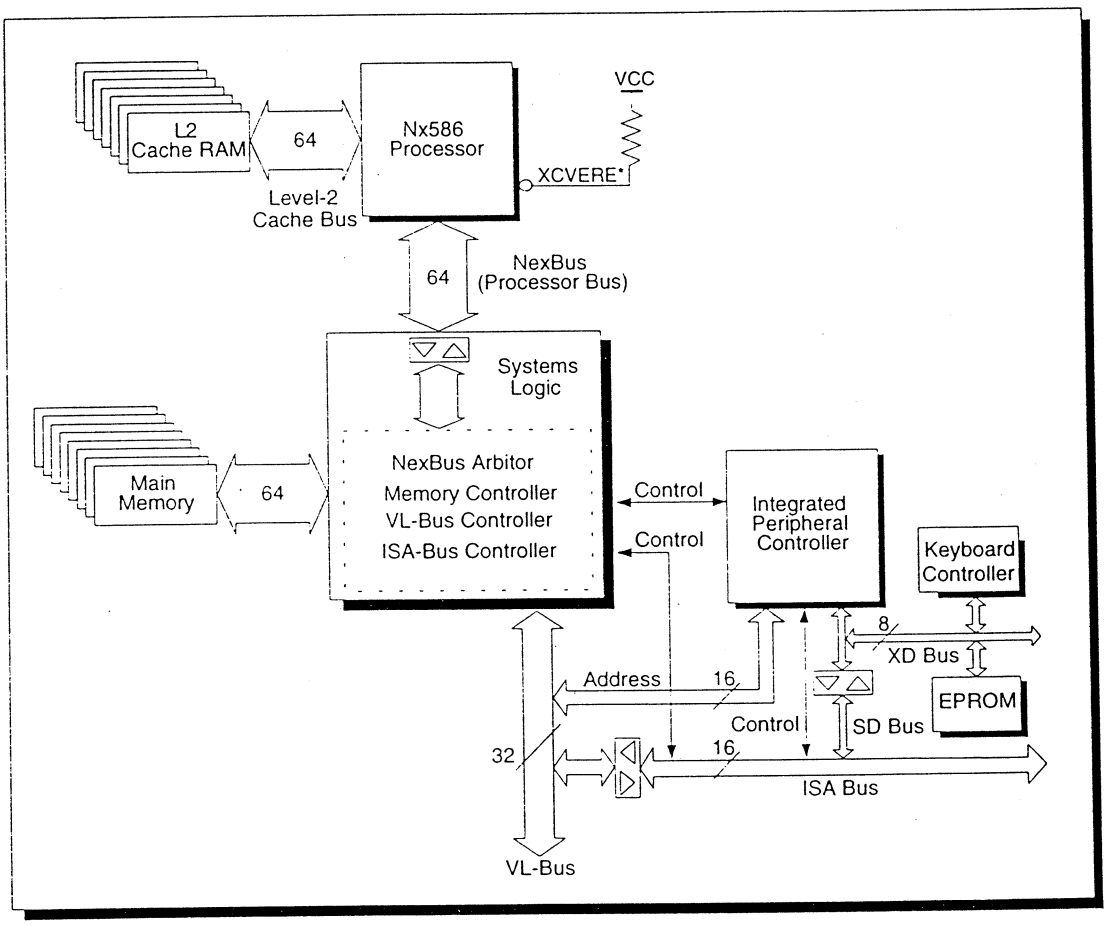

Figure 16 Nx586 System with Systems Logic containing NexBus Transceivers.

When XCVERE* is tied low, the Nx586 generates NexBus^5 directly. NexBus^5 is a 64-bit synchronous, multiplexed bus that supports all signals and bus protocols needed for cache-coherency. A modified write-once MESI protocol is used for cache coherency. The processor continually monitors the NexBus^5 to guarantee cache coherency.

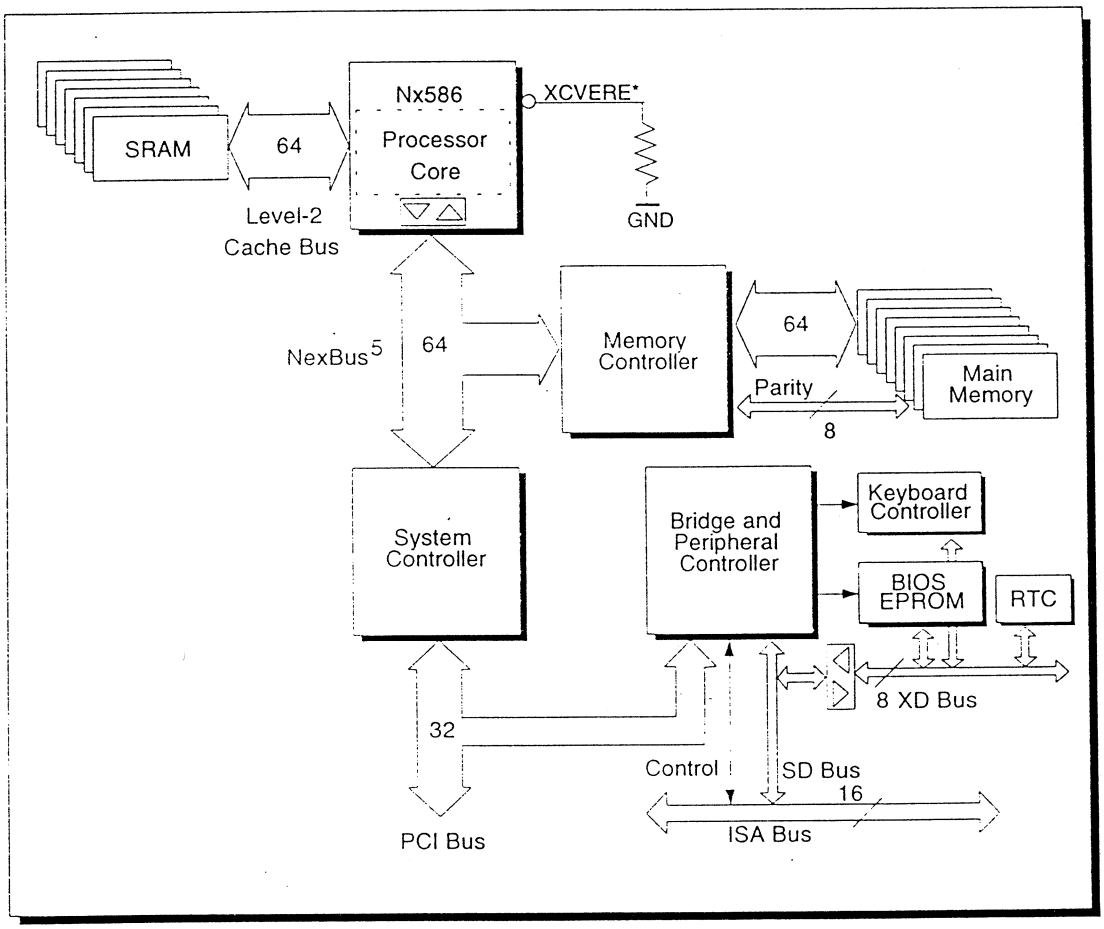

Figure 17 Example System with the Nx586 and NexBus^5.

The Nx586 based PCI system shown in figure 17 uses a chipset to connect the Nx586 via the NexBus^5 system bus. The example chipset is divided into two major components, the memory and systems controller. The system controller contains the NexBus^5 arbiter, and the PCI bus controller. Note, both the memory controller and the system controller are NexBus^5 devices and can respond directly to the processor. The ISA bus is generated by a PCI to ISA bridge chip.

L2 Cache Bus

The 64-bit L2 cache bus is dedicated to external SRAM cache. The bus carries one to eight bytes of cache data, or the tags and state bits for one to four cache banks (sets). The L2 cache write-policy can be programmed for write-back or write-through. Optionally, at power-on reset the L2 cache controller can be programmed for synchronous or asynchronous SRAMs. Bus accesses for each mode are identical except for the existence of the SRAM clocking signal on CKMODE for the synchronous SRAMs. Note, the synchronous SRAMs operate at the same frequency as the processor not at half the frequency.

The processor manages cache-coherency for both L2 and L1 caches. The 64-bit L2 cache bus is fully isolated and decoupled from the processor local bus also known as NexBus. The L2 cache bus does not require any arbitration to gain control of the bus. In fact, the L2 cache controller can start a L2 cache cycle on any processor clock. In addition, speculative cycles are supported on the L2 cache bus. The processor can request data from the L2 cache controller and terminate the cycle at any time during the access. 32-bytes is the unit of transfer between the memory and the cache. There is no data bursting from the L2 cache memory. Since no arbitration is necessary, the L1 cache line fills are just back-to-back read cycles.

Internal 64-bit Execution Unit Bus

The Nx586 contains an internal 64-bit bus dedicated to the optional floating-point execution unit. Discrete arbitration signals implement a simple protocol between the two devices. 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 Floating-Point execution unit a tightly coupled component of the execution pipeline.

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

Operating Frequencies

There are four operating frequencies associated with the processor, as shown in Figure 18:

- *NexBus/NexBus*^{5}—Operates at the frequency of the system clock (NxCLK).
- *Processor*—Operates at twice the frequency of the NxCLK. The Nx586 processor and the Floating Point Execution Unit both operate at the same frequency.
- *L1 (On-Chip) Cache*—Operates at twice the frequency of the processor clock.
- *L2 (Off-Chip) Cache*—Operates at the same frequency as the NxCLK. Transfers between L2-cache and the processor occur at the peak rate of one octet every two processor clocks, but the transfers (which can be back-to-back) can begin on any processor clock. Data is returned to the processor on the third clock phase after an access is started.

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

Figure 18 shows the relative clocking frequencies for a Nx586 processor. The NxCLK clock determines the systems overall operating speed. The NxCLK clock sets the NexBus/NexBus^5 operating frequency. The processor's on-board PLL doubles the frequency of NxCLK making the Nx586 operate at twice the frequency of NexBus. The dual port nature of the L1 caches requires the L1 cache controllers to operate at double the frequency of the processor. The effective operating frequency of the L2-cache is half of the processor.

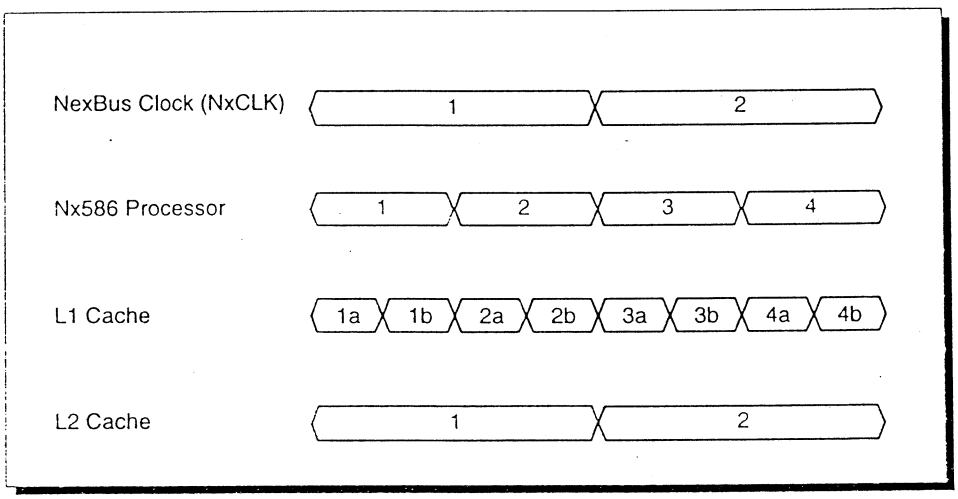

Figure 18 System Clocking Relationships

The processor uses an on-chip phase-locked loop and NxCLK to internally generates a two phase non-overlapping clock, shown in Figure 18 as the 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.

Internal Architecture

Figure 19 shows the relationship between functional units within the Nx586 processor. The main processing pipeline is distributed across five units:

- Decode Unit
- Address Unit
- Cache and Memory Unit
- 2 Integer Units
- Floating Point Execution Unit (optional)

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 (NexBus or NexBus^5) that is visible outside the processor.

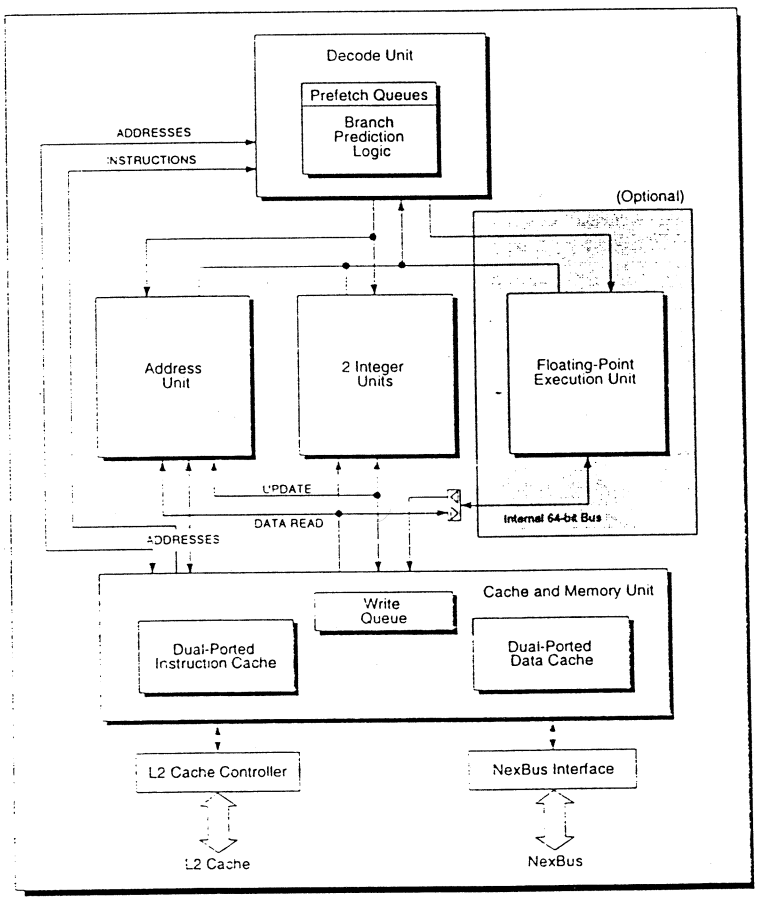

Figure 19 Nx586 Internal Architecture

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 cache-coherency protocol (MESI) 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) branch prediction capability 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, a system using a memory controller chip that maintains a prefetch queue between the L2 cache and main memory can continuously pre-load 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 20 shows this hierarchy during a read cycle in a system. Figure 21 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^5 is multiplexed between address/status and data, the L2-cache data bus carries only data at 64 bits every NexBus^5 clock cycle. The disk subsystem and software disk cache are included in the figures for completeness of the hierarchy; the software disk cache is maintained in memory by some operating systems.

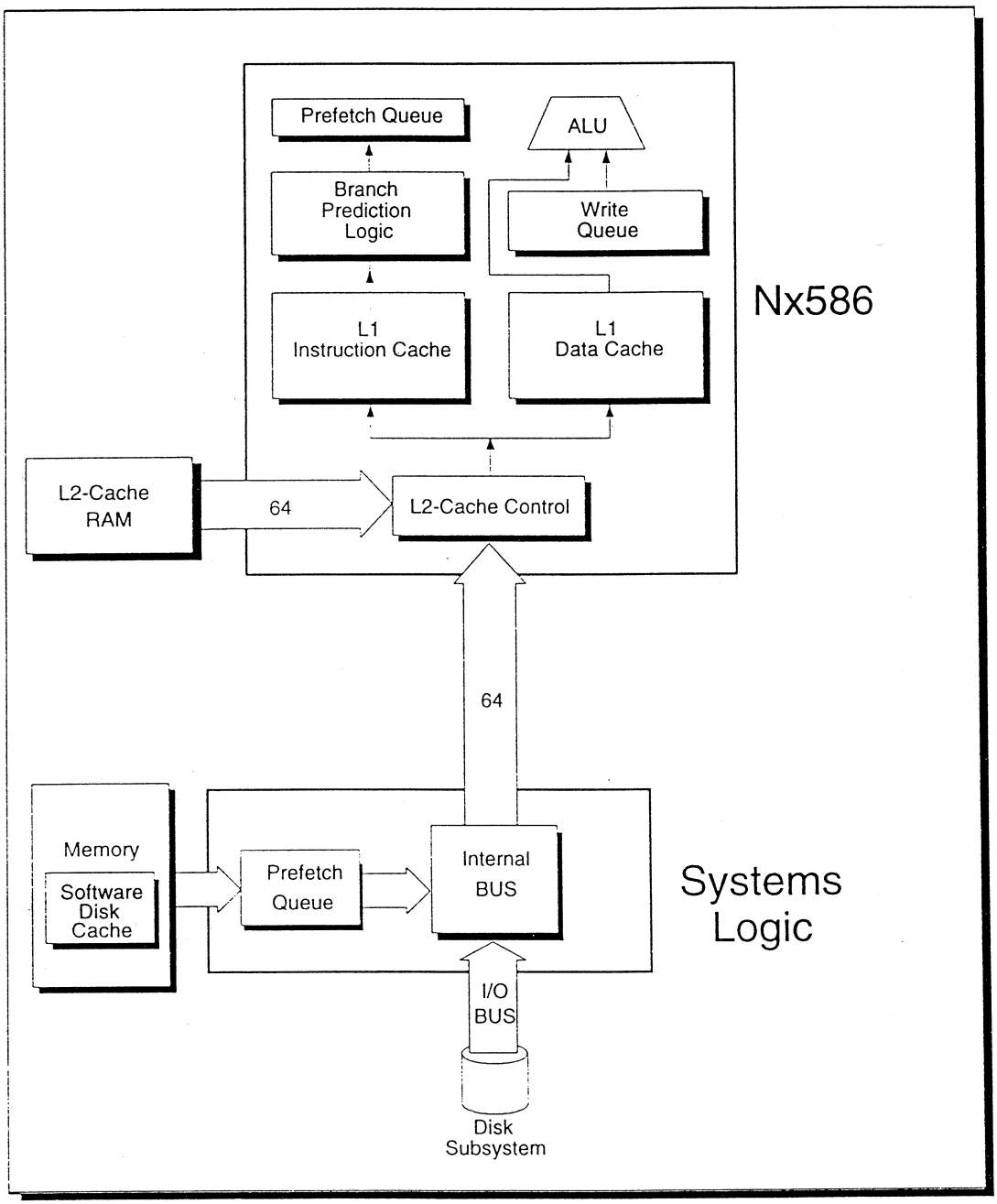

Figure 20 Storage Hierarchy (Reads)

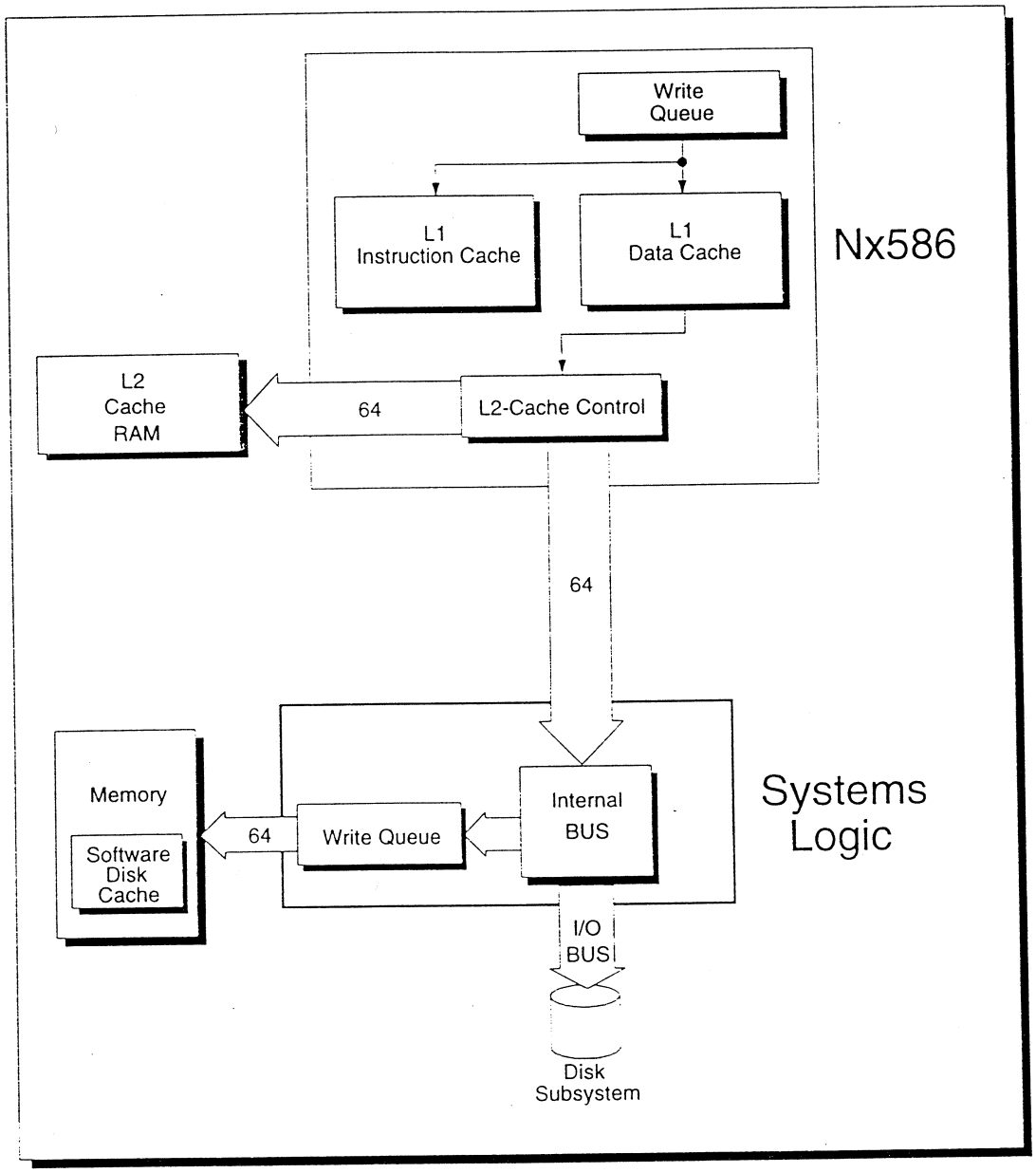

Figure 21 Storage Hierarchy (Writes)

Transaction Ordering

Interlocks enforce transaction ordering in a manner that optimizes read accesses. Interlocks are conditions where the execution of one function is deferred until the conflicting function has completed execution. With the exceptions detailed below, the *general rules* for transaction ordering are:

- *Memory Reads*—Memory reads (whether cache hits or reads on NexBus) are re-ordered ahead of writes, are performed out of order with respect to other reads, and are done speculatively. 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 and Memory-Mapped 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.
- *Locked Cycles*—Locked read-modify-writes are stalled until the write queue is emptied.
- *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 or 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 NexBus (cache misses) if any of the following conditions exist:
  - Reads are I/O or Memory-Mapped I/O.
  - 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.

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 16kB in size and dual-ported. The L2 cache is either 256kB or 1MB and single-ported. It can be built from an array of eight asynchronous (using 8-bit devices) or four synchronous (using 16/18-bit devices), or two synchronous (using 32/36-bit devices) SRAMs. 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 external 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 the L2 cache controller coupled very closely 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. 32-bytes is the unit of transfer between memory and cache.

	L1 Cache		*L2 Cache*
Contents	Instructions (I Cache)	Data (D Cache)	Instructions and Data (Unified Cache)
Location	processor	processor	on-chip controller; 64-bit SRAM bus
Cache Size	16kB	16kB	256kB or 1MB
Ports	2	2	1
Clock Frequency, Relative to Processor Clock	2x	2x	0.5x

Figure 22 Cache Characteristics

If a write needs to go to NexBus for cache-coherency purposes, it does so before it goes to a cache. Whether the write is needed on 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[5] 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 to the processor L2 cache and L1 caches. Reads coming in from the L2 cache are routed first to the processor, then to the L1 caches. Write-backs go only 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 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. Both L1 caches may 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.

Level-2 Cache Power-on RESET Configurations

The Nx586 supports two types of Level-2 cache SRAMs, asynchronous and synchronous. Asynchronous SRAMs should be implemented for low speed cost effective system designs. On the other hand, synchronous SRAMs need to be used for systems operating the Nx586 at very high speeds (typically, 100MHz or higher). A group of pins are examined at power-on RESET to determine what mode the L2 cache controller is operating. The key pin is SRAMMODE. If SRAMMODE is pulled high, the on-chip L2 cache controller is configured for synchronous SRAMs. In synchronous SRAM mode, the CKMODE pin generates the SRAM clocks and COEB* generates global L2 SRAM write enables after RESET is inactive. While in synchronous SRAM mode, SCLKE is used to determine the output of CKMODE. If SCLKE is asserted, CKMODE generates a clock equal to the processor's internal frequency (double NxCLK). Due to loading and the loss of COEB* in synchronous mode, the type of synchronous SRAMs necessary are wide I/O or 32/36-bit. The basic access style for the synchronous SRAMs is "Flow Through". While SCLKE is inactive, CKMODE is driven low. When SRAMMODE is left unconnected (floating), the internal pull down resistor configures the Nx586 for asynchronous SRAMs. Figure 23 is a shows how to configure the Nx586 for the particular L2 Cache SRAM mode desired.

SRAM Mode Type	SRAMMODE	RESET*	SCLKE	CKMODE
—	0	1	X	PLL Mode Select
Asynchronous	0	0	X	floating
Synchronous	1	0	0	0 (low)
Synchronous	1	0	1	2x NxCLK

Figure 23 L2 Cache SRAM Interface Modes

Cache Coherency

The processor monitors (snoops) NexBus5 operations by bus masters to guarantee coherency with data cached in the processor's L2 cache, L1 caches, and branch prediction logic. 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 or invalid state) but that has subsequently been written to. Also called *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 NexBus5 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 NexBus5 cache-control bits.
- *Caches data* in a specific state by watching NexBus5 cache-control input signals from system logic and the slave being accessed.
- *Snoops* NexBus5 to detect other NexBus5 transactions that hit in the processor's caches.
- *Intervenes* in the operations of other NexBus5 devices 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 an update or cache replacement 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 24 shows only the basic state transitions for write-back 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 cache write.

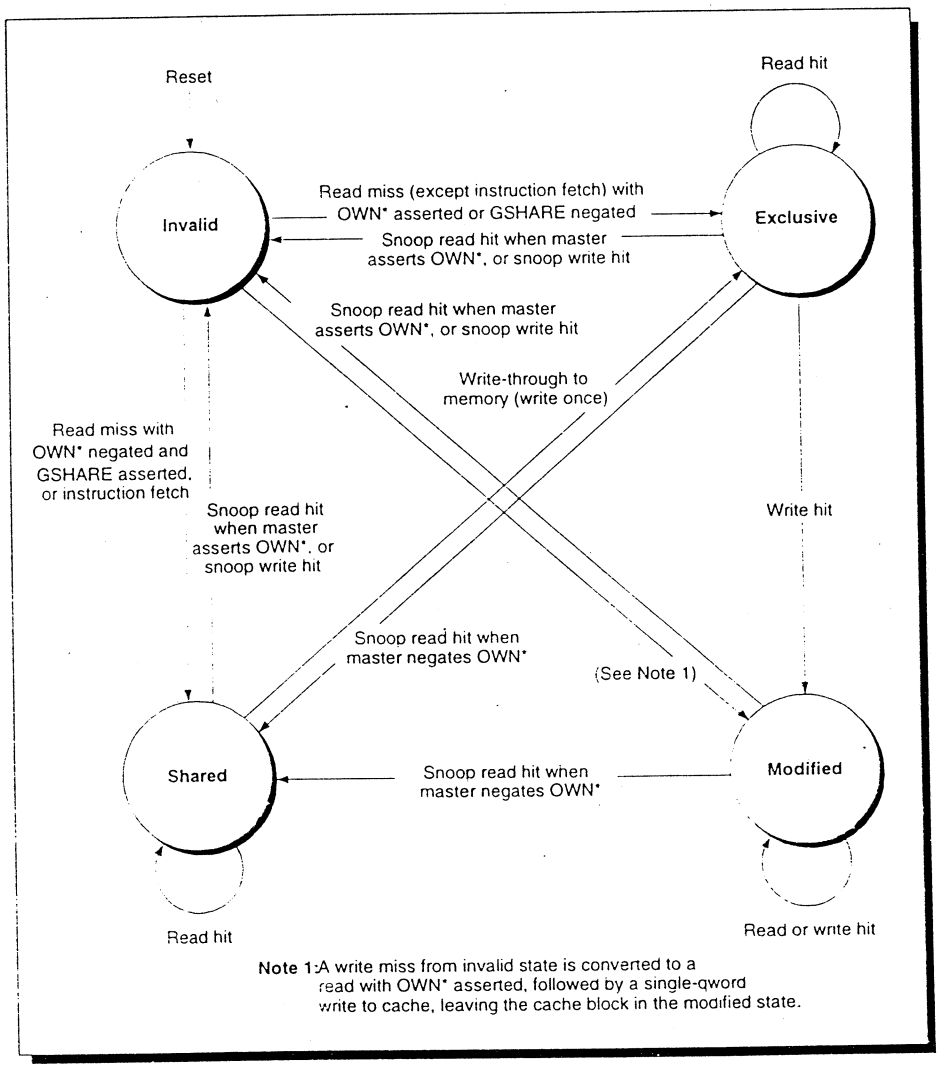

Figure 24 Basic Cache-State Transitions

Figure 25 describes the primary signals and status bits that affect the state transitions shown in Figure 24. The OWN* and SHARE* signals control many transitions. The assertion of OWN* implies that the data is both snoopable (SNPNBL) and cacheable (CACHBL). Figure 26 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 for write-backs and reads from the stack. If such an operation hits in the cache of another master, that master writes its data back (if copy is modified) and changes the state of its copy to *invalid*. If OWN* is negated during a read or write, another master may not assume that the copy is in *shared* state when not asserting SHARE* 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 NexBus5 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 systems using logic that does not have an OWNABL signal, 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 NexBus5 master during block reads by another NexBus5 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 may be promoted to the *exclusive* state because no other NexBus5 device declared via SHARE* that it has cached a copy. Code fetches will stay in the *shared* state.

Figure 25 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. This signal is negated by the processor on write-backs.
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[5]. The assertion of DCL* is the only way in which atomic 32-byte cache-block fills by another NexBus[5] 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[5] 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 26 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 may 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.

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 a write by another master hits in an exclusive block, the processor invalidates the block. 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 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* or invalidates the block during the operation. For external block writes (unlike the single-qword writes described above), the processor does not perform an intervenor operation with a write-back because the other master overwrites the entire cache block(s). If an external block write hits a modified processor block it invalidates the block.

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 NxCLK 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.

Systems logic using the 82C206 integrated peripheral controller (IPC) or equivalent, use the IPC to handle interrupts. The systems logic typically 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 a INTA* output.

For Nx586s with the optional floating-point execution unit, the Nx586 generates unmasked floating point error interrupts on the NPIRQ* pin. The NPIRQ* function is included for PC-AT compatibility. This pin is typically inverted and then connected to IRQ13. Floating-point errors are cleared in the same manner as in a compatible PC-AT. However, the Nx586 detects and traps the I/O writes which normally clears the error and performs the clearing internally. Therefore, the supporting AT compatible chipset does not require a dedicated signal to the processor to clear floating-point errors.

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, NxCLK, PHE1, and PHE2. These signals determine one of four modes: Phase-Locked Loop (the normal operating mode), External Phase Inputs, Reserved, or External Processor Clock, as shown in Figure 27 and described in the sections below. Note, each clocking mode is determined at power-on RESET*. PHE2 determines the relationship between the internal non-overlapping clocks. When pulled low, narrow non-overlapped clocks are generated. Wide non-overlapped clocks are produced for PHE2 pulled high.

Mode Type	*Mode #*	*RESET**	*CKMODE*	*XSEL*	*PHE1*
Phase-Locked Loop (normal operating mode)	0	↑	0	0	0
External Processor Clock	1	↑	0	1	Input at 2x the NxCLK frequency
Reserved Mode	2	↑	1	0	
External Phase Inputs	3	↑	1	1	Externally supplied at 2x the NxCLK frequency

Figure 27 Clocking Modes

Mode #0: In the *phase-locked loop* mode, the internal clock phases are derived from the external NxCLK clock via a phase-locked loop (PLL). In all modes, the NxCLK 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.

Mode #1: In the *external processor clock* mode, the internal clock phases are derived from PHE1 input signal. The PHE1 input signal operates at twice the frequency of NxCLK. The falling edge of the internal phase2 will occur before the rising edge of XREF, which is a buffered NxCLK output, and can be observed on the XPH2 output. This mode allows bypassing the internal PLL for test purposes or to change the clock frequency, as when entering or leaving a low-power mode.

Mode #2: This is a reserved mode.

Mode #3: 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 internally to drive the processor clock distribution system.