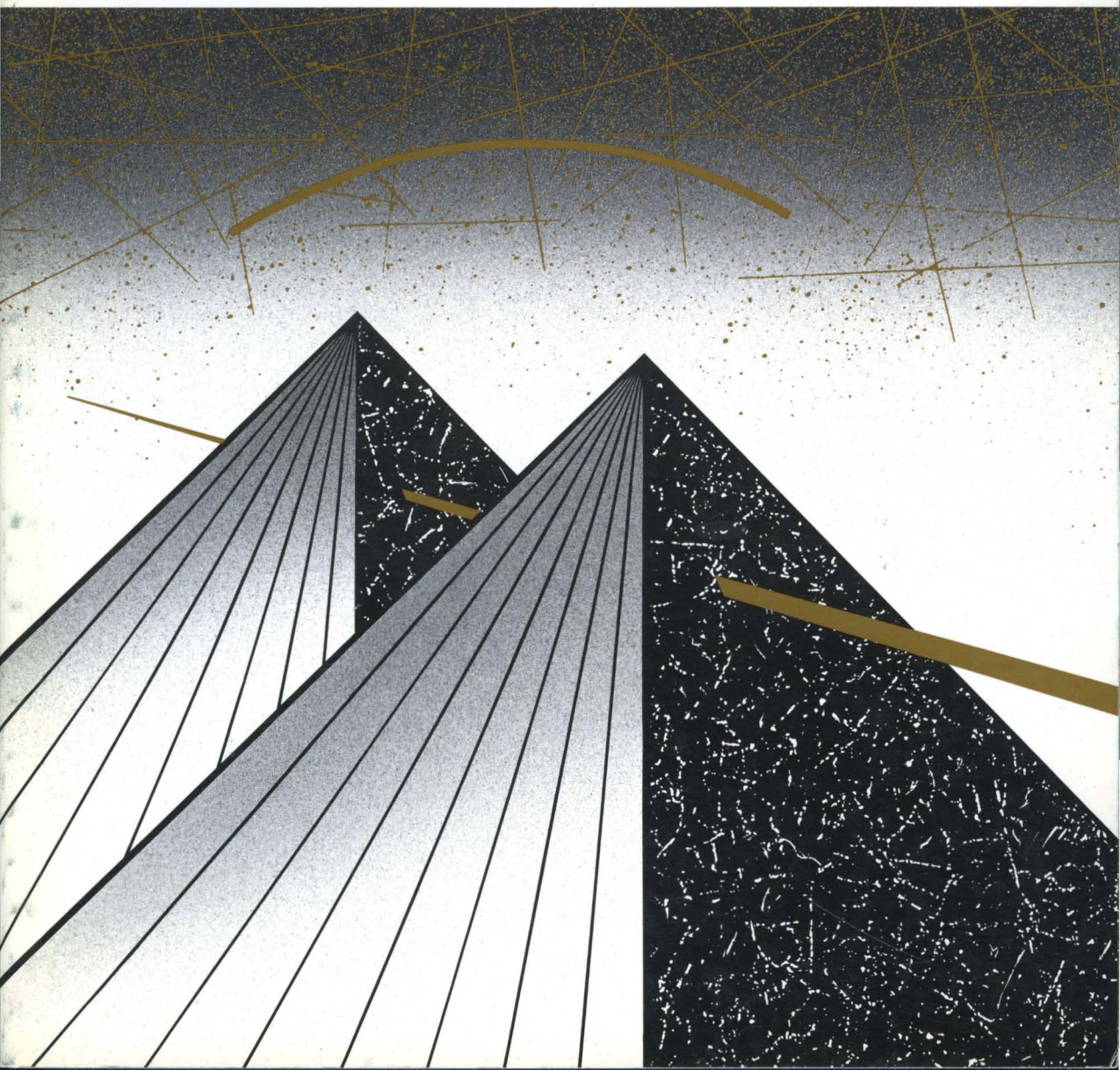
intel

i860™ XP MICROPROCESSOR DATA BOOK



PRELIMINARY

i860™ XP MICROPROCESSOR

- **Parallel Architecture that Supports Up to Three Operations per Clock**
  - **One Integer or Control Instruction**
  - **Up to Two Floating-Point Results**
- **High Performance Design**
  - **40/50 MHz Clock Rate**
  - **100 Peak Single Precision MFLOPS**
  - **75 Peak Double Precision MFLOPS**
  - **64-Bit External Data Bus**
  - **64-Bit Internal Code Bus**
  - **128-Bit Internal Data Bus**
- **High Integration on One Chip**
  - **32-Bit Integer and Control Unit**
  - **32/64-Bit Pipelined Floating-Point**
  - **64-Bit 3-D Graphics Unit**
  - **Paging Unit with 64 Four-Kbyte and 16 Four-Mbyte Pages**
  - **16 Kbyte Code Cache**
  - **16 Kbyte Data Cache**
- **Fast, Multiprocessor-Oriented Bus**
  - **Burst Cycles Move 400 Mbyte/Sec**
  - **Hardware Cache Snooping**
  - **MESI Cache Consistency Protocol**
  - **Supports Second-Level Cache**
  - **Supports DRAM**
- **Compatible with Industry Standards**
  - **ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
  - **Intel 386™/Intel 486™/i860™ Data Formats and Page Table Entries**
  - **Binary Compatible with i860™ XR Applications Instruction Set**
  - **Detached Concurrency Control Unit (CCU) Supports Parallel Architecture Extensions (PAX)**
  - **JEDEC 262-pin Ceramic Pin Grid Array Package**
  - **IEEE Standard 1149.1/D6 Boundary-Scan Architecture**
- **Easy to Use**
  - **On-Chip Debug Register**
  - **UNIX*/860**
  - **APX Attached Processor Executive**
  - **Assembler, Linker, Simulator, Debugger, C and FORTRAN Compilers, FORTRAN Vectorizer, Scalar and Vector Math Libraries**
  - **Graphics Libraries**

The Intel i860 XP Microprocessor (order code A80860XP) delivers supercomputing performance in a single VLSI component. The 32/64-bit architecture of the i860 XP microprocessor balances integer, floating point, and graphics performance for applications such as engineering workstations, scientific computing, 3-D graphics workstations, and multiuser systems. Its parallel architecture achieves high throughput with RISC design techniques, multiprocessor support, pipelined processing units, wide data paths, large on-chip caches, 2.5 million transistor design, and fast 0.8-micron silicon technology.

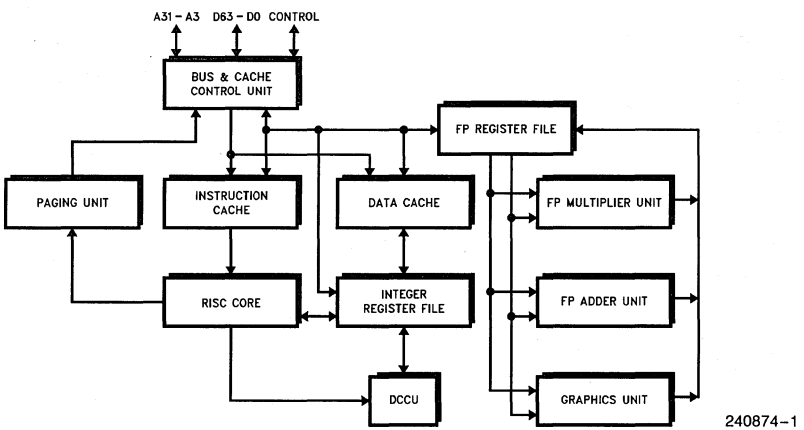


Figure 0.1. Block Diagram

*UNIX is a registered trademark of UNIX System Laboratories, Inc.
Intel, i860, Intel386 and Intel486 are trademarks of Intel Corporation.

Order Number: 240874-001

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

PAGE

FIGURES

CONTENTS

PAGE

CONTENTS

PAGE

CONTENTS

PAGE

CONTENTS

PAGE

TABLES

CONTENTS

PAGE

1.0 FUNCTIONAL DESCRIPTION

As shown by the block diagram on the front page, the i860 XP Microprocessor consists of the following units:

1. Integer Registers and Core Execution Unit
2. Floating-Point Registers and Control Unit
3. Floating-Point Adder Unit
4. Floating-Point Multiplier Unit
5. Graphics Unit
6. Paging Unit
7. Instruction Cache
8. Data Cache
9. Bus and Cache Control Unit
10. Detached Concurrency Control Unit

The core execution unit controls overall operation of the i860 XP microprocessor. It executes load, store, integer, bit, I/O, and control-transfer operations, and fetches instructions for the floating-point unit as well. A set of 32 × 32-bit general-purpose registers are provided for the manipulation of integer data. Load and store instructions move 8-, 16-, and 32-bit data to and from these registers. Its full set of integer, logical, and control-transfer instructions give the core unit the ability to execute complete systems software and applications programs. A trap mechanism provides rapid response to exceptions and external interrupts. Debugging is supported by the ability to trap on data or instruction reference.

The floating-point hardware is connected to a separate set of floating-point registers, which can be accessed as 16 × 64-bit registers or as 32 × 32-bit registers. Load and store instructions can also access these same registers as 8 × 128-bit registers. All floating-point and graphics instructions use these registers as their source and destination operands.

The floating-point control unit controls both the floating-point adder and the floating-point multiplier, issuing instructions, handling all source and result exceptions, and updating status bits in the floating-point status register. The adder and multiplier can operate in parallel, producing up to two results per clock. The floating-point data types, floating-point instructions, and exception handling all support the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985).

The floating-point adder performs addition, subtraction, comparison, and conversions on 64- and 32-bit floating-point values. An adder instruction executes in three clocks; however, in pipelined mode, a new result is generated every clock.

The floating-point multiplier performs floating-point and integer multiply as well as floating-point reciprocal operations on 64- and 32-bit floating-point values. A multiplier instruction executes in three to four clocks; however, in pipelined mode, a new result can be generated every clock for single-precision and every other clock for double precision.

The graphics unit supports three-dimensional drawing in a graphics frame buffer, with color intensity shading and hidden surface elimination via the Z-buffer algorithm. The graphics unit recognizes the pixel as an 8-, 16-, or 32-bit integer data type. It can compute individual red, blue, and green color intensity values within a pixel; but it does so with parallel operations that take advantage of the 64-bit internal word size and 64-bit external bus. The graphics features of the i860 XP microprocessor assume that the surface of a solid object is drawn with polygon patches which, like the pieces of a puzzle, collectively approximate the shape of the original object. The color intensities of the vertices of the polygon and their distances from the viewer are known, but the distances and intensities of the other points must be calculated by interpolation. The graphics instructions of the i860 XP microprocessor directly aid such interpolation.

The paging unit implements protected, paged, virtual memory. The paging unit uses two four-way set-associative cache memories called TLBs (Translation Lookaside Buffers) to perform the translation of logical address to physical address, and to check for access violations. The access protection scheme employs two levels of privilege: user and supervisor. One TLB supports 4 Kbyte pages, and has 64 entries; the other supports 4 Mbyte pages, and has 16 entries.

The instruction cache is a four-way set-associative memory of 16 Kbytes, with 32-byte lines. It transfers up to 64 bits per clock (400 Mbyte/sec at 50 MHz).

The data cache is a four-way set-associative memory of 16 Kbytes, with 32-byte lines. It transfers up to 128 bits per clock (800 Mbyte/sec at 50 MHz). The i860 XP microprocessor normally uses write-back caching, i.e. memory writes update the cache (if applicable) without necessarily updating memory immediately; however, under both software and hardware control, write-through and write-once policies can be implemented, or caching can be inhibited. The caches are transparent to applications software.

The bus and cache control unit performs data and instruction accesses for the core unit. It receives cycle requests and specifications from the core unit, performs the data-cache or instruction-cache miss processing, controls TLB translation, and provides

the interface to the external bus. Its pipelined structure supports up to three outstanding bus cycles. Its burst mode transfers data at up to 400 Mbyte/sec at 50 MHz. In multiprocessor systems, it maintains cache consistency by monitoring bus activity in parallel with other CPU functions.

The DCCU (detached concurrency control unit) is a compatible subset of the external CCU that expedites loop-level parallelism and synchronization in multiprocessor systems. The DCCU consists of registers and a counter that allow a single i860 XP microprocessor to run binary code compiled for a multiprocessor system adhering to the PAX parallel applications binary interface (ABI).

The i860 XP microprocessor may to be used with or without an external, secondary cache built from 82495XP and 82490XP cache components. An 82495XP and 82490XP cache provides up to 512 Kbytes of high-speed storage for data and instruction combined. In most cases, an 82495XP and 82490XP cache can provide data to the CPU with zero wait states. The larger size of an external cache can provide an increased hit rate when the size or number of data structures and programs exceeds the size of the internal caches. In multiprocessor systems, the external cache serves as local memory, and can reduce bus traffic. An external cache also hides the processor from rest of system, which is a double advantage:

1. The processor can be upgraded without affecting design of the memory and other subsystems.
2. Slower and less expensive memory and I/O subsystem designs can be employed without unduly lowering overall system performance.

Refer to the *82495XP Cache Controller/82490XP Cache RAM Data Sheet* (Intel Order #240956) for more information.

2.0 PROGRAMMING INTERFACE

The programmer-visible aspects of the architecture of the i860 XP microprocessor include data types, registers, instructions, and traps.

2.1 Data Types

The i860 XP microprocessor provides operations for integer and floating-point data. Integer operations are performed on 32-bit operands with some support also for 64-bit operands. Load and store instructions can reference 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit operands. Floating-point operations are performed on IEEE-standard 32- and 64-bit formats. Graphics instructions operate on arrays of 8-, 16-, or 32-bit pixels.

2.1.1 INTEGER

An integer is a 32-bit signed value in standard two's complement form. A 32-bit integer can represent a value in the range $-2{,}147{,}483{,}648$ (-2^{31}) to $2{,}147{,}483{,}647$ ($+2^{31} - 1$). Arithmetic operations on 8- and 16-bit integers can be performed by sign-extending the 8- or 16-bit values to 32 bits, then using the 32-bit operations.

There are also add and subtract instructions that operate on 64-bit long integers.

Load and store instructions may also reference (in addition to the 32- and 64-bit formats previously mentioned) 8- and 16-bit items in memory. When an 8- or 16-bit item is loaded into a register, it is converted to an integer by sign-extending the value to 32 bits. When an 8- or 16-bit item is stored from a register, the corresponding number of low-order bits of the register are used.

2.1.2 ORDINAL

Arithmetic operations are available for 32-bit ordinals. An ordinal is an unsigned integer. An ordinal can represent values in the range 0 to $4{,}294{,}967{,}295$ ($+2^{32} - 1$).

Also, there are add and subtract instructions that operate on 64-bit ordinals.

2.1.3 SINGLE- AND DOUBLE-PRECISION REAL

Figure 2.1 shows the real number formats. A single-precision real (also called "single real") data type is a 32-bit binary floating-point number. Bit 31 is the sign bit; bits 30..23 are the exponent; and bits 22..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a single-precision real is defined as follows:

1. If $\mathbf{e} = 0$ and $\mathbf{f} \neq 0$ or $\mathbf{e} = 255$ then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If $0 < \mathbf{e} \leq 255$, then the value is $(-1)^{s} \times 1.\mathbf{f} \times 2^{e-127}$.
3. If $\mathbf{e} = 0$ and $\mathbf{f} = 0$, then the value is signed zero.

A double-precision real (also called "double real") data type is a 64-bit binary floating-point number. Bit 63 is the sign bit; bits 62..52 are the exponent; and bits 51..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a double-precision real is defined as follows:

1. If $\mathbf{e} = 0$ and $\mathbf{f} \neq 0$ or $\mathbf{e} = 2047$, then generate a floating-point source-exception trap when encountered in a floating-point operation.

2. If $0 < \mathbf{e} < 2047$, then the value is $(-1)^{s} \times 1.\mathbf{f} \times 2^{e-1023}$.
3. If $\mathbf{e} = 0$ and $\mathbf{f} = 0$, then the value is signed zero.

The special values infinity, NaN ("Not a Number"), indefinite, and denormal generate a trap when encountered. The trap handler implements IEEE-standard results.

A double real value occupies an even/odd pair of floating-point registers. Bits 31..0 are stored in the even-numbered floating-point register; bits 63..32 are stored in the next higher odd-numbered floating-point register.

2.1.4 PIXEL

A pixel may be 8-, 16-, or 32-bits long, depending on color and intensity resolution requirements. Regardless of the pixel size, the i860 XP microprocessor always operates on 64 bits of pixel data at a time. The pixel data type is used by two kinds of instructions:

- The selective pixel-store instruction that helps implement hidden surface elimination.
- The pixel add instruction that helps implement 3-D color intensity shading.

To perform color intensity shading efficiently in a variety of applications, the i860 XP microprocessor defines three pixel formats according to Table 2.1.

Figure 2.2 illustrates one way of assigning meaning to the fields of pixels. These assignments are for illustration purposes only. The i860 XP microprocessor defines only the field sizes, not the specific use of each field. Other ways of using the fields of pixels are possible.

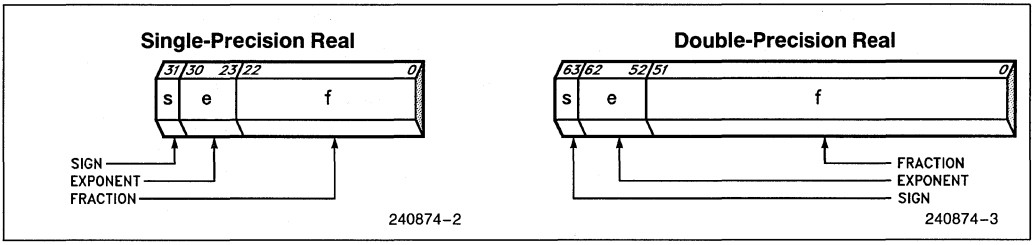
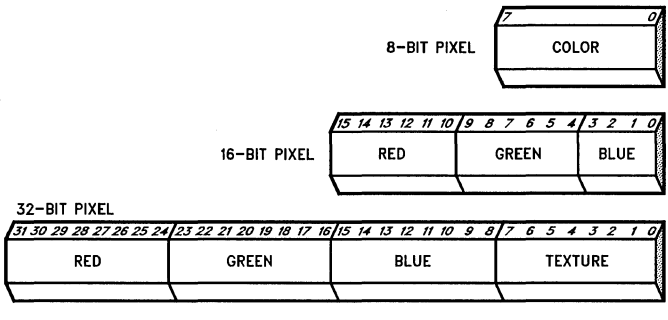


Figure 2.1. Real Number Formats

Table 2.1. Pixel Formats

Pixel Size (in bits)	Bits of Color 1 Intensity(1)	Bits of Color 2 Intensity(1)	Bits of Color 3 Intensity(1)	Bits of Other Attribute (Texture, Color)
8	N (≤8) bits of intensity(2)			8–N
16M	6	6	4	0
32	8	8	8	8

NOTES:
1. The intensity attribute fields may be assigned to colors in any order convenient to the application.
2. With 8-bit pixels, up to 8 bits can be used for intensity; the remaining bits can be used for any other attribute, such as color or texture. Bits that require interpolation (shading), such as those for intensity, must be the low-order bits of the pixel.



NOTE:
These assignments of specific meanings to the fields of pixels are for illustration only. Only the field sizes are defined, not the specific use of each field.

Figure 2.2. Pixel Format Example

2.2 Register Set

As Figure 2.3 shows, the i860 XP microprocessor has the following registers:

- An integer register file
- A floating-point register file
- Control registers **psr, epsr, db, dirbase, fir, fsr, bear, ccr, p3, p2, p1, p0**
- Special-purpose registers KR, KI, T, MERGE, STAT, and NEWCURR

The control registers are accessible only by load and store control-register instructions; the integer and floating-point registers are accessed by arithmetic operations and load and store instructions. The special-purpose registers KR, KI, and T are used by floating-point instructions; MERGE is used by graphics instructions. NEWCURR and STAT are used for concurrency control; they are accessed by memory load and store instructions.

2.2.1 INTEGER REGISTER FILE

There are 32 integer registers, each 32 bits wide, referred to as **r0** through **r31**, which are used for address computation and scalar integer computations. Register **r0** always returns zero when read.

2.2.2 FLOATING-POINT REGISTER FILE

There are 32 floating-point registers, each 32-bits wide, referred to as **f0** through **f31**, which are used for floating-point computations. Registers **f0** and **f1** always return zero when read. The floating-point registers are also used by a set of integer operations, primarily for graphics computations.

When accessing 64-bit floating-point or integer values, the i860 XP microprocessor uses an even/odd pair of registers. When accessing 128-bit values, it uses an aligned set of four registers (**f0, f4, f8, f12, f16, f20, f24**, or **f28**). The instruction must designate the lowest register number of the set of registers containing 64- or 128-bit values. Misaligned register numbers produce undefined results. The register with the lowest number contains the least significant part of the value. For 128-bit values, the register pair with the lower number contains the value from the lower memory address; the register pair with the higher number contains the value from the higher address.

The 128-bit load and store instructions, along with the 128-bit data path between the floating-point registers and the data cache, help to sustain an extraordinarily high rate of computation.

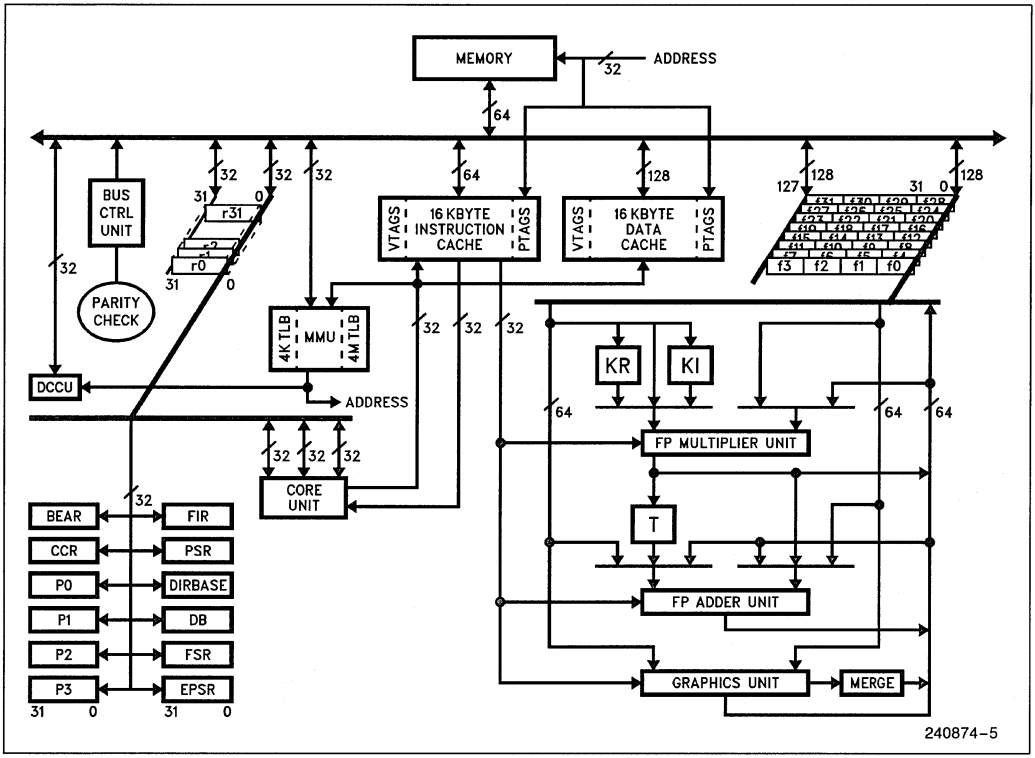


Figure 2.3. Registers and Data Paths

2.2.3 PROCESSOR STATUS REGISTER

The processor status register (**psr**) contains miscellaneous state information for the current process. Figure 2.4 shows the format of the **psr**.

- BR (Break Read) and BW (Break Write) enable a data access trap when the operand address matches the address in the **db** register and a read or write (respectively) occurs.
- Various instructions set CC (Condition Code) according to tests they perform. The branch-on-condition-code instructions test its value. The **bla** instruction sets and tests LCC (Loop Condition Code).
- IM (Interrupt Mode), if set, enables external interrupts on the INT pin; disables interrupts on INT if clear. IM does not affect parity error interrupts or interrupts on the BERR pin.
- U (User Mode) is set when the i860 XP microprocessor is executing in user mode; it is clear when the i860 XP microprocessor is executing in supervisor mode. In user mode, writes to some control registers are inhibited. This bit also controls the memory protection mechanism.
- PIM (Previous Interrupt Mode) and PU (Previous User Mode) save the corresponding status bits (IM and U) on a trap, because those status bits are changed when a trap occurs. They are restored into their corresponding status bits when returning from a trap handler with a branch indirect instruction when a trap flag is set in the **psr**.
- FT (Floating-Point Trap), DAT (Data Access Trap), IAT (Instruction Access Trap), IN (Interrupt), and IT (Instruction Trap) are trap flags. They are set when the corresponding trap condition occurs. The trap handler examines these bits (and other trap bits in the **epsr)** to determine which condition or conditions have caused the trap.
- DS (Delayed Switch) is set if a trap occurs during the instruction before dual-instruction mode is entered or exited. If DS is set and DIM (Dual Instruction Mode) is clear, the i860 XP microprocessor switches to dual-instruction mode one instruction after returning from the trap handler. If DS and DIM are both set, the i860 XP microprocessor switches to single-instruction mode one instruction after returning from the trap handler.

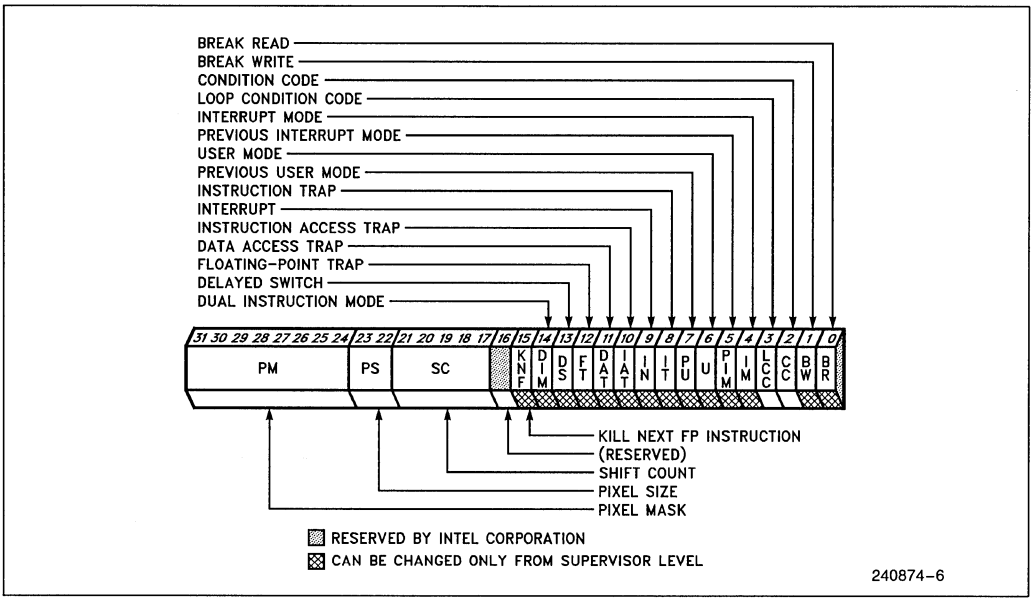


Figure 2.4. Processor Status Register

- When a trap occurs, the i860 XP microprocessor sets DIM if it is executing in dual-instruction mode; it clears DIM if it is executing in single-instruction mode. If DIM is set after returning from a trap handler, the i860 XP microprocessor resumes execution in dual-instruction mode.
- When KNF (Kill Next Floating-Point Instruction) is set, the next floating-point instruction is suppressed (except that its dual-instruction mode bit is interpreted). A trap handler sets KNF if the trapped floating-point instruction should not be reexecuted.
- SC (Shift Count) stores the shift count used by the last right-shift instruction. It controls the number of shifts executed by the double-shift instruction.
- PS (Pixel Size) and PM (Pixel Mask) are used by the pixel-store and other graphics instructions. The values of PS control pixel size as defined by Table 2.2. The bits in PM correspond to pixels to be updated by the pixel-store instruction **pst.d**. The low-order bit of PM corresponds to the low-order pixel of the 64-bit source operand of **pst.d**. The number of low-order bits of PM that are actually used is the number of pixels that fit into 64-bits, which depends upon PS. If a bit of PM is set, then **pst.d** stores the corresponding pixel. Refer also to the **pst.d** instruction in section 10.

Table 2.2. Values of PS

Value	Pixel Size in Bits	Pixel Size in Bytes
00	8	1
01	16	2
10	32	4
11	(undefined)	(undefined)

2.2.4 EXTENDED PROCESSOR STATUS REGISTER

The extended processor status register (**epsr**) contains additional state information for the current process beyond that stored in the **psr**. Figure 2.5 shows the format of the **epsr**.

- The processor type is 2 for the i860 XP microprocessor.
- The stepping number has a unique value that distinguishes among different revisions of the processor.
- IL (Interlock) is set if a trap occurs after a **lock** instruction but before the last BRDY# of the load or store following the subsequent **unlock** instruction. IL indicates to the trap handler that a locked sequence has been interrupted. When the trap handler finds IL set, it should scan backwards for the **lock** instruction and restart at that point. The absence of a **lock** instruction within 30–33 instructions of the trap indicates a programming error.

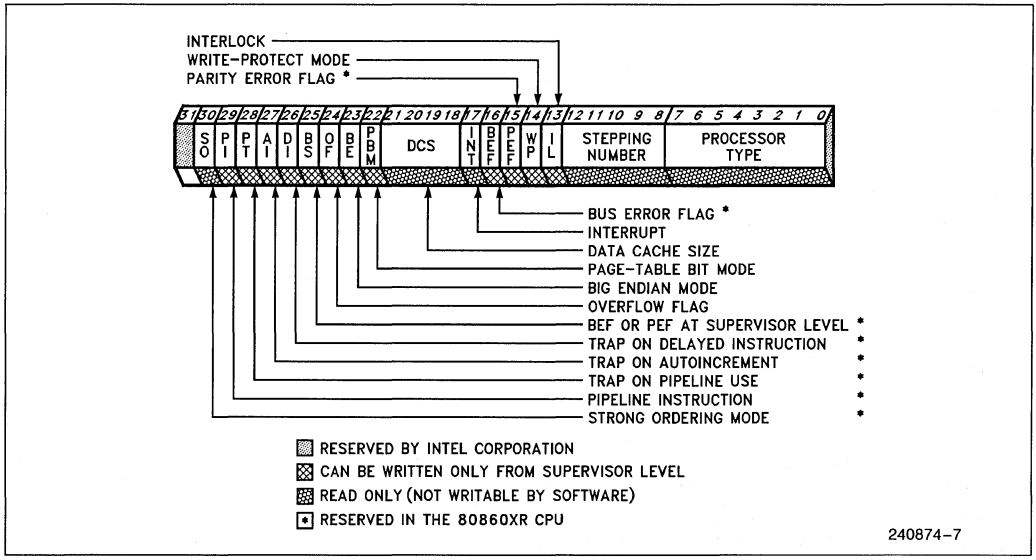


Figure 2.5. Extended Processor Status Register

- WP (write protect) controls the semantics of the W bit of page table entries. A clear W bit in either the directory or the page table entry causes writes to be trapped. When WP is clear, writes are trapped in user mode, but not in supervisor mode. When WP is set, writes are trapped in both user and supervisor modes.
- PEF (parity error flag) is set by the i860 XP microprocessor when a parity error trap occurs. As soon as PEF is set, further parity error and bus error traps are masked. Software must clear PEF to reenable such traps. PEF is set at RESET.
- BEF (bus error flag) is set by the i860 XP microprocessor when the BERR pin is asserted, indicating a bus error. As soon as BEF is set, further parity error and bus error traps are masked. Software must clear BEF to reenable such traps. BEF is set at RESET.
- INT (Interrupt) is the value of the INT input pin.
- DCS (Data Cache Size) is a read-only field that tells the size of the on-chip data cache. The number of bytes actually available is 2^{12} + DCS; therefore, a value of zero indicates 4 Kbytes, one indicates 8 Kbytes, etc. The value of DCS for the i860 XP microprocessor is two, which indicates 16 Kbytes.
- PBM (Page-Table Bit Mode) has no effect in the i860 XP microprocessor. PBM is used by the i860 XR microprocessor.
- BE (Big Endian) controls the ordering of bytes within a data item in memory. Normally (i.e. when BE is clear) the i860 XP microprocessor operates in little endian mode, in which the addressed byte is the low-order byte. When BE is set (big endian mode), the low-order three bits of all 32-bit data load and store addresses are complemented, then masked to the appropriate boundary for alignment. This causes the addressed byte to be the most significant byte. Big endian mode affects not only the memory load and store instructions but also the **ldio, stio, ldint,** and **scyc** instructions.
- OF (Overflow Flag) is set by **adds**, **addu**, **subs**, and **subu** when integer overflow occurs. For **adds** and **subs**, OF is set if the carry from bit 31 is different than the carry from bit 30. For **addu**, OF is set if there is a carry from bit 31. For **subu**, OF is set if there is no carry from bit 31. Under all other conditions, it is cleared by these instructions. OF may be changed by arithmetic instructions in either user or supervisor mode. It may be changed by the **st.c** instruction in supervisor mode only. OF controls the function of the **intovr** instruction. Inside the trap handler, OF may not be valid for traps other than one caused by **intovr**.
- BS (bus or parity error trap in supervisor mode) is set by the i860 XP microprocessor when a bus or parity error occurs while the processor is in supervisor mode. The operating system can use this bit to decide, for example, whether to abort the process (user mode) or reboot the system (supervisor mode).

- DI (trap on delayed instruction) is set by the i860 XP microprocessor when a trap occurs on a delayed instruction (the instruction located after a delayed branch instruction). When DI is set, the trap handler must restart the interrupted procedure from the branch instruction rather than at the address in **fir**.
- AI (trap on autoincrement instruction) is set by the i860 XP microprocessor when a trap occurs on an instruction with autoincrement (including the **bla** instruction). When AI is set, the trap handler should undo the autoincrement (that is, restore *src2* to its original value).
- PT (trap on pipeline use) indicates to the i860 XP microprocessor that a trap should be generated and PI should be set when it executes an instruction that uses the floating-point or graphics unit. Such instructions include all the instructions designated "Floating-Point Unit" in Table 2.9, plus the **pfld** instruction. PT is set and cleared only by software. It can be used by the trap handler to avoid unnecessary saving and restoring of the pipelines (refer to section 2.8). When a trap due to PT occurs, the floating-point operation has not started, and the pipelines have not been advanced. Such a trap also sets the IT bit of **psr**.
- The behavior of PI (pipeline instruction) depends on the setting of PT. If PT = 0, the i860 XP microprocessor sets PI when any pipelined instruction or **pfld** is executed. If PT = 1, the processor sets PI and traps when it decodes any instruction that uses the pipes, whether scalar or pipelined. Refer to section 2.8.
- SO (strong ordering) indicates whether the processor is in strong ordering mode (SO = 1) or weak ordering mode (SO = 0). SO is set if the EWBE# pin is active (LOW) at RESET. (Refer to the paragraphs on write cycle reordering in section 5.)

2.2.5 DATA BREAKPOINT REGISTER

The data breakpoint register (**db**) is used to generate a trap when the i860 XP microprocessor accesses an operand at the virtual address stored in this register. The trap is enabled by BR and BW in **psr**. When comparing, a number of low order bits of the address are ignored, depending on the size of the operand. For example, a 16-bit access ignores the low-order bit of the address when comparing to **db**; a 32-bit access ignores the low-order two bits. This ensures that any access that overlaps the address contained in the register will generate a trap. The trap occurs *before* the register or memory update by the load or store instruction.

2.2.6 DIRECTORY BASE REGISTER

The directory base register **dirbase** (shown in Figure 2.6) controls address translation, caching, and bus options.

- ATE (Address Translation Enable), when set, enables the virtual-address translation algorithm.
- DPS (DRAM Page Size) controls how many bits to ignore when comparing the current bus-cycle address with the previous bus-cycle address to generate the NENE# signal. This feature allows for higher speeds when using static column or page-mode DRAMs and consecutive reads and writes access the same column or page. The comparison ignores the low-order 12 + DPS bits. A value of zero is appropriate for one bank of 256K × *n* RAMs, 1 for 1M × *n* RAMS, etc. For interleaved memory, increase DPS by one for each power of interleaving—add one for 2-way, two for 4-way, etc.

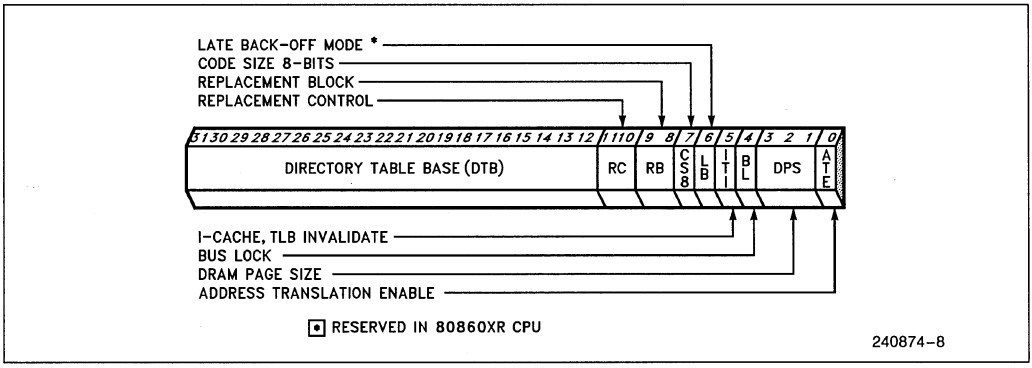


Figure 2.6. Directory Base Register

- When BL (Bus Lock) is set, external bus accesses are locked. The LOCK# signal is asserted with the next bus cycle (excluding instruction fetch and write-back cycles) whose internal bus request is generated after BL is set. It remains set on every subsequent bus cycle as long as BL remains set. The LOCK# signal is deasserted on the next load or store instruction after BL is cleared. Traps immediately clear BL. The **lock** and **unlock** instructions control the BL bit. The result of modifying BL with the **st.c** instruction is not defined.
- ITI (Cache and TLB Invalidate), when set in the value that is loaded into **dirbase**, causes all entries in the instruction cache and virtual tags in the address-translation cache (TLB) to be invalidated. Also invalidates all virtual tags in the data cache. The ITI bit does not remain set in **dirbase**. ITI always appears as zero when reading **dirbase**.
- When software sets the LB bit, the i860 XP microprocessor enters two-clock late back-off mode. This mode gives two additional clock periods of decision time to the external logic that may need to use the BOFF# signal to cancel a bus cycle or data transfer. If the processor enters one-clock late back-off mode during RESET via configuration pin strapping, the LB bit has no effect, and it is impossible to enter two-clock late back-off mode. Furthermore, software cannot exit two-clock late back-off mode once it is activated; the LB bit cannot be cleared except by resetting the processor.
- When CS8 (Code Size 8-Bit) is set, instruction cache misses are processed as 8-bit bus cycles. When this bit is clear, instruction cache misses are processed as 64-bit bus cycles. This bit can not be set by software; hardware sets this bit at initialization time. It can be cleared by software (one time only) to allow the system to execute out of 64-bit memory after bootstrapping from 8-bit EPROM. A nondelayed branch to code in 64-bit memory should directly follow the **st.c** (store control register) instruction that clears CS8, in order to make the transition from 8-bit to 64-bit memory occur at the correct time. The branch instruction must be aligned on a 64-bit boundary.
- RB (Replacement Block) identifies the cache line (block) to be replaced by cache replacement algorithms. RB conditions the cache flush instruction **flush**, which is discussed in Section 10. Table 2.3 explains the values of RB.
- RC (Replacement Control) controls cache replacement algorithms. Table 2.4 explains the significance of the values of RC.
- DTB (Directory Table Base) contains the high-order 20 bits of the physical address of the page directory when address translation is enabled (i.e. ATE = 1). The low-order 12 bits of the address are zeros.

Table 2.3. Values of RB

Value	Replace TLB Block	Replace Instruction and Data Cache Block
0 0	0	0
0 1	1	1
1 0	2	2
1 1	3	3

Table 2.4. Values of RC

Value	Meaning
00	Selects the normal (random) replacement algorithm where any block in the set may be replaced on cache misses in all caches.
01	Instruction, data, and TLB cache misses replace the block selected by RB. This mode is used for cache and TLB testing.
10	Data cache misses replace the block selected by RB. Instruction and TLB caches use random replacement. This mode is used when flushing the data cache with the **flush** instruction.
11	Disables data cache replacement. Instruction and TLB caches use random replacement.

2.2.7 FAULT INSTRUCTION REGISTER

When a trap occurs, this register contains the address of the trapping instruction (not necessarily the instruction that created the conditions that required the trap). The **fir** is a read-only register. In single-instruction mode, using a **ld.c** instruction to read the **fir** anytime except the first time after a trap saves in *idest* the address of the **ld.c** instruction; in dual-instruction mode, the address of its floating-point companion (address of the **ld.c** − 4) is saved.

2.2.8 FLOATING-POINT STATUS REGISTER

The floating-point status register (**fsr**) contains the floating-point trap and rounding-mode status for the current process. Figure 2.7 shows its format.

- If FZ (Flush Zero) is clear and underflow occurs, a result-exception trap is generated. When FZ is set and underflow occurs, the result is set to zero, and no trap due to underflow occurs.
- If TI (Trap Inexact) is clear, inexact results do not cause a trap. If TI is set, inexact results cause a trap. The sticky inexact flag (SI) is set whenever an inexact result is produced, regardless of the setting of TI.
- RM (Rounding Mode) specifies one of the four rounding modes defined by the IEEE standard. Given a true result *b* that cannot be represented by the target data type, the i860 XP microprocessor determines the two representable numbers *a* and *c* that most closely bracket *b* in value ($a < b < c$). The i860 XP microprocessor then rounds (changes) *b* to *a* or *c* according to the mode selected by RM as defined in Table 2.5. Rounding introduces an error in the result that is less than one least-significant bit.

Table 2.5. Values of RM

Value	Rounding Mode	Rounding Action
00	Round to nearest or even	Closer to *b* of *a* or *c*; if equally close, select even number (the one whose least significant bit is zero).
01	Round down (toward $-\infty$)	*a*
10	Round up (toward $+\infty$)	*c*
11	Chop (toward zero)	Smaller in magnitude of *a* or *c*.

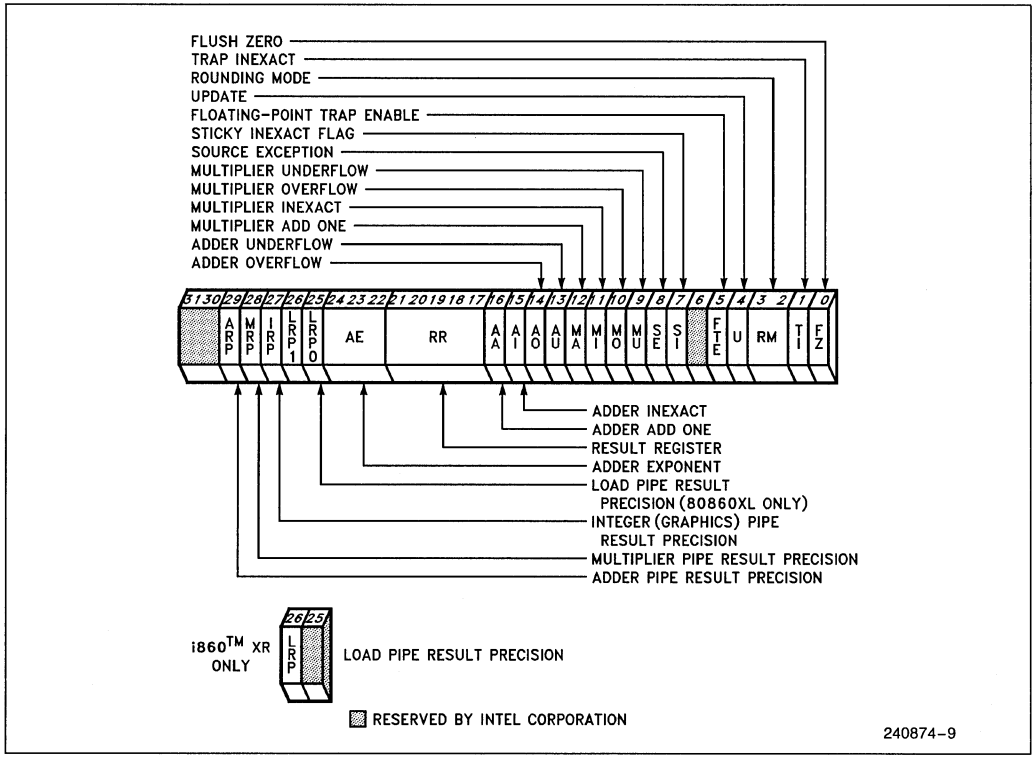


Figure 2.7. Floating-Point Status Register

- The U-bit (Update Bit), if set in the value that is loaded into **fsr** by a **st.c** instruction, enables updating of the result-status bits (AE, AA, AI, AO, AU, MA, MI, MO, and MU) in the first-stage of the floating-point adder and multiplier pipelines. If this bit is clear, the result-status bits are unaffected by a **st.c** instruction; **st.c** ignores the corresponding bits in the value that is being loaded. An **st.c** always updates **fsr** bits 21..17 and 8..0 directly. The U-bit does not remain set; it always appears as zero when read.
- The FTE (Floating-Point Trap Enable) bit, if clear, disables all floating-point traps (invalid input operand, overflow, underflow, and inexact result).
- SI (Sticky Inexact) is set when the last-stage result of either the multiplier or adder is inexact (i.e. when either AI or MI is set). SI is "sticky" in the sense that it remains set until reset by software. AI and MI, on the other hand, can by changed by the subsequent floating-point instruction.
- SE (Source Exception) is set when one of the source operands of a floating-point operation is invalid; it is cleared when all the input operands are valid. Invalid input operands include denormals, infinities, and all NaNs (both quiet and signaling).
- When read from the **fsr**, the result-status bits MA, MI, MO, and MU (Multiplier Add-One, Inexact, Overflow, and Underflow, respectively) describe the last-stage result of the multiplier.

  When read from the **fsr**, the result-status bits AA, AI, AO, AU, and AE (Adder Add-One, Inexact, Overflow, Underflow, and Exponent, respectively) describe the last-stage result of the adder. The high-order three bits of the 11-bit exponent of the adder result are stored in the AE field.

  The Adder Add-One and Multiplier Add-One bits indicate that the absolute value of the result fraction grew by one least-significant bit due to rounding. AA and MA are not influenced by the sign of the result.

  After a floating-point operation in a given unit (adder or multiplier), the result-status bits of that unit are undefined until the point at which result exceptions are reported.

  When written to the **fsr** with the U-bit set, the result-status bits are placed into the first stage of the adder and multiplier pipelines. When the processor executes pipelined operations, it propagates the result-status bits of a particular unit (multiplier or adder) one stage for each pipelined floating-point operation for that unit. When they reach the last stage, they replace the normal result-status bits in the **fsr** and generate traps, if enabled. When the U-bit is not set, result-status bits in the word being written to the **fsr** are ignored.

  In a floating-point dual-operation instruction (e.g. add- and-multiply or subtract-and-multiply), both the multiplier and the adder may set exception bits. The result-status bits for a particular unit remain set until the next operation that uses that unit.
- RR (Result Register) specifies which floating-point register (**f0-f31**) was the destination register when a result-exception trap occurs due to a scalar operation.
- IRP (Integer (Graphics) Pipe Result Precision), MRP (Multiplier Pipe Result Precision), and ARP (Adder Pipe Result Precision) aid in restoring pipeline state after a trap or process switch. Each defines the precision of the last-stage result in the corresponding pipeline. One of these bits is set when the result in the last stage of the corresponding pipeline is double precision; it is cleared if the result is single precision.
- LRP1 and LRP0 (Load Pipe Result Precision) together define the size of the last-stage result of the load pipeline. They are encoded as Table 2.6 shows.

Table 2.6. Values of LRP1 and LRP0

LRP1	LRP0	pfld Length
0	0	*(reserved)*
0	1	4 Bytes
1	0	8 Bytes
1	1	16 Bytes

2.2.9 KR, KI, T, AND MERGE REGISTERS

The KR, KI, and T registers are special-purpose registers used by the dual-operation floating-point instructions **pfam**, **pfsm**, **pfmam**, and **pfmsm**, which initiate both an adder operation and a multiplier operation. The KR, KI, and T registers can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions. (Refer to Figure 2.16.)

The MERGE register is used only by the graphics instructions. The purpose of the MERGE register is to accumulate (or merge) the results of multiple-addition operations that use as operands the color-intensity values from pixels or distance values from a Z-buffer. The accumulated results can then be stored in one 64-bit operation.

Two multiple-addition instructions and an OR instruction use the MERGE register. The addition instructions are designed to add interpolation values to each color-intensity field in an array of pixels or to each distance value in a Z-buffer.

Refer to the instruction descriptions in section 10 for more information about these registers.

2.2.10 BUS ERROR ADDRESS REGISTER

The **bear** helps the trap handler determine faulty memory locations. The i860 XP microprocessor loads a valid address into **bear** under these conditions:

- For bus errors, the **bear** receives the address of the cycle for which the BERR signal is asserted, if external hardware synchronizes assertion of BERR with BRDY# for that cycle.
- For parity errors on a read, the **bear** receives the address of the cycle during which the processor detects the error, if external hardware asserts PEN# with BRDY# for that cycle.

If external hardware does not meet these conditions, the contents of the **bear** are undefined.

A valid address in **bear** is accurate to 29 bits; that is, address signals A31–A3 are latched in the high-order 29 bits of **bear**. At RESET and after every parity and bus error trap, software must read the **bear** before further parity and bus error traps can occur. The **bear** is a read-only register.

2.2.11 PRIVILEGED REGISTERS

The registers **p0, p1 p2,** and **p3** are provided for the operating system to use. They do not affect processor operation. They can be accessed by the **ld.c** and **st.c** instructions, but they can be written only in supervisor mode. They may be used to store information such as the interrupt stack pointer, current user stack pointer at the beginning of the trap handler, register values during trap handling, processor ID in a multiprocessor system, or for any other purpose.

2.2.12 CONCURRENCY CONTROL REGISTER

The concurrency control register (**ccr**) controls the operation of the internal Concurrency Control Unit (CCU), which is described in section 2.5. The **ccr** can be written in supervisor mode only, but can be read in user or supervisor mode. Figure 2.8 shows the format of the **ccr**.

DO (Detached Only) bit and CO (CCU On) bit together specify the CCU configuration. DO, when set, indicates that there is no external CCU. CO (CCU On) bit, when set, indicates that the Concurrency Control Architecture is enabled. Table 2.7 summarizes the modes defined by CO and DO bits. The reserved combinations should not be used by software.

If the DCCU is on (CO = DO = 1), the processor intercepts and interprets all memory loads and stores which are to the CCU address space, which is the two pages defined by CCUBASE. Loads and stores to that address range do not go to memory, but to the DCCU.

Table 2.7. Values of CO and DO

CO	DO	Mode
0	0	External CCU, or no CCU
0	1	*reserved*
1	0	*reserved*
1	1	Internal CCU (DCCU) only

CCUBASE is the virtual address of the memory area into which the CCU registers are mapped. Software must set bit 12 to zero, because the CCUBASE must be aligned on a two page (8 Kbyte) boundary. This is because an external CCU contains supervisor registers mapped to the second page.

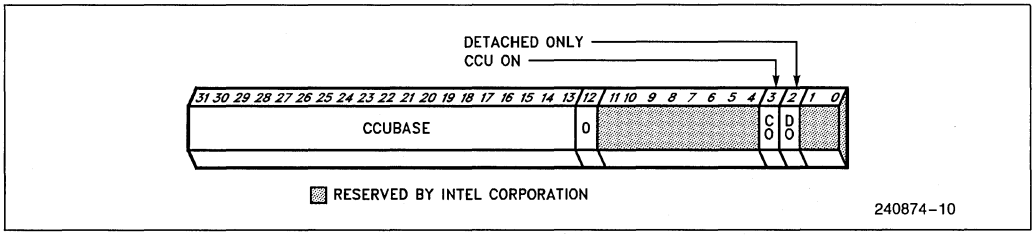


Figure 2.8. Concurrency Control Register

2.2.13 NEWCURR REGISTER

The NEWCURR register is part of the detached CCU (concurrency control unit). It a 32-bit counter that supplies an iteration count for loop execution. (Refer to section 2.5.)

NEWCURR is architecturally a 64-bit register, but only the low-order 32 bits are provided in this implementation. Compiler and operating-system data structures should provide for a 64-bit size for future implementation.

2.2.14 STAT REGISTER

The STAT register is part of the detached CCU (concurrency control unit). As Figure 2.9 shows, it contains the following bits:

InLoop Indicates that the processor is currently executing a concurrent loop. This bit is set when a processor starts a concurrent, non-nested loop, and it is cleared when the processor enters serial code when not nested or idle. It can also be read or written directly.

Nested Indicates whether the processor is in the nested state. InLoop is copied into this bit when starting a nested loop. Otherwise, it can be read or written directly.

Detached Always contains the value of **ccr** bit DO.

STAT is architecturally a 64-bit register. Compiler and operating-system data structures should provide for a 64-bit size for future implementation.

2.3 Addressing

Memory is addressed in byte units with a paged virtual-address space of 2^{32} bytes. Data and instructions can be located anywhere in this address space. Address arithmetic uses 32-bit input values and produces 32-bit results. The low-order 32 bits of the result are used in case of overflow.

Normally, multibyte data values are stored in memory in little endian format, i.e. with the least significant byte at the lowest memory address. As an option, the ordering can be dynamically selected by software in supervisor mode. The i860 XP microprocessor also offers big endian mode, in which the most significant byte of a data item is at the lowest address. Figure 2.10 defines by example how data is transferred from memory over the bus into a register in both modes. Big endian and little endian data areas should not be mixed within a 64-bit data word. Illustrations of data structures in this data sheet show data stored in little endian mode, i.e. the rightmost (low-order) byte is at the lowest memory address.

Code accesses are always done with little endian addressing. This implies that instructions appear differently than documented here when accessed as big endian data. Intel Corporation recommends that disassemblers running in a big endian system convert instructions that have been read as data back to little endian form and present them in the format documented here.

Page directories and page tables are also accessed in little endian mode, regardless of the value of the BE bit.

Big endian mode affects not only the memory load and store instructions but also the **ldio, stio, ldint,** and **scyc** instructions.

Alignment requirements are as follows (any violation results in a data-access trap):

- 128-bit values are aligned on 16-byte boundaries when referenced in memory (i.e. the four least significant address bits must be zero).
- 64-bit values are aligned on 8-byte boundaries when referenced in memory (i.e. the three least significant address bits must be zero).
- 32-bit values are aligned on 4-byte boundaries when referenced in memory (i.e. the two least significant address bits must be zero).

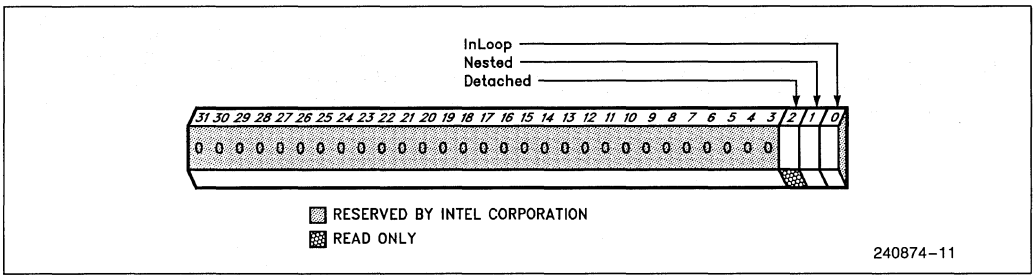
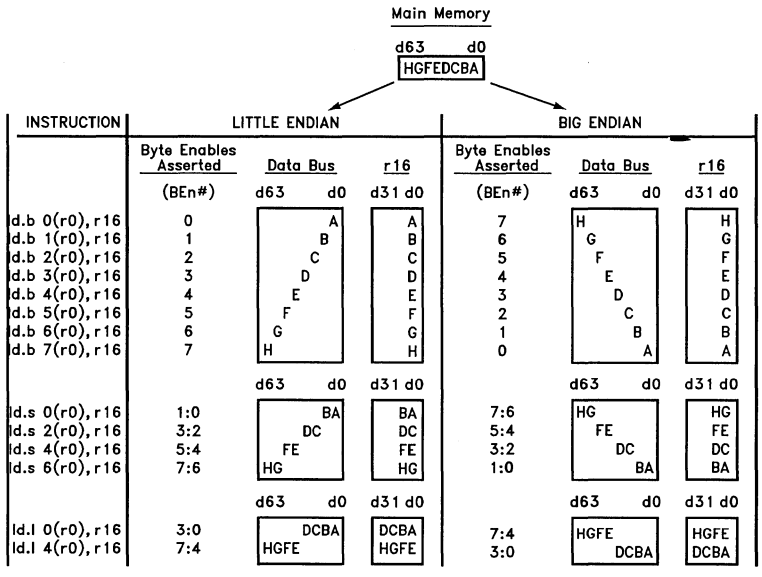


Figure 2.9. Concurrency Status Register



NOTE:
64- and 128-bit big endian accesses are treated the same as little endian accesses

Figure 2.10. Little and Big Endian Memory Transfers

- 16-bit values are aligned on 2-byte boundaries when referenced in memory (i.e. the least significant address bit must be zero).

2.4 Virtual Addressing

When address translation is enabled, the processor maps instruction and data virtual addresses into physical addresses before referencing memory. This address transformation is compatible with that of the Intel 386 and Intel 486 microprocessors and implements the basic features needed for page-oriented virtual-memory systems and page-level protection.

The address translation is optional. Address translation is disabled when the processor is reset. It is enabled when a store (**st.c**) to **dirbase** sets the ATE bit. The operating system typically does this during software initialization. Address translation is disabled again when **st.c** clears the ATE bit. The ATE bit must be set if the operating system is to implement page-oriented protection or page-oriented virtual memory.

2.4.1 PAGE FRAME

A *page frame* is a unit of contiguous addresses of physical main memory. A *page* is the collection of data that occupies a page frame when that data is present in main memory or occupies some location in secondary storage when there is not sufficient space in main memory.

The i860 XP microprocessor architecture supports two sizes of pages and page frames: four Mbytes and four Kbytes. Four Kbyte page frames begin on four Kbyte boundaries and are fixed in size. Four Mbyte page frames begin on four Mbyte boundaries and are fixed in size. The four Kbyte address transformation is compatible with that of the Intel 486 microprocessor.

2.4.2 VIRTUAL ADDRESS

A virtual address refers indirectly to a physical address by specifying a page and an offset within that page. Figure 2.11 shows the formats of virtual addressess. The format for virtual addresses that refer to four Mbyte pages is different from that of four Kbyte pages.

Figure 2.12 shows how the i860 XP microprocessor converts a virtual address into the physical address by consulting page tables. The addressing mechanism uses the DIR field as an index into a page directory. For 4K pages, it uses the PAGE field as an index into the page table determined by the page directory and uses the OFFSET field to address a byte within the page determined by the page table. For 4M pages, the page directory entry determines the page address, and the OFFSET field addresses a byte within that page table.

2.4.3 PAGE TABLES

A page table is simply an array of 32-bit page specifiers. A page table is itself a page, and contains 4 Kbytes of data or at most 1K 32-bit entries.

At the highest level is a page directory. The page directory holds up to 1K entries that address either page tables of the second level or 4-Mbyte pages.

A page table of the second level addresses up to 1K 4-Kbyte pages. All the tables addressed by one page directory, therefore, can address 1M 4-Kbyte pages.

Whether 4-Mbyte pages, 4-Kbyte pages, or some combination of the two are used, one page directory can cover the entire four gigabyte physical address space of the i860 XP microprocessor (1K page directory entries × 4M page or 1K page directory entries × 1K page table entries × 4K page).

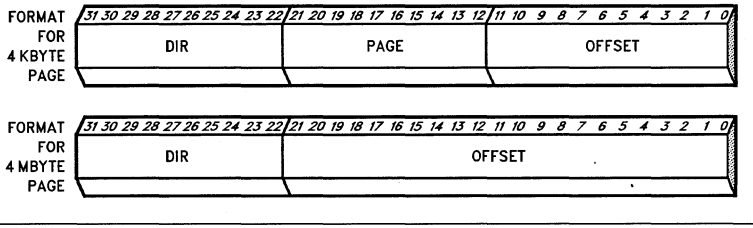


Figure 2.11. Formats of Virtual Addresses

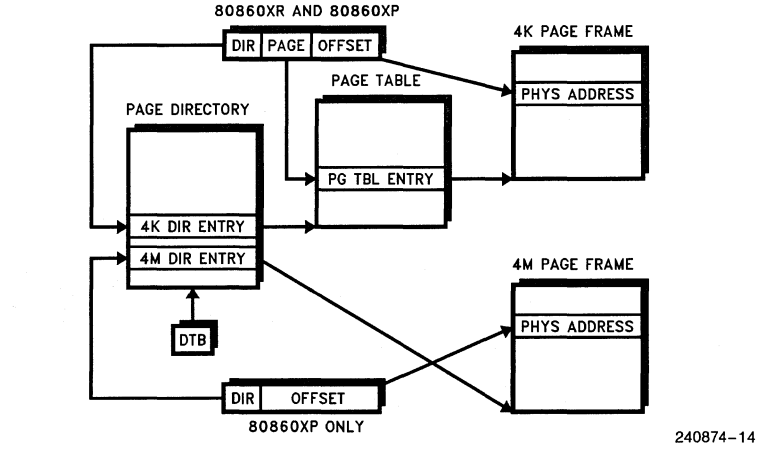


Figure 2.12. Address Translation

The physical address of the current page directory is stored in the DTB field of the **dirbase** register. Memory management software has the option of using one page directory for all processes, one page directory for each process, or some combination of the two.

2.4.4 PAGE-TABLE ENTRIES

Page-table entries (PTEs) have one of the formats shown by Figure 2.13.

2.4.4.1 Page Frame Address

The page frame address specifies the physical starting address of a page. In a page directory, the page frame address is either the address of a page table or the address of the four Mbyte page frame that contains the desired memory operand. In a second-level page table, the page frame address is the address of the 4-Kbyte page frame that contains the desired memory operand.

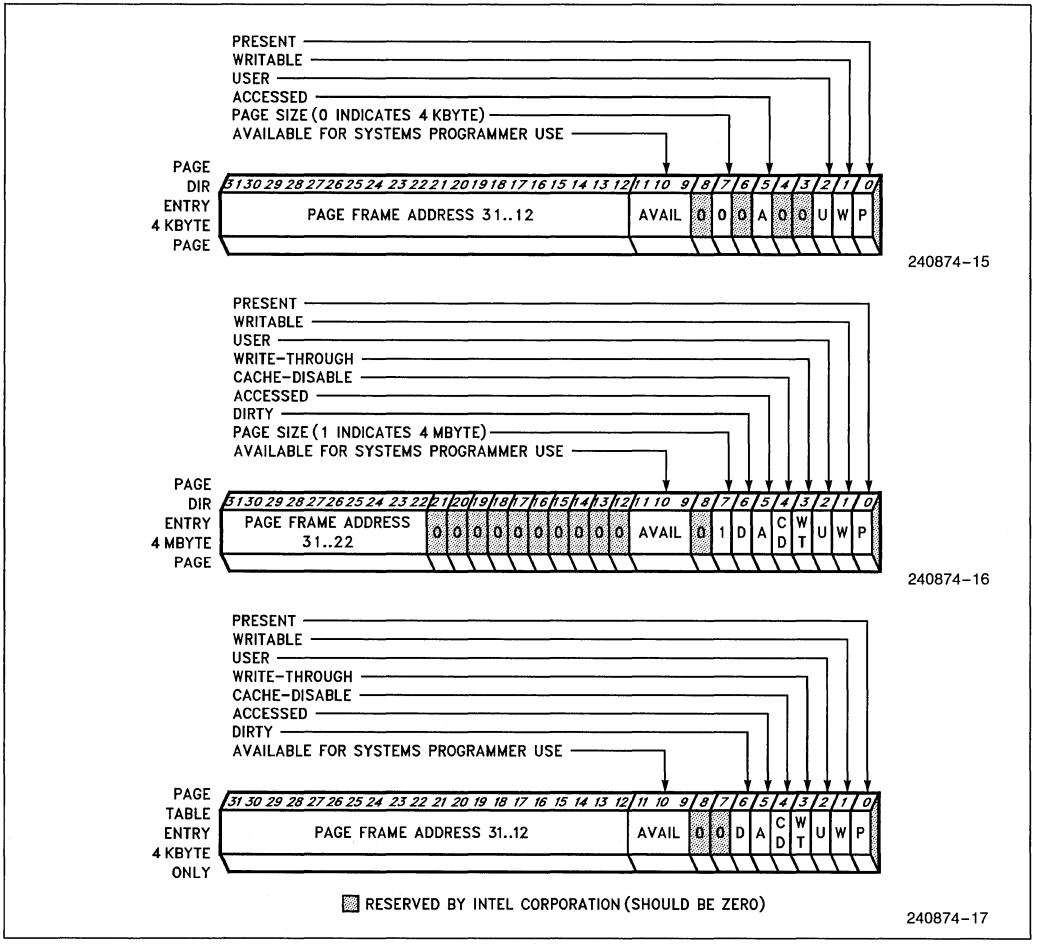


Figure 2.13. Formats of Page Table Entries

2.4.4.2 Present Bit

The P (present) bit indicates whether a page table entry can be used in address translation. P=1 indicates that the entry can be used. When P=0 in either level of page tables, the entry is not valid for address translation, and the rest of the entry is available for software use; none of the other bits in the entry is tested by the hardware. If P=0 in either level of page tables when an attempt is made to use a page-table entry for address translation, the processor signals either a data-access fault or an instruction-access fault. In software systems that support paged virtual memory, the trap handler can bring the required page into physical memory.

Note that there is no P bit for the page directory itself. The page directory may be not-present while the associated process is suspended, but the operating system must ensure that the page directory indicated by the **dirbase** image associated with the process is present in physical memory before the process is dispatched.

2.4.4.3 Writable and User Bits

The W (writable) and U (user) bits are used for page-level protection, which the i860 XP microprocessor performs at the same time as address translation. The concept of privilege for pages is implemented by assigning each page to one of two levels:

Supervisor level (U=0) — For the operating system and other systems software and related data.

User level (U=1) — For applications procedures and data.

The U bit of the **psr** indicates whether the i860 XP microprocessor is executing at user or supervisor level. The i860 XP microprocessor maintains the U bit of **psr** as follows:

- The i860 XP microprocessor clears the **psr** U bit to indicate supervisor level when a trap occurs (including when the **trap** instruction causes the trap). The prior value of U is copied into PU.
- The i860 XP microprocessor copies the **psr** PU bit into the U bit when an indirect branch is executed and one of the trap bits is set. If PU was one, the i860 XP microprocessor enters user level.

With the U bit of **psr** and the W and U bits of the page table entries, the i860 XP microprocessor implements the following protection rules:

- When at user level, a read or write of a supervisor-level page causes a trap.
- When at user level, a write to a page whose W bit is not set causes a trap.
- When at user level, a store (**st.c**) to certain control registers is ignored.
- When at user level, privileged instructions (**ldio, stio, scyc, ldint**) have no effect.

When the i860 XP microprocessor is executing at supervisor level, all pages are addressable, but, when it is executing at user level, only pages that belong to the user level are addressable.

When the i860 XP microprocessor is executing at supervisor level, all pages are readable. Whether a page is writable depends upon the write-protection mode controlled by WP of **epsr**:

WP=0 All pages are writable.

WP=1 A write to page whose W bit is not set causes a trap.

When the i860 XP microprocessor is executing at user level, only pages that belong to user level and are marked writable are actually writable; pages that belong to supervisor level are neither readable nor writable from user level.

2.4.4.4 Write-Through Bit

The i860 XP microprocessor implement both write-back and write-through caching policies for the on-chip instruction and data caches. If WT is set, the write-through policy is applied to data from the corresponding page. If WT is clear, the normal write-back policy is applied to data from the page.

For four-Mbyte pages, the WT bit of the page directory entry is used. For four-Kbyte pages, only the WT bit of the second-level page table entry is used; the WT bit of the page directory entry is not referenced by the processor, but is *reserved*.

The value of the WT bit is driven externally on the PWT pin, so that external caches can employ the same policy used internally.

2.4.4.5 Cache Disable Bit

If a page's CD (cache disable) bit is set, data from the page is not placed in the internal instruction or data caches (regardless of the value of the WT bit). Clearing CD permits the processor to place data from the associated page into internal caches.

For four-Mbyte pages, the CD bit of the page directory entry is used. For four-Kbyte pages, only the CD bit of the second-level page table entry is used; the CD bit of the page directory entry is not referenced by the processor, but is *reserved*.

The value of the CD bit is driven externally on the PCD pin, so that cacheability can be the same in both internal and external caches.

2.4.4.6 Accessed and Dirty Bits

The A (accessed) and D (dirty) bits provide data about page usage in both levels of the page tables.

The i860 XP microprocessor sets the A-bit before a read or write operation to a page. For four-Kbyte pages, it sets the A-bit of both levels of page tables.

The processor tests the dirty bit before a write, and, under certain conditions, causes traps. The trap handler then has the opportunity to maintain appropriate values in the dirty bits. For four-Mbyte pages, the D bit of the page directory entry is used. For four-Kbyte pages, only the D bit of the second-level page table entry is used; the D bit of the page directory entry is not referenced by the processor, but is *reserved*. The precise algorithm for using these bits is specified in section 2.4.5.

An operating system that supports paged virtual memory can use the D and A bits to determine what pages to eliminate from physical memory when the demand for memory exceeds the physical memory available. The D and A bits are normally initialized to zero by the operating system. The processor sets the A bit when a page is accessed either by a read or write operation. When a data-access fault occurs, the trap handler sets the D bit if an allowable write is being performed, then reexecutes the instruction.

The operating system is responsible for coordinating its updates to the accessed and dirty bits with updates by the CPU and by other processors that may share the page tables. The i860 XP microprocessor automatically asserts the LOCK# signal while testing and setting the A bit.

2.4.4.7 Page Tables for Trap Handlers

When paging is enabled (ATE = 1), software that creates page tables and directories must assure that A = 1 always in the PTEs and PDEs for the code pages of the trap handler and the first data page accessed by the handler. Preallocation of these pages is required in case a trap occurs during a lock sequence. Otherwise, recursive traps would be generated, as the A-bit would need to be set by the translation hardware, which is a trapping situation in itself.

2.4.4.8 Combining Protection of Both Levels of Page Tables

For any four-Kbyte page, the protection attributes of its page directory entry may differ from those of its page table entry. The i860 XP microprocessor computes the effective protection attributes for a page by examining the protection attributes in both the directory and the page table and choosing the more restrictive of the two.

2.4.5 ADDRESS TRANSLATION ALGORITHM

The following algorithm defines the translation of each virtual address to a physical address. Let DIR, PAGE, and OFFSET be the fields of the virtual address; let PFA1 and PFA2 be the page frame address fields of the first and second level page tables respectively; DTB is the page directory table base address stored in the **dirbase** register.

1. Read the PDE (Page Directory Entry) at the physical address formed by DTB:DIR:00.
2. If P in the PDE is zero, generate a data- or instruction-access fault.
3. If W in the PDE is zero, the operation is write, and either the U bit of the PSR is set or WP = 1, generate a data-access fault.
4. If the U bit in the PDE is zero and U bit in the **psr** is set, generate a data- or instruction-access fault.
5. If A in the PDE is zero and the TLB miss occurred inside a locked sequence, generate a data or instruction access fault. (The trap allows software to set A to one and restart the sequence. This helps external bus hardware determine unambiguously what address corresponds to a locked semaphore.)
6. If bit 7 of the PDE is one (four Mbyte page), and the operation is write, and D = 0 in the PDE, generate a data-access fault.
7. If A = 1 in the PDE, continue at step 11. Otherwise, assert LOCK#.
8. Perform the PDE read as in step 1 and the P, W and U bit checks as in steps 2 through 4.
9. Write the PDE with A bit set.
10. Deassert LOCK#.
11. If bit 7 of the PDE is one (four Mbyte page), form the physical address as PFA1:OFFSET, and exit address translation. In this case, PFA1 is 10 bits and OFFSET is 22 bits.
12. The remaining steps are for four Kbyte pages. If the A-bit in the PDE was zero before translation began, assert LOCK#.

13. Fetch the PTE at the physical address formed by PFA1:PAGE:00.
14. Perform the P-, W-, U-, and A-bit checks as in steps 2 through 5 with the second-level PTE. If A = zero in the PTE, and the TLB miss occurred inside a locked sequence, generate a data or instruction access fault. LOCK# remains active.
15. If the operation is write, and D in the PTE is zero, generate a data access fault.
16. If the A-bit in the PDE was already active before translation began, and the A-bit in the PTE is already active, go to step 20.
17. If LOCK# is not already active, assert it and refetch the PTE.
18. Perform the U-, W-, and P-bit checks and A-bit setting in the PTE as in steps 8 through 9. Do the locked write update of the PTE to unlock the bus, even if the A-bit in the PTE is already one.
19. Deassert LOCK#.
20. Form the physical address as PFA2:OFFSET. In this case, PFA2 is 20 bits and OFFSET is 12 bits.

During translation, the i860 XP microprocessor looks only in external memory for page directories and page tables. The data cache is not searched. Therefore, any code that modifies page directories or page tables must keep them out of the cache. The tables should either be kept in noncacheable memory or in write-through pages or should be flushed from the cache.

The i860 XP microprocessor expects page directories and page tables to be in little endian format. The operating system must maintain these tables in little endian format either by setting BE to zero when manipulating the tables or by complementing bit two of the 32-bit address when loading or storing entries.

2.4.6 ADDRESS TRANSLATION FAULTS

The address translation fault can be signalled as either an instruction access fault or a data-access fault. The instruction causing the fault can be reexecuted upon returning from the trap handler.

2.5 Detached CCU

The i860 XP microprocessor supports parallel processing, where multiple processors work simultaneously on different parts of the same problem. The Concurrency Control Unit (CCU) controls work sharing among CPUs, in multiprocessor systems. The CCU is a VLSI chip that allows multiple processors to work together to execute portions of a single program in parallel. The CCU performs the iteration assignment for loop parallelization. Accesses to the CCU for synchronization are much faster than accesses to shared memory semaphores. The CCU is memory mapped, and its internal registers are accessed via memory load and store operations.

To take advantage of the parallel architecture, software must be compiled by parallelizing compilers that generate instructions to access the CCU. However, such instructions cannot run on a system that does not include a CCU. To allow an application compiled for parallel execution to run on any system based on the i860 XP microprocessor, a "Detached Only" CCU (DCCU, also referred to as "internal CCU") is implemented in the i860 XP microprocessor. The DCCU is a compatible subset of the external CCU, consisting of the minimal set of features required for a single CPU. The DCCU alone neither increases performance nor concurrency, but does allow software designed for parallel processing to run unmodified on a single CPU.

2.5.1 DCCU INITIALIZATION

After reset, the i860 XP microprocessor DCCU is disabled (CO and DO bits in **ccr** are cleared). To enable the DCCU, the CO and DO bits in **ccr** must be set by software. Before turning on the CCU, the operating system must invalidate the TLB and flush the data cache to make sure that they do not contain data from the CCU pages. The TLB is invalidated by setting ITI = 1 in the **dirbase** register. Also, the **flush** instruction must be used once per each line of the data cache to invalidate the physical address of the cache entry, if the two pages at the CCUBASE address may have been cached. The flush is unneeded if page tables or external hardware have prohibited caching of the CCUBASE pages.

Neither the external CCU nor the DCCU can be accessed within four instructions after **ccr** is modified.

2.5.2 DCCU ADDRESSING

The CCU facilities are memory-mapped, manipulated by normal load and store instructions. The DCCU is memory-mapped to a single 4 Kbyte user page. When the DCCU is active, all accesses to this page are satisfied by the DCCU, and no external bus cycle is generated. The address space of two adjacent pages beginning on an 8 Kbyte boundary is reserved for the CCU. The first (lower address) page contains

locations accessible in user mode (which includes the DCCU registers), and the second page contains locations accessible in supervisor mode (used for external CCU only). The base address of these pages is specified by the CCUBASE field in **ccr**. Accesses to the second page in DCCU-only mode have no effect on the DCCU, and are treated as normal memory accesses.

When the DCCU is active, accesses to its address page use only the virtual address, and no translation is done on the DCCU access. However, the accesses to an external CCU go through normal address translation. The operating system should make sure that the page table entries for the CCU pages are set so that no fault occurs during address translation. If an external CCU is used, the two PTEs for the CCU should have CD = 1 (caching disabled) and page frame addresses that match the external hardware addresses of the CCU. Accesses to the DCCU that cause a TLB miss do not cause the PTE to be loaded into the TLB.

If the external CCU is used when address translation is disabled (ATE=0), external hardware must deactivate KEN# for such accesses, to avoid caching external CCU accesses.

2.5.3 DCCU INTERNALS

The DCCU consists of an address decoder, a 32-bit counter (NEWCURR), and three bits of state information (InLoop, Nested, and Detached). InLoop, Nested and Detached). InLoop, Nested and Detached correspond to bits 0, 1, and 2 respectively of the external CCU STAT register. The Detached bit always reflects the value of the DO bit in **ccr**.

Several addresses within the DCCU memory page are decoded to cause actions to NEWCURR, InLoop, and Nested state bits. The CCU register to be accessed is specified by address bits 11–3. The valid CCU addresses are shown in Table 2.8 with their mnemonics. Accesses to these address may also have side effects within the DCCU. Refer to the *i860™ Microprocessor Family Programmer's Reference Manual* for programming information. Loads from any other addresses within the DCCU memory page return zero; stores to any other addresses have no effect. Access to the DCCU by any load or store instructions other than **ld.x** and **st.x** produce undefined results.

Assemblers should encode address bits 2–0 as zero for accesses in little-endian mode. However, in big-endian mode (**epsr** BE bit = 1), DCCU accesses should have address bit 2 active. Thus, software for big-endian access to the DCCU must differ from little-endian software. That allows an external CCU to be accessed in both big and little endian modes.

When reading from the DCCU, the access latency is the same as reading data from the data cache—the data is ready for use as a source by the second instruction after the load. The first instruction after the load may use the data, but that instruction will experience a one-clock freeze before the data becomes available.

2.6 Instruction Set

Table 2.9 shows the complete set of instructions for the i860 XP microprocessor, grouped by function within processing unit. Refer to Section 10 for an algorithmic definition of each instruction. The instruction set of the i860 XP microprocessor is fully upward compatible with that of the i860 XR microprocessor, extended in a few ways to better serve certain application domains. User-level software applications written for the i860 XR microprocessor will run unmodified on the i860 XP microprocessor, but some supervisor code (for example, trap handlers) may need minor modifications. The i860 XR microprocessor instruction set has been extended with the following instructions:

- **ldio, stio**: I/O load and store instructions
- **ldint**: Load interrupt instruction to perform an interrupt acknowledge cycle and read the interrupt vector. Used to emulate the Intel 486 interrupt acknowledge sequence.
- **scyc**: A special-cycle instruction, used to generate bus cycles that signal invalidation and synchronization of an external cache.
- **pfld.q**: A pipelined, floating-point load of 128 bits.

Table 2.8. CCU Addresses

Mnemonic	A11–A8	A7–A4	Little Endian A3–A0	Big Endian A3–A0
cbr_*j*	0000	0**abc**	**b**000	**d**100
cget	1111	0110	0000	0100
cnewcurr	1111	1100	0000	0100
cstat	1111	1100	1000	1100
cstatci	1111	1101	0000	0100
cstatn	1111	1101	1000	1100
cclm	1111	1110	1000	1100
cver	1111	1111	1000	1100

NOTE:
Variable **i** is a 4-bit index formed by A6–A3. Let its binary form be represented by the symbols **abcd**.

Table 2.9. Instruction Set (1 of 2)

Core Unit	
Mnemonic	**Description**
Load and Store Instructions	
ld.x	Load integer
st.x	Store integer
fld.y	F-P load
fst.y	F-P store
pfld.y	Pipelined F-P load
pst.d	Pixel store
Register to Register Move	
ixfr	Transfer integer to F-P register
Integer Arithmetic Instructions	
addu	Add unsigned
adds	Add signed
subu	Subtract unsigned
subs	Subtract signed
Shift Instructions	
shl	Shift left
shr	Shift right
shra	Shift right arithmetic
shrd	Shift right double
Logical Instructions	
and	Logical AND
andh	Logical AND high
andnot	Logical AND NOT
andnoth	Logical AND NOT high
or	Logical OR
orh	Logical OR high
xor	Logical exclusive OR
xorh	Logical exclusive OR high
Control-Transfer Instructions	
br	Branch direct
bri	Branch indirect
bc	Branch on CC
bc.t	Branch on CC taken
bnc	Branch on not CC
bnc.t	Branch on not CC taken
bte	Branch if equal
btne	Branch if not equal
bla	Branch on LCC and add
call	Subroutine call
calli	Indirect subroutine call
intovr	Software trap on integer overflow
trap	Software trap

Floating-Point Unit	
Mnemonic	**Description**
Register to Register Move	
fxfr	Transfer F-P to integer register
F-P Multiplier Instructions	
fmul.p	F-P multiply
pfmul.p	Pipelined F-P multiply
pfmul3.dd	3-Stage pipelined F-P multiply
fmlow.p	F-P multiply low
frcp.p	F-P reciprocal
fsqr.p	F-P reciprocal square root
F-P Adder Instructions	
fadd.p	F-P add
pfadd.p	Pipelined F-P add
famov.r	F-P adder move
pfamov.r	Pipelined F-P adder move
fsub.p	F-P subtract
pfsub.p	Pipelined F-P subtract
pfgt.p	Pipelined greater-than compare
pfeq.p	Pipelined equal compare
fix.v	F-P to integer conversion
pfix.v	Pipelined F-P to integer conversion
ftrunc.v	F-P to integer truncation
Dual-Operation Instructions	
pfam.p	Pipelined F-P add and multiply
pfsm.p	Pipelined F-P subtract and multiply
pfmam.p	Pipelined F-P multiply with add
pfmsm.p	Pipelined F-P multiply with subtract
Long Integer Instructions	
fisub.z	Long-integer subtract
pfisub.z	Pipelined long-integer subtract
fiadd.z	Long-integer add
pfiadd.z	Pipelined long-integer add
Graphics Instructions	
fzchks	16-bit Z-buffer check
pfzchds	Pipelined 16-bit Z-buffer check
fzchkl	32-bit Z-buffer check
pfzchkl	Pipelined 32-bit Z-buffer check
faddp	Add with pixel merge
pfaddp	Pipelined add with pixel merge
faddz	Add with Z merge
pfaddz	Pipelined add with Z merge
form	OR with MERGE register
pform	Pipelined OR with MERGE register

Table 2.9. Instruction Set (2 of 2)

Core Unit	
Mnemonic	**Description**
I/O Instructions	
ldio.x	Load I/O
stio.x	Store I/O
ldint.x	Load interrupt vector
System Control Instructions	
flush	Cache flush
ld.c	Load from control register
st.c	Store to control register
lock	Begin interlocked sequence
unlock	End interlocked sequence
scyc.x	Special bus cycles
Assembler Pseudo-Operations	
Register to Register Move	
mov	Integer move
fmov.r	F-P reg-reg move
pfmov.r	Pipelined F-P reg-reg move
nop	Core no-operation
fnop	F-P no-operation
pfle.p	Pipelined F-P less-than or equal

The architecture of the i860 XP microprocessor uses parallelism to increase the rate at which operations may be introduced into the unit. Parallelism in the i860 XP microprocessor is **not** transparent; rather, programmers have complete control over parallelism and therefore can achieve maximum performance for a variety of computational problems.

2.6.1 PIPELINED AND SCALAR OPERATIONS

One type of parallelism used within the floating-point unit is "pipelining". The pipelined architecture treats each operation as a series of more primitive operations (called "stages") that can be executed in parallel. Consider just the floating-point adder as an example. Let **A** represent the operation of the adder. Let the stages be represented by $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$. The stages are designed such that $\mathbf{A}_{i+1}$ for one adder instruction can execute in parallel with $\mathbf{A}_i$ for the next adder instruction. Furthermore, each $\mathbf{A}_i$ can be executed in just one clock. The pipelining within the multiplier and graphics units can be described similarly, except that the number of stages may be different.

Figure 2.14 illustrates three-stage pipelining as found in the floating-point adder (also in the floating-point multiplier when single-precision input operands are employed). The central columns of the table represent the three stages of the pipeline. Each stage holds intermediate results and also (when introduced into the first stage by software) holds status information pertaining to those results. The table assumes that the instruction stream consists of a series of consecutive floating-point instructions, all of one type (i.e. all adder instructions or all single-precision multiplier instructions). The instructions are represented as **A**, **B**, etc. The rows of the table represent the states of the unit at successive clock cycles. Each time a pipelined operation is performed, the result of the last stage of the pipeline is stored in the destination register *fdest*, the pipeline is advanced one stage, and the input operands of the operation are transferred to the first stage of the pipeline.

Clock	Instruction	Pipeline			Result
		Stage 1	Stage 2	Stage 3	
1	A	A			
2	B	B	A		
3	C	C	B	A	
4	D	D	C	B	A → *fdest* of D
5	E	E	D	C	B → *fdest* of E
6	F	F	E	D	C → *fdest* of F

Figure 2.14. Pipelined Instruction Execution

In the i860 XP microprocessor, the number of pipeline stages ranges from one to three. A pipelined operation with a three-stage pipeline stores the result of the third prior operation. A pipelined operation with a two-stage pipeline stores the result of the second prior operation. A pipelined operation with a one-stage pipeline stores the result of the prior operation.

There are four floating-point pipelines: one for the multiplier, one for the adder, one for the graphics unit, and one for floating-point loads. The adder pipeline has three stages. The number of stages in the multiplier pipeline depends on the precision of the source operands in the pipeline; it may have two or three stages. The graphics unit has one stage for all precisions. The load pipeline has three stages for all precisions.

Changing the FZ (flush zero), RM (rounding mode), or RR (result register) bits of **fsr** while there are results in either the multiplier or adder pipeline produces effects that are not defined.

2.6.1.1 Scalar Mode

In addition to the pipelined execution mode, the i860 XP microprocessor also can execute floating-point instructions in "scalar" mode. Most floating-point instructions have both pipelined and scalar variants, distinguished by a bit in the instruction encoding. In scalar mode, the floating-point unit does not start a new operation until the previous floating-point operation is completed. The scalar operation passes through all stages of its pipeline before a new operation is introduced, and the result is stored automatically. Scalar mode is used when the next operation depends on results from the previous few floating-point operations (or when the compiler or programmer does not want to deal with pipelining).

2.6.1.2 Pipelining Status Information

Result status information in the **fsr** consists of the AA, AI, AO, AU, and AE bits, in the case of the adder, and the MA, MI, MO, and MU bits, in the case of the multiplier. This information arrives at the **fsr** via the pipeline in one of two ways:

1. It is calculated by the last stage of the pipeline. This is the normal case.
2. It is propagated from the first stage of the pipeline. This method is used when restoring the state of the pipeline after a preemption. When a store instruction updates the **fsr** and the the U bit being written into the **fsr** is set, the store updates the result status bits in the first stage of both the adder and multiplier pipelines. When software changes the result-status bits of the first stage of a particular unit (multiplier or adder), the updated result-status bits are propagated one stage for each pipelined floating-point operation for that unit. In this case, each stage of the adder and multiplier pipelines holds its own copy of the relevant bits of the **fsr**. When they reach the last stage, they override the normal result-status bits computed from the last-stage result.

At the next floating-point instruction (or at certain core instructions), after the result reaches the last stage, the i860 XP microprocessor traps if any of the status bits of the **fsr** indicate exceptions. Note that the instruction that creates the exceptional condition is not the instruction at which the trap occurs.

2.6.1.3 Precision in the Pipelines

In pipelined mode, when a floating-point operation is initiated, the result of an earlier pipelined floating-point operation is returned. The result precision of the current instruction applies to the operation being initiated. The precision of the value stored in *fdest* is that which was specified by the instruction that initiated that operation.

If *fdest* is the same as *fsrc1* or *fsrc2*, the value being stored in *fdest* is used as the input operand. In this case, the precision of *fdest* must be the same as the source precision.

The multiplier pipeline has two stages when the source operands are double-precision and three stages when they are single. This means that a pipelined multiplier operation stores the result of the second previous multiplier operation for double-precision inputs and third previous for single-precision inputs (except when changing precisions).

2.6.1.4 Transition between Scalar and Pipelined Operations

When a scalar operation is executed, it passes through all stages of the pipeline; therefore, any unstored results in the affected pipeline are lost. To avoid losing information, the last pipelined operations before a scalar operation should be dummy pipelined operations that unload unstored results from the affected pipeline.

After a scalar operation, the values of all pipeline stages of the affected unit (except the last) are undefined. No spurious result-exception traps result when the undefined values are subsequently stored by pipelined operations; however, the values should not be referenced as source operands.

For best performance a scalar operation should not immediately precede a pipelined operation whose *fdest* is nonzero.

2.6.1.5 Pipelined Loads

The **pfld** instruction is optimized for accesses that miss the data cache and transfer directly from memory. Therefore, even when there is a data cache hit, a **pfld** may generate a bus cycle. The data from the internal cache is used only if it was modified. Otherwise, data is taken from the external bus, even if it resides in the on-board cache.

The **pfld** FIFO can be extended externally, due to the facts that a **pfld** always generates a bus cycle and that such a cycle can be identified externally by the value on the CTYP pin. Software written for an externally-extended **pfld** pipeline must ensure that it does not **pfld** from a location that was modified in the data cache. When a **pfld** cache hit to a modified line occurs, the **pfld** pipeline length used by the i860 XP microprocessor is three stages. The modified data from the cache is put into the internal three-stage data FIFO, and the third **pfld** instruction after the data cache hit will update its *fdest* register with the modified data.

2.6.2 DUAL-INSTRUCTION MODE

Another form of parallelism results from the fact that the i860 XP microprocessor can execute both a floating-point and a core instruction simultaneously. Such parallel execution is called *dual-instruction mode*. When executing in dual-instruction mode, the instruction sequence consists of 64-bit aligned instruction pairs, with a floating-point instruction in the lower 32 bits and a core instruction in the upper 32 bits. Table 2.9 identifies which instructions are executed by the core unit and which by the floating-point unit.

Programmers specify dual-instruction mode either by including in the mnemonic of a floating-point instruction a **d.** prefix or by using the Assembler directives **.dual** . . . **.enddual**. Both of the specifications cause the D-bit of floating-point instructions to be set. If the i860 XP microprocessor is executing in single-instruction mode and encounters a floating-point instruction with the D-bit set, one more 32-bit instruction is executed before dual-mode execution begins. If the i860 XP microprocessor is executing in dual-instruction mode and a floating-point instruction is encountered with a clear D-bit, then one more pair of instructions is executed before resuming single-instruction mode. Figure 2.15 illustrates two variations of this sequence of events: one for extended sequences of dual-instructions and one for a single instruction pair.

Note that **d.fnop** cannot be used to initiate dual instruction mode.

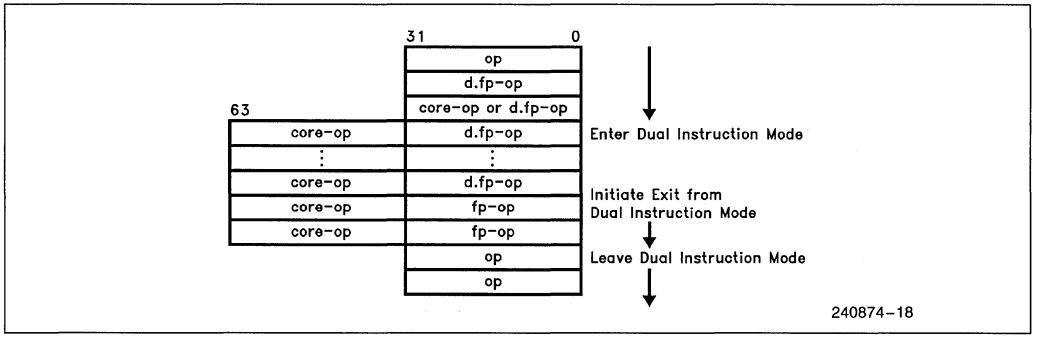


Figure 2.15. Dual-Instruction Mode Transitions (1 of 2)

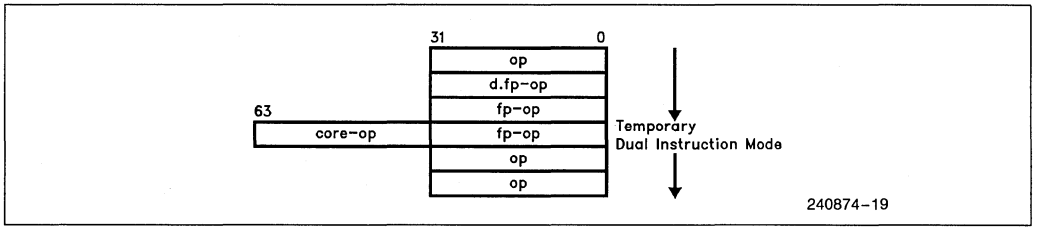


Figure 2.15. Dual-Instruction Mode Transitions (2 of 2)

When a 64-bit dual-instruction pair sequentially follows a delayed branch instruction in dual-instruction mode, both 32-bit instructions are executed.

2.6.3 DUAL-OPERATION INSTRUCTIONS

Special dual-operation floating-point instructions (add-and-multiply, subtract-and-multiply) use both the multiplier and adder units within the floating-point unit in parallel to efficiently execute such common tasks as evaluating systems of linear equations, performing the Fast Fourier Transform (FFT), and performing graphics transformations.

The instruction classes **pfam** *fsrc1, fsrc2, fdest*, **pfmam** *fsrc1, fsrc2, fdest* (add and multiply), **pfsm** *fsrc1, fsrc2, fdest*, and **pfmsm** *fsrc1, fsrc2, fdest* (subtract and multiply) initiate both an adder operation and a multiplier operation. Six operands are required, but the instruction format specifies only three operands; therefore, there are special provisions for specifying the operands. These special provisions consist of:

- Three special registers (KR, KI, and T) that can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions.
  - — The constant registers KR and KI can store the value of *fsrc1* and subsequently supply that value to the multiplier pipeline in place of *fsrc1*.
  - — The transfer register T can store the last-stage result of the multiplier pipeline and subsequently supply that value to the adder pipeline in place of *fsrc1*.
- A four-bit data-path control field in the opcode (DPC) that specifies the operands and loading of the special registers.
  1. Operand-1 of the multiplier can be KR, KI, or *fsrc1*.
  2. Operand-2 of the multiplier can be *fsrc2*, the last-stage result of the multiplier pipeline, or the last-stage result of the adder pipeline.
  3. Operand-1 of the adder can be *fsrc1*, the T-register, the last-stage result of the multiplier pipeline, or the last-stage result of the adder pipeline.
  4. Operand-2 of the adder can be *fsrc2*, the last-stage result of the multiplier pipeline, or the last-stage result of the adder pipeline.

Figure 2.16 shows all the possible data paths surrounding the adder and multiplier. The DPC field in these instructions selects different data paths. Section 10 shows the various encodings of the DPC field.

Note that the mnemonics **pfam.p**, **pfsm.p**, **pfmam.p**, and **pfmsm.p** are never used as such in the assembly language; these mnemonics are used here to designate classes of related instructions. Each value of DPC has a unique mnemonic associated with it.

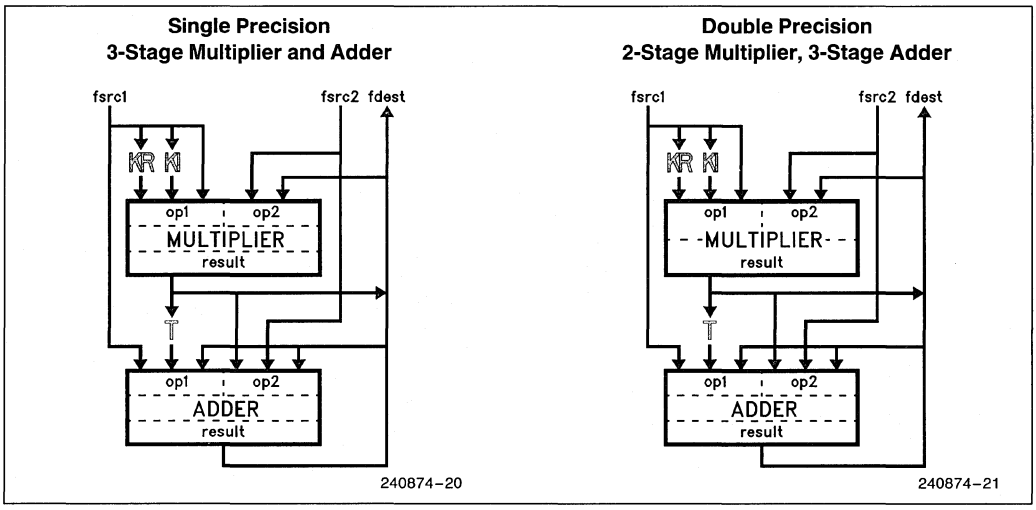


Figure 2.16. Dual-Operation Data Paths

2.7 Addressing Modes

Data access is limited to load and store instructions. Memory addresses are computed from two fields of load and store instructions: *isrc1* and *isrc2*.

1. *isrc1* either contains the identifier of a 32-bit integer register or contains an immediate 16-bit address offset.
2. *isrc2* always specifies a register.

Because either *isrc1* or *isrc2* may be null (zero), a variety of useful addressing modes result:

offset + ***register***	Useful for accessing fields within a record, where *register* points to the beginning of the record. Useful for accessing items in a stack frame, where *register* is **r3**, the register used for pointing to the beginning of the stack frame.
register + ***register***	Useful for two-dimensional arrays or for array access within the stack frame.
register	Useful as the end result of any arbitraryaddress calculation.
offset	Absolute address into the first or last 32K of the logical address space.

In addition, the floating-point load and store instructions may select autoincrement addressing. In this mode *isrc2* is replaced by the sum of *isrc1* and *isrc2* after performing the load or store. This mode makes stepping through arrays more efficient, because it eliminates one address-calculation instruction.

2.8 Traps and Interrupts

Traps are caused by exceptional conditions detected in programs or by external interrupts. Traps cause interruption of normal program flow to execute a special program known as a trap handler. Traps are divided into the types shown in Table 2.10.

2.8.1 TRAP HANDLER INVOCATION

This section applies to traps other than reset. When a trap occurs, execution of the current instruction is aborted. Except for bus error and parity error traps, the instruction is restartable. The processor takes the following steps while transferring control to the trap handler:

1. Copies U (user mode) of the **psr** into PU (previous U).
2. Copies IM (interrupt mode) into PIM (previous IM).
3. Sets U to zero (supervisor mode).
4. Sets IM to zero (interrupts disabled).
5. If the processor is in dual instruction mode, it sets DIM; otherwise it clears DIM.
6. If the processor is in single-instruction mode and the next instruction will be executed in dual-instruction mode or if the processor is in dual-instruction mode and the next instruction will be executed in single-instruction mode, DS is set; otherwise, it is cleared.
7. The appropriate trap type bits in **psr** and **epsr** are set (IT, IN, IAT, DAT, FT, OF, IL, PI, PT, BEF, PEF). Several bits may be set if the corresponding trap conditions occur simultaneously.
8. An address is placed in the fault instruction register (**fir**) to help locate the trapped instruction. In single-instruction mode, the address in **fir** is the address of the trapped instruction itself. In dual-instruction mode, the address in **fir** is that of the floating-point half of the dual instruction. If an instruction or data access fault occurred, the associated core instruction is the high-order half of the dual instruction (**fir** + 4). In dual-instruction mode, when a data access fault occurs in the absence of other trap conditions, the floating-point half of the dual instruction will already have been executed (except in the case of the **fxfr** instruction).

The processor begins executing the trap handler by transferring execution to virtual address 0xFFFFFF00. The trap handler begins execution in single-instruction mode. The trap handler must examine the trap-type bits in **psr** (IT, IN, IAT, DAT, FT) and **epsr** (OF, IL, PT, PI, BEF, PEF) to determine the cause or causes of the trap.

2.8.2 INSTRUCTION FAULT

This fault is caused by any of the following conditions. In all cases the processor sets the IT bit before entering the trap handler.

1. By the **trap** instruction. When **trap** is executed in dual-instruction mode, the floating-point companion of the **trap** instruction is not executed before the trap is taken.
2. By the **intovr** instruction. The trap occurs only if OF in **epsr** is set when **intovr** is executed. To distinguish between cases 1 and 2, the trap handler must examine the instruction addressed by **fir**. The trap handler should clear OF before returning. When **intovr** causes a trap in dual-instruction mode, the floating-point companion of the **intovr** instruction is completely executed before the trap is taken.

Table 2.10. Types of Traps

Type	Indication			Caused by	
	psr	epsr	fsr	Condition	Instruction
Instruction Fault	IT	OF		Software traps	**trap** **intovr**
		IL		Missing **unlock**	Any
		PT & PI		Pipeline usage	Any scalar or pipelined instruction that uses a pipeline
Floating Point Fault	FT		SE	Floating-point source exception	Any M- or A-unit except **fmlow**
				Floating-point result exception	Any M- or A-unit except **fmlow**, **pfgt**, and **pfeq**. Reported on any F-P instruction, **pst, fst,** and sometimes **fld, pfld,** and **ixfr**
			AO, MO	overflow	
			AU, MU	underflow	
			AI, MI	inexact result	
Instruction Access Fault	IAT			Address translation exception during instruction fetch	Any
Data Access Fault	DAT			Load/store address translation exception	Any load/store
				Misaligned operand address	Any load/store
				Operand address matches **db** register	Any load/store
Parity Error Fault	IN	PEF		Parity error on data pins during bus read operation when PEN# pin active	
Bus Error Fault	IN	BEF		External interrupt signal on BERR pin	
Interrupt	IN	INT		External interrupt signal on INT pin	
Reset	None	PEF, BEF		Hardware RESET signal	

3. By violation of lock/unlock protocol, explained below. (Note that **trap** and **intovr** should not be used within a locked sequence; otherwise, it would be difficult to distinguish between this and the prior cases.)
4. By execution of an instruction that uses a pipeline when the PT bit of **epsr** is set. (Refer to section 2.8.2.2.)

2.8.2.1 Lock Protocol

The lock protocol requires the following sequence of activities:

1. **lock**
2. Any load or store instruction. For compatibility with future processor generations, this should be a load.
3. **unlock**
4. Any load or store instruction. For compatibility with future processor generations, this should be a store.

There may be other instructions between any of these steps. The bus is locked after step 2, and remains locked until step 4. Step 4 must follow step 1 by 30 instructions or less; otherwise, an instruction trap occurs. In case of a trap, IL is also set. If the load or store instruction of step 2 accesses a previously unaccessed page (A=0), the bus is locked briefly while the A bit is set, unlocked, then locked again to satisfy the **lock** instruction and start the locked sequence.

2.8.2.2 Using PT and PI Bits

The PI and PT bits are provided to help the trap handler avoid unnecessarily saving and restoring the pipelines (refer to the section "Pipeline Preemption" in the *i860 Microprocessor Family Programmer's Reference Manual*).

Trap handlers that use PI or PT must initially examine **fsr**. If a pending trap exists—that is, if the FTE (floating-point trap enable) bit is set and any of the

floating-point exception bits (AI, AO, AU, MI, MO, MU) is active—the trap handler must save the pipelines. The i860 XP microprocessor, like the i860 XR microprocessor, may set an **fsr** exception bit before the floating-point trap is generated, and this pending trap relies on information in the pipeline. For example, an external interrupt might invoke the trap handler between the scalar floating-point instruction that produces an overflow and the next floating-point operation—the one that would cause a branch to the trap handler for the floating-point trap.

If no pending trap exists, the handler can follow either of the following two methods:

- **Using both PT and PI:** Upon invocation, the trap handler saves the state of PI and PT (in **epsr**), but does not save the pipes. If PI is found set (which means that the interrupted code needs the state information currently in the floating-point pipelines), the handler sets PT and clears PI (with a single **st.c** to **epsr** instruction), then continues with trap processing. If the pipes are used during trap handling (even by a scalar instruction), a trap will be generated with IT and PI set by hardware. The trap handler may then check PI and PT, and if both are set, clear PT, PI, and IT, save the pipes, set an indication that they were saved, and restart execution from the instruction that caused the trap. At the end of trap handling, the trap handler restores the pipes if they were saved, and restores PI and PT to their values before the trap. This method avoids both saving and restoring the pipes, assuming that most trap handling sequences do not alter the pipes, and therefore a trap for PT = 1 will not happen very often.
- **Using only PI:** Another approach is to leave PT = 0, using only the PI bit, which the processor sets each time a pipelined instruction or **pfld** is executed. The trap handler saves PI, saves the pipes if PI is set, sets an indication that they were saved, and clears PI. At the end of trap handling, the trap handler restores the pipes if they were saved, and restores PI to its value before the trap. With this method, the pipes are sometimes saved and restored unnecessarily if the trap handler code does not use the pipes. This method is advised when it is known that the trap handler uses the pipes.

2.8.3 FLOATING-POINT FAULT

The floating-point fault is reported on floating-point instructions, **pst**, **fst**, and sometimes **fld**, **pfld**, and **ixfr**. The floating-point faults of the i860 XP microprocessor support the floating-point exceptions defined by the IEEE standard as well as some other useful classes of exceptions. The i860 XP microprocessor divides these into two classes: source exceptions and result exceptions. The numerics library supplied by Intel provides the IEEE standard default handling for all these exceptions.

2.8.3.1 Source Exception Faults

All exceptional operands, including infinities, denormalized numbers and NaNs, cause a floating-point fault and set SE in the **fsr**. Source exceptions are reported on the instruction that initiates the operation. For pipelined operations, the pipeline is not advanced.

SE is undefined for faults on **fld**, **pfld**, **fst**, **pst**, and **ixfr** instructions under these conditions:

- In single-instruction mode, always.
- In dual-instruction mode, when the companion instruction is not a multiplier or adder operation.

2.8.3.2 Result Exception Faults

The result exceptions include:

- Overflow. The absolute value of the rounded true result would exceed the largest positive finite number in the destination format.
- Underflow (when FZ is clear). The absolute value of the rounded true result would be smaller than the smallest positive finite number in the destination format.
- Inexact result (when TI is set). The result is not exactly representable in the destination format. For example, the fraction 1/3 cannot be precisely represented in binary form. This exception occurs frequently and indicates that some (generally acceptable) accuracy has been lost.

The point at which a result exception is reported depends upon whether pipelined operations are being used:

- **Scalar (nonpipelined) operations.** Result exceptions are reported on the next floating-point, **fst.x**, or **pst.x** (and sometimes **fld**, **pfld**, **ixfr**) instruction after the scalar operation. When a trap occurs, the last-stage of the affected unit contains the result of the scalar operation.
- **Pipelined operations.** Result exceptions are reported when the result is in the last stage and the next floating-point (and sometimes **fld**, **pfld**, **ixfr**) instruction is executed. When a trap occurs, the pipeline is not advanced, and the last-stage results (that caused the trap) remain unchanged.

When no trap occurs (either because FTE is clear or because no exception occurred), the pipeline is ad-

vanced normally by the new floating-point operation. The result-status bits of the affected unit are undefined until the point that result exceptions are reported. At this point, the last-stage result-status bits (bits 29..22 and 16..9 of the **fsr**) reflect the values in the last stages of both the adder and multiplier. For example, if the last-stage result in the multiplier has overflowed and a **pfadd** is started, a trap occurs and MO is set.

For scalar operations, the RR bits of **fsr** report in which register the result was stored. RR is updated when the scalar instruction is initiated. The result exception trap, however, occurs on a subsequent instruction. Programmers must prevent intervening stores to **fsr** from modifying the RR bits. Prevention may take one of the following forms:

- Before any store to **fsr** when a result exception may be pending, execute a dummy floating-point operation to trigger the result-exception trap.
- Always read from **fsr** before storing to it, and mask updates so that the RR bits are not changed.

For pipelined operations, RR is cleared; the result is in the last stage of the pipeline of the appropriate unit. The trap handler must flush the pipeline, saving the results and the status bits.

In either pipelined or scalar mode, the trap handler must compute the result to be returned. In either case, the result delivered by the CPU has the same significand as the true result and has an exponent that is the low-order bits of the true result. The trap handler can inspect the delivered result, compute the result appropriate for that instruction (a NaN or an infinity, for example), and store the computed result. If RR is nonzero, the trap handler must store the computed result in the register specified by RR; if RR is zero, it must load the last stage of the pipeline with the computed result instead of the saved result.

Result exceptions may be reported for both the adder and multiplier at the same time. In this case, the trap handler should fix up the last stage of both pipelines.

2.8.4 INSTRUCTION ACCESS FAULT

This trap occurs during address translation for instruction fetches in any of these cases:

- The address fetched is in a page whose P (present) bit in the page table is clear (not present).
- The address fetched is in a supervisor mode page, but the processor is in user mode.
- The address fetched is in a page whose PTE has A = 0, and the access occurs during a locked sequence (i.e. between **lock** and **unlock**).

Note that several instructions are fetched at one time, either due to instruction prefetching or to instruction caching. Therefore, a trap handler can change from supervisor to user mode and continue to execute instructions fetched from a supervisor page. An instruction access trap occurs only when the next group of instructions is fetched from a supervisor page (up to eight instructions later). If, in the meantime, the handler branches to a user page, no instruction access trap occurs. No protection violation results, because the processor does not permit data accesses to supervisor pages while running in user mode.

2.8.5 DATA ACCESS FAULT

This trap results from an abnormal condition detected during data operand fetch or store. Such an exception can be due only to one of the following causes:

- An attempt is being made to write to a page whose D (dirty) bit is clear.
- A memory operand is misaligned (is not located at an address that is a multiple of the length of the data).
- The address stored in the debug register is equal to one of the addresses spanned by the operand.
- The operand is in a not-present page.
- An attempt is being made from user level to write to a read-only page or to access a supervisor-level page.
- The operand is in a page whose PTE has A = 0, and the access occurs during a locked sequence (i.e. between **lock** and **unlock**).
- Write protection (determined by **epsr** bit WP = 1) is violated in supervisor mode.

When a data access trap is taken on a pipelined floating-point instruction that occurs immediately after the load or store instruction that causes the trap, the destination register of the pipelined floating-point instruction may be partially updated. Correct execution will occur when the trap handler resumes execution after handling the DAT, because the pipelined floating-point instruction will then correctly update its destination register.

2.8.6 PARITY ERROR TRAP

If the PEN# pin is active and the bus unit detects a parity error during a bus read operation, the processor sets PEF and IN, then generates a trap. Further parity error traps are masked as soon as PEF is set. To reenable such traps, software must clear PEF and unfreeze BEAR by executing **ld.c bear,** *rdest*.

The interrupted program is not restartable. BS (bus or parity error trap in supervisor mode) is set by the i860 XP microprocessor when a parity error occurs while the processor is in supervisor mode. The operating system can use this bit to decide, for example, whether to abort the process (user mode) or reboot the system (supervisor mode).

2.8.7 BUS ERROR TRAP

When external hardware asserts the BERR pin, the processor sets BEF (bus error flag) and IN (interrupt), and then traps. Further BERR traps are masked as soon as BEF is set by hardware. To reenable such traps, software must clear BEF and unfreeze BEAR by executing **ld.c bear,** *rdest*.

The interrupted program is not restartable. BS (bus or parity error trap in supervisor mode) is set by the i860 XP microprocessor when a bus error occurs while the processor is in supervisor mode. The operating system can use this bit to decide, for example, whether to abort the process (user mode) or reboot the system (supervisor mode).

2.8.8 INTERRUPT TRAP

An interrupt is an event that is signaled from an external source. If the processor is executing with interrupts enabled (IM set in the **psr**), the processor sets the interrupt bit IN in the **psr** and INT in the **epsr**, then generates an interrupt trap.

Vectored interrupts are implemented by interrupt controllers and software. Software can use the **ldint** instruction to generate an interrupt acknowledge (INTA) cycle. This instruction generates a bus cycle with INTA cycle specifications, and places the data returned from the bus to the destination register. Tags are not checked in the data cache for hit, and the cycle is not burstable.

The Intel 486 microprocessor generates two INTA cycles as a response to an interrupt and inserts four idle clocks in between. To generate an interrupt acknowledge sequence that is compatible with the Intel 486 microprocessor, the **ldint** instruction sequence documented in section 5.1.4 should be executed.

2.8.9 RESET TRAP

When the i860 XP microprocessor is reset, execution begins in single-instruction mode at virtual address 0xFFFFFF00. This is the same address as for other traps. The reset trap can be distinguished from other traps by the fact that no trap bits are set. The instruction cache is flushed. The bits DPS, BL, and ATE in **dirbase** are cleared. CS8 is initialized by the value at the INT pin at the end of reset. The read-only fields of the **epsr** are set to identify the processor, while the IL, WP, and PBM bits are cleared. The bits U, IM, BR, and BW in **psr** are cleared, as are the trap bits FT, DAT, IAT, IN, and IT. All other bits of **psr** and all other register contents are **undefined**. Refer to Table 2.11 for a summary of these initial settings.

The software must ensure that the control registers are properly initialized before performing operations that depend on the values of those registers.

Reset code must initialize the floating-point pipeline state to zero with floating-point traps disabled to ensure that no spurious floating-point traps are generated.

After a RESET the i860 XP microprocessor starts execution at supervisor level (U = 0). Before branching to the first user-level instruction, the RESET trap handler or subsequent initialization code has to set PU and a trap bit so that an indirect branch instruction will copy PU to U, thereby changing to user level.

2.9 Debugging

The i860 XP microprocessor supports debugging with both data and instruction breakpoints. The features of the i860 XP microprocessor architecture that support debugging include:

- **db** (data breakpoint register), which permits specification of a data address that the i860 XP microprocessor will monitor.
- BR (break read) and BW (break write) bits of the **psr**, which enable trapping of either reads or writes (respectively) to the address in **db**.
- DAT (data access trap) bit of the **psr**, which allows the trap handler to determine when a data breakpoint was the cause of the trap.
- **trap** instruction that can be used to set breakpoints in code. Any number of code breakpoints can be set. The values of the *isrc1* and *isrc2* fields help identify which breakpoint has occurred.
- IT (instruction trap) bit of the **psr**, which allows the trap handler to determine when a **trap** instruction was the cause of the trap.

Table 2.11. Register and Cache Values after Reset

Registers	Initial Vaule
Integer Registers	*Undefined*
Floating-Point Registers	*Undefined*
psr	U, IM, BR, BW, FT, DAT, IAT, IN, IT = 0; others are *undefined*
epsr	IL, WP, PBM, BE, PT = 0; BEF, PEF = 1; Processor Type, Stepping Number, DCS, SO are read only; others are *undefined*
db	*Undefined*
dirbase	DPS, BL, LB, ATE = 0; others are *undefined*
fir	*Undefined*
fsr	*Undefined*
bear	*Undefined*
p3–p0	*Undefined*
ccr	CO, DO = 0; others are *undefined*
KR, KI, T, MERGE	*Undefined*
NEWCURR	*Undefined*
STATUS	InLoop, Nested, Detached = 0
Caches	**Initial Value**
Instruction Cache	All entries invalid
Data Cache	All entries invalid
TLB	All entries invalid

3.0 ON-CHIP CACHES

By holding data, instructions, and address translation on-chip, the caches of the i860 XP microprocessor provide the following advantages:

1. Low chip count for the CPU subsystem.
2. Wide processor-to-cache path: 16 bytes for data, 8 bytes for instructions.
3. Fast access without requiring much additional high-speed design in the system. The fast (50 MHz) cache-access circuitry is hidden on chip; the external bus can respond more slowly without significantly degrading performance.

3.1 Address Translation Caches

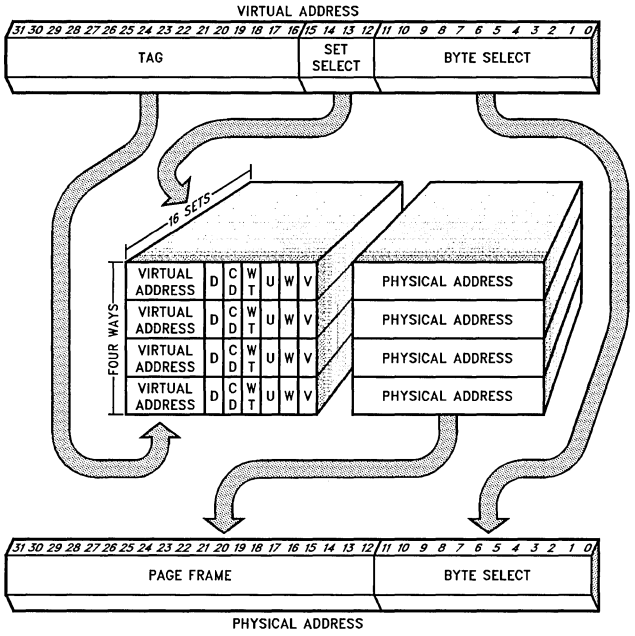
The i860 XP microprocessor allows both four Kbyte and four Mbyte page sizes, and a separate translation look-aside buffer (TLB) is used to cache address translation information for each page size. The TLB for four-Kbyte pages (Figure 3.1) has 64 entries, and the TLB for four-Mbyte pages (Figure 3.2) has 16 entries. Both are four-way set associative. The TLBs function when paging is enabled. When a page is first accessed, its translation information is saved in the appropriate TLB along with other page attributes, such as access rights and cacheability. Every address translation operation looks up the virtual address simultaneously in both TLBs. Only if the necessary paging information is not in either of the caches must the paging tables in memory be referenced. Both TLBs employ a random replacement algorithm to choose which of the four ways to replace.

If an instruction's virtual address is found in the instruction cache, the virtual address is not translated, and code access rights are not verified. However, when an instruction's virtual address is not found in the cache, address translation does occur, and all access rights are verified. The virtual addresses of data are always translated, and access rights are always verified.

The i860 XP microprocessor requires simultaneous access to data and instruction caches, but the TLBs can service only one address translation at a time. Data address translation has higher priority in the TLBs than instruction address translation, if both are required at the same time.

Any data or instruction access fault halts address translation at once, and the TLB is not updated. If a directory read causes an access fault, the page table is not read at all.

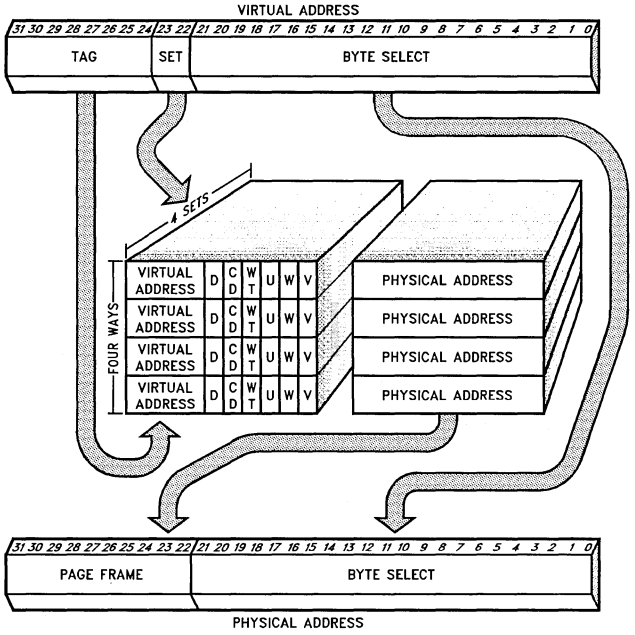
If the paging unit generates a fault (in setting the D bit for the first write to a nondirty page, for example), the corresponding entry is deleted from the TLB. Therefore, software does not need to invalidate the TLB entry in response to DAT or IAT faults.



NOTES:
D Dirty
CD Cache Disable
WT Write-Through
U User Mode
W Writable
V Validity

240874-22

Figure 3.1. 4K TLB Organization



NOTES:
D Dirty
CD Cache Disable
WT Write-Through
U User Mode
W Writable
V Validity

240874-23

Figure 3.2. 4M TLB Organization

If TLB replacement is initiated during a locked sequence generated by the **lock** instruction and if another locked sequence has to be executed to set the A-bit, the paging unit generates an access fault. This helps external hardware implement "locking by address" by preventing generation of nested lock sequences.

3.2 Internal Instruction and Data Caches

The i860 XP microprocessor has separate data and instruction caches on-chip. Having separate caches for instructions and data allows simultaneous cache look-up. Up to two instructions and 128 bits of data can be accessed simultaneously from these caches. The data and instruction caches hold 16 Kbytes each. A line can be filled from memory with a four-transfer burst.

The caches are fully transparent to applications software. Snooping (address monitoring) is designed into both instruction and data caches, to maintain cache consistency in multiprocessor systems.

Each cache has two sets of tags: *virtual* tags used for internal access, and *physical* tags used for snooping. Figure 3.3 shows how the bits of both virtual and physical addresses are mapped for caching. The presence of both virtual and physical tags supports *aliasing*, a situation in which the TLBs associate a single physical address with two or more virtual addresses.

Any area of memory can be cached, although both software and hardware can disallow certain areas from being cached—software by setting the CD bit in their page table entries; hardware by deasserting the KEN# signal for bus cycles with addresses that fall in those areas. (Data reads from the two four-Kbyte pages pointed to by the CCUBASE field of **ccr** are not cached (and the CACHE# signal is inactive), if the DCCU is activated by setting CO of the **ccr** register. This is independent of the value of KEN#.) When both software and hardware agree that a requested datum is cacheable, the i860 XP microprocessor fetches an entire 32-byte line and places it into the appropriate cache. Cache line fills are generated only for read misses, not for write misses. A store that misses the cache does not copy the missed line into cache from memory, but rather posts the datum in a write buffer, then sends it to the external bus when the bus is available.

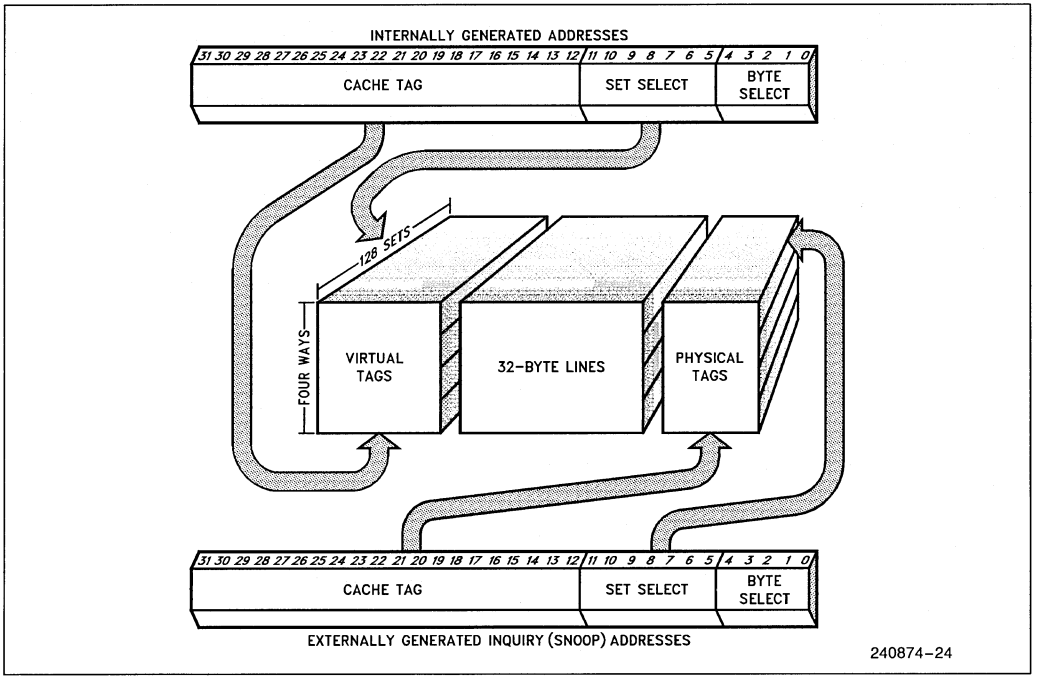


Figure 3.3. Cache Address Usage

3.2.1 DATA CACHE

Figure 3.4 shows the organization of the data cache. The data cache has two status bits per physical tag and one validity status bit for the virtual tag. A virtual tag hit is possible only when the validity bit of the virtual tag is set and the state of the physical tag is M, E, or S.

Aliasing support is built into the cache look-up algorithm. Even though a physical line may be aliased, the processor never enters the line twice in the data cache. If a virtual address is not found among the virtual tags in the data cache, a bus cycle is initiated (except a read is not issued at this time if the bus pipeline is full) and, at the same time, the physical tags are searched for the physical address (which by this time has been retrieved from the paging unit). For reads, if the physical address is found, the data returned from the bus is ignored, on-chip data is used, and the virtual tag is replaced with the new one. For writes, if a virtual address is not found, the write is issued on the bus and memory is updated. If the physical address is found, the line in cache is updated, and the virtual tag is replaced with the new one. However, the cache state (M, E, or S) of the physical-address tag does not change when the virtual tag is overwritten.

Note that the BE (big endian) bit of **epsr** has no influence on data cache behavior. Data items are kept in cache in exactly the same ordering as in external memory. Byte-shifting operations invoked by the BE bit upon loads and stores occur at the input to the register files only.

3.2.1.1 Data Cache Update Policies

To minimize bus traffic, a *write-back* policy is normally used. The write-back policy (also called *copy-back* and *deferred-write*) reduces bus traffic by eliminating many unnecessary writes. Writes to a line in the cache are not immediately forwarded to main memory; instead, they are accumulated in the cache. The modified cache line is written to main memory only when its cache space is needed for other data, when the modified data is needed by another processor, or when a flush procedure is executed.

Under the write-back policy, a write that hits the cache utilizes it for two cycles (one to check the virtual tags for hit, another to update the cache line). However, the cache pipeline allows successive store hits to operate at one per cycle. The processor's internal write buffers can hold two successive stores, preventing a freeze upon store miss.

Under a *write-through* policy, a write request to a line in the cache triggers updates to both cache and main memory. An address decoder, for example, can select the write-through policy for writes to video RAM, where it is necessary that writes be seen on the video display. Software, by setting the WT page-table bit, can select the write-through policy for specific areas of memory—those that are used for interprocessor message queues, for example.

A *write-once* policy combines write-through with write-back. Write-through is employed for the first write to a cache line, while subsequent writes to the same line follow the write-back policy. Write-once is valuable in multiprocessor systems to maintain cache consistency with the least possible bus traffic. The first write broadcasts to other processor nodes the fact that a line has been modified. Write-once is also used if a second-level cache is attached to the i860 XP microprocessor to maintain consistency between the first- and second-level caches.

The external system can dynamically change the update policy (write-back, write-through, write-once) of the i860 XP microprocessor with each cache line.

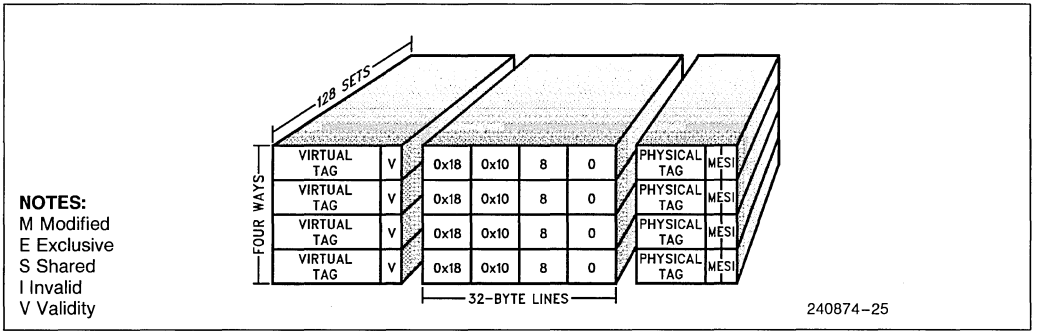


Figure 3.4. Data Cache Organization

3.2.2 INSTRUCTION CACHE

Figure 3.5 shows the organization of the instruction cache. The instruction cache has one validity bit that is common to both virtual and physical tags. Aliasing support for instructions consists not simply of changing the virtual tag, but rather fetching a line whenever a virtual tag miss occurs. If the physical address already exists in the instruction cache, its line and its tags are overwritten. So, even though a physical line may be aliased, the processor never enters the line twice in the instruction cache.

3.2.3 CACHE REPLACEMENT ALGORITHM

The data, instruction, and address-translation caches all use similar algorithms to choose which of the four cache blocks will be overwritten when a miss causes a line fetch.

First, the first invalid line (if any) in a set of four is replaced (in the order 0, 1, 2, 3). When there are no more invalid lines in a set, a pseudorandom replacement algorithm chooses which valid lines to replace. The algorithm is controlled by counters inside the chip. RESET initializes these counters to zero, so that the "randomness" is deterministic and two i860 XP CPUs executing the same code on identical boards have exactly the same series of cache hits, misses, and replacements.

Setting ITI to invalidate the caches and TLBs also resets the counters used to select the set used for cache line replacement. This brings the i860 XP microprocessor cache-replacement mechanism to a known state without resetting the whole chip.

When the **flush** instruction is used to write back modified lines in the data cache, the flush routine must alter the RC (replacement control) field of **dirbase**. Therefore, replacement is not random. Instead, the block (or "way") replaced is the one selected by the RB (replacement block) field of **dirbase**.

3.2.4 CACHE CONSISTENCY PROTOCOL

The i860™ XP Microprocessor implements cache consistency via its use of a MESI (Modified, Exclusive, Shared, Invalid) protocol.

3.2.4.1 Data Cache States

Each line of the data cache of the i860 XP microprocessor can be in one of the states defined in Table 3.1. Note that the instruction cache of the i860 XP only implements the "SI" part of the MESI protocol, because the instruction cache is not writable.

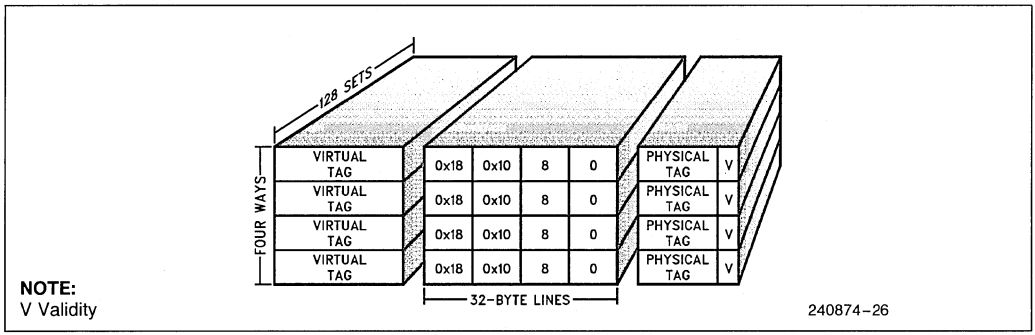


Figure 3.5. Instruction Cache Organization

Table 3.1. MESI Cache Line States

Cache Line State:	M Modified	E Exclusive	S Shared	I Invalid
This cache line is valid?	Yes	Yes	Yes	No
The memory copy is . . .	. . . out of date	. . . valid	. . . valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line . . .	. . . does not go to bus	. . . does not go to bus	. . . goes to bus and updates the cache	. . . goes directly to bus

Table 3.2. Internally Initiated Cache State Transitions

State	Next State after Read	Next State after Write*
I	If WB/WT# = 1; E; else S Line fill	Write-through I
S	S	Write-through If WB/WT# = 1, E; else S
E	E	M
M	M	M

NOTE:
* "Write" does not include write-backs due to replacement. Those can only cause an M to I transition.

The state of a cache line can change as the result of either internal or external activity related to that line. Table 3.2 presents the line state transitions that result from internal activity of the i860 XP microprocessor in the data cache.

External cache-consistency support is provided through *inquiry cycles*. Inquiry cycles are initiated by other processors in a multiprocessor system to check whether an address is cached in the internal cache of the i860 XP microprocessor. Table 3.3 shows the line state transitions initiated by inquiry cycles.

Table 3.3. Inquiry-Initiated Cache State Transitions

State	INV = 0	INV = 1
I	I	I
S	S	I
E	S	I
M	S; write back the line	I; write back the line

3.2.4.2 Write-Once Policy

A write-once cache policy can be implemented through use of the WB/WT# input pin. The signal on this pin is sampled in both read and write cycles. A read miss causes a line to enter either S or E after the line fill. If WB/WT# is sampled LOW at the time of NA# or the first BRDY# activation, the line enters S state, forcing the next write hit to this line to show up on the bus. If WB/WT# is sampled HIGH, the line enters E state. In write-through cycles, the state of a line is changed from S to E when WB/WT# is sampled HIGH, so that subsequent writes will not be written through to the bus. Thus, if this signal is driven LOW on read cycles and HIGH on write cycles, a write-once cache policy is implemented. The easiest way to implement write-once (in systems not using the 82495XP cache controller) is to tie this pin to the W/R# output of the processor.

If the WT bit in the page table entry is set, the i860 XP microprocessor ignores the WB/WT# signal for the cycles that hit that page and always performs a write-through. In other words, hardware cannot override software's selection of the write-through policy.

3.2.4.3 Locked Access

Locked accesses are those data loads and stores that occur after a **lock** instruction up to and including the first load or store after the corresponding **unlock** instruction.

State transitions for locked accesses differ from those in Table 3.2 in ways that guarantee that locked accesses are seen by all processors in the system. Any locked load or store generates both a cache look-up and an external bus cycle, regardless of cache hit or miss.

1. In a locked read:
   a. If the required data is not found in the cache, the data from the bus is used. The data is placed in the cache if it is cacheable and KEN# is also asserted.
   b. If the required data is found in an unmodified (E or S) state, the data from the bus is used.
   c. If the data is found in the cache in a modified (M) state, the cached data is used, and the bus data is ignored, as long as no inquiry write-back occurs before the BRDY# of the bus cycle. If, however, an intervening inquiry write-back changes the line to S or I state, the bus data is used.
2. A locked store is forced through the cache and issued on the bus. No more data accesses occur until the last BRDY# for the store. If the store hits the internal cache, the cache update is done after the last BRDY# from the bus. Note that the line written by a locked store remains in M state in spite of the write-through to the bus, because the length of the write-through is less than the line size of 32 bytes.

Locked accesses are totally *serializing* in the sense that:

1. All loads and stores that precede the **lock** instruction are issued on the bus (if they miss the cache) before the first locked access is issued. The locked access can be issued before the last BRDY# of the prior cycle if NA# is activated in response to the prior cycle.
2. No load or store after the last locked access is issued internally or on the bus until the final BRDY# for all locked accesses.

To maximize performance, instruction fetches during the locked sequence are *not* serializing. When NA# invokes pipelining, instruction fetches may be issued while locked data fetches or stores remain on the bus.

3.3 Internal Cache Consistency

Both the instruction and the data caches can be snooped by externally generated inquiry cycles, and the result of the look-up is presented on the HIT# and HITM# output pins. These inquiry cycles help maintain consistency with caches of other processors. However, software must take care not to create inconsistencies such as the following among the internal caches (including the TLBs):

1. Changing the address space while leaving virtual-address tags from the prior space in the instruction or data cache.
2. Changing instructions in memory (or in the data cache) without changing them in the instruction cache.
3. Changing page table information in memory (or in the data cache) without changing the same information in the TLBs.

Under certain circumstances, such as I/O references, self-modifying code, page-table updates, or shared data in a multiprocessing system, it is necessary to bypass, to invalidate, or to flush the caches. The i860 XP microprocessor provides the following methods for doing this:

- **Bypassing Instruction and Data Caches.**
  1. If deasserted during cache-miss processing, the KEN# pin disables instruction and data caching of the referenced data.
  2. If the CD bit of the associated page table is set, caching of a page is disabled. The value of the CD bit is output on the PCD pin for use by external caches.
  3. If the WT bit of the associated page table is set, caching is not disabled, but writes pass through the cache. The value of the WT bit is output on the PWT pin for use by external caches. (Note that WT does not affect policy for the instruction cache, because the instruction cache is not writable. However, when an instruction from a page having the WT bit of the PTE set is placed in the data cache, the write-through policy applies just as for a data page.)
- **Invalidating Cache Entries.** Storing to the **dirbase** register with the ITI bit set invalidates each line of the instruction and address-translation caches. In the data cache, it invalidates the virtual tags, but not the physical tags.
- **Flushing the Data Cache.** The data cache is flushed by a software routine that uses the **flush** instruction. The **flush** instruction speeds up write-backs. The same effect (writing back modified lines) can be achieved with the load instruction **ld.l**, but this would be more than twice as slow—the load must first do four bus transfers to get new data, then write back the modified line. The **flush** instruction causes the write-backs without requiring a read from external memory to replace the modified line.

3.3.1 ADDRESS SPACE CONSISTENCY

In a multitasking virtual-address system, the operating system may intentionally employ aliasing, where several processes use the same physical memory while accessing it with different virtual addresses. When the operating system switches control from one process to the next, it changes the DTB field of the **dirbase** to point to a different page directory that defines the new address space. When this happens, all caches must be invalidated: the TLBs, so that the new page directory is read into the TLBs; the data and instruction caches, so that virtual addresses from the new space don't accidently match cached virtual addresses from the old space.

The caches are invalidated by setting the ITI bit when writing to **dirbase**. Invalidating the instruction cache invalidates both the physical and the virtual tags, because the instruction cache has one status (valid) bit, which is common to both physical and virtual tags. In the data cache, setting ITI does not invalidate physical tags. However, any modified lines will eventually be written back when their space is required for lines from the new address space or when external agents on the bus express a need for the modified data via inquiry cycles.

The caches are invalidated by setting the ITI bit when writing to **dirbase**. Note, however, that the operating system code that flushes the caches must be present during the flushing. Typically this code has the same virtual address for all processes.

NOTE:
The mapping of the page(s) containing the currently executing instruction, the next six instructions, and any data referenced by these instructions should not be different in the new page tables when the DTB is changed.

Enabling or disabling address translation (via the ATE bit) is similar to changing the DTB, in that the address mapping is changed. The virtual tags in the data and instruction cache must be invalidated prior to changing ATE.

3.3.2 INSTRUCTION CACHE CONSISTENCY

When software modifies a page containing instructions (as when a debugger replaces an instruction with the **trap** instruction to set a breakpoint), the instruction cache can become inconsistent for any of the following reasons:

- Because the data cache uses a write-back policy, changes to cached instruction pages do not immediately update memory.
- Changes to instructions do not automatically update the instruction cache.
- Instruction cache misses are not checked in the data cache.

Software must ensure that modified lines containing instructions are written to main memory before the instruction cache tries to read them. There are two methods for this:

1. Flush the data cache using the **flush** instruction. Note that to make the instruction cache consistent with the data cache, the data cache must be flushed *before* invalidating the instruction cache.
2. Mark all instruction pages as WT (write through) so that modifications to instructions are immediately written to memory. This is the better alternative.

In either case, the instruction cache must be invalidated (by a store to **dirbase** with ITI set) after a code page has been modified, so that the updated instructions will be read from memory.

3.3.3 PAGE TABLE CONSISTENCY

When the operating system modifies page tables or directories, the TLBs can become inconsistent with the modifications for any of the following reasons:

- Because the data cache uses a write-back policy, updates to cached page tables do not immediately update memory.
- Changes to page tables do not automatically update the TLB.
- The i860 XP microprocessor searches only external memory for page directories and page tables in the translation process. The data cache is not searched. (Data is not transferred from the data cache to the TLBs during TLB replacement cycles.)

Software must ensure that modified lines containing page table entries are written to main memory before the paging unit tries to read them. There are two methods for this:

1. Keep page tables and directories in noncacheable memory or write-through pages.
2. Flush the data cache using the **flush** instruction.

The processor itself invalidates the affected TLB entry, when a trap is triggered by the need to set the A or D bit. In other cases, after a page table or directory has been modified, software must invalidate the TLBs (by a store to **dirbase** with ITI set) so that the updated entries will be read from memory.

The data cache does not need flushing if the program is modifying only the P, U, W, A, or D bits of a PTE (as long as the page frame address is not changed and the PTE itself is not in the data cache.) The i860 XP CPU does not use the TLB for cache line write-backs; it writes to the address in the physical tag.

Thus, a trap handler can service a data access trap for D-bit zero merely by setting D = 1. When setting the P or A bits, there is no need to invalidate or flush any caches, because the processor does not load entries into the TLB that have P = 0 or A = 0.

Two potential TLB inconsistencies are avoided automatically by the i860 XP microprocessor.

1. If the paging unit issues a write cycle (to set the A bit, for example), this cycle is snooped by the data cache for invalidation.
2. Any TLB entry that causes a DAT or IAT is automatically invalidated.

3.3.4 CONSISTENCY OF CACHEABILITY

Normally, an operating system ensures that the page attributes (CD and WT) of a memory access are consistent with the cache contents. However, the operating system can fail to maintain consistency by the following actions:

- Changing the CD or WT bits while related lines are in the cache.
- Aliasing a physical address with virtual addresses that have differing CD or WT bits.

In these situations, the i860 XP microprocessor gives priority to cache state. For example:

1. If a read or write request is to a noncacheable page (CD = 1), but the data (or code) is found in cache, the request is satisfied by the cache, and no external cycle is issued.
2. If the physical address of a read or write request hits in the cache but the virtual address misses, the virtual tag is overwritten by the new virtual address, but the CD bit of the new virtual address is ignored.
3. If a store to a write-through page (WT = 1) hits a cache line in E or M state, no write-through cycle is issued; only the cache is updated.

3.3.5 LOAD PIPE CONSISTENCY

The **pfld** (pipelined floating-point load) instruction facilitates transfer of data from memory to registers, and avoids placing data in the data cache. When large amounts of data are used, **pfld** allows the programmer to keep rarely-used data out of the cache. The i860 XP microprocessor ensures consistency between cached data and **pfld** references. It checks the data cache and, upon a data cache hit to a modified line, forwards data from cache into the three-stage **pfld** pipeline.

3.3.6 SUMMARY

Table 3.4 summarizes flush and invalidation requirements, assuming that WT is set in the PTEs of instruction and page-table pages:

Table 3.4. Summary of Cache Flushing And Invalidation

Action	Flush Data Cache	Invalidate Caches (ITI)
Setting A	No	No
Setting P	No	No
Clearing P	No	Yes
Setting D	No	No
Changing protection (U,W)	No	Yes
Setting CD or WT	Yes	Yes
Changing PFA in a used(1) PTE	No	Yes
Changing **dirbase** DTB	No	Yes
Changing **dirbase** ATE	No	Yes
Changing **epsr** WP	No	No
Setting **ccr** DO and CO	Yes(2)	Yes(2)
Modifying code	No(3)	Yes

NOTES:
1. "Used" means a PTE that at some past time had P set.
2. If data from either of the CCU pages could have been cached.
3. Assuming all instructions and their page directories and page tables are in write-through or noncacheable pages.

4.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

4.1 Pins Overview

Figure 4.1 identifies functional groupings of the pins. Table 4.1 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output pins are tristate, except BREQ, HIT#, HITM#, HLDA, LOCK#, and PCHK#.

Table 4.1. Pin Summary

Pin ID	Name	Active Level	When Floated Synch/Asynch	Internal Resistor
Output Pins				
ADS#	Address Status	LOW	HLDA, clock after BOFF#	
BE7#–BE0#	Byte Enable	LOW	HLDA, BOFF#	
BREQ	Bus Request	HIGH		
CACHE#	Cache	LOW	HLDA, BOFF#	
CTYP	Cycle Type	HIGH	HLDA, BOFF#	
D/C#	Data/Code		HLDA, BOFF#	
HIT#	Snoop Hit Cache	LOW		
HITM#	Snoop Hit Modified Line	LOW		
HLDA	Hold Acknowledge	HIGH		
KB0,KB1	Cache Block	HIGH	HLDA, BOFF#	
LEN	Length	HIGH	HLDA, BOFF#	
LOCK#	Address Lock	LOW		
M/IO#	Memory/IO		HLDA, BOFF#	
NENE#	Next Near	LOW	HLDA, BOFF#	
PCD	Page Cache Disable	HIGH	HLDA, BOFF#	
PCHK#	Parity Check	LOW		
PCYC	Page Cycle	HIGH	HLDA, BOFF#	
PWT	Page Write-Through	HIGH	HLDA, BOFF#	
TDO	Test Output		Nonscan Mode	
W/R#	Write/Read		HLDA, BOFF#	
Input/Output Pins				
A31–A3	Address	HIGH	AHOLD, HLDA, BOFF#	
D63–D0	Data	HIGH	HLDA, BOFF#	
DP7–DP0	Data Parity	HIGH	HLDA, BOFF#	
Input Pins				
AHOLD	Address Hold	HIGH	Synch	
BERR	Bus Error	HIGH	Synch	
BOFF#	Back-Off	LOW	Synch	
RSRVD	*Intel Reserved*			
BRDY#	Burst Ready	LOW	Synch	
BYPASS#	*Intel Reserved*	LOW		
CLK	Clock			
RESET	Reset	HIGH	Asynch	
EADS#	External Address Status	LOW	Synch	
EWBE#	External Write Buffer Empty	LOW	Synch	
FLINE#	Flush Line	LOW	Synch	
HOLD	Bus Hold	HIGH	Synch	
INT/CS8	Interrupt/Code-Size 8	HIGH	Asynch	
INV	Invalidate	HIGH	Synch	
KEN#	Cache Enable	LOW	Synch	
NA#	Next Address	LOW	Synch	
PEN#	Parity Enable	LOW	Synch	
TCK	Test Clock			
TDI	Test Data Input		Synch	Pull-up
TMS	Test Mode Select		Synch	Pull-up
TRST#	Test Reset	LOW	Asynch	Pull-up
WB/WT#	Write-Back/Write-Through		Synch	
SPARE	*Intel Reserved*			

The pins D/C#, W/R#, and M/IO# define bus cycle types. They are summarized in Table 4.2. For data transfers to or from memory, two additional pins, CTYP and PCYC, provide further information regarding the type of transfer, as shown in Table 4.3. Table 4.4 shows how the LEN and CACHE# pins determine cycle length.

Table 4.2. ADS# Initiated Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Table 4.3. Memory Data Transfer Cycle Types

PCYC	CTYP	W/R#	Data Transfer Type
0	0	0	Normal read
0	1	0	Pipelined load (**pfld** instruction)
1	0	0	Page directory read
1	1	0	Page table read
0	0	1	Write-through (S-state hit)
0	1	1	Store miss or write-back
1	0	1	Page directory update
1	1	1	Page table update

NOTE:
PCYC and CTYP are defined only for memory data transfer cycles (D/C# = 1, M/IO# = 1)

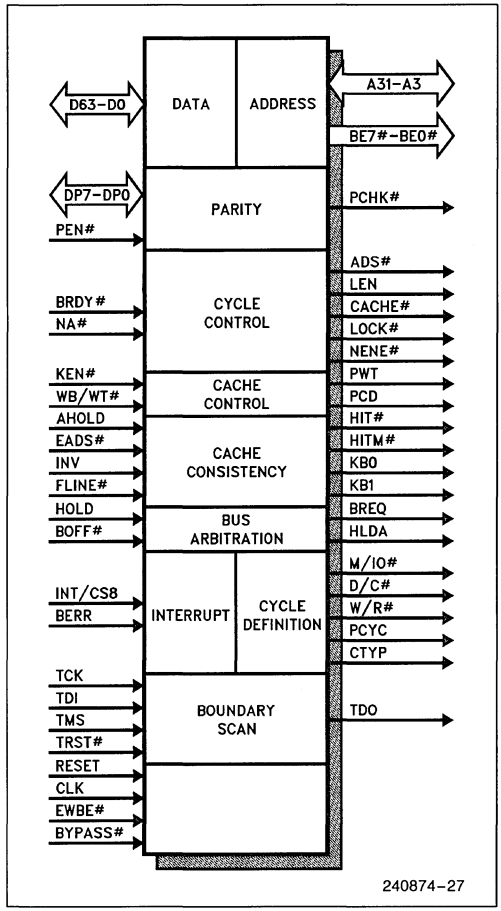


Figure 4.1. Signal Grouping

Table 4.4. Cycle Length Definition

W/R#	LEN	CACHE#	KEN#	Cycle Description	Burst Length
0	0	1	—	Noncacheable** 64-bit (or less) read	1
0	0	—	1	Noncacheable 64-bit (or less) read	1
1	0	1	—	64-bit (or less) write	1
—	0	1	—	I/O and Special Cycles	1
0	1	1	—	Noncacheable 128-bit read **(p)fld.q**	2
0	1	—	1	Noncacheable 128-bit read **(p)fld.q**	2
1	1	1	—	128-bit write **fst.q**	2
0	—	0	0	Cache line fill	4
1	—	0	—	Cache write-back	4

NOTE:
** Includes CS8-mode code fetches, which may be cached by the processor.
—Indicates "don't care" values.

4.2 Signal Description

In this section descriptions of all pins are presented in alphabetical order.

4.2.1 A31–A3 (ADDRESS PINS)

The 29-bit address bus (A31–A3) identifies addresses to a 64-bit location. Separate byte-enable signals (BE7#–BE0#) identify which bytes should be accessed within the 64-bit location.

The address lines are bidirectional. The i860 XP microprocessor drives the address lines unless it is in a hold state. The system drives address lines A31–A5 to perform cache line inquiries (refer to the EADS# signal description).

4.2.2 ADS# (ADDRESS STATUS)

The i860 XP microprocessor asserts ADS# to identify the first clock period of each bus cycle, the clock period during which new values become valid on the address bus and cycle-definition pins. This signal is held active for one clock.

If BOFF# is asserted, the processor floats ADS# two clocks after sampling BOFF# (and not, like all other pins, on the next clock). This is to ensure that ADS# is deasserted before it floats, and therefore is never left floating active.

ADS# can be asserted while AHOLD is active to initiate a cache write-back cycle.

4.2.3 AHOLD (ADDRESS HOLD)

The external system asserts AHOLD to perform a cache inquiry. In response to assertion of AHOLD, the i860 XP microprocessor immediately (in the next clock) stops driving the address bus (A31–A3 lines). The other buses remain active, and data can be transferred for previously issued read or write bus cycles during address hold. AHOLD is recognized even during RESET and LOCK#. The earliest that AHOLD can be deasserted is the clock after EADS# is asserted to start the inquiry.

If HITM# has activated due to an inquiry, the i860 XP microprocessor asserts ADS# while AHOLD is active to start the write-back of the modified line that was the target of the inquiry.

4.2.4 BE7#–BE0# (BYTE ENABLES)

The byte-enable pins are driven with the address. BE7# applies to D63–D56, BE0# applies to D7–D0.

In write cycles (noncacheable writes as well as cache line write-backs), the BE*n*# signals determine which bytes must be written into external memory for the current cycle.

In read cycles, the BEn# values indicate which byte the load instruction has requested. In all noncacheable read cycles (CACHE# or KEN# deasserted), the byte enables match the length and address of the requested data. Cacheable read cycles (KEN# asserted), however, result in four 64-bit memory transfers to fill an entire 32-byte cache line. The BE*n*# pins activated are those that represent the operand of the load instruction that caused the line fill, and these same BE*n*# pins remain activated for as long as A31–A5. All 64 bits must be returned for each cacheable cycle without regard for the BE*n*# signals.

While in CS8 mode, BE2#–BE0# serve as (active-high) lower-order address bits for instruction fetches (from the ROM). Data fetches and stores are not affected by CS8 mode, and BE2#–BE0# retain their normal byte-enable function for data.

4.2.5 BERR (BUS ERROR)

This is a nonmaskable interrupt input, which supports bus error handling or other urgent circumstances. BERR is not masked by the IM bit of the **psr** nor by lock cycles. When BERR is activated, the i860 XP microprocessor vectors to the trap handler and sets the bus error flag (BEF) in the **epsr**. BERR causes the physical address of the current bus cycle to be latched into the BEAR control register; thus, if asserted the clock of BRDY# or the clock after BRDY#, it causes the bus address to be latched for software to examine. BERR is rising-edge sensitive. Once the trap has occurred, further BEF traps cannot occur until software has cleared BEF and read BEAR.

BERR does not terminate outstanding bus cycles. Therefore, the system must still activate BRDY# a sufficient number of times or activate BOFF# for those cycles. Even though activating BOFF# temporarily halts the erring cycles, the i860 XP microprocessor will retry them when BOFF# is deasserted, in spite of BERR.

Timing of BERR is not influenced by late back-off mode.

4.2.6 BOFF# (BACK-OFF)

The system can assert this signal to abort all outstanding bus cycles that have not yet completed. In response to BOFF#, the i860 XP microprocessor

immediately (in the next clock) floats its bus, except for ADS#, which is floated one clock later. The processor floats all the same pins normally floated during bus hold; however, unlike a bus hold, HLDA is not asserted. (HLDA is asserted only in response to HOLD; no acknowledgment is required for BOFF#.). Any data and BRDY# returned to the processor while BOFF# is asserted are ignored. The processor remains in bus hold until BOFF# is deasserted, at which time it restarts the bus cycles by driving the address and cycle definition pins and asserting ADS#. When BOFF# deactivates, ADS# may be asserted the following clock. Thus a BOFF# duration of one clock results in not floating ADS# at all. BOFF# cannot be used to force the pins to float during RESET; use HOLD for that purpose.

4.2.7 BRDY# (BURST READY)

BRDY # indicates either that the external system has driven valid data on the data pins in response to a read request or that the external system has latched the data in response to a write request. The CPU ignores this signal when no bus requests are outstanding. During a bus cycle, BRDY # is sampled at each clock, starting with the clock after assertion of ADS # and continuing until all data for the cycle has been transferred. When BRDY # is sampled active in a read cycle, the data present on the pins is sampled.

4.2.8 BREQ (BUS REQUEST)

BREQ allows the i860 XP microprocessor to share the local bus with other bus masters. An external bus arbiter can use BREQ to implement an "on demand only" policy for granting the bus to the i860 XP microprocessor. The i860 XP microprocessor asserts BREQ the clock after it realizes an internal request for the bus. The system should sample this pin only when the i860 XP microprocessor is not in control of the bus (that is, when HLDA, BOFF#, or AHOLD is active). BREQ is undefined when the i860 XP microprocessor is driving the bus. BREQ may be deasserted between assertions of ADS#, but this does not imply that the CPU does not need the bus.

4.2.9 BYPASS# (BYPASS)

This pin is reserved by Intel Corporation and should be tied HIGH to V_{CC} through a resistor. When LOW, the phase-locked loop that generates the internal clock is unused. In this case, the internal clock has more skew relative to the external CLK, and the A.C. timing parameters are not guaranteed.

4.2.10 CACHE# (CACHEABILITY)

This output signal indicates internal cacheability of a bus request. Its timing follows that of the address bus.

The i860 XP microprocessor asserts CACHE# for cacheable reads and code fetches to announce its intention to cache the data. If CACHE# is asserted on a read cycle and if the KEN# input is active, the cycle is a burst line fill. If CACHE# is inactive in a read cycle, the i860 XP microprocessor does not cache the returned data, regardless of the KEN# pin. CACHE# is also asserted for cache line write-backs.

CACHE# is inactive for noncacheable reads (for example, **pfld, ldio, ldint**), TLB replacements, and store misses.

Table 4.4 shows how cacheability determines the number of data transfers in a cycle.

Note that the CACHE# output is always inactive for CS8 (Code-Size 8 bits) mode instruction fetches so that the instructions are fetched with single-transfer cycles. However, the code fetched may then be placed in the instruction cache, unless KEN# was inactive.

4.2.11 CLK (CLOCK)

The CLK input determines execution rate and timing of the i860 XP microprocessor. External timing parameters are specified relative to the rising edge of this signal. The i860 XP microprocessor can utilize a clock rate of 50 Mhz. The internal operating frequency is the same as the external clock. This signal requires TTL levels.

4.2.12 CTYP (CYCLE TYPE)

CTYP is one of the bus cycle definition signals. Tables 4.2 and 4.3 show the types of bus cycle generated. CTYP is defined only for data write and read requests. The value of this pin changes only when ADS# is asserted.

4.2.13 D/C# (DATA/CODE)

D/C# specifies whether the current request is for data or instructions. The data/code line is one of the bus cycle definition pins. Tables 4.2 and 4.3 show the types of bus cycle generated. The value of this pin changes only when ADS# is asserted.

4.2.14 D63–D0 (DATA PINS)

The bus interface has 64 bidirectional data pins (D63–D0) to transfer data in eight- to 64-bit quantities. Pins D7–D0 transfer the least significant byte; pins D63–D56 transfer the most significant byte. In read cycles, all 64 bits of the data bus are latched, even in CS8-mode instruction fetches when only the low-order eight bits are used. In write cycles, the i860 XP microprocessor does not drive D63–D0 in the clock of ADS#, but in the following clock.

4.2.15 DP7–DP0 (DATA PARITY)

There is one parity signal for each byte of the data bus. They are driven by the i860 XP microprocessor with even parity information on writes with the same timing as write data. Likewise, if parity checking is enabled by PEN#, the system must drive even parity information on these pins with the same timing as read information to ensure that the correct parity check status is indicated by the i860 XP microprocessor. "Even parity" means that the total number of set bits in a byte, including the parity bit, is even. Refer also to the PCHK# signal.

4.2.16 EADS# (EXTERNAL ADDRESS STATUS)

This signal indicates that a valid external address has been driven onto address pins A31–A5 of the i860 XP microprocessor to be used for a cache inquiry. This signal is recognized while the processor is in hold (HLDA is driven active), while forced off the bus with BOFF# input, or while AHOLD is asserted. The i860 XP microprocessor ignores EADS# at all other times. EADS# is not recognized if HITM# is active, nor during the clock after ADS#, nor during the clock after a valid assertion of EADS#. Table 4.5 shows when EADS is first sampled. It is then sampled in every clock as long as the hold remains active and HITM# remains inactive.

Table 4.5. EADS# Sample Time

Trigger	EADS# First Sampled
AHOLD	Second clock after AHOLD asserted
HOLD	First clock after HLDA asserted
BOFF#	Second clock after BOFF# asserted

INV and FLINE# are sampled in the same clock period that EADS# is validly asserted. HIT# and HITM# may be asserted as the results of a cache inquiry.

4.2.17 EWBE# (EXTERNAL WRITE BUFFER EMPTY)

At RESET, the value on EWBE# determines the ordering mode. The processor enters strong ordering mode if EWBE# is sampled active for at least the last three clocks before RESET deactivates; otherwise, it enters weak ordering mode.

In weak ordering mode, the value of EWBE# after reset does not affect processor operation.

in strong ordering mode, the external system asserts EWBE# as long as all external write buffers are empty. If an external write buffer is not empty (EWBE# deasserted) or the intenal write buffer is not empty, the processor delays data cache updates so as to keep the external order of writes the same as the programmed order.

In systems that do not have external write buffers, EWBE# can be tied to V_{SS}, if strong ordering is desired, or to V_{CC}, if weak ordering is acceptable. Refer to sections 5.3.3 and 5.3.4 for more explanation and for other ways to control write ordering.

4.2.18 FLINE# (FLUSH LINE)

The system asserts FLINE# to request that the i860 XP microprocessor write back a modified cache line before other outstanding bus cycles are completed, if the line is hit by an external inquiry. If this pin is active in the same clock that EADS# is asserted, the write-back cycle is initiated, and the i860 XP microprocessor expects BRDY#s for the write-back before outstanding cycles (if any) are returned. If data transfer for another cycle is currently in progress when FLINE# is asserted (i.e. first BRDY# returned before HITM# asserted), the i860 XP microprocessor waits until the data transfers for that burst have completed, and only then does it assert the ADS# for the write-back. If the first BRDY# has not yet occurred for an outstanding cycle, NA# must be activated to trigger ADS# for the write-back.

At RESET, the value on FLINE# determines configuration. The processor enters one-clock late backoff mode if FLINE# is sampled active for at least the last three clocks before RESET deactivates.

4.2.19 HIT# (CACHE INQUIRY HIT)

This pin is one output of inquiry cycles. If an inquiry cycle hits a valid line in the caches of the i860 XP microprocessor (either data or instruction), HIT# is asserted two clocks after EADS# is activated. If the inquiry cycle misses the caches, this pin is negated two clocks after EADS# activation.

This pin changes its value only as a result of EADS# activation during AHOLD, HOLD, or BOFF# and retains its value until two clocks after the next valid activation of EADS#.

HIT# can be used to control the WB/WT# pin of other processors in a multiprocessor system. Activation of HIT# indicates that the inquiring processors should cache the line as S-state, not E-state.

4.2.20 HITM# (HIT MODIFIED LINE)

This pin is an output of inquiry cycles. When an inquiry hits a modified line in the internal data cache, the i860 XP microprocessor asserts HITM# two clocks after EADS# is activated. (Refer also to the EADS# signal.) The HITM# signal stays active until the last BRDY# for the corresponding write-back cycle. At all other times, HITM# is inactive. HIT# is also asserted when HITM# is asserted (except for the special case of an inquiry after the ADS# of a write-back).

4.2.21 HLDA (BUS HOLD ACKNOWLEDGE)

The i860 XP microprocessor activates HLDA in response to a hold request presented on the HOLD pin. Assertion of HLDA indicates that the i860 XP microprocessor has given the bus to another local bus master. It is driven active in the same clock that the i860 XP microprocessor floats its bus. All output pins are floated except LOCK#, BREQ, HLDA, PCHK#, HIT#, and HITM#.

The time required to acknowledge a hold request is one clock plus the number of clocks needed to finish any outstanding bus cycles (maximum of four outstanding cycles of four burst transfers each for total of 16 transfers). If this hold latency is too long for a given application, BOFF# can be used instead.

When leaving a bus hold, the i860 XP microprocessor deactivates HLDA and, in the same clock period, initiates a pending bus cycle, if any.

4.2.22 HOLD (BUS HOLD)

This pin, along with the output signal HLDA, is used for local bus arbitration. At some time after the HOLD signal is asserted, the i860 XP microprocessor releases control of the local bus and puts most bus interface outputs in floating state, then asserts HLDA—all during the same clock period. It maintains this state until HOLD is deasserted. Instruction execution stops only if required instructions or data cannot be read from the on-chip instruction and data caches. The i860 XP microprocessor ignores HOLD until all outstanding bus cycles are complete (until the last BRDY#). The i860 XP microprocessor recognizes HOLD even during RESET and LOCK#. HOLD cannot be used when the 82495XP cache controller is attached.

4.2.23 INV (INVALIDATE)

The external system asserts this signal to invalidate the cache-line state in the case of an inquiry cycle hit. It is sampled together with A31–A5 in the clock EADS# is active.

4.2.24 INT/CS8 (INTERRUPT/CODE-SIZE EIGHT BITS)

This input, like the BERR input, allows interruption of the current instruction stream. The processor samples INT as instruction boundaries. If interrupts are enabled (IM set in **psr**) when INT is sampled active, the i860 XP microprocessor fetches the next instruction from virtual address 0xFFFFFF00. INT is level triggered. To assure that an interrupt is recognized, INT should remain asserted until the software acknowledges the interrupt (by executing an interrupt-acknowledge cycle, for example). The interrupt may be ignored by the processor if the INT signal does not remain active.

Interrupt latency (the maximum time between assertion of INT and execution of the first instruction of the trap handler) depends both on the internal context and on the external system. After INT is asserted, the i860 XP microprocessor finishes all instructions currently being executed, including any outstanding bus cycles, before starting the trap handler. The following instruction sequence is an example of the worst case:

```
pfld.q
pfld.q
ld.l
br
ld.l
st.l
```

If INT is asserted during the execution stage of the last **ld.l** instruction, the execution of the trap handler may have to wait for:

- Two 2-transfer bursts (the **pfld** instructions)
- Two data cache line fills (misses by the **ld.l** instructions)
- Two data cache line write-backs (eliminating modified lines to open space for the fills)
- Two instruction cache line fills (the target of the **br** and the first instruction of the trap handler)
- Three TLB miss sequences of up to six nonpipelined accesses each (the **br**, the last **ld.l**, and the trap handler)

The time to finish the above bus activities can be extended by inquiry cycles and associated write-backs initiated by an external cache or bus controller.

Besides the bus-related delays, the i860 XP microprocessor has internal freeze conditions that can delay interrupt response by up to 10 additional clocks.

During a locked sequence, the INT pin is ignored, and the INT bit of **epsr** reflects the value on the INT pin. To limit the time that INT is ignored, the **lock** instruction can assert LOCK# for only 30–33 instructions before trapping.

This input is asynchronous, but appropriate setup and hold times must be met to insure recognition on any specific clock.

If INT is asserted for at least the last three clock periods before the falling edge of RESET, the i860 XP microprocessor enters eight-bit code-size (CS8) mode.

4.2.25 KB0, KB1 (CACHE BLOCK)

For reads, these output signals define which cache block (line) is going to receive the data. For write-backs, these lines specify which block is being flushed. They are driven together with cycle definition for cacheable data reads, TLB replacement, code fetch cycles, and write-backs. External hardware can use these signals to observe changes to cache blocks.

4.2.26 KEN# (CACHE ENABLE)

The i860 XP microprocessor samples KEN# to determine whether the data being read for the current cache-miss cycle is to be cached. When the i860 XP microprocessor generates a read cycle that can be cached (CACHE# output active) and KEN# is active, the cycle is transformed into a burst line fill. By activating KEN#, the memory system commits to a four-transfer burst. The entire 64 bits of the data bus are used for the read, regardless of the state of the byte-enable pins.

If KEN# is sampled inactive, code fetches are not transferred in bursts, but 128-bit data items may still be transferred with a burst length of two.

KEN# is sampled together with NA# or BRDY#, whichever comes first. It is sampled only with the *first* BRDY# of a burst; its value at any other time has no effect.

4.2.27 LEN (DATA LENGTH)

The LEN output pin specifies the number of burst transfers for each cycle. This pin and the CACHE# output pin are used by the system to determine the burst length for each cycle (refer to Table 4.4). The i860 XP microprocessor can generate 1, 2, or 4-transfer bursts for reads and writes.

LEN is inactive if the internal request is for 64 bits or less. If LEN is active, the internal request is for 128 bits or more, and the cycle should be returned as a two- or four-transfer burst. LEN is always active for 128-bit data accesses. LEN is always inactive for code accesses.

A cacheable read (CACHE# active) can be automatically converted to a four-transfer burst regardless of LEN by assertion of KEN#.

Table 4.4 summarizes different cycle lengths as they are calculated from the LEN and CACHE# signals. LEN has the same timing as the address.

4.2.28 LOCK# (ADDRESS LOCK)

This signal is used to provide atomic (indivisible) read-modify-write sequences in multiprocessor systems. The address to be locked is the one being driven on A31–A3 when LOCK# is activated. A multiprocessor bus arbiter must permit only one processor a locked read, locked write, or unlocked write to that address and must maintain the lock of that location across cycle boundaries until LOCK# deactivates. The simplest arbitration hardware can just lock the entire bus against all other accesses during LOCK# assertion; however, software must never assume that this implemenïation is being used.

The i860 XP microprocessor coordinates the external LOCK# signal with the **lock** and **unlock** instructions. Programmers do not have to be concerned about the fact that bus activity is not always synchronous with instruction execution. LOCK# is asserted with ADS# for the address operand of the first load or store instruction executed after the **lock** instruction.

After an **unlock** instruction, LOCK# is deasserted with the next load or store. The i860 XP microprocessor deactivates LOCK# one clock after ADS# for the last locked bus cycle. Unlike the i860 XR microprocessor, the i860 XP microprocessor does not deassert LOCK# immediately when a trap occurs. Instead, the trap handler must execute a load or store instruction to deassert LOCK#. (The handler does not have to execute an **unlock** instruction, however. The unlocking function is performed by the processor's trap logic.)

The i860 XP microprocessor also asserts LOCK# during TLB miss processing for updates of the accessed bit in page-directory and page-table entries. The maximum time that LOCK# can be asserted in this case is the time required to perform a nonpipelined, four-byte, read-modify-write sequence.

Between locked sequences, at least one cycle of no LOCK# is guaranteed by the behavior of the **unlock** instruction.

Between **lock** and **unlock** instructions, the INT pin is ignored.

Instruction fetches do not alter the LOCK# signal.

4.2.29 M/IO# (MEMORY-I/O)

M/IO# specifies whether the current cycle is for the memory address space or for the I/O address space. M/IO# is one of the bus cycle definition pins. Tables 4.2 and 4.3 show the types of bus cycle generated. The value of this pin changes only when ADS# is asserted.

4.2.30 NA# (NEXT ADDRESS REQUEST)

NA# makes address pipelining possible. The system asserts NA# for at least one clock to indicate that it is ready to accept the next address from the i860 XP microprocessor. (If the system does not implement pipelining, NA# must not be activated.) The i860 XP microprocessor samples NA# every clock, starting one clock after the activation of ADS#. If the i860 XP microprocessor has a new cycle pending internally when NA# is activated, it initiates that cycle in the clock after NA# is asserted. Up to three bus cycles can be outstanding simultaneously.

NA# is latched internally; the i860 XP microprocessor remembers that NA# was asserted until it has an internal request to send to the bus; so, assertion of NA# for a single clock can trigger an ADS# several clocks later. NA# is ignored in the clock of ADS#.

KEN# and WB/WT# inputs for the current cycle are sampled with NA#, if NA# is asserted before the first BRDY# of the current cycle.

NA# is also used in conjunction with FLINE# to invoke write-back of a modified line during outstanding bus cycles.

4.2.31 NENE# (NEXT NEAR)

The i860 XP microprocessor asserts NENE# when the current address is in the same DRAM page as the previous bus cycle. This signal allows higher-speed reads and writes in the case of consecutive accesses to static column or page-mode DRAMs. The i860 XP microprocessor determines the DRAM page size by inspecting the software-controlled DPS field in the **dirbase** register. The page size can range from 2^9 to 2^{16} 64-bit words, supporting DRAM sizes from 256K $\times$ 1 to 4G $\times$ *n*. The value of this pin changes only when ADS# is asserted. NENE# is never asserted for the next bus cycle after the address bus has been floating (after AHOLD, BOFF#, or HLDA is deasserted).

4.2.32 PCD (PAGE CACHE DISABLE)

PCD provides a cacheability indication on a page by page basis. This signal, together with PWT, is set to an attribute bit in the page table entry for the current cycle. When paging is enabled, PCD corresponds to the CD bit (bit 4) of the page table entry. The i860 XP microprocessor does not perform a cache fill to any page for which CD of the page table entry is set. When paging is disabled, or for any cycle that is not paged (**ldio, stio, ldint, scyc**), the i860 XP microprocessor drives PCD inactive.

During TLB miss processing, PCD is inactive while the address translation hardware is accessing the first level page directory. During accesses to the second-level page-table entry, PCD reflects the CD values taken from the first level page-table entry.

The value of this pin changes only when ADS# is asserted.

4.2.33 PCHK# (PARITY CHECK)

This output shows the result of the parity check on data pins in the previous clock of a read cycle. It is

asserted for one clock when incorrect parity has been detected. It reflects the parity status for the entire data bus. It should be ignored by external hardware except during the clock after BRDY# is activated for reads.

PCHK# does not terminate outstanding bus cycles, so the system must still activate BRDY# a sufficient number of times or activate BOFF# for those cycles.

4.2.34 PCYC (PAGE CYCLE)

The page cycle line is active during memory read or write cycles to distinguish page-table accesses from other accesses. The types of bus cycle generated are indicated in Tables 4.2 and 4.3. The value of this pin changes only when ADS# is asserted.

4.2.35 PEN# (PARITY ENABLE)

The i860 XP microprocessor samples this signal for read cycles on the same clock edge at which BRDY# is found asserted. If sampled active, the i860 XP microprocessor feeds the parity check result into the interrupt logic. If a parity error is encountered, the i860 XP microprocessor vectors to the trap handler. The BEAR register latches the offending address, as described with the BERR signal. This interrupt is not masked by the IM bit of the PSR, nor is it masked during lock cycles.

The system should deassert PEN# any time the DP7–DP0 pins are known not to reflect the parity of the full eight-byte bus (for example, reads from I/O devices or ROMs that are not parity protected).

4.2.36 PWT (PAGE WRITE-THROUGH)

PWT provides a write-back/write-through indication on a page by page basis. This signal, together with PCD, is set to an attribute bit in the page table entry for the current cycle. When paging is enabled, PWT corresponds to the WT bit (bit 3), and write-back caching is implemented for this page only if WT is clear. When paging is disabled, or for any cycle that is not paged (**ldio, stio, ldint, scyc**), the i860 XP microprocessor drives PWT inactive.

During TLB miss processing, PWT is inactive while the address translation hardware is accessing the first level page directory. During accesses to the second-level page-table entry, PWT reflects the WT value taken from the first level page-table entry.

The value of this pin changes only when ADS# is asserted.

4.2.37 RESET (SYSTEM RESET)

Asserting RESET for at least ten CLK periods causes initialization of the i860 XP microprocessor. On power up, RESET should remain active at least one millisecond after V_{CC} and CLK have reached their proper DC and AC specs. RESET is synchronous with CLK.

4.2.38 RSRVD, SPARE

The RSRVD input is reserved by Intel. Tie it high. The spare input should be left unconnected.

4.2.39 TCK (TEST CLOCK)

This is the clock input for the TAP (test access port). If the TAP is to be used, this signal must be connected to a clock synchronous to CLK. If the TAP is not used, TCK can be tied low. TCK does not need to be kept running when boundary scan is not active.

The rising edge of TCK must be externally synchronized to CLK. The boundary scan latches retain their state when TCK is stopped at either logic zero or one.

4.2.40 TDI (TEST DATA INPUT)

TDI is the input for test instructions and data to the TAP. TDI is sampled on the rising edge of TCK. It is provided with an internal pull-up resistor, so that an open circuit at TDI produces a result equivalent to driving continuous HIGH signals.

4.2.41 TDO (TEST DATA OUTPUT)

This is the serial output of the TAP. The contents of TAP registers are shifted out through TDO on the falling edge of TCK. The data is moved from TDI to TDO without inversion, which allows easy serial cascading of different components for scanning.

TDO is held in high-impedance state, except while scanning is in progress. This allows parallel connection of these outputs for several components.

4.2.42 TMS (TEST MODE SELECT)

This input is decoded by the TAP to select the operation of the TAP. It is sampled at the rising edge of TCK. It is provided with an internal pull-up resistor to assure deterministic behavior for open-circuit failure at this pin. If boundary scan is not used, TMS can be tied high or left unconnected.

4.2.43 TRST# (TEST RESET)

This input resets the TAP. If the TAP is not used, TRST# should be tied LOW. To ensure deterministic behavior of the test logic, TMS should be held HIGH while TRST# changes from LOW to HIGH.

4.2.44 V_{CC} (SYSTEM POWER) AND V_{SS} (GROUND)

The i860 XP microprocessor has 54 pins for power and 56 for ground. All pins must be connected to the appropriate low-inductance power and ground signals in the system.

4.2.45 V_{CC}CLK (CLOCK POWER)

This is the power supply for the internal CLK buffer. It should be connected to the same V_{CC} plane as the other V_{CC} pins.

4.2.46 WB/WT# (WRITE-BACK/WRITE-THROUGH)

This input signal defines cache policy for the line being accessed in the current bus cycle. The processor samples WB/WT# for both reads and writes on the same clock edge at which it finds NA# or the first BRDY# asserted, whichever comes first. If this signal is sampled low, the write-through policy is applied to the cache line—if an internal write hits this line, it causes a write-through cycle. If this signal is sampled high, the write-back policy is applied—future write hits to this line do not show up on the bus.

4.2.47 W/R# (WRITE/READ)

This pin specifies whether a bus cycle is a read (LOW) or write (HIGH) cycle. Tables 4.2 and 4.3 show the types of bus cycle generated. The value of this pin changes only when ADS# is asserted.

5.0 BUS OPERATION

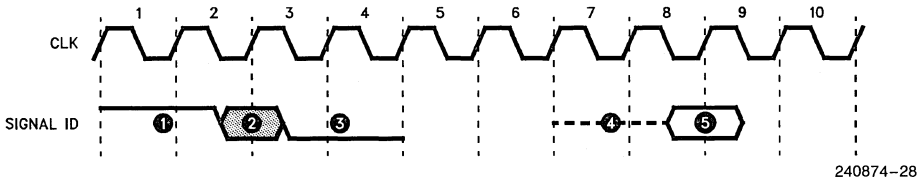
The interaction among signals is illustrated by timing diagrams. Figure 5.1 shows the conventions used in the timing diagrams.

5.1 Bus Cycles

A bus cycle begins when the i860 XP microprocessor activates ADS# and ends when the system activates the last of a predetermined number of BRDY# signals. Figure 4.4 shows how the i860 XP microprocessor and the external system cooperate to determine the number of BRDY# activations in each cycle. The processor starts sampling BRDY# one clock after assertion of ADS# and continues sampling in every clock until the last BRDY# becomes active.

The i860 XP microprocessor supports several different types of bus cycle. These are introduced in order of complexity:

1. Single-transfer cycles
2. Multiple-transfer (burst) cycles
3. Pipelined cycles
4. Cache inquiry cycles



NOTES:
1. HIGH (high voltage)
2. Don't care or undefined
3. LOW (low voltage)
4. High-impedance (floating)
5. Either HIGH or LOW

Figure 5.1. Timing Diagram Conventions

5.1.1 SINGLE-TRANSFER CYCLE

The simplest bus cycle is the single-transfer, non-cacheable, 64-bit cycle either with or without wait states. The shortest bus cycle is two clock periods long. Read and write cycles of this type are shown in Figure 5.2.

A *wait state* is any clock in which the i860 XP microprocessor samples BRDY# but the system does not assert it. The system can add wait states to any cycle. Figure 5.3 shows cycles with two wait states added. Any number of wait states can be added to i860 XP microprocessor bus cycles by maintaining BRDY# inactive.

5.1.2 BURST CYCLES

When a bus request requires more than a single data transfer (refer to Table 4.4), the i860 XP microprocessor requires that the memory system perform a burst data transfer. Burst cycles allow the maximum bus transfer rate by eliminating unnecessary driving of the address bus. The addresses of the data items in burst cycles all fall within the same 32-byte aligned area (corresponding to an internal i860 XP microprocessor cache line). Given the address of the first transfer, external hardware can calculate the addresses of subsequent transfers. With these addresses eliminated from the bus, a new data item can be sampled into the i860 XP microprocessor every clock period.

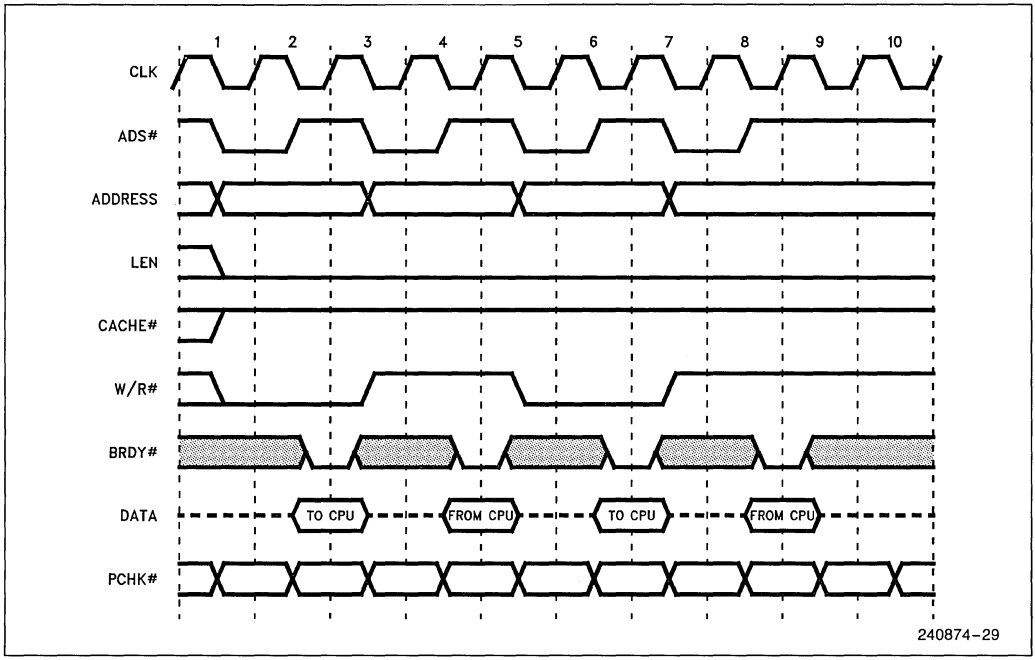


Figure 5.2. Fastest Single-Transfer Cycles

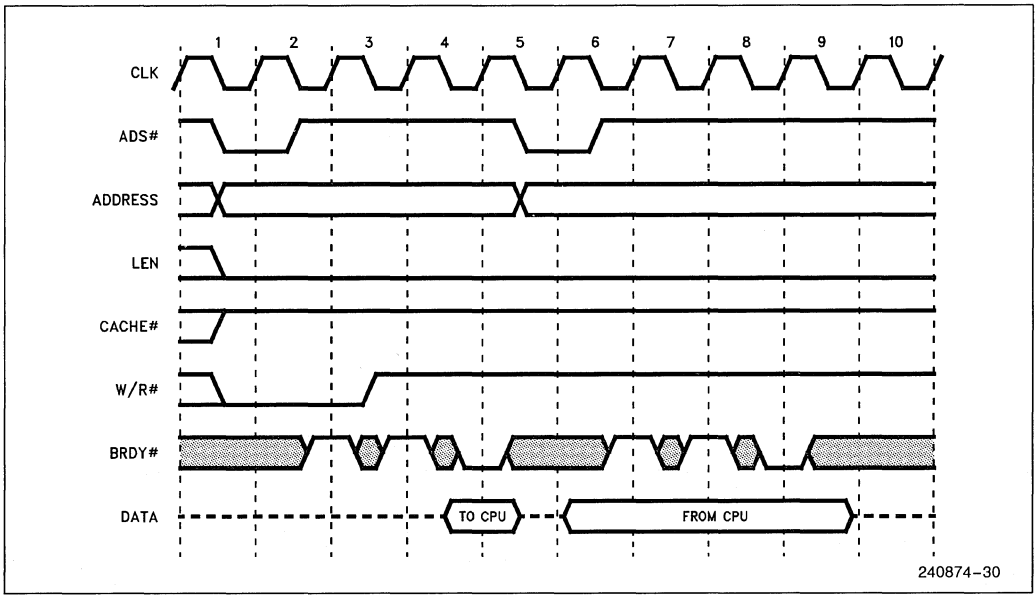
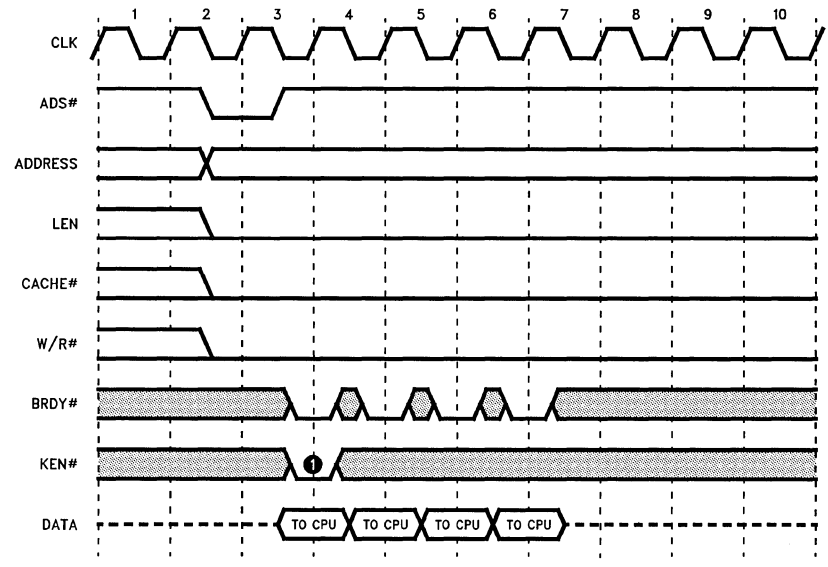


Figure 5.3. Single-Transfer Cycles with Wait States

The fastest possible burst cycle requires two clock periods for the first data item: one clock for ADS# and one clock for BRDY#; subsequent data items are transferred every clock period. One such bus cycle is shown in Figure 5.4. Note that, in this case, the initial cycle generated by the i860 XP microprocessor could be satisfied by a single data transfer, but the system transforms it into a multiple-transfer cache line fill by activating KEN# in the clock period of the first BRDY#. KEN# has this effect only if the CACHE# pin is active, which means the cycle is internally cacheable in the i860 XP microprocessor.

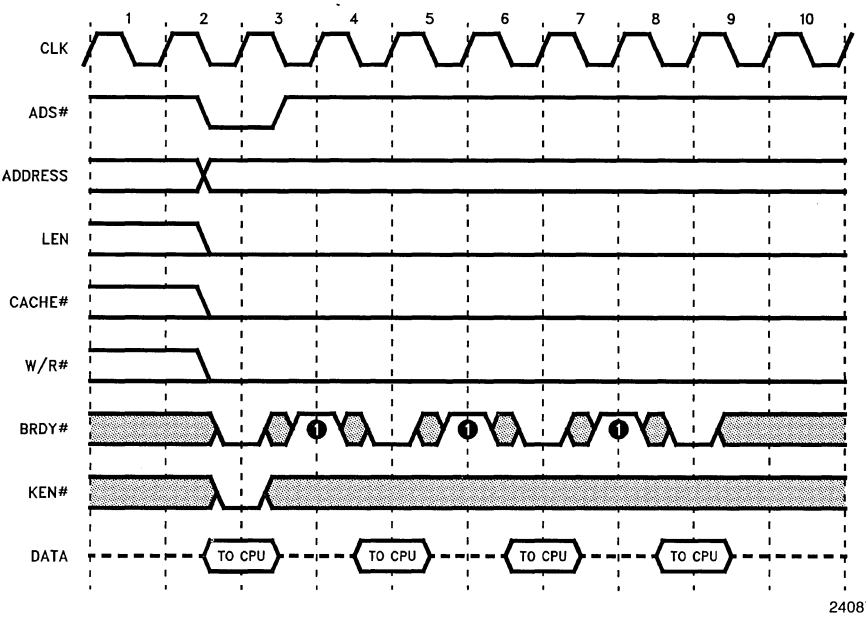
Read data is sampled only in the clock period in which BRDY# is returned, which means that data need not be sent to the i860 XP microprocessor every clock period in the burst cycle. Figure 5.5 shows an example of a burst cycle in which two clock periods are required for every burst item.

The burst length attributes LEN and CACHE# are driven with the address. Figure 5.6 illustrates two consecutive burst cycles with differing length attributes: the first one is a noncacheable 128-bit read, and the second one is a cache line fill initiated by a cacheable 64-bit read.



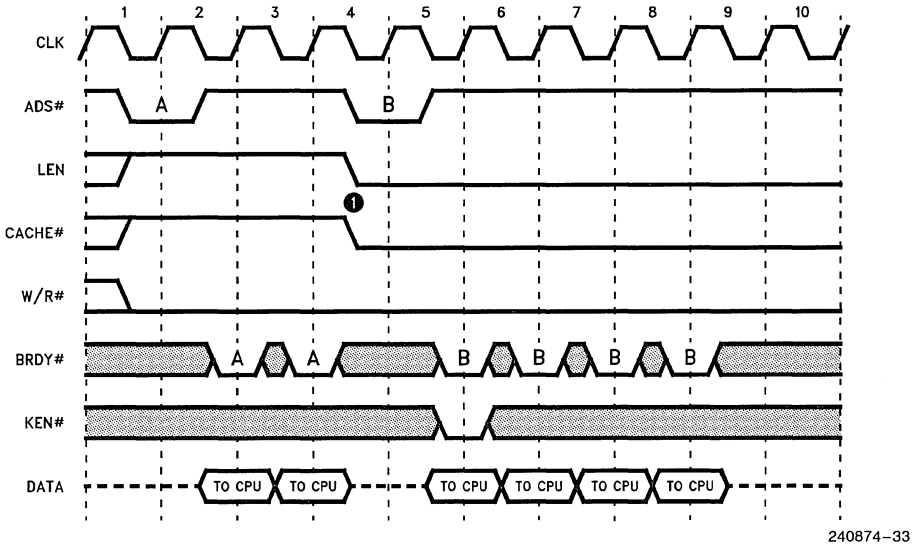
NOTE:
1. KEN# driven with first assertion of BRDY#

Figure 5.4. Basic Burst Cycle



NOTE:
1. Wait states added by delaying assertion of BRDY#

Figure 5.5. Slow Burst Cycle



NOTE:
1. Length attributes driven with ADS#

Figure 5.6. Different Lengths of Burst Cycles

The timing of write bursts is similar to that of read bursts. The i860 XP microprocessor does not put data on D63–D0 for writes until the clock period after ADS#.

When initiating any read, the i860 XP microprocessor presents the address for the data item requested. When the cycle is converted into a cache fill, the first data item returned corresponds to the address sent out by the i860 XP microprocessor. The remaining items must be returned in the order shown in Table 5.1. This ordering is optimized for two-bank memories, but works equally well with noninterleaved memories.

In i860 XP microprocessor systems, memory must support the burst order as defined in Table 5.1 for reads. For writes, the burst addresses are always increasing, so writes with four transfers match the first line of the table. In CS8 (code-size 8 bits) mode, instructions are not fetched in bursts.

Note that the i860 XP microprocessor drives only the first address of a burst cycle; the memory system is responsible for calculating subsequent addresses as shown in the table. The addresses can be derived by complementing A3 after every transfer, and complementing A4 after two transfers.

Table 5.1. Burst Order for Cache Line Transfers

1st Address	2nd Address	3rd Address	4th Address
0	8	0x10	0x18
8	0	0x18	0x10
0x10	0x18	0	8
0x18	0x10	8	0

5.1.3 PIPELINED CYCLES

A *pipelined* cycle is one that starts while one or two other bus cycles are outstanding. A cycle is considered *outstanding* until the last BRDY# is asserted to terminate that cycle. A *nonpipelined* cycle is one that starts when no other bus cycles are outstanding. Both types of cycle can be either read or write cycles. To allow high transfer rates in large memory systems, the i860 XP microprocessor supports two-level pipelining. New cycles can start as often as every other clock until three cycles are outstanding.

The system asserts NA# to indicate that the i860 XP microprocessor can start another cycle before the current one is completed. (NA# can even

be asserted while BRDY# is active.) The i860 XP microprocessor begins sampling NA# in the next clock after ADS# is asserted. If the following conditions are met, a new (pipelined) cycle begins:

1. NA# having been active
2. An internal request pending
3. Compatibility between the pending request and the outstanding requests (refer to Table 5.2)
4. HOLD, BOFF#, and AHOLD not active
5. Fewer than three cycles outstanding

The following "compatibility" rules determine when the processor does not issue a pipelined ADS# (they are the source of Table 5.2):

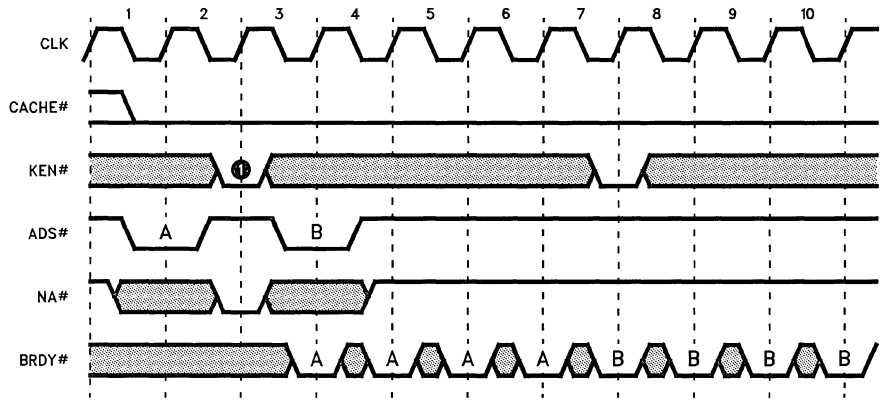
- Data cache line fills are pipelined into each other only in the case of an aliasing virtual tag miss with a physical tag hit.
- Reads can be pipelined into TLB miss writes. TLB misses for instructions can be pipelined into data accesses, and *vice versa*.
- No data cycle is ever pipelined while LOCK# is active.
- I/O cycles, special cycles, and **Idint** cycles never begin when any cycle is outstanding.

NA# may be asserted before, simultaneously with, or after the first BRDY# of the current cycle. If NA# is asserted before the first BRDY#, the cacheability (KEN#) and cache policy (WB/WT#) indicators for the current cycle are sampled during the same clock period as NA# is sampled active; otherwise, they are sampled with the first BRDY#. Figure 5.7 shows an example of four-transfer, pipelined, back-to-back reads. Note the timing of KEN#. Because NA# is asserted before the first BRDY# of the cycle A, KEN# is sampled with the NA# for cycle B.

Table 5.2. Pipeline Cycle Compatibility

	B	If A is Outstanding, can B be pipelined?								
	A	Data Cache Line Fill	Data Cache Store Miss, Write-Thru	Data Cache Read Miss KEN# = 1	Write-Back**	Instruction Fetch	pfld	TLB Miss	Idio, stio, Idint, scyc	LOCK# Active
PREVIOUS CYCLE	Data Cache Line Fill	YES*	YES*	YES*	YES	YES	YES*	YES	NO	YES
	Data Cache Store Miss, Write-Thru	YES	YES	YES	YES	YES	YES	YES	NO	YES
	Data Cache Read Miss KEN# = 1	YES	YES	YES	YES	YES	YES	YES	NO	YES
	Write-Back	YES	YES	YES	YES	YES	YES	YES	NO	YES
	Instruction Fetch	YES	YES	YES	YES	YES	YES	YES	NO	YES
	pfld	YES	YES	YES	YES	YES	YES	YES	NO	YES
	TLB Miss	YES	YES	YES	YES	YES	YES	YES	NO	YES
	Idio, stio, Idint, scyc	YES	YES	YES	YES	YES	YES	YES	NO	YES
	LOCK# Active	NO	NO	NO	NO	YES	NO	YES	NO	NO

NOTE:
* Pipelining can occur if the first ADS# is for an aliasing virtual tag miss with a physical tag hit.
**Inquiry write-backs are not pipelined into prior cycle unless FLINE# is asserted.



NOTES:
A Four-transfer, cache line fill cycle
B Four-transfer, cache line fill cycle
1. KEN# for A simultaneous with NA#

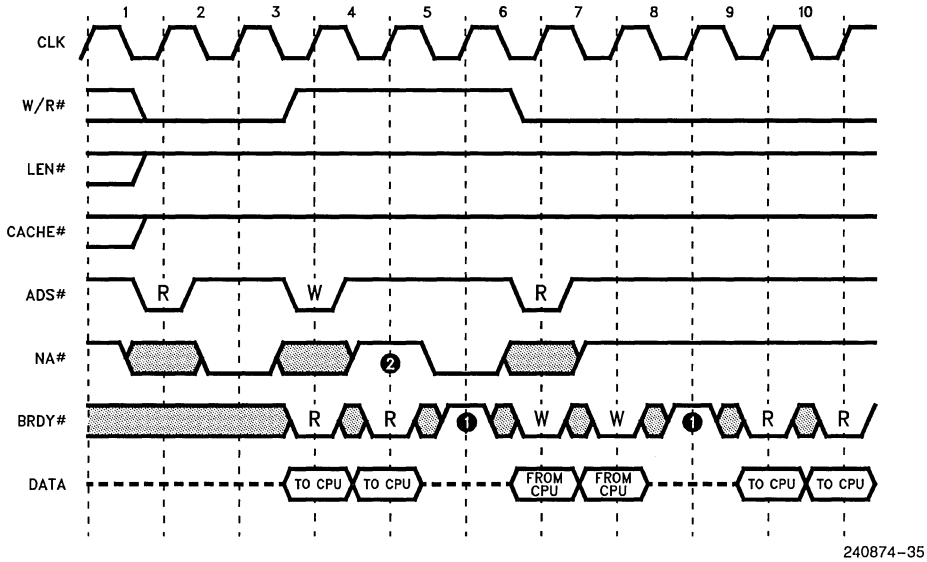
Figure 5.7. Pipelined Cache Line Fills

Write cycles can be pipelined into read cycles and *vice versa*, but, in both cases, one clock period should be left idle between bursts to allow bus turnover. Pipelined back-to-back read and write cycles are shown in Figure 5.8. On writes, assertion of NA# does not cause the values on the data bus to change; it just enables new address and cycle specification outputs.

5.1.4 INTERRUPT ACKNOWLEDGE CYCLES

In response to a trap caused by assertion of the INT pin, trap-handling software can generate interrupt acknowledge cycles by executing a procedure similar to the following.

```
//The following lock instruction must be on a 32-byte boundary:
  lock                          // Lock the bus
  ldint.b src2,  rdest          // First INTA cycle. Src2 contains 8.
  or      rdest, r0, rdest      // Won't proceed until rdest loaded.
  unlock                        // Unlock the bus after the next ldint
//nop                           // Insert 4 + <number of NOPs> idle
//nop                           //   clocks for 8259A recovery.
  ldint.b r0,    rdest          // Second INTA cycle
```



NOTES:
R Two-transfer, noncacheable read cycle
W Two-transfer, noncacheable write cycle
1. Idle clock for bus turnaround
2. Second assertion of NA# could be here

Figure 5.8. Pipelined Back-to-Back Read and Write Cycles

Figure 5.9 shows the interrupt acknowledge cycles generated by the code sequence. Interrupt acknowledge cycles are generated in locked pairs. The interrupt vector is returned during the second cycle. Each of the interrupt acknowledge cycles is terminated when the external system responds by asserting BRDY#. Wait states can be added by withholding BRDY#. There must be a number of idle clocks between the first and second cycles to allow for 8259A recovery time. The software controls the number of intervening clocks via the number of **nop** instructions in the interrupt acknowledge routine.

5.1.5 SPECIAL BUS CYCLES

The i860 XP microprocessor provides a special cycle to indicate to the external system that certain internal conditions have occurred. The special bus cycle (indicated by M/IO# = 0, D/C# = 0, and W/R# = 1) is generated by the i860 XP microprocessor as a response to **scyc** instruction execution. This cycle (defined in Table 5.3) is used to flush or invalidate a secondary cache. The defined value of byte enables can be generated by using an appropriate address operand in the **scyc** instruction. The **scyc** instruction does not have any effect on the internal caches. External hardware must acknowledge a special bus cycle by asserting BRDY# once. The data driven on the data bus with BRDY# is *undefined*. The effect of **scyc** is determined by decoders in external hardware.

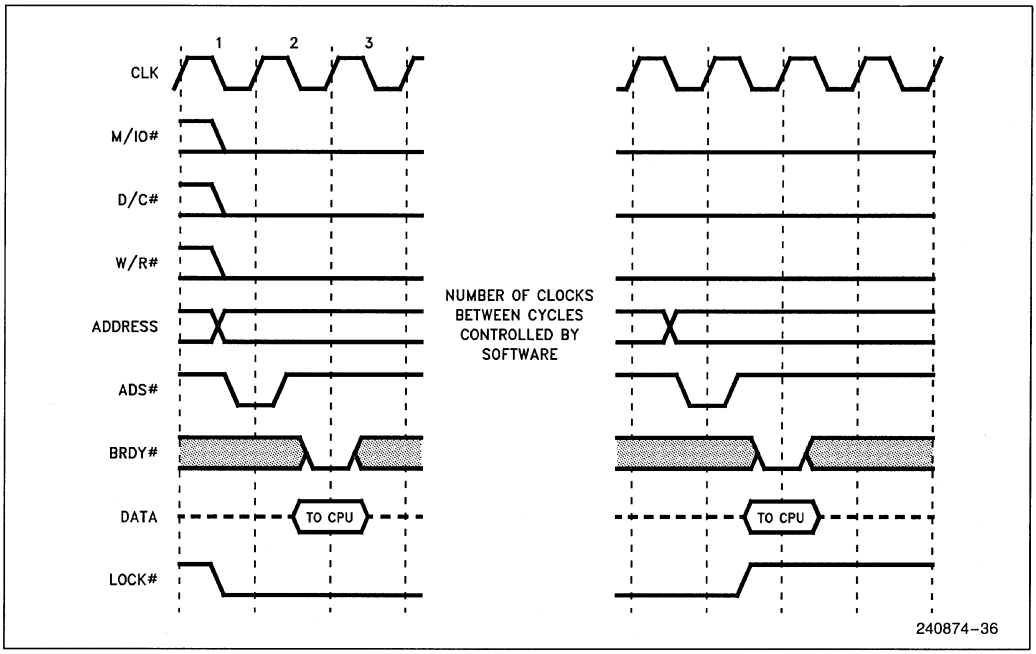


Figure 5.9. Example Interrupt Acknowledge Sequence

Table 5.3. Encoding of Special Bus Cycles

BE7# – BE0#	Special Bus Cycle
1 1 1 1 0 1 1 1	Write Back External Cache and Invalidate
1 1 1 1 1 0 1 1	Halt
1 1 1 1 1 1 0 1	Invalidate External Cache
1 1 1 1 1 1 1 0	Shut Down

All other encodings are *reserved.*

5.2 Bus Arbitration

The i860 XP microprocessor responds to three different signals that tell it to stop driving the bus:

HOLD Finishes outstanding cycles before giving up the bus.

BOFF# Aborts outstanding cycles and gives up bus immediately.

AHOLD Stops driving address bus and permits a cache inquiry.

AHOLD results in a partial hold state, which is covered in Section 5.3. The present section concentrates on HOLD and BOFF#.

When in a hold state (due either to HOLD or BOFF#), the i860 XP microprocessor uses BREQ to request control of the bus. If holding due to HOLD, AHOLD, or BOFF#, the processor activates BREQ in the clock after an internal bus request is generated. (In the case of HOLD, BREQ is asserted even though HLDA is asserted.) If holding due to BOFF# and cycles need to be restarted or there is a new internal request, it asserts the BREQ signal within four clock periods after the assertion of BOFF#. In all cases, BREQ remains active at least until the clock after ADS# is activated for the requested cycle.

5.2.1 HOLD AND HLDA ARBITRATION

HOLD indicates to the i860 XP microprocessor that another bus master needs control of the bus. When HOLD is asserted, the i860 XP microprocessor keeps control of the bus until all outstanding cycles are completed. Then it floats the output signals (except BREQ, HLDA, LOCK#, PCHK#, HIT#, and HITM#) and asserts HLDA. These outputs remain at the high-impedance state until HOLD is deasserted.

HLDA may be asserted as soon as the clock period after the one in which HOLD is asserted. HLDA may be deasserted as soon as the clock after the one in which HOLD is deasserted.

An example HOLD/HLDA transaction is shown in Figure 5.10. The i860 XP microprocessor recognizes HOLD even while RESET is asserted, and it drives HLDA in this case as well.

HOLD is recognized even when BOFF# is active, and the i860 XP microprocessor responds with HLDA the same as when the bus is idle.

5.2.2 BUS CYCLE BACK-OFF AND RESTART

The i860 XP microprocessor provides the ability to abort bus cycles and restart them again. It is necessary to abort cycles for reasons such as the following:

1. Retry after an error is detected by ECC or parity logic.
2. Escape from a deadlock; for example, when the i860 XP microprocessor is using A31–A3 to load a new cache line, but the 82495XP cache controller needs A31–A5 to invalidate a line in the CPU cache which the 82495XP cache controller is replacing in its cache in order to satisfy the CPU's line-fill request.
3. Maintain cache consistency; for example, the i860 XP microprocessor is attempting to read or write to a line that has been modified in the cache of another CPU.
4. Prevent illegal access to an address already locked by another CPU in a multiprocessor system.

5.2.2.1 Cycle Back-Off

Bus cycles are aborted when the system asserts BOFF#. The i860 XP microprocessor samples this pin in every clock period that it is driving the bus. When BOFF# is asserted, the i860 XP microprocessor immediately (in the next clock period) floats the bus. It floats the ADS# pin one clock period later, thereby giving time for ADS# to be deasserted so that it is not left floating active. The i860 XP microprocessor floats the same pins as for HOLD, but HLDA is not asserted. If a bus cycle is in progress at the time BOFF# is asserted, the cycle is aborted, and, in a read cycle, any data returned to the processor while BOFF# is active is ignored. BOFF# overrides BRDY#; so, if both are sampled active in the same clock, BRDY# is ignored. BOFF# aborts

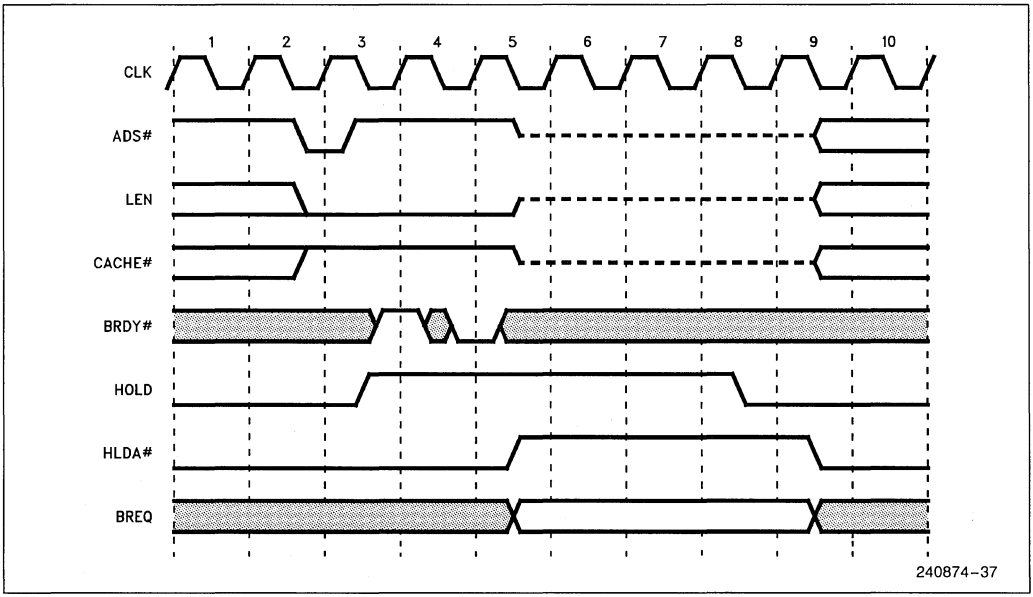


Figure 5.10. HOLD/HLDA Handshake

a burst cycle even if it arrives with the last BRDY# of the cycle. However, for read bursts, data transfers completed before assertion of BOFF# are used by the processor if they satisfy an internal request. Cacheable data is cached in spite of BOFF#; however, the cached data is overwritten when the cycle is restarted.

The bus remains in the high-impedance state until BOFF# is deasserted. If cycles need to be restarted or if a new internal request has been generated, the BREQ signal is asserted within four clock periods after the assertion of BOFF#.

5.2.2.2 Cycle Restart

When the system deasserts BOFF#, the i860 XP microprocessor restarts aborted bus cycles from the beginning by driving the address and status (A31–A3, W/R#, D/C#, etc.) and asserting ADS#. If more than one cycle was outstanding when BOFF# was asserted, the i860 XP microprocessor restarts all outstanding cycles in the same order. If HITM# is active due to an inquiry, the write-back for it will be the first cycle after deassertion of BOFF#. BOFF# restarts all aborted cycles except:

- The stale cycles mentioned in section 5.3.5.
- The read that may have been generated by an alias hit (virtual tag miss, but physical tag hit).
- The read that may have been generated by a **pfld** that hit the data cache.

If the processor's KEN# pin was active (with NA# or first BRDY#) before the cycle was aborted, external hardware must activate it again after the cycle is restarted. In other words, the system cannot use BOFF# to change the cacheability of a cycle via KEN#.

The LOCK# signal is not affected by restarted cycles; it retains its state in spite of BOFF# assertion.

5.2.2.3 Late Back-Off Modes

In some cases the logic that needs to assert BOFF# cannot make the necessary decision in time to cancel the relevant cycle or data transfer. For example:

1. The result of checking ECC or parity may not be available until one or two cycles after the BRDY# to which it corresponds.
2. When the i860 XP microprocessor is attempting to read or write to a line that might be modified in the cache of another processor on the same bus, it may be advantageous to let part of a burst run in parallel with inquiries to the other processors, rather than delay the entire burst until the inquiries are finished.

For such situations, the i860 XP microprocessor provides *late back-off mode*. For a read cycle in this mode, the processor employs a buffer to internally delay data and BRDY#, which allows BOFF# assertion to be delayed relative to the external BRDY#. Likewise, for a write cycle in this mode, BOFF# assertion can be delayed relative to BRDY#. However, data for a write cycle is not delayed.

Two flavors of late back-off mode are provided:

1. One allows BOFF# to be delayed by one clock period relative to the data transfer. The processor enters one-clock late back-off mode when the FLINE# pin has been sampled active for at least three clock periods when RESET deactivates.
2. The other allows BOFF# to be delayed by up to two clock periods relative to the data transfer. The i860 XP microprocessor enters this mode when software sets the LB bit of the **dirbase** register.

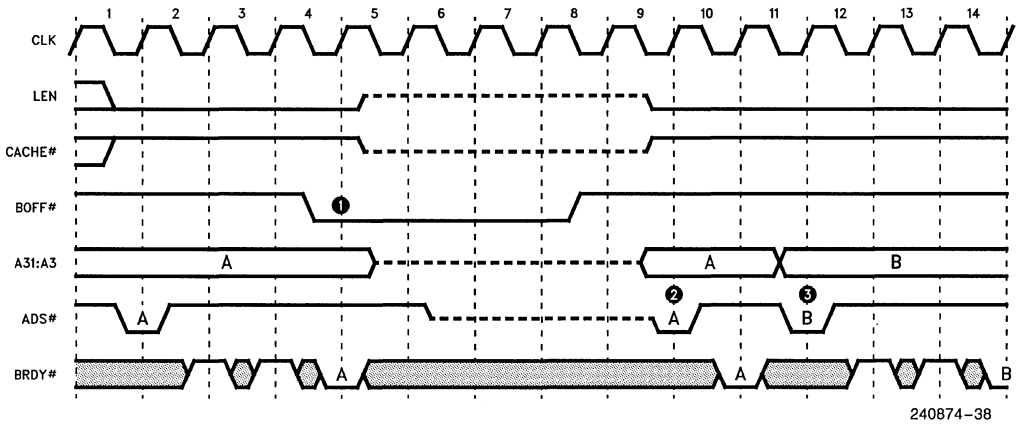
If the processor enters one-clock late back-off mode during RESET, it is impossible to enter two-clock late back-off mode. The LB bit has no effect. Furthermore, software cannot exit two-clock late back-off mode once it is activated, and the LB bit cannot be cleared except by resetting the processor.

Figures 5.12–5.17 illustrate variations on late back-off mode cycles. BOFF# can be (and usually is) asserted longer than one clock period, as Figure 5.11 shows; the remaining figures show an active time of only one clock.

5.2.2.4 One-Clock Late Back-Off Mode

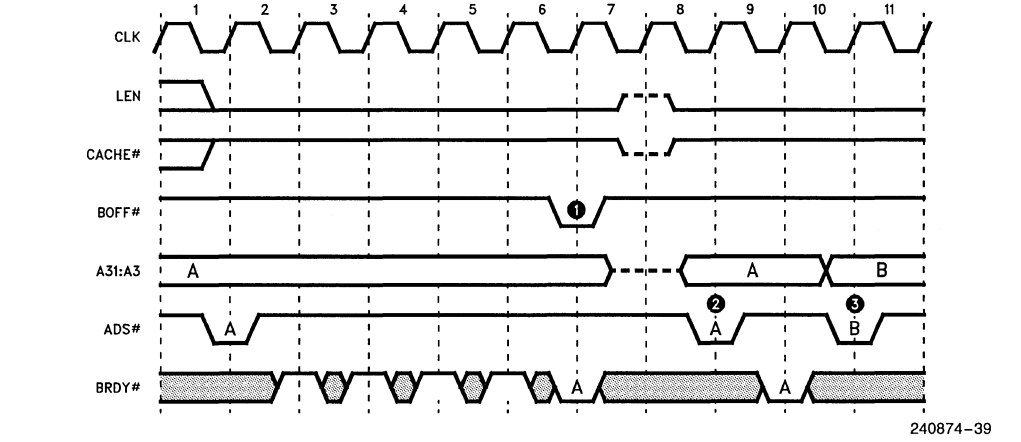
In *one-clock late back-off mode* the data is delayed internally by one clock before it is used.

In this mode, data and BRDY# are seen by internal logic one clock period later than they appear on the bus, which is equivalent to adding an extra wait state to reads on the external bus (Figure 5.13). All responses to BRDY# (assertion of the ADS# for the next cycle, assertion of HLDA in response to a HOLD request, and deassertion of HITM#) are delayed by one clock period compared to the normal mode of operation. Not delayed, however, are write data on D63–D0 and sampling of KEN# and WB/WT#. KEN# and WB/WT# must be valid with the first BRDY# assertion.



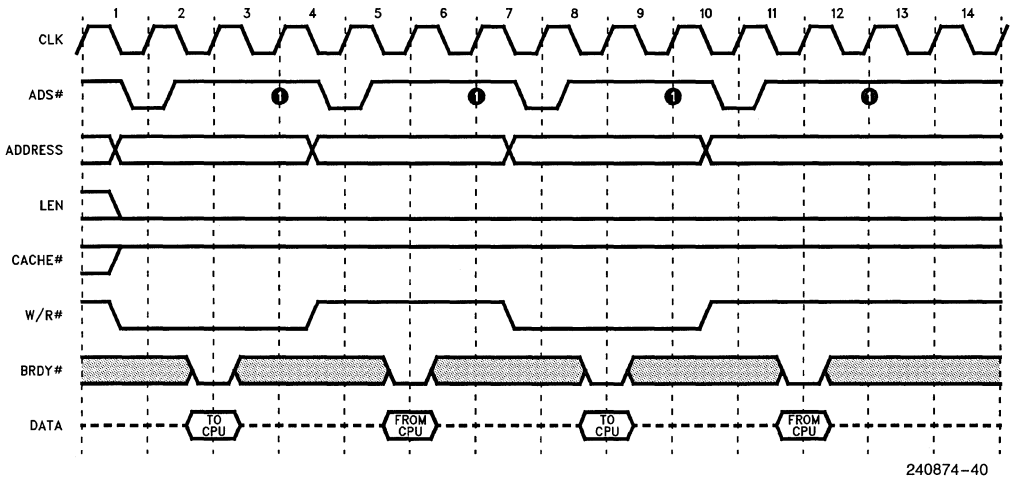
NOTES:
A Noncacheable, 64-bit cycle (one transfer)
B Next cycle (any type)
1. BOFF# cancels cycle and data transfer
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.11. Normal Back-Off



NOTES:
A Noncacheable, 64-bit cycle (one transfer)
B Next cycle (any type)
1. BOFF# cancels cycle and data transfer
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.12. One-Clock Normal Back-Off



NOTE:
1. Idle clock due to internal delay of BRDY#

Figure 5.13. Fastest Nonpipelined Cycles in One-Clock Late Back-Off Mode

If BOFF# is asserted as late as the second BRDY# (Figure 5.14), it cancels the entire cycle, ignores data latched with the first BRDY#, and ignores the data being driven with the second BRDY#. This is true of a two-transfer burst (shown) as well as a four-transfer burst (not shown).

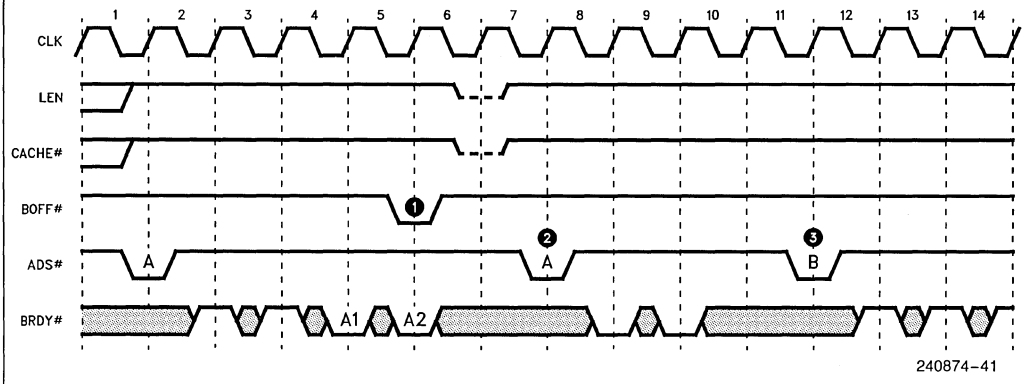
In a two-transfer burst, if BOFF# is asserted in the clock after the second BRDY# (Figure 5.15), it still cancels the cycle.

In a four-transfer burst, if BOFF# is asserted within one clock after the last BRDY# (Figure 5.16), it still forces a retry of the cycle, but previously transferred read data is used by the processor if it satisfies the read request.

5.2.2.5 Two-Clock Late Back-Off Mode

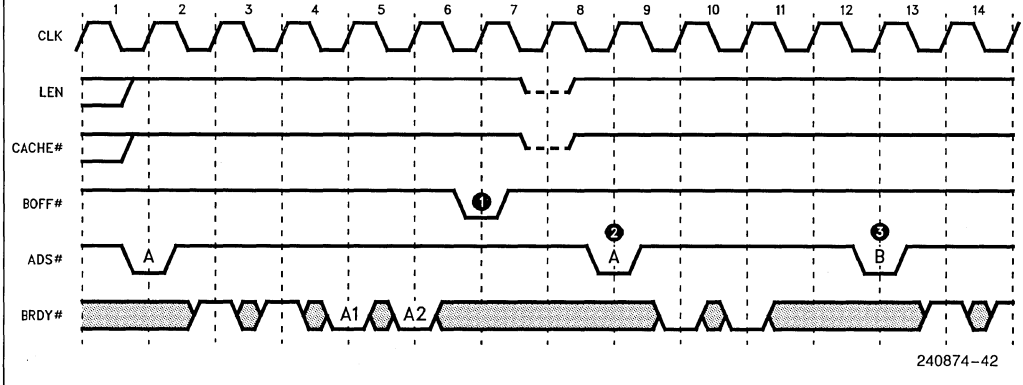
Two-clock late back-off mode gives external logic even more time to decide to use BOFF#. In this mode, data delivery is delayed by either one or two clock periods, depending on external activity. For any BRDY#, the data is delayed by one clock period. If in the next clock period BRDY# is again asserted, the previous data is used. However, if in that next clock period BRDY# remains inactive, the data is delayed for one extra clock period before it is used. The responses to BRDY# (assertion of the ADS# for the next cycle, assertion of HLDA, and deassertion of HITM#) are delayed by one or two clock periods, depending on the value of BRDY# in the next clock.

The **st.c dirbase** instruction that sets the LB bit must be aligned on a 32-byte boundary and must be followed by seven **nop** instructions. Software must not enable late back-off mode when the processor is used with the 82495XP external cache controller.



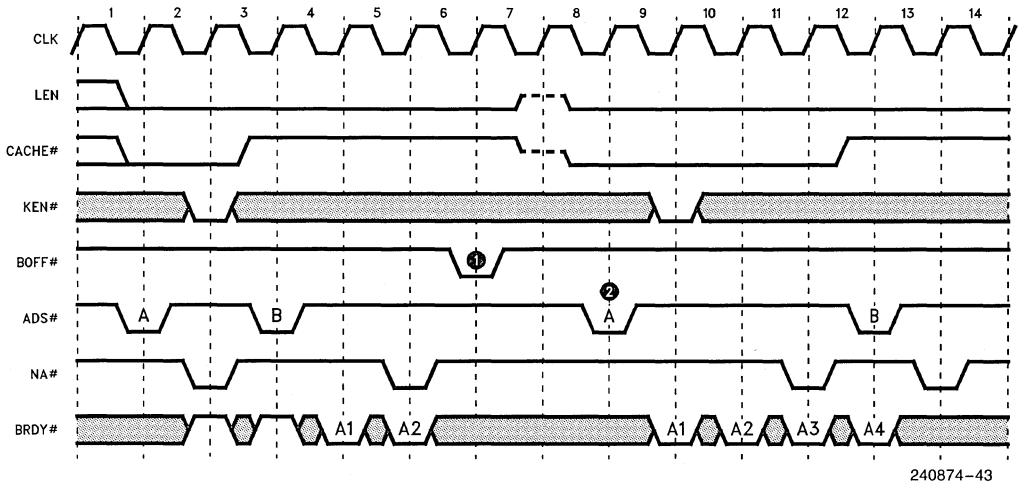
NOTES:
A Noncacheable, 128-bit cycle (two transfers)
B Next cycle (any type)
1. BOFF# cancels both transfers (A1 in buffer, A2 on D63–D0)
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.14. One-Clock Late Back-Off Mode (Case 1)



NOTES:
A Noncacheable, 128-bit cycle (two transfers)
B Next cycle (noncacheable)
1. BOFF# cancels both transfers (A2 in buffer is needed to satisfy request)
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.15. One-Clock Late Back-Off Mode (Case 2)



NOTES:
A Cacheable 64-bit (or less) cycle (four transfers)
B Next cycle (any type)
1. BOFF# cancels A2 and A3 transfers, but A1 transfer has already satisfied request
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.16. One-Clock Late Back-Off Mode (Case 3)

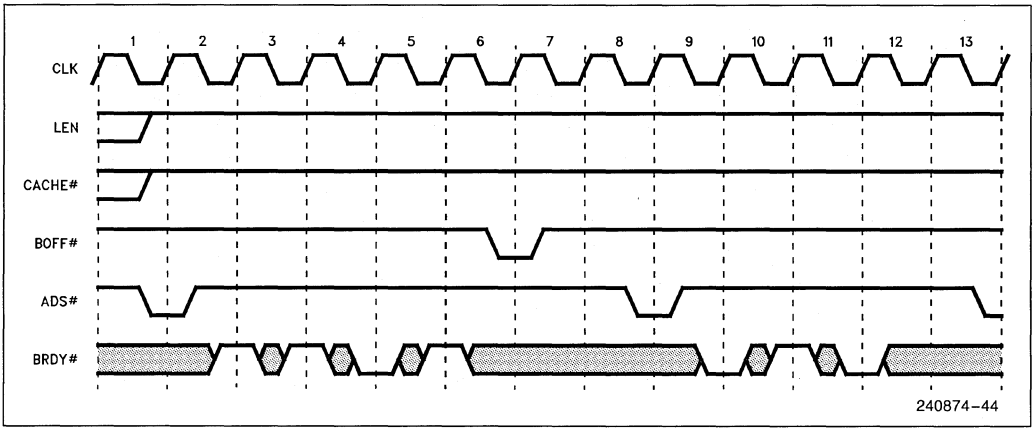


Figure 5.17. Two-Clock Late Back-Off Mode

5.3 Cache Inquiry Cycles (Snooping)

Another processor initiates an inquiry cycle to check whether an address is cached in the internal data or instruction cache of the i860 XP microprocessor. An inquiry cycle differs from any other cycle in that it is initiated externally to the i860 XP microprocessor, and the signal for beginning the cycle is EADS# (External Address Status) instead of ADS#. The address bus of the i860 XP microprocessor is bidirectional in order to allow the address of inquiry to be driven by the system. An inquiry cycle can begin during any hold state:

1. While HOLD and HLDA are asserted.
2. While BOFF# is asserted.
3. While AHOLD (address hold) is asserted.

If neither a HOLD nor a BOFF# is in effect, the system can assert AHOLD to interrupt the current bus activity.

EADS# is first sampled two clocks after BOFF# or AHOLD assertion, or one clock after HLDA. This allows time for the processor to float A31–A5 and for the system to stabilize the inquiry address there.

In the clock in which EADS# is asserted, the i860 XP microprocessor samples these inputs, which qualify the type of inquiry:

INV Specifies whether the line (if found) must be invalidated (that is, changed to I-state).

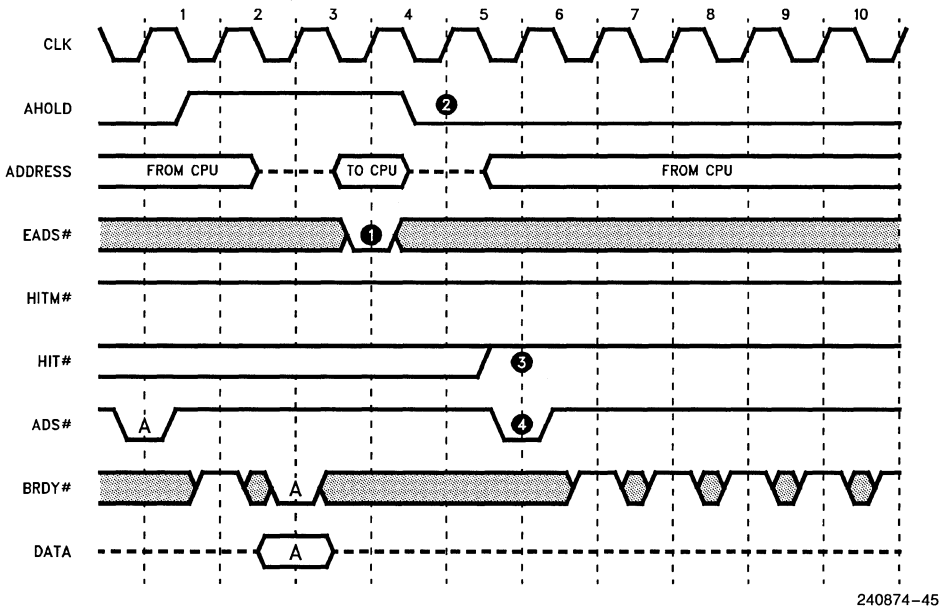
FLINE# Specifies whether the line (if found in M-state) must be written back immediately or after outstanding bus cycles are completed.

The i860 XP microprocessor compares the address of the inquiry request with addresses of lines in cache and of any line in the write-back buffer waiting to be transferred on the bus. It does not, however, compare with the address of write-miss data in the write buffers. Two clock periods after sampling EADS#, the i860 XP drives the results of the inquiry look-up on these output pins:

HIT# Specifies whether the address was found (active) or not found (inactive).

HITM# If active, the line found was in the M-state; if inactive, the line was in E- or S-state, or was not found.

Figure 5.18 shows an inquiry with AHOLD that misses the cache. When the system asserts AHOLD, the i860 XP microprocessor floats A31–A3 in the next clock period. It does not, however, assert HLDA; no acknowledge is required. Once the address pins are floating, external logic drives the address for the inquiry on A31–A5 and starts the inquiry cycle by activating EADS#. The i860 XP microprocessor does not begin sampling EADS# until the second clock after AHOLD is activated. EADS# activation may be delayed any number of clocks.



NOTES:
A Outstanding cycle (for example, a single-transfer read) finishes during the inquiry
1. Earliest assertion of EADS# is two clocks after assertion of AHOLD
2. Earliest deassertion of AHOLD is one clock after assertion of EADS#
3. HIT# is valid two clocks after assertion of EADS#
4. Earliest assertion of ADS# for next cycle is one clock after deassertion of AHOLD

Figure 5.18. Inquiry Miss Cycle

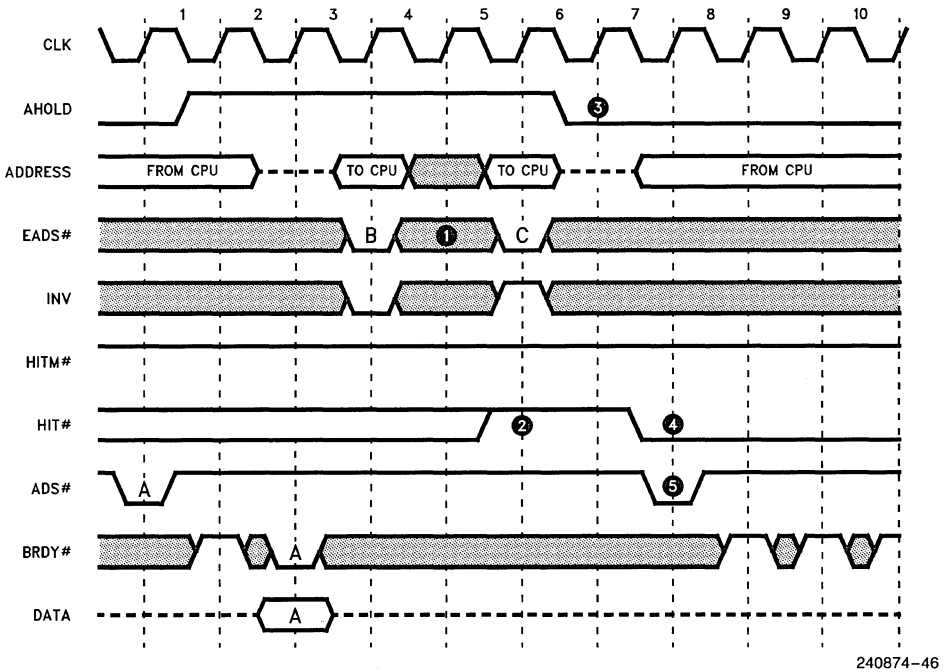
The earliest that AHOLD can be deasserted is the clock after EADS# assertion. However, by maintaining AHOLD active, multiple inquiry cycles can be executed in one AHOLD session (Figure 5.19). The i860 XP microprocessor can accept inquiry cycles at a rate of one every other clock period, unless a write-back is required. The earliest that ADS# can be asserted for the next cycle is the clock after AHOLD deassertion.

The second inquiry in Figure 5.19 hits an unmodified line in the cache. When a cache line with matching address is found and the INV input signal is asserted (as in this case), that line is invalidated (changed to I-state). If the INV signal is inactive, the line enters S-state.

5.3.1 INQUIRY WRITE-BACK CYCLES

If an inquiry cycle hits a dirty (M-state) line in the i860 XP microprocessor cache, the i860 XP microprocessor asserts the HITM# signal to indicate that the line will be written on the bus. The HITM# output becomes valid in the same clock period as HIT#. In this case the modified line is written out, and the cache entry is changed to either I or S state according to INV. The HITM# signal stays active through the last BRDY# for the corresponding write-back cycle.

An inquiry write-back cycle is similar to ordinary write-back cycles. It is initiated by assertion of ADS#. ADS# is asserted even when the AHOLD



NOTES:
A Outstanding cycle (for example, a single-transfer read) finishes during the inquiry
B Earliest inquiry, no invalidation
C Earliest successive inquiry, with invalidation
1. EADS# is not sampled in the clock after its assertion
2. Inquiry B misses cache
3. Earliest deassertion of AHOLD is one clock after last assertion of EADS#
4. Inquiry C hits cache, invalidates line
5. Earliest assertion of ADS# for next cycle is one clock after deassertion of AHOLD

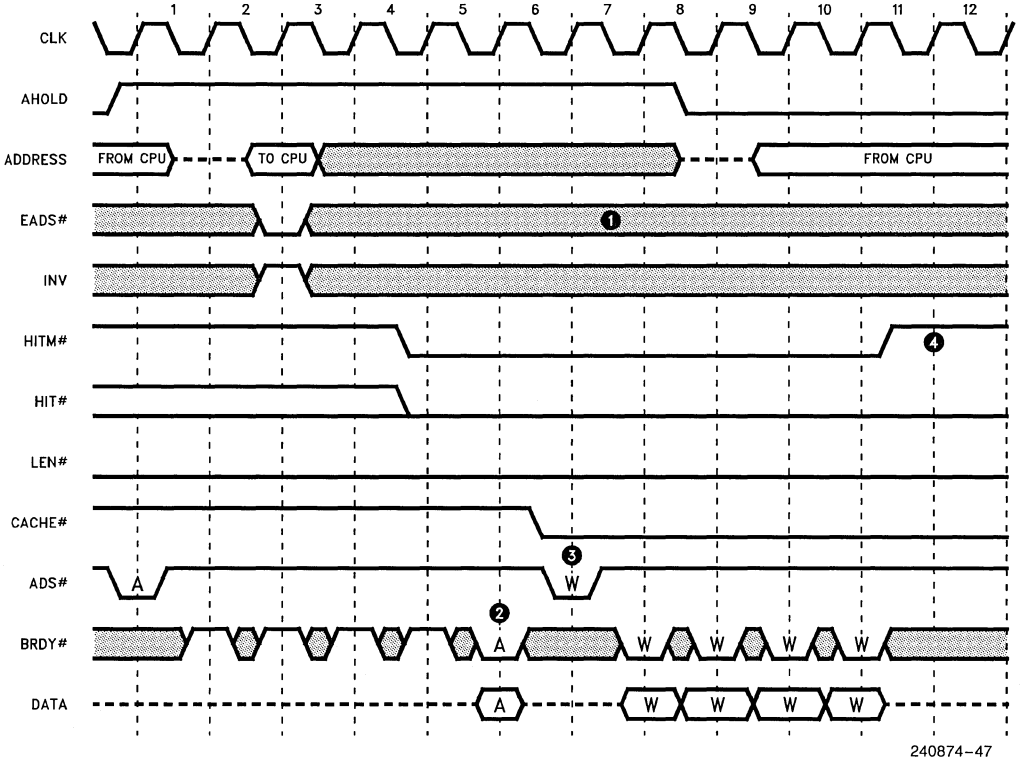
Figure 5.19. Fastest Inquiry Cycles (Miss and Hit)

signal is active. The cycle definition signals are driven properly by the processor, however, the address pins are not driven, because activation of AHOLD forces the i860 XP microprocessor off the address bus. If, however, AHOLD is deasserted before or during the write-back cycle, the i860 XP microprocessor drives the correct address for the write-back.

For all types of inquiry, the write-backs are not pipelined into an outstanding cycle, except when the FLINE# pin is used (refer to section 5.3.5). ADS# for the inquiry write-back is asserted from one to four clock periods after the HITM# pin is driven active or after the last BRDY# is returned for any outstanding cycle, whichever occurs later.

Bursts for a HITM# write-back, as for any write-back, are in the order 0, 8, 0x10, 0x18, because the i860 XP microprocessor ignores A4–A3 of the inquiry address.

Figure 5.20 shows an inquiry cycle that hits an M-state line.



NOTES:
A Outstanding cycle (for example, a single-transfer read)
W Write-back cycle
1. EADS# is not sampled while HITM# is active
2. Earliest ADS# assertion if not delayed by outstanding cycle
3. ADS# for write-back delayed by outstanding cycle
4. HITM# deactivates after last BRDY# of write-back

Figure 5.20. Inquiry Hit Cycle with Write-Back

The fact that a write-back cycle is initiated while address lines are floating supports multiple inquiries (with write-backs) during a single AHOLD session. This is especially useful during secondary cache replacement processing, when the secondary-cache line is larger than that of the i860 XP microprocessor.

Note that EADS# is ignored as long as HITM# is active. If the system is executing a series of inquiries, it might happen that the HITM# assertion for one inquiry masks the EADS# for a subsequent inquiry. In that case the system must reassert EADS# to restart the masked inquiry.

Inquiries can occur during a hold due to HOLD/HLDA or BOFF#. However, in these cases, the cycle definition pins and ADS# are floating. If an inquiry requires a write-back, the HOLD or BOFF# must be deasserted so that the cycle definition pins and ADS# can be driven to start the write-back cycle.

5.3.2 SNOOPING RESPONSIBILITY LIMITS

The i860 XP microprocessor takes responsibility for responding to inquiry cycles for a cache line only during the time that the line is actually in the cache or in a write-back buffer. There are times during the cache line fill cycle and during the cache replacement cycle when the line is "in transit", and inquiry (snooping) responsibility must be taken by other system components.

Systems designers should consider the possibility that an inquiry cycle may arrive at the same time as a cache line fill or replacement for the same address. This situation can occur:

- In multiprocessor systems that have external (secondary) caches with separate CPU and memory busses, thereby allowing concurrent activity on the two busses. In such systems, it is desirable to run invalidation cycles concurrently with other i860 XP microprocessor bus activity. It can happen that writes on the memory bus cause invalidation requests to the i860 XP microprocessor at the same time that the i860 XP microprocessor fetches data from the secondary cache. Such events can occur at any time relative to each other.
- In multiprocessor systems with no secondary cache, if memory is dual-ported. In such systems, two processors can simultaneously read the same line, each sending an inquiry to the other.

The simultaneous activities considered here may be for different data items in the same cache line. Unless the inquiry request is timed carefully with respect to the cache fill cycle, the cache-consistency mechanism may be subverted, and data inconsistencies may result (for example, both CPUs may get the line in E-state on a read). If the 82495XP and 82490XP cache is being used, the timing with respect to the i860 XP microprocessor is handled correctly by the cache controller; however, the same problem may arise between the memory system and the secondary cache.

There are two cases to consider:

1. Inquiry for a line that is being cached.
2. Inquiry for a line that is being replaced.

5.3.2.1 Inquiry for a Line Being Cached

The i860 XP microprocessor accepts an inquiry cycle at any time, even if it hits the line being cached at that time. Regardless of the timing of the cycle, the i860 XP microprocessor delivers the read data to the load instruction that initiated the read request. However, the timing of the invalidation cycle determines whether the line is placed in the cache and what value the i860 XP microprocessor drives on HIT#. Table 5.4 summarizes the different cases.

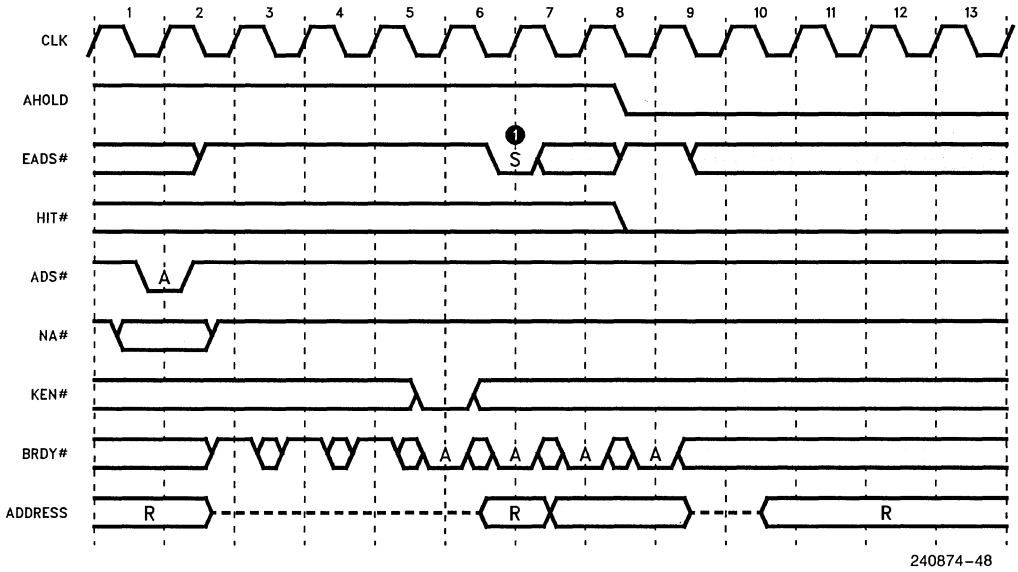
Table 5.4. Inquiry for a Line being Cached

	EADS# before or with NA# or 1st BRDY#	**EADS# after NA# or 1st BRDY#**
Line is cached?	YES	NO
HIT# =	Inactive	Active
Data/Instruction used by CPU?	YES	YES

If EADS# is asserted before or with the sampling of KEN#, the processor cannot match the address of the line being cached with an invalidation request. Thus, the processor does not assert HIT#. The external system must satisfy the inquiry with the correct data and WB/WT# status. If invalidation of that line is required, the system must do one of the following:

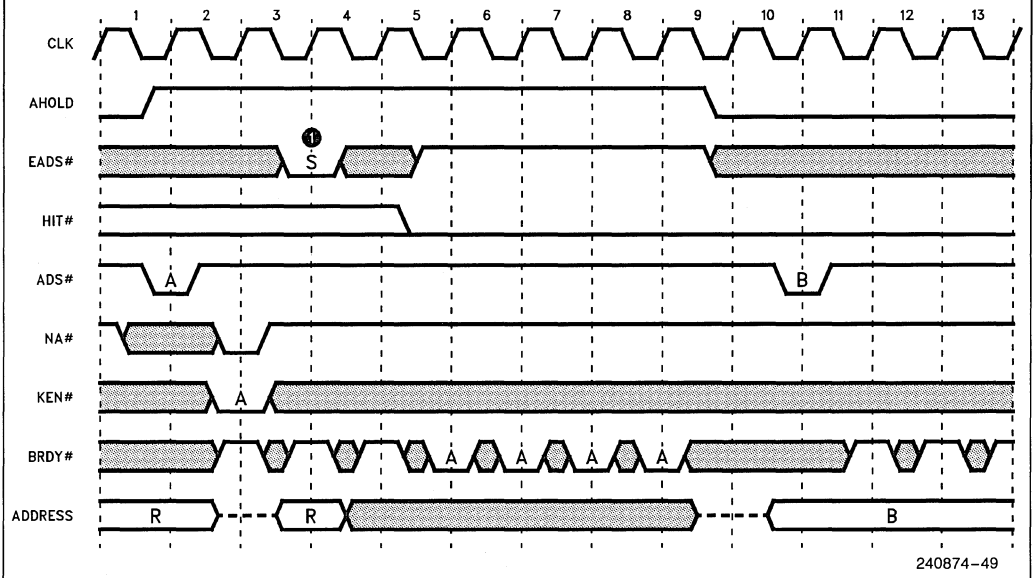
- Delay assertion of EADS# until one clock after assertion of KEN#.
- Reassert EADS# after KEN#.
- Make KEN# inactive at the first BRDY# or NA#, thereby preventing the line from being cached.

Figures 5.21 and 5.22 show when the i860 XP microprocessor picks up responsibility for inquiries for a line that it is caching. Figure 5.21 shows the earliest EADS# assertion that invalidates the line being cached relative to the first BRDY# for nonpipelined cycles. Figure 5.22 shows the earliest EADS# assertion that invalidates the line being cached relative to the first NA# for pipelined cycles. These timings hold for normal and late back-off modes.



NOTES:
A Cache line fill cycle
S Snoop (inquiry) cycle
R Addresses of cache line fill and snoop are the same
1. Earliest EADS# assertion that can invalidate line being filled

Figure 5.21. Snoop Responsibility Pickup (Nonpipelined Cycle)



NOTES:
A Cache line fill cycle
B Next cycle (any type)
S Snoop (inquiry) cycle
R Addresses of cycles A and S are the same
1. Earliest EADS# assertion that can invalidate line being filled

Figure 5.22. Snoop Responsibility Pickup (Pipelined Cycle)

5.3.2.2 Inquiry for a Line Being Replaced

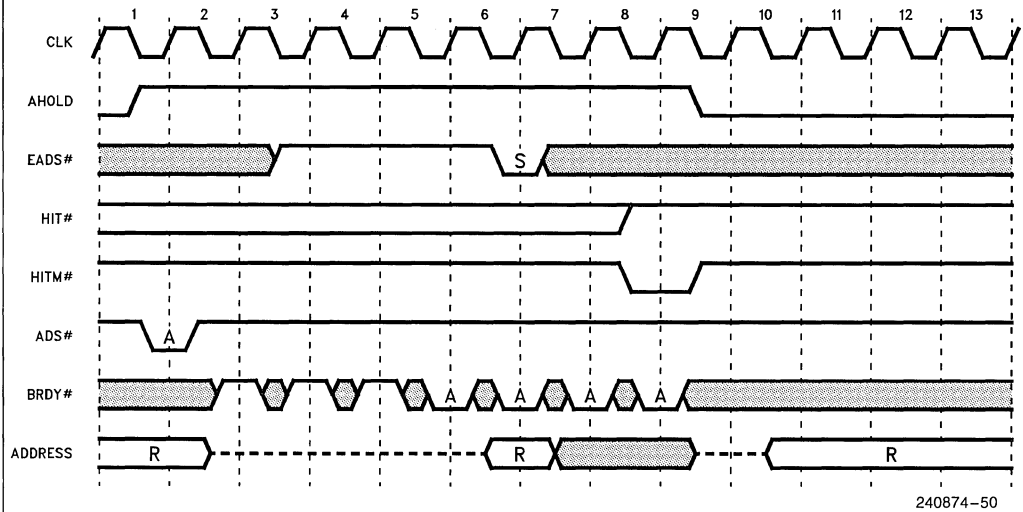
When the i860 XP microprocessor is replacing a line, there are two cases:

1. If the replacement does not require write-back, the address being replaced can be matched by an inquiry until assertion of NA# or first BRDY# of the line-fill cycle. From that point on, the inquiry has no effect.
2. If the replacement requires a write-back, the address being replaced can be matched by an inquiry until assertion of the last BRDY# for the write-back. An EADS# as late as two clocks before the last BRDY# can cause HITM# to be asserted.

Figures 5.23 through 5.25 show when the i860 XP microprocessor drops responsibility for recognizing inquiries for a line that it is writing back. They show the latest EADS# assertion that can cause HITM# assertion. In late back-off mode, EADS# can be asserted later, because BRDY# is internally delayed (Figures 5.24 and 5.25).

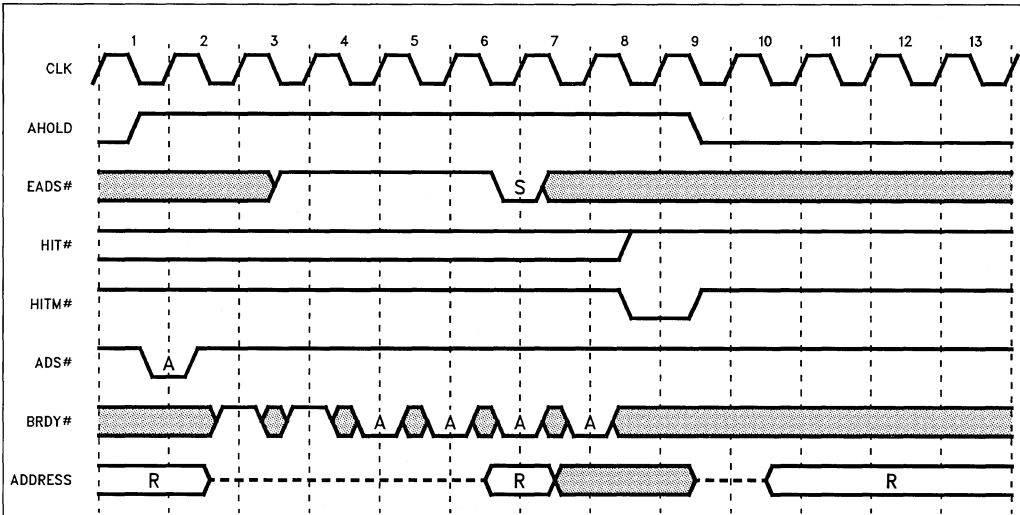
In all these cases, HITM# remains active for only one clock period. HITM#, as always, remains active through the last BRDY# of the corresponding write-back; in these cases the write-back has already completed.

If an inquiry cycle hits the write-back address after its ADS# has been issued, the i860 XP microprocessor asserts HITM#; however, HIT# is deasserted. This unique combination of values on HIT# and HITM# indicates that the write-back cycle corresponding to the HITM# has already been issued.



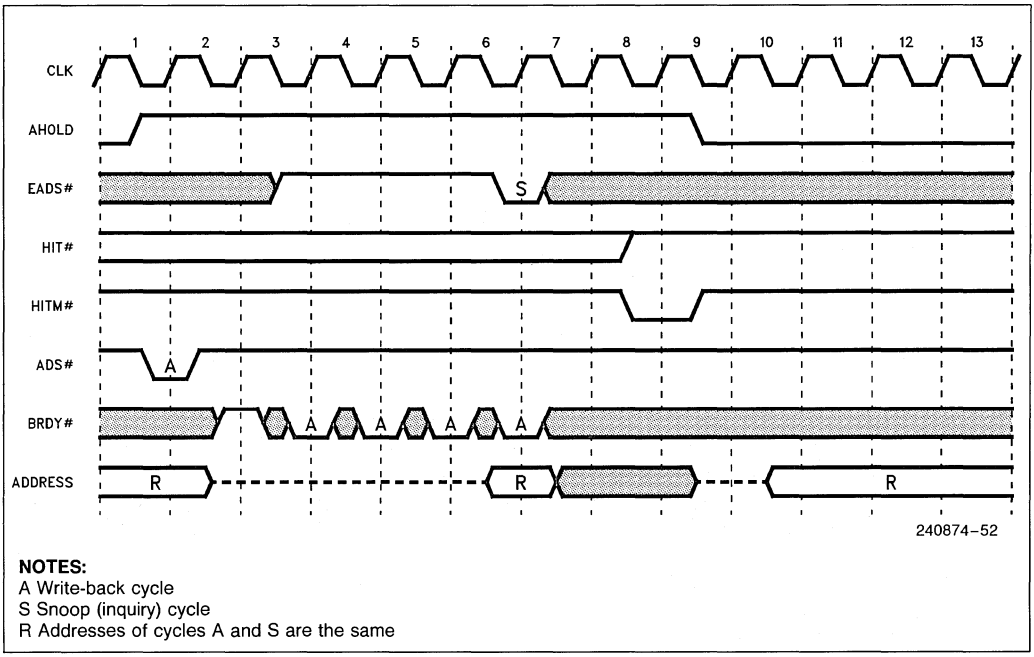
NOTES:
A Write-back cycle
S Snoop (inquiry) cycle
R Addresses of cycles A and S are the same

Figure 5.23. Latest Snooping of Write-Back (Not Late Back-Off Mode)



NOTES:
A Write-back cycle
S Snoop (inquiry) cycle
R Addresses of cycles A and S are the same

Figure 5.24. Latest Snooping of Write-Back (One-Clock Late Back-Off Mode)



NOTES:
A Write-back cycle
S Snoop (inquiry) cycle
R Addresses of cycles A and S are the same

Figure 5.25. Latest Snooping of Write-Back (Two-Clock Late Back-Off Mode)

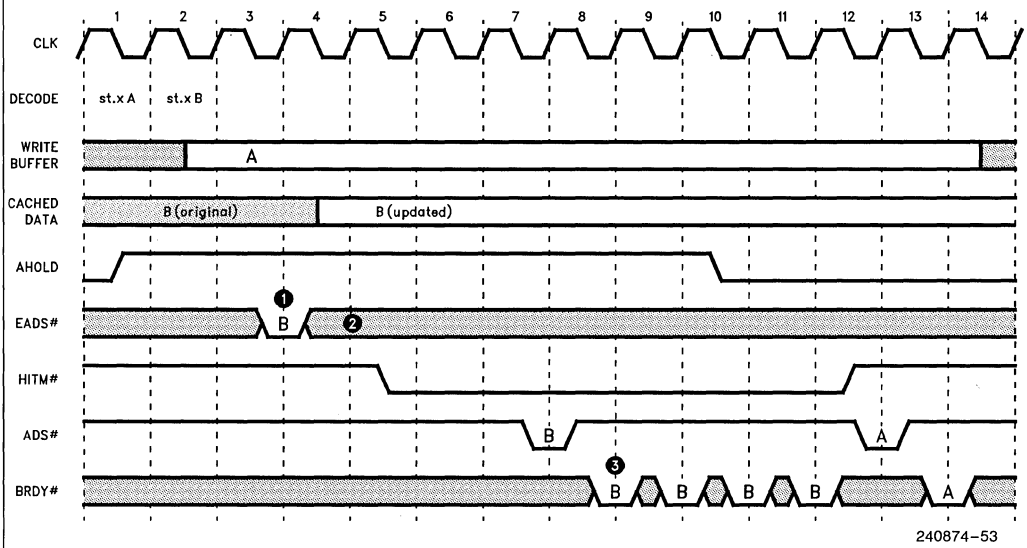
5.3.3 WRITE CYCLE REORDERING DUE TO BUFFERING

The MESI cache protocol and the ability to perform and respond to inquiry cycles guarantee that writes to the cache are logically equivalent to writes that go to memory. In particular, the *order* of read and write operations on cached data is the same as if the operations were on data in memory. Even uncached memory read and write requests usually occur on the external bus in the same order that they are issued in the program. For example, when a write miss is followed by a read miss, the write data goes onto the bus before the read request is put on the bus. However, the posting of writes in write buffers coupled with inquiry cycles may cause the order of writes seen on the external bus to differ from the order they appear in the program. Consider the following example, which is illustrated in Figure 5.26:

1. Three bus cycles are outstanding.
2. Processor 1 executes a store to address A, which misses the cache. This store is posted; that is, the data is latched in the write buffer while the processor continues execution without waiting for the store to be completed on the bus. In this case the store is not even put on the bus because there are already three outstanding cycles.
3. Processor 1 executes a store to address B, which hits the cache.
4. Processor 2 executes an inquiry for address B. Processor 1 looks in its cache, finds the modified line, asserts HIT# and HITM#, and executes a write-back cycle to address B, while the data for address A is still in the write buffer.
5. Processor 1 issues the write to address A on the bus.

In this example, the original order of the writes has been changed. In most cases it is not necessary that the ordering of writes be strictly maintained. But there are cases (for example, semaphore updates in a multiprocessor system) that require stores to be observed externally in the same order as programmed. There are several ways to ensure serialization of stores:

1. Bracket one of the stores with the **lock** and **unlock** instructions. That forces serialization of the stores (refer to section 5.4). In the above example of a store-miss followed by store-hit, locking either store would ensure that the internal store-hit does not update the cache until the miss gets to the external bus.
2. Apply the write-through policy to the critical data, by setting WT = 1 in the page table entries or by driving the WB/WT# pin low.



NOTES:
A Data written by **st.x A** instruction
B Data written by **st.x B** instruction
1. Snoop for address of B
2. Snoop look-up in tag array occurs here; finds B modified
3. Write-back of line containing B occurs before write of A

Figure 5.26. Write Reordering due to Buffering

3. Configure the processor for Strong Ordering Mode by asserting EWBE# during RESET.

Option 1 is implementable by user-level programs, while option 2 is an operating-system level solution, not directly implementable by user-level code. Option 3, the hardware solution, is discussed in greater detail in section 5.3.4.

5.3.4 STRONG ORDERING MODE

In strong ordering mode, the processor delays updates to its internal data cache in either of these conditions:

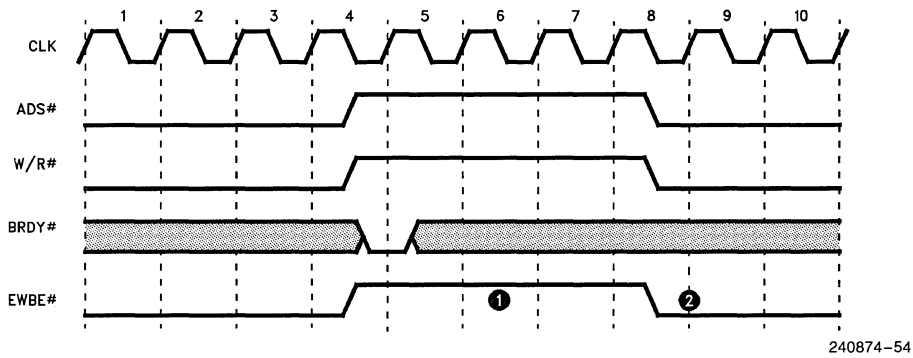
1. The internal write buffer is not empty.
2. An external write buffer is not empty (the external system signals this condition by deactivating the EWBE# signal).

By delaying the cache update until all write buffers are empty, the i860 XP microprocessor avoids the out-of-order sequence shown in section 5.3.3.

In strong ordering mode, EWBE# can be asserted only between the ADS# and the last BRDY# of a store. The earliest assertion is the clock after ADS#; the latest assertion is together with the last BRDY#. EWBE# can be deasserted any time, except when the processor is performing an inquiry write-back. In other words, EWBE# must not deactivate while HITM# is active. When EWBE# is deasserted, the processor completes any cache update that may have been delayed by its assertion.

Figure 5.27 shows how an external cache can use EWBE# when a store miss in the i860 XP microprocessor is also a miss in the external cache.

An external cache controller should also refrain from updating the external cache while EWBE# is active.



NOTE:
1. Assumes the external cache needs five cycles to write the data to memory.
2. Pending internal data cache updates are delayed until the clock in which EWBE# is sampled LOW.

Figure 5.27. Timing of EWBE#

5.3.5 SCHEDULING INQUIRY WRITE-BACK CYCLES

In order to preserve system-wide ordering of memory transactions in multiprocessor systems that have a pipelined or split-transaction memory bus, it may be necessary to get the data corresponding to an inquiry hit before outstanding bus cycles are completed. Another bus master can always request an inquiry while the i860 XP microprocessor has cycles outstanding on the bus. However, when AHOLD is asserted, the i860 XP microprocessor normally completes outstanding cycles before it performs any write-back that may be required. The i860 XP microprocessor provides two methods for causing the inquiry write-back before outstanding cycles are completed:

FLINE# When FLINE# is asserted during the EADS# of an inquiry that hits an M-state line, the i860 XP microprocessor issues a write-back cycle and writes the dirty line to memory before the outstanding bus cycles are completed.

BOFF# If there are outstanding cycles on the bus, asserting BOFF# clears the bus pipeline. If an inquiry causes HITM# to be asserted, then the first cycle issued by the i860 XP microprocessor after deassertion of BOFF# is the inquiry write-back cycle. After the inquiry write-back, it reissues the aborted cycles.

5.3.5.1 Choosing between FLINE# and BOFF#

FLINE#, although the more efficient choice, cannot handle all situations. Under certain circumstances, it can happen that outstanding stores on the bus correspond to data that is obsolete relative to the data in the cache, because a subsequent store has updated the cache after the ADS# for the outstanding store has occurred. For example:

- An *aliasing store hit*, in which a cache virtual-tag miss occurs and the ADS# is issued at the same time as a physical-tag hit. Then the cached data would be updated before external memory, and a subsequent store to the new virtual address could also update cache before the outstanding bus store completed.
- *Back-to-back writes* to the same line can also update the cache more recently than the bus when the write-once update policy is employed. The first write updates the cache and generates a bus write request, but the second write only updates the cache.

In both of these examples the outstanding stores on the bus are obsolete relative to the data in the cache line. If an inquiry cycle hits a line and this line is written back out of order (that is, before outstanding stores are completed), special care should be taken to discard the outstanding stores.

The easiest way to avoid this situation is not to assert FLINE# when stores are outstanding, but use BOFF# instead. If out-of-order write-back is implemented with BOFF#, the i860 XP microprocessor does not restart the outstanding store to that line if such a store has been obsoleted by a later cache hit store. That is, the i860 XP microprocessor detects this condition and kills the obsolete data. However, lock-bracketed stores (including the last store in the lock sequence) are restarted by the i860 XP microprocessor, because lock-bracketed stores update the cache only after BRDY# is returned.

If, on the other hand, out-of-order write-back is implemented by using only the FLINE# pin, the external system must return BRDY#s for outstanding stores, but the data must be ignored if it has already been written out by an inquiry write-back.

Note that if a replacement write-back is in progress (ADS# has been issued, but last BRDY# has not occurred) and an inquiry hits the same line that is being written back, the FLINE# pin is ignored. The system can recognize this special case by the fact that HITM# is asserted while HIT# is deasserted. If other cycles are outstanding and it is necessary to write the line back before the other cycles, BOFF# can be used.

5.3.5.2 Reordering Write-Backs with FLINE#

FLINE# must be active during the EADS# that initiates an inquiry. BRDY# must not be asserted forthe previously issued cycles while HITM# is active. If HITM# is asserted while the data transfer of the outstanding cycle is in progress (i.e. first BRDY# has been asserted, but the entire transfer has not yet been completed), the i860 XP microprocessor waits for the current cycle to complete, and only then issues the write-back. After the last BRDY# for the ongoing burst (if any), BRDY# is ignored until the clock period after ADS# is asserted for the write-back.

From the viewpoint of the i860 XP microprocessor, an inquiry write-back cycle is just another bus cycle; so, if there is an outstanding cycle at the time of FLINE# and HITM# activation, the system must assert NA# to initiate the write-back.

Figure 5.28 illustrates simple cycle reordering, when FLINE# is not asserted during the data transfer of another cycle. The outstanding request could be either a read or write.

Figure 5.29 shows the case in which FLINE# is asserted after data transfer for the outstanding cycle has already started. In this case, the i860 XP microprocessor does not issue a write-back until the outstanding transfer is completed. NA# is needed in this example only if other outstanding cycles remain.

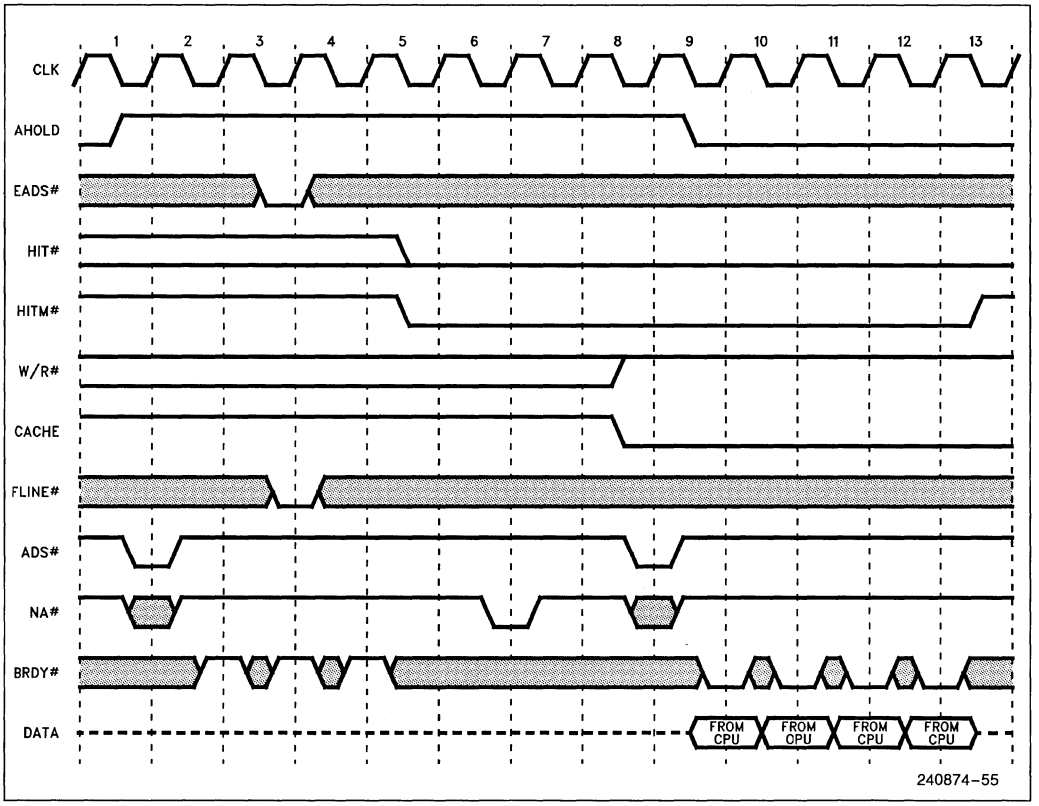
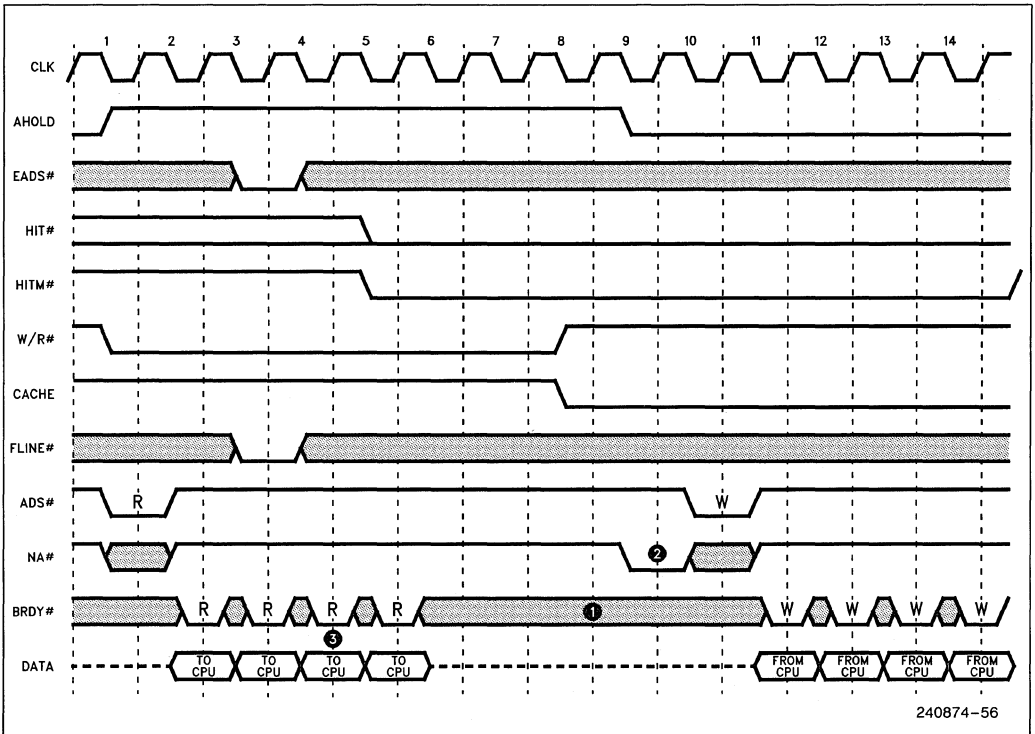


Figure 5.28. Cycle Reordering via FLINE# (No Ongoing Burst)



NOTES:
1. BRDY# is ignored by CPU from end of ongoing burst through ADS# of write-back, even if other cycles remain outstanding
2. NA# required only if another cycle is outstanding
3. If the first BRDY# is asserted here or sooner (relative to HITM#), the outstanding cycle completes before the FLINE# write-back.

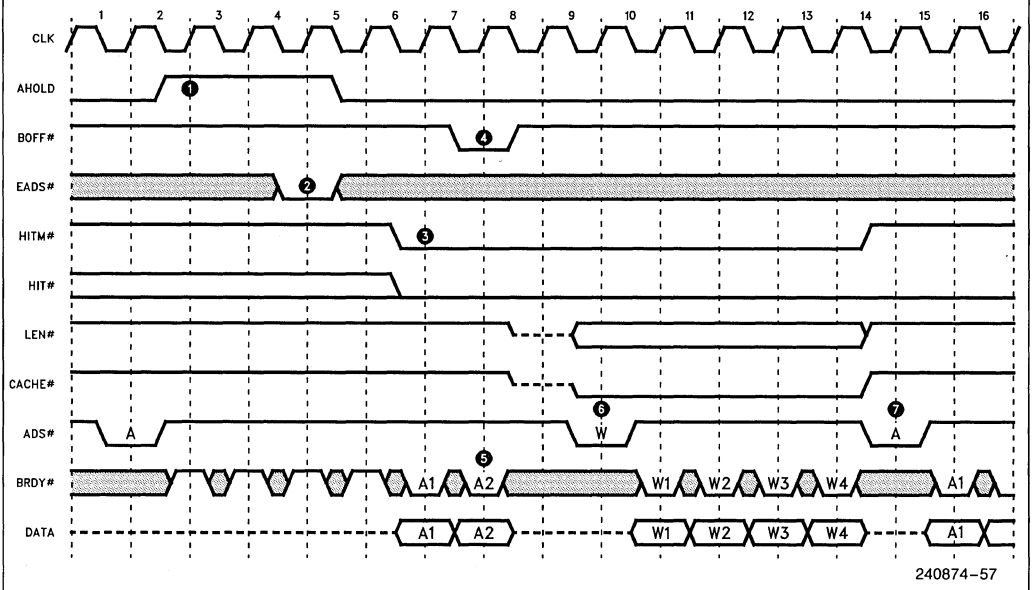
Figure 5.29. Cycle Reordering via FLINE# (Ongoing Burst)

5.3.5.3 Reordering Write-Backs with BOFF#

Back-off cycles are discussed in general in Section 5.2.2. Figure 5.30 shows how BOFF# can be used to cancel outstanding cycles so that an inquiry write-back can take place immediately.

5.4 The LOCK# Cycle Attribute

The processor asserts the LOCK# signal when several accesses to a single memory location must be effectively uninterruptible. By causing LOCK# to be asserted, a programmer can, for example, increment the contents of a memory variable and be assured that the variable will not be accessed between the read and the update of that variable.



NOTES:
A Outstanding cycle (for example, noncacheable 128-bit read) W Write-back cycle
1. AHOLD begins an inquiry while one cycle is outstanding.
2. Earliest assertion of EADS# is two clocks after assertion of AHOLD
3. Inquiry hits modified line.
4. Assertion of BOFF# aborts the outstanding cycle.
5. BRDY# asserted during BOFF# is ignored by CPU.
6. Write-back begins after deassertion of BOFF#.
7. Earliest assertion of ADS# for restart of cycle A (assuming no pipelining).

Figure 5.30. Cycle Reordering via BOFF# (Ongoing Burst)

The memory location to be locked is the one whose address is driven during the cycle in which LOCK# is first activated. In multiprocessor systems, external hardware should guarantee that no other processor is granted a locked read, locked write, or unlocked write to the same location until LOCK# is deasserted. The i860 XP microprocessor has no hardware provision to prevent another master from also locking the variable; this responsibility falls on the bus arbiter. In the simplest implementation, the arbiter can globally prevent other masters from accessing the bus.

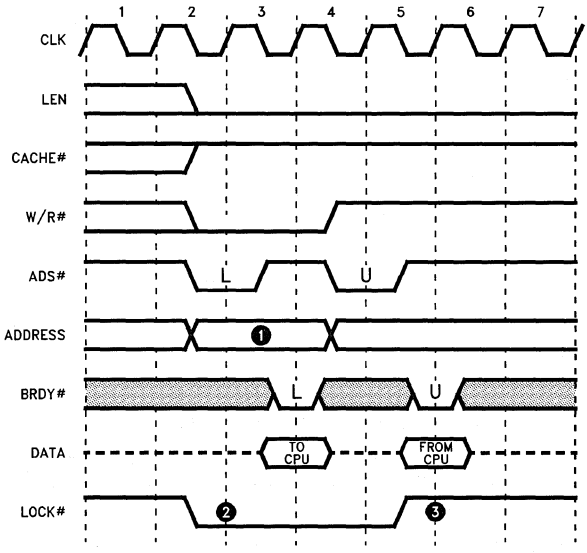
Not all cycles affect the value of LOCK#. Code fetches, write-backs due to replacement or inquiry, and cycles restarted due to BOFF# do not affect LOCK#. Any other type of cycle can be used to initiate or terminate LOCK#, including cache line fills, interrupt acknowledge, I/O, and special cycles.

Data accesses with LOCK# asserted are not pipelined, and other data cycles are not pipelined while a LOCK# cycle remains outstanding. Instruction fetches, however, may be pipelined during lock.

The i860 XP microprocessor can run very long lock sequences; therefore, to guarantee reasonable bus turnover latency in multimaster systems, the i860 XP microprocessor recognizes bus hold (HOLD), address hold (AHOLD), and back-off (BOFF#) while the LOCK# signal is active. In spite of such intervening conditions, the arbiter should prevent any other bus master from also locking or updating the variable the i860 XP microprocessor locked. In simple systems the HOLD input can be masked by the LOCK# output (that is, the external logic that generates HOLD can AND the LOCK# signal with other hold conditions). More sophisticated systems, however, may allow the bus to be turned over while LOCK# is asserted.

Whatever the lock implementation, arbiter design must, in one case, allow another processor to write the locked variable. That case is when another i860 XP microprocessor or master asserts HITM# in response to the inquiry generated by the locking processor's initial read. That other master must write back the locked variable before the i860 XP microprocessor can read it. This HITM# write-back must always be allowed.

The timing of LOCK# is shown in Figure 5.31. Note that LOCK# is asserted in the same clock period as ADS# for the locked address, but is deasserted in the clock period *after* ADS# for the unlocking load or store.



240874-58

NOTES:
L Locking access
U Unlocking access
1. This address is to be locked
2. LOCK# is asserted with ADS#
3. LOCK# is deasserted one clock after ADS#

Figure 5.31. LOCK# Timing

5.5 RESET Initialization

Initialization of the i860 XP microprocessor is caused when the system asserts the RESET signal for at least ten clocks. Table 5.5 shows the status of output pins during the time that RESET is asserted. Note that the bidirectional data pins (D63–D0 and DP7–DP0) are floated during RESET, though the bidirectional A31–A3 pins are not. If the i860 XP microprocessor is used with 82495XP and 82496XP cache, however, the latter do float the bidirectional pins they share with i860 XP microprocessor during RESET. Note that HOLD requests are honored during RESET and that the HLDA output signal may also become active. The status of output pins depends on whether a HOLD request is being acknowledged. Note also that the test logic may be active during RESET and that the EXTEST instruction may drive other values on the output pins.

Some aspects of processor configuration are determined by asserting input signals during RESET. To select a given option, the corresponding input must be asserted for at least the last three clocks before the falling edge of RESET; to deselect, the corresponding input must be deasserted for at least the last three clocks before the falling edge of RESET:

EWBE# Enter strong ordering mode.

FLINE# Enter one clock late back-off mode.

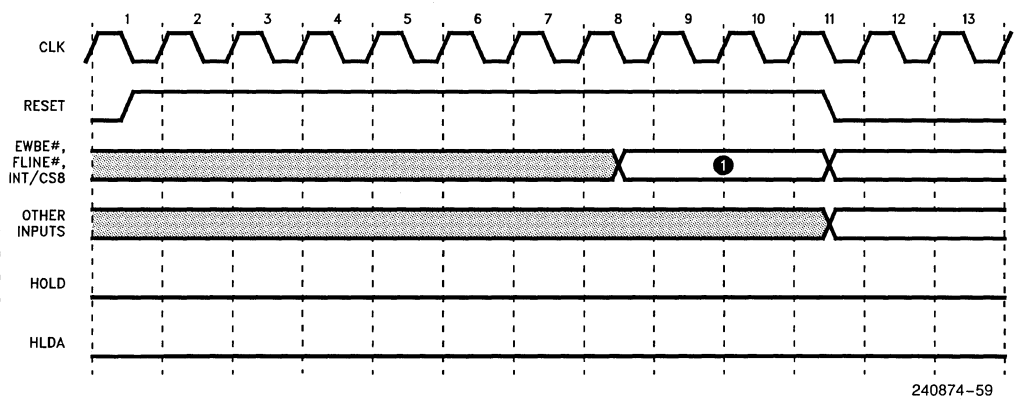
INT/CS8 Enter eight-bit code-size mode.

Figure 5.32 shows how configuration pins are sampled during the three clock periods just before the falling edge of RESET. No inputs besides EWBE#, HOLD, FLINE#, and INT/CS8 are sampled during RESET.

Table 5.5. Output Pin Status during Reset

Pin Name	Pin Value	
	HOLD Not Acknowledged	HOLD Acknowledged
BREQ	LOW	LOW
HLDA	LOW	HIGH
W/R#, PWT, PCD	LOW	Tristate OFF
ADS#	HIGH	Tristate OFF
D63–D0, DP7–DP0	Tristate OFF	Tristate OFF
A31–A3, BE7#0–BE0#, NENE# CACHE#, CTYP, D/C#, KB0, KB1, LEN, M/IO#, PCYC	Undefined	Tristate OFF
PCHK#, HIT#	Undefined	Undefined
HITM#, LOCK#	HIGH	HIGH

NOTE:
This table does not apply if the test logic is running the EXTEST instruction.



NOTE:
1. The CPU samples these inputs in the clock preceding the falling edge of reset.

Figure 5.32. Reset Activities

While in eight-bit code-size mode, instruction cache misses are one-byte reads (transferred on D7–D0 of the data bus) instead of eight-byte reads. This allows the i860 XP microprocessor to be bootstrapped from an eight-bit ROM. For these code reads, byte enables BE2#–BE0# are redefined to be the low order three bits of the address, so that a complete byte address is available. The entire eight-byte data bus continues to be parity-checked by the i860 XP microprocessor during CS8-mode instruction fetches; therefore, external hardware must either generate good parity on all eight bytes or disable parity traps by deasserting PEN# during CS8 mode.

While in this mode, instructions must reside in an eight-bit wide memory, while data must reside in a separate 64-bit wide memory. After the code has been loaded into 64-bit memory, initialization code can initiate 64-bit code fetches by clearing the CS8 bit of the **dirbase** register (refer to section 2). Once eight-bit code-size mode is disabled by software, it cannot be reenabled except by resetting the i860 XP microprocessor.

Instruction fetches in CS8 mode update the instruction cache if KEN# is asserted during NA# or all of the first eight BRDY#s (refer to section 4.2.26). They are pipelined if NA# is asserted. When used with the 82495XP and 82496XP cache, CS8 mode works only if the ROM locations are made non-cacheable.

6.0 TESTABILITY

The i860 XP microprocessor provides testability features compatible with the proposed *Standard Test Access Port and Boundary-Scan Architecture* (IEEE Std. P1149.1/D6). The subset of the standard test logic implemented in the i860 XP microprocessor provides for testing the interconnections between the i860 XP microprocessor and other integrated circuits once they have been assembled onto a printed circuit board.

The test logic consists of a boundary-scan register and other building blocks that are accessed through a test access port (TAP). The TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes.

The TAP can be controlled by a bus master. The bus master can be either automatic test equipment or a component that interfaces to a four-pin test bus.

6.1 Test Architecture

The test logic contains the following elements:

- Test access port (TAP), which consists of input pins TMS, TCK, TDI, and TRST#; and output pin TDO.
- TAP controller, which receives the dedicated test clock (TCK) and interprets the signals on the test mode select (TMS) line. The TAP controller generates clock and control signals for the instruction and test data registers and for other parts of the test logic.
- Instruction register (IR), which allows instruction codes to be shifted into the test logic. The instruction codes are used to select the test to be performed or the test data register to be accessed.
- Test data registers: Bypass Register (BPR), Device Identification Register (DID), and Boundary-Scan Register (BSR).

The instruction and test data registers are separate shift-register paths connected in parallel and having a common serial data input and a common serial data output connected to the TAP TDI and TDO signals respectively.

6.2 Test Data Registers

The test logic contains the following data registers:

- **Bypass Register (BPR)**: BPR is a one-bit shift register that provides a minimum-length path between TDI and TDO when no test operation of the component is required. This allows more rapid movement of test data to and from other board components that are required to perform test operations. While running through BPR, the data is transferred without inversion from TDI to TDO.
- **Device Identification Register (DID)**: This register contains the manufacturer's identification code, part number code, and version code in the format shown by Figure 6.1. The values are: manufacturer's identification code (9), part number code (61A0), version code (8), entire 32-bit value (0x861A0013).
- **Boundary Scan Register (BSR)**: The BSR is a single shift-register path containing 150 cells that are connected to all input and output pins of the i860 XP microprocessor. Figure 6.2 shows the logical structure of the BSR. Input cells only capture data; they do not affect operation of the i860 XP microprocessor. Data is transferred without inversion from TDI to TDO through the BSR during scanning. The BSR can be operated by the EXTEST and SAMPLE instructions.

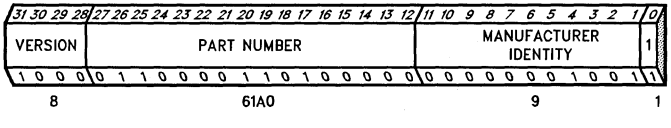


Figure 6.1. Format of DID Register

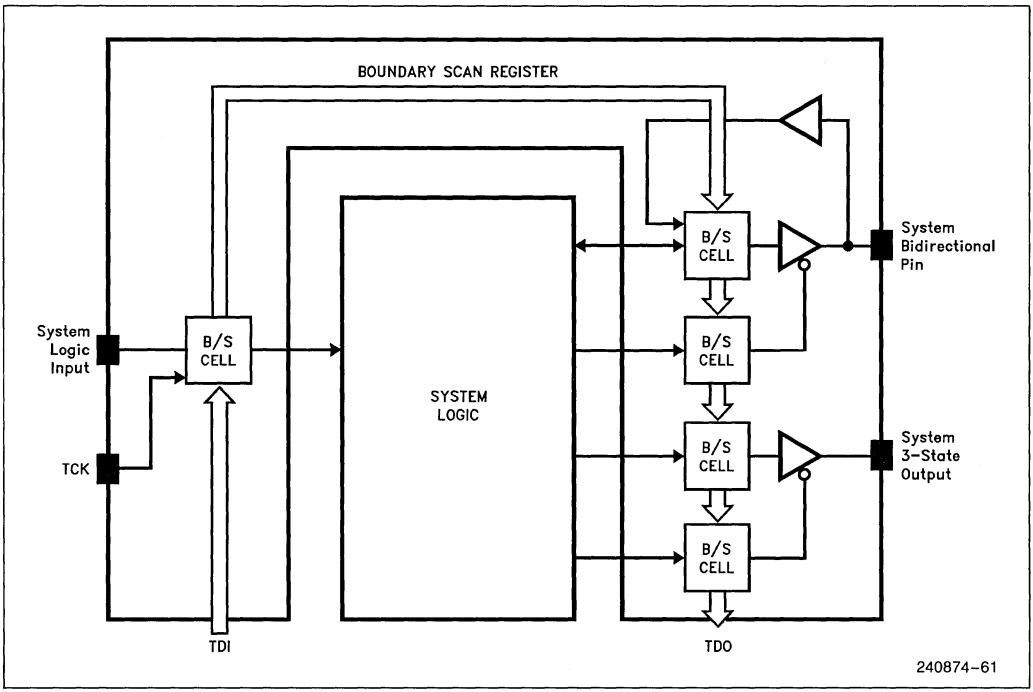


Figure 6.2. Logical Structure of BSR Register

6.3 Instruction Register

The Instruction Register (IR) selects the test to be performed and the test data register to be accessed. It is four bits wide, with no parity bit. Table 6.1 shows the encoding of the instructions supported by the TAP controller of the i860 XP microprocessor. The rightmost bit is the least significant and is the first shifted out on TDO.

Table 6.1. TAP Instruction Encoding

Instruction Code	Instruction
0000	EXTEST Boundary Scan
0001	SAMPLE Boundary Scan
0010	IDCODE
0011 . . . 1110	*Intel reserved* **CAUTION***
1111	BYPASS

* **CAUTION:** Operation of these private instructions may cause damage to the component.

EXTEST The BSR cells associated with output pins drive the output pins of the i860 XP microprocessor. Values scanned into the BSR cells become the output values. The BSR cells associated with input pins sample the inputs of the i860 XP microprocessor. Note that I/O pins can be input or output for this test, depending on their control setting. The values shifted to the input latches are not used by the internal logic of the i860 XP microprocessor. After use of the EXTEST command, the i860 XP microprocessor must be reset (with the RESET signal) before normal use.

SAMPLE The BSR cells associated with output pins sample the value driven by the i860 XP microprocessor. BSR cells associated with input pins sample on the rising edge of TCK the values driven to the i860 XP

microprocessor. BSR cells associated with I/O pins sample the value on the respective pin. The I/O pin can be driven by the i860 XP microprocessor or by external hardware. The values shifted to the input latches are not used by the internal logic of the i860 XP microprocessor.

IDCODE The identification code of the i860 XP microprocessor from the DID register is passed to TDO. The DID register is not altered by data shifted in on TDI.

BYPASS Test data is passed from TDI to TDO via the single-bit BPR, effectively bypassing the test logic of the i860 XP microprocessor. Because of its special encoding, this instruction can be entered by holding TDI HIGH while completing an instruction-scan cycle. This reduces the demands on the host test system in cases where access is required, for example, only to chip 57 on a 100-chip board.

Note that an open circuit fault in the board-level test data path causes the BPR register to be selected following an instruction-scan cycle, because the TDI input has a pull-up resistor. Therefore, no unwanted interference with the operation of the on-chip system logic can occur.

Table 6.2 defines which registers are active during execution of each instruction.

6.4 TAP Controller

The TAP Controller is a synchronous, finite state machine. It controls the sequence of operations of the test logic. The TAP Controller changes state only in response to the following events:

1. A rising edge of TCK.
2. A transition to logic zero at the TRST# input.
3. Power-up.

The value of the TMS input signal at a rising edge of TCK controls the sequence of state changes. The state diagram for the TAP controller is shown in Figure 6.3. Test designers must consider the operation of the state machine in order to design the correct sequence of values to drive on TMS.

6.4.1 TEST-LOGIC-RESET STATE

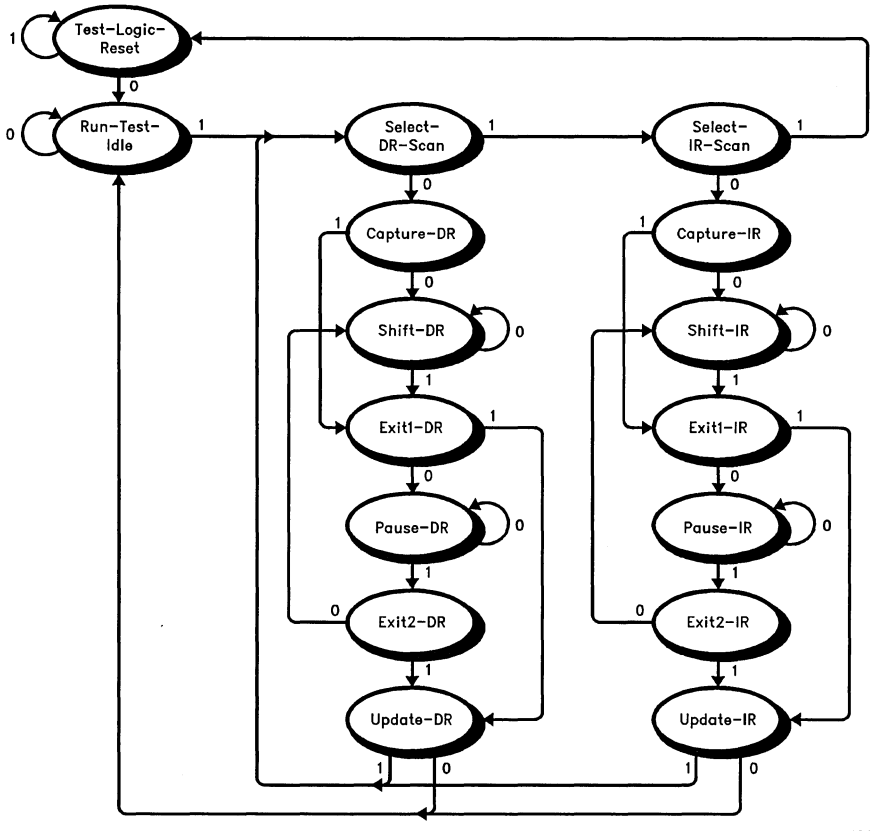
In this state, the test logic is disabled so that normal operation of the i860 XP microprocessor can continue unhindered. This is achieved by initializing the instruction register such that the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters *Test-Logic-Reset* when the TMS input is held HIGH for at least five rising edges of TCK. The controller remains in this state while TMS is HIGH.

If the controller leaves the *Test-Logic-Reset* state as a result of an erroneous LOW signal on the TMS line at the time of a rising edge of TCK (for example, a glitch due to external interference), it returns to the *Test-Logic-Reset* state following three rising edges of TCK while the TMS signal at the intended HIGH logic level. The operation of the test logic is such that no disturbance is caused to on-chip system logic operation as the result of such an error. On leaving the *Test-Logic-Reset* state, the controller moves into the *Run-Test/Idle* state, where no action occurs because the current instruction has been set to select operation of the DID register. The test logic is also inactive in the *Select-DR-Scan* and *Select-IR-Scan* states.

The TAP controller is also forced to the *Test-Logic-Reset* state by applying a LOW logic level to the TRST# input and at power-up.

Table 6.2. Registers Active by Instruction

Mode	Register		
	BSR	DID	BPR
EXTEST	TDI → BSR → TDO	Inactive	Inactive
SAMPLE	TDI → BSR → TDO	Inactive	Inactive
IDCODE	Inactive	DID → TDO	Inactive
BYPASS	Inactive	Inactive	TDI → BPR → TDO



NOTE:
0,1 The values present on TMS at the time of a rising edge on TCK.

Figure 6.3. TAP Controller State Diagram

6.4.2 RUN-TEST/IDLE STATE

The controller enters this state between scan operations. Once in this state, the controller remains in this state as long as TMS is held LOW. No activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is HIGH and a rising edge is applied to TCK, the controller moves to the *Select-DR-Scan* state.

6.4.3 SELECT-DR-SCAN STATE

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held LOW and a rising edge is applied to TCK when in this state, the controller moves into the *Capture-DR* state, and a scan sequence for the selected test data register is initiated. If TMS is held HIGH and a rising edge is applied to TCK, the controller moves to the *Select-IR-Scan* state.

The instruction does not change in this state.

6.4.4 SELECT-IR-SCAN STATE

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held LOW and a rising edge is applied to TCK when in this state, the controller moves into the *Capture-IR* state, and a scan sequence for the instruction register is initiated. If TMS is held HIGH and a rising edge is applied to TCK, the controller moves to the *Test-Logic-Reset* state.

The instruction does not change in this state.

6.4.5 CAPTURE-DR STATE

In this state, the BSR captures input pin data if the current instruction is EXTEST or SAMPLE. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the *Exit1-DR* state if TMS is HIGH or the *Shift-DR* state if TMS is LOW.

6.4.6 SHIFT-DR STATE

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the *Exit1-DR* state if TMS is HIGH or remains in the *Shift-DR* state if TMS is LOW.

6.4.7 EXIT1-DR STATE

This is a temporary state. If TMS is held HIGH, a rising edge applied to TCK while in this state causes the controller to enter the *Update-DR* state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the *Pause-DR* state.

The test data register selected by the current instruction retains its previous state unchanged. The instruction does not change in this state.

6.4.8 PAUSE-DR STATE

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. This might be necessary, for example, to allow the tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous state. The instruction does not change in this state.

The controller remains in this state as long as TMS is LOW. When TMS goes HIGH and a rising edge is applied to TCK, the controller moves to the *Exit2-DR* state.

6.4.9 EXIT2-DR STATE

This is a temporary state. If TMS is held HIGH and a rising edge is applied to TCK, the scanning process terminates, and the TAP controller enters the *Update-DR* state. If TMS is held LOW and a rising edge is applied to TCK, the controller enters the *Shift-DR* state.

The test data register selected by the current instruction retains its previous state unchanged. The instruction does not change in this state.

6.4.10 UPDATE-DR STATE

The BSR register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE instructions. When the TAP controller is in this state and the BSR register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in test data registers selected by the current instruction retain their previous state unchanged. The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the *Select-DR-Scan* state if TMS is held HIGH or the *Run-Test/Idle* state if TMS is held LOW.

6.4.11 CAPTURE-IR STATE

In this controller state the shift register contained in the instruction register loads the fixed value 0001 on the rising edge of TCK.

The test data register selected by the current instruction retains its previous state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the *Exit1-IR* state if TMS is held HIGH or the *Shift-IR* state if TMS is held LOW.

6.4.12 SHIFT-IR STATE

In this state, the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruction retains its previous state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the *Exit1-IR* state if TMS is held HIGH or remains in the *Shift-IR* state if TMS is held LOW.

6.4.13 EXIT1-IR STATE

This is a temporary state. If TMS is held HIGH, a rising edge applied to TCK while in this state causes the controller to enter the *Update-IR* state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the *Pause-IR* state.

The test data register selected by the current instruction retains its previous state unchanged. The instruction does not change in this state, and the instruction register retains its state.

6.4.14 PAUSE-IR STATE

This state allows the shifting of the instruction register to be temporarily halted.

The test data register selected by the current instruction retains its previous state. The instruction does not change in this state, and the instruction register retains its state.

The controller remains in this state as long as TMS is LOW. When TMS goes HIGH and a rising edge is applied to TCK, the controller moves to the *Exit2-IR* state.

6.4.15 EXIT2-IR STATE

This is a temporary state. If TMS is held HIGH and a rising edge is applied to TCK, the scanning process terminates, and the TAP controller enters the *Update-IR* state. If TMS is held LOW and a rising edge is applied to TCK, the controller enters the *Shift-IR* state.

The test data register selected by the current instruction retains its previous state unchanged. The instruction does not change in this state, and the instruction register retains its state.

6.4.16 UPDATE-IR STATE

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain the previous state.

6.5 Boundary Scan Register Cell Ordering

Figure 6.4 shows the order of cells in the BSR. There are 150 cells including TDO. TDI is *not* a BSR cell.

The DCTL, ACTL, TCTL, and OCTL cells do not correspond to pins of the i860 XP microprocessor; rather, they control the bidirectional and tristate pins:

DCTL D63–D0, DP7–DP0

ACTL A31–A3

TCTL Tristate outputs: ADS#, BE7#–BE0#, CACHE#, CTYP, D/C#, KB0, KB1, LEN, M/IO#, NENE#, PCD, PCYC, PWT, W/R#

OCTL Outputs not floated in normal operation: BREQ, HIT#, HITM#, HLDA, LOCK#, PCHK#

If a value of one is loaded into any of these control latches, the associated pins will not drive the external bus while running EXTEST.

The values of DCTL, ACTL, TCTL, and OCTL are *undefined* during the SAMPLE instruction.

The values and direction of I/O and outputs do not change during the scanning process (that is, during *Shift-DR* states). They only change after scanning is completed (in the *Update-DR* state).

The decision table, Table 6.3, defines how the boundary scan instructions EXTEST and SAMPLE/PRELOAD utilize BSR.

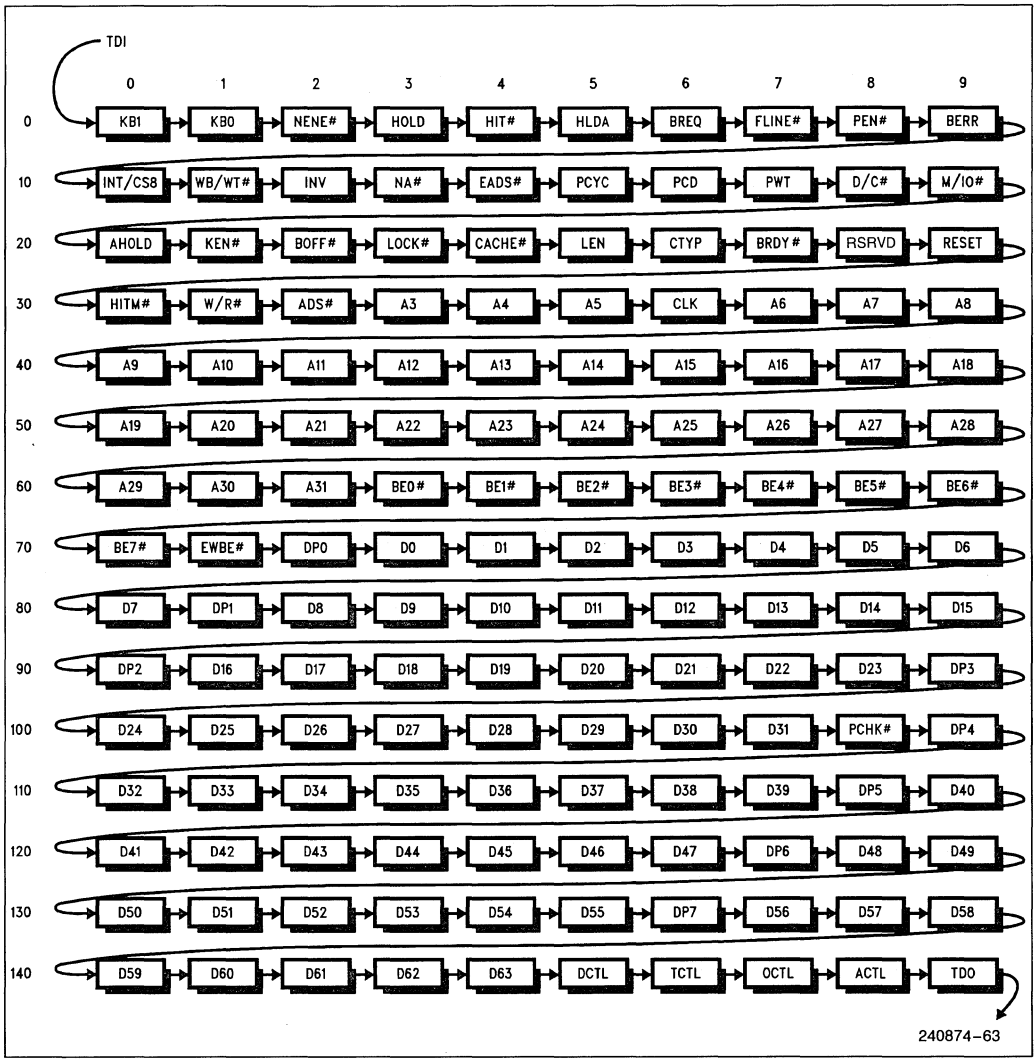


Figure 6.4. Boundary Scan Register Ordering

6.6 TAP Controller Initialization

TAP can be initialized by applying a high signal level on the TMS input for five periods of TCK or by activating the TRST# input pin. TCK does not have to be running in order to initialize TAP with the TRST# pin. TRST# is provided with an internal pull-up resistor; so, even if an open circuit fault occurs, the TAP logic can still be used.

7.0 MECHANICAL DATA

Figures 7.1 and 7.2 show the locations of pins; Tables 7.1 and 7.2 help to locate pin identifiers.

Table 6.3. Instruction Functions

Instruction:	**EXTEST**		**SAMPLE/PRELOAD**	
Control Cell:	**LOW**	**HIGH**	**LOW**	**HIGH**
Input BSR cells . . .	. . . sample values driven to processor by system		. . . sample values driven to processor by system	
Values of input cells used by processor?	NO		NO	
Output BSR cells . . .	. . . drive output pins with cell values		. . . sample values driven by processor	
Input/output BSR cells:	Treat as output	Treat as input	Treat as output	Treat as input

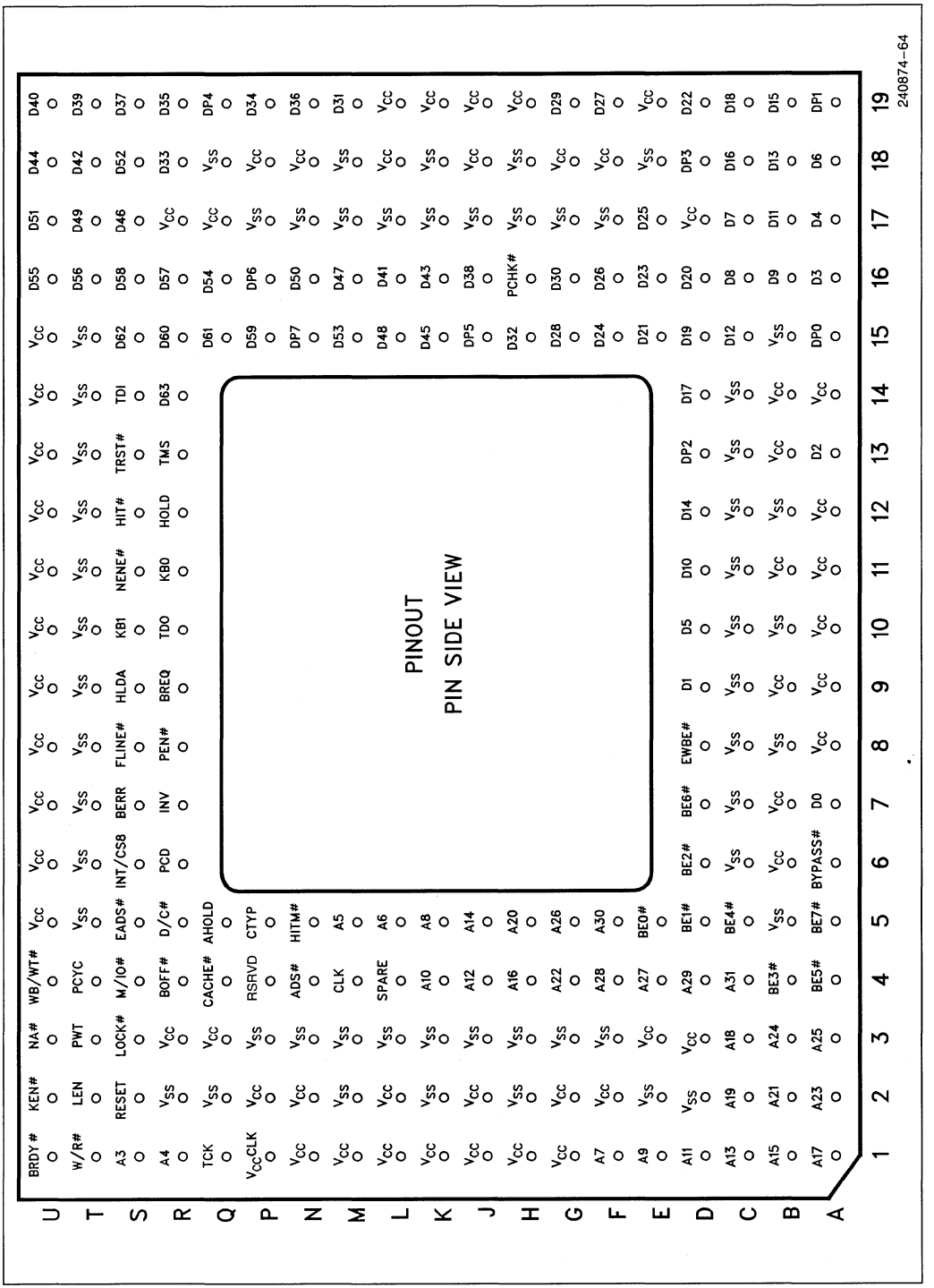


Figure 7.1. i860™ XP Microprocessor Pin Configuration—View from Pin Side

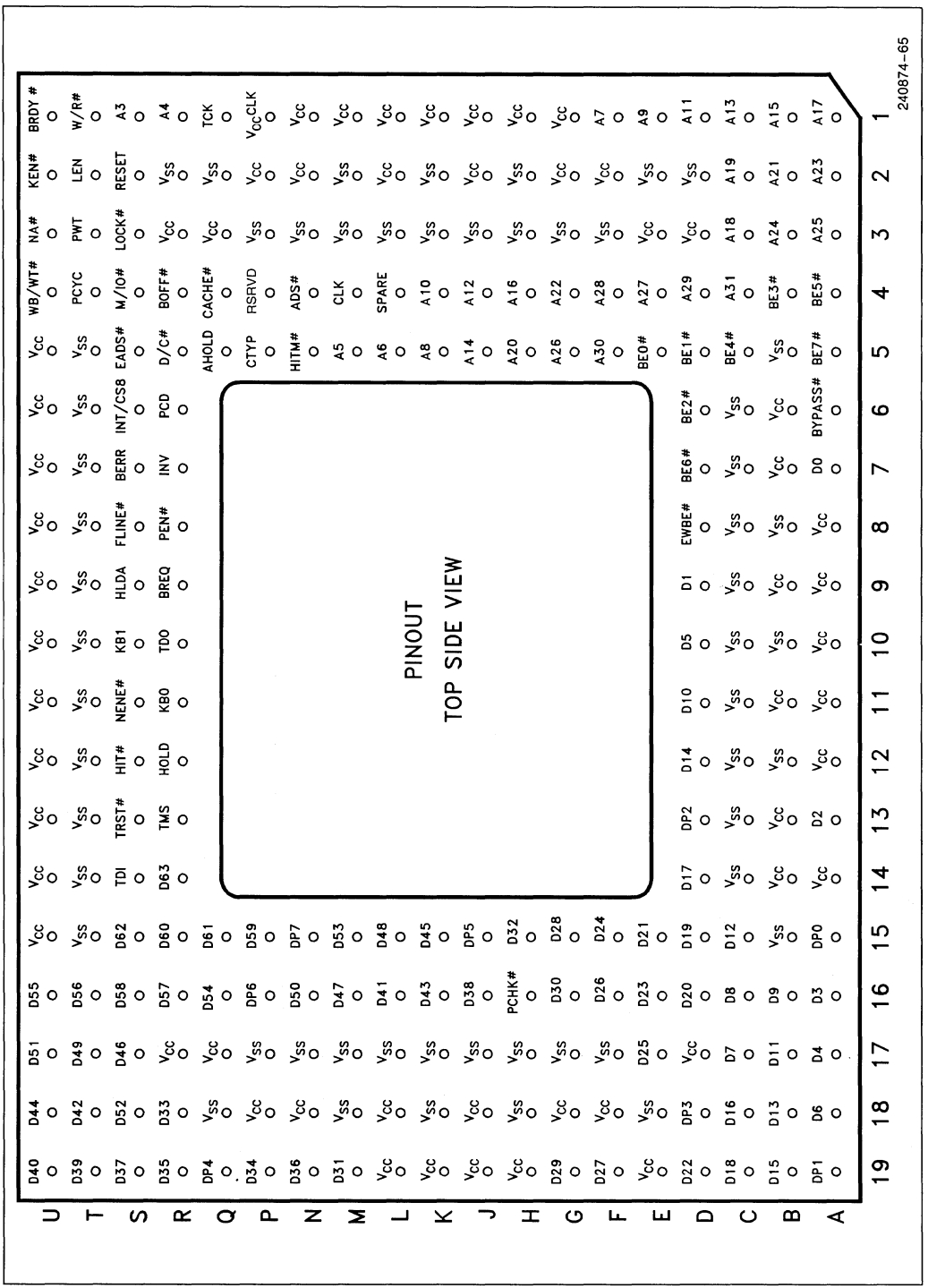


Figure 7.2. i860™ XP Microprocessor Pin Configuration—View from Top Side

Table 7.1. Pin Cross Reference by Location

Location	Signal	Location	Signal	Location	Signal	Location	Signal
A01	A17	C15	D12	G18	V_{CC}	N01	V_{CC}
A02	A23	C16	D8	G19	D29	N02	V_{CC}
A03	A25	C17	D7	H01	V_{CC}	N03	V_{SS}
A04	BE5#	C18	D16	H02	V_{SS}	N04	ADS#
A05	BE7#	C19	D18	H03	V_{SS}	N05	HITM#
A06	BYPASS#	D01	A11	H04	A16	N15	DP7
A07	D0	D02	V_{SS}	H05	A20	N16	D50
A08	V_{CC}	D03	V_{CC}	H15	D32	N17	V_{SS}
A09	V_{CC}	D04	A29	H16	PCHK#	N18	V_{CC}
A10	V_{CC}	D05	BE1#	H17	V_{SS}	N19	D36
A11	V_{CC}	D06	BE2#	H18	V_{SS}	P01	V_{CC} CLK
A12	V_{CC}	D07	BE6#	H19	V_{CC}	P02	V_{CC}
A13	D2	D08	EWBE#	J01	V_{CC}	P03	V_{SS}
A14	V_{CC}	D09	D1	J02	V_{CC}	P04	RSRVD
A15	DP0	D10	D5	J03	V_{SS}	P05	CTYP
A16	D3	D11	D10	J04	A12	P15	D59
A17	D4	D12	D14	J05	A14	P16	DP6
A18	D6	D13	DP2	J15	DP5	P17	V_{SS}
A19	DP1	D14	D17	J16	D38	P18	V_{CC}
B01	A15	D15	D19	J17	V_{SS}	P19	D34
B02	A21	D16	D20	J18	V_{CC}	Q01	TCK
B03	A24	D17	V_{CC}	J19	V_{CC}	Q02	V_{SS}
B04	BE3#	D18	DP3	K01	V_{CC}	Q03	V_{CC}
B05	V_{SS}	D19	D22	K02	V_{SS}	Q04	CACHE#
B06	V_{CC}	E01	A9	K03	V_{SS}	Q05	AHOLD
B07	V_{CC}	E02	V_{SS}	K04	A10	Q15	D61
B08	V_{SS}	E03	V_{CC}	K05	A8	Q16	D54
B09	V_{CC}	E04	A27	K15	D45	Q17	V_{CC}
B10	V_{SS}	E05	BE0#	K16	D43	Q18	V_{SS}
B11	V_{CC}	E15	D21	K17	V_{SS}	Q19	DP4
B12	V_{SS}	E16	D23	K18	V_{SS}	R01	A4
B13	V_{CC}	E17	D25	K19	V_{CC}	R02	V_{SS}
B14	V_{CC}	E18	V_{SS}	L01	V_{CC}	R03	V_{CC}
B15	V_{SS}	E19	V_{CC}	L02	V_{CC}	R04	BOFF#
B16	D9	F01	A7	L03	V_{SS}	R05	D/C#
B17	D11	F02	V_{CC}	L04	SPARE	R06	PCD
B18	D13	F03	V_{SS}	L05	A6	R07	INV
B19	D15	F04	A28	L15	D48	R08	PEN#
C01	A13	F05	A30	L16	D41	R09	BREQ
C02	A19	F15	D24	L17	V_{SS}	R10	TDO
C03	A18	F16	D26	L18	V_{CC}	R11	KB0
C04	A31	F17	V_{SS}	L19	V_{CC}	R12	HOLD
C05	BE4#	F18	V_{CC}	M01	V_{CC}	R13	TMS
C06	V_{SS}	F19	D27	M02	V_{SS}	R14	D63
C07	V_{SS}	G01	V_{CC}	M03	V_{SS}	R15	D60
C08	V_{SS}	G02	V_{CC}	M04	CLK	R16	D57
C09	V_{SS}	G03	V_{SS}	M05	A5	R17	V_{CC}
C10	V_{SS}	G04	A22	M15	D53	R18	D33
C11	V_{SS}	G05	A26	M16	D47	R19	D35
C12	V_{SS}	G15	D28	M17	V_{SS}	S01	A3
C13	V_{SS}	G16	D30	M18	V_{SS}	S02	RESET
C14	V_{SS}	G17	V_{SS}	M19	D31	S03	LOCK#

Table 7.1. Pin Cross Reference by Location (Continued)

Location	Signal	Location	Signal	Location	Signal	Location	Signal
S04	M/IO#	S18	D52	T13	V_{SS}	U08	V_{CC}
S05	EADS#	S19	D37	T14	V_{SS}	U09	V_{CC}
S06	INT/CS8	T01	W/R#	T15	V_{SS}	U10	V_{CC}
S07	BERR	T02	LEN	T16	D56	U11	V_{CC}
S08	FLINE#	T03	PWT	T17	D49	U12	V_{CC}
S09	HLDA	T04	PCYC	T18	D42	U13	V_{CC}
S10	KB1	T05	V_{SS}	T19	D39	U14	V_{CC}
S11	NENE#	T06	V_{SS}	U01	BRDY#	U15	V_{CC}
S12	HIT#	T07	V_{SS}	U02	KEN#	U16	D55
S13	TRST#	T08	V_{SS}	U03	NA#	U17	D51
S14	TDI	T09	V_{SS}	U04	WB/WT#	U18	D44
S15	D62	T10	V_{SS}	U05	V_{CC}	U19	D40
S16	D58	T11	V_{SS}	U06	V_{CC}		
S17	D46	T12	V_{SS}	U07	V_{CC}		

Table 7.2. Pin Cross Reference by Pin Name

Signal	Location	Signal	Location	Signal	Location	Signal	Location
A3	S01	AHOLD	Q05	D21	E15	D49	T17
A4	R01	BE0#	E05	D22	D19	D4	A17
A5	M05	BE1#	D05	D23	E16	D50	N16
A6	L05	BE2#	D06	D24	F15	D51	U17
A7	F01	BE3#	B04	D25	E17	D52	S18
A8	K05	BE4#	C05	D26	F16	D53	M15
A9	E01	BE5#	A04	D27	F19	D54	Q16
A10	K04	BE6#	D07	D28	G15	D55	U16
A11	D01	BE7#	A05	D29	G19	D56	T16
A12	J04	BERR	S07	D2	A13	D57	R16
A13	C01	BOFF#	R04	D30	G16	D58	S16
A14	J05	RSRVD	P04	D31	M19	D59	P15
A15	B01	BRDY#	U01	D32	H15	D5	D10
A16	H04	BREQ	R09	D33	R18	D60	R15
A17	A01	CACHE#	Q04	D34	P19	D61	Q15
A18	C03	CLK	M04	D35	R19	D62	S15
A19	C02	CTYP	P05	D36	N19	D63	R14
A20	H05	D0	A07	D37	S19	D6	A18
A21	B02	D10	D11	D38	J16	D7	C17
A22	G04	D11	B17	D39	T19	D8	C16
A23	A02	D12	C15	D3	A16	D9	B16
A24	B03	D13	B18	D40	U19	D/C#	R05
A25	A03	D14	D12	D41	L16	DP0	A15
A26	G05	D15	B19	D42	T18	DP1	A19
A27	E04	D16	C18	D43	K16	DP2	D13
A28	F04	D17	D14	D44	U18	DP3	D18
A29	D04	D18	C19	D45	K15	DP4	Q19
A30	F05	D19	D15	D46	S17	DP5	J15
A31	C04	D1	D09	D47	M16	DP6	P16
ADS#	N04	D20	D16	D48	L15	DP7	N15

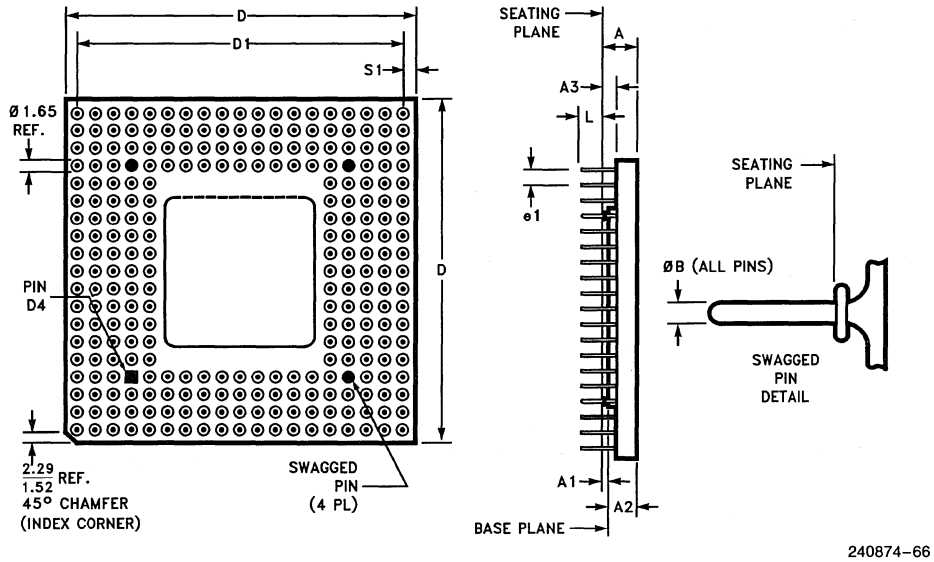
Table 7.2. Pin Cross Reference by Pin Name (Continued)

Signal	Location	Signal	Location	Signal	Location	Signal	Location
EADS#	S05	V_{CC}	B06	V_{CC}	R17	V_{SS}	H17
FLINE#	S08	V_{CC}	B07	V_{CC}	U05	V_{SS}	H18
HIT#	S12	V_{CC}	B09	V_{CC}	U06	V_{SS}	J03
HITM#	N05	V_{CC}	B11	V_{CC}	U07	V_{SS}	J17
HLDA	S09	V_{CC}	B13	V_{CC}	U08	V_{SS}	K02
HOL	R12	V_{CC}	B14	V_{CC}	U09	V_{SS}	K03
INT/CS8	S06	V_{CC}	D03	V_{CC}	U10	V_{SS}	K17
INV	R07	V_{CC}	D17	V_{CC}	U11	V_{SS}	K18
KB0	R11	V_{CC}	E03	V_{CC}	U12	V_{SS}	L03
KB1	S10	V_{CC}	E19	V_{CC}	U13	V_{SS}	L17
KEN#	U02	V_{CC}	F02	V_{CC}	U14	V_{SS}	M02
LEN	T02	V_{CC}	F18	V_{CC}	U15	V_{SS}	M03
LOCK#	S03	V_{CC}	G01	$V_{CC}CLK$	P01	V_{SS}	M17
M/IO#	S04	V_{CC}	G02	V_{SS}	B05	V_{SS}	M18
NA#	U03	V_{CC}	G18	V_{SS}	B08	V_{SS}	N03
NENE#	S11	V_{CC}	H01	V_{SS}	B10	V_{SS}	N17
PC	R06	V_{CC}	H19	V_{SS}	B12	V_{SS}	P03
PCHK#	H16	V_{CC}	J01	V_{SS}	B15	V_{SS}	P17
PCYC	T04	V_{CC}	J02	V_{SS}	C06	V_{SS}	Q02
PEN#	R08	V_{CC}	J18	V_{SS}	C07	V_{SS}	Q18
PWT	T03	V_{CC}	J19	V_{SS}	C08	V_{SS}	R02
RESET	S02	V_{CC}	K01	V_{SS}	C09	V_{SS}	T05
SPARE	L04	V_{CC}	K19	V_{SS}	C10	V_{SS}	T06
EWBE#	D08	V_{CC}	L01	V_{SS}	C11	V_{SS}	T07
BYPASS#	A06	V_{CC}	L02	V_{SS}	C12	V_{SS}	T08
TCK	Q01	V_{CC}	L18	V_{SS}	C13	V_{SS}	T09
TDI	S14	V_{CC}	L19	V_{SS}	C14	V_{SS}	T10
TDO	R10	V_{CC}	M01	V_{SS}	D02	V_{SS}	T11
TMS	R13	V_{CC}	N01	V_{SS}	E02	V_{SS}	T12
TRST#	S13	V_{CC}	N02	V_{SS}	E18	V_{SS}	T13
V_{CC}	A08	V_{CC}	N18	V_{SS}	F03	V_{SS}	T14
V_{CC}	A09	V_{CC}	P02	V_{SS}	F17	V_{SS}	T15
V_{CC}	A10	V_{CC}	P18	V_{SS}	G03	W/R#	T01
V_{CC}	A11	V_{CC}	Q03	V_{SS}	G17	WB/WT#	U04
V_{CC}	A12	V_{CC}	Q17	V_{SS}	H02		
V_{CC}	A14	V_{CC}	R03	V_{SS}	H03		

Table 7.3. Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is noncumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1", and "C" are nominal.
5. Details of Pin 1 identifier are optional.



240874-66

Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		.140	.180	
A1	0.64	1.14	Solid Lid	.025	.045	Solid Lid
A2	2.79	3.56	Solid Lid	.110	.140	Solid Lid
A3	1.14	1.40		.045	.055	
B	0.43	0.51		.017	.020	
D	49.28	49.96		1.940	1.967	
D1	45.59	45.85		1.795	1.805	
e1	2.29	2.79		.090	.110	
L	2.54	3.30		.100	.130	
N	240	280		240	280	
S1	1.52	2.54		.060	.100	
ISSUE	9/90					

Figure 7.3. 262-Lead Ceramic PGA Package Dimensions

8.0 PACKAGE THERMAL SPECIFICATIONS

For this section, let:

- P = maximum power consumption
- T_C = case temperature
- T_A = ambient air temperature
- θ_{CA} = thermal resistance from case to ambient air
- θ_{JC} = thermal resistance from junction to case
- θ_{JA} = thermal resistance from junction to ambient air

The i860 XP microprocessor is specified for operation when T_C is within the range of 0°C-85°C. T_C may be measured in any environment to determine whether the i860 XP microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

T_A can be calculated from θ_{CA} with the following equation:

$$T_A = T_C - P * \theta_{CA}$$

Typical values for θ_{CA} at various airflows and for θ_{JC} are given in Table 8.1 for the 1.95 sq. in., 262 pin, ceramic PGA. θ_{JC} is shown so that θ_{JA} can be calculated by:

$$\theta_{JA} = \theta_{JC} - \theta_{CA}$$

Note that θ_{JC} with a heatsink differs from θ_{JC} without a heatsink because case temperature is measured differently. Case temperature for θ_{JC} with heatsink is measured at the center of the heat fin base. Case temperature for θ_{JC} without heatsink is measured at the center of the package top surface.

Table 8.2 shows the maximum T_A allowable (without exceeding T_C) at various airflows.

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *D.C. Characteristics* of section 9.

Figure 8.1 gives typical I_{CC} derating with case temperature. For more information on heat sinks, measurement techniques, or package characteristics, refer to *Intel Packaging Handbook*, order number 240800.

Table 8.1. Thermal Resistance—In °C/Watt

	θ_{JC}	θ_{CA} as a Function of Airflow — ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
With Heat Sink*	1.6	10.1	6.3	4.3	3.2	2.5	2.2
Without Heat Sink	1.0	13.5	11.0	8.0	6.5	5.5	5.0

NOTE:
* Nine-fin, unidirectional heat sink (fin dimensions: 0.250″ height, 0.040″ fin width, 0.100″ center-to-center spacing, 1.730″ length)

Table 8.2. Maximum T_A at Various Airflows—In °C

		Airflow — ft/min (m/sec)					
	f_{CLK} (MHz)	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink*	50	24	47	59	66	70	72
T_A without Heat Sink	50	4	19	37	46	52	55
T_A with Heat Sink*	40	34.5	53.5	63.5	69	72.5	74
T_A without Heat Sink	40	17.5	30	45	52.5	57.5	60

NOTE:
* Nine-fin, unidirectional heat sink (fin dimensions: 0.250″ height, 0.040″ fin width, 0.100″ center-to-center spacing, 1.730″ length)

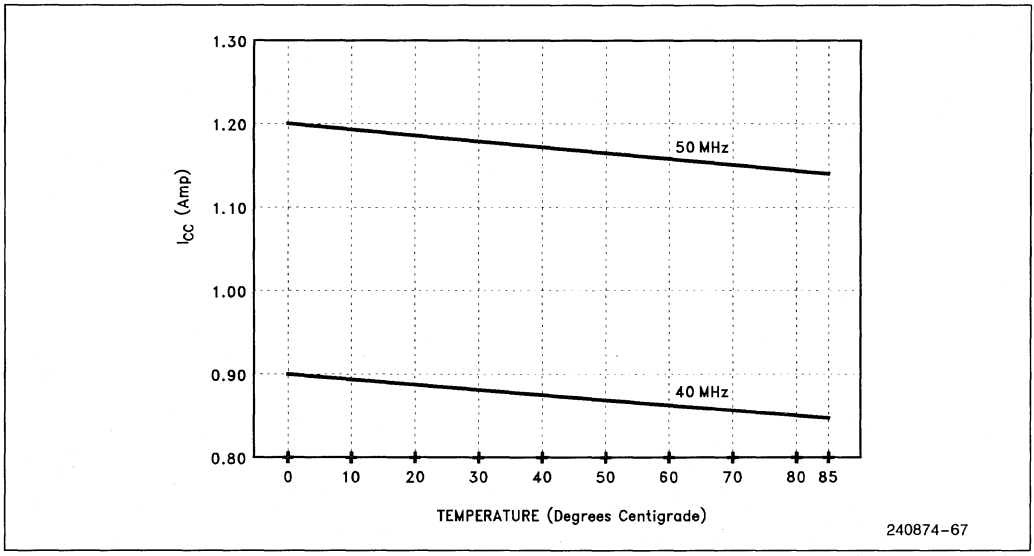


Figure 8.1. I_{CC} Derating with Case Temperature

9.0 ELECTRICAL DATA

All TTL input and output timings are specified relative to the 1.5V input level of the rising edge of CLK and refer to the point that the signal reaches 1.5V.

NOTE:
This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice.

9.1 Absolute Maximum Ratings

Case Temperature T_C under Bias .....0°C to 85°C

Storage Temperature .....−65°C to +150°C

Voltage on Any Pin with Respect to Ground.....−0.5 to V_{CC}+0.5V

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

9.2 D.C. Characteristics

Table 9.1. D.C. Characteristics Operating Conditions: V_{CC} = 5V ±5%; T_C = 0°C to 85°C

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Input LOW voltage (TTL)	−0.3	+0.8	V	
V_{IH}	Input HIGH voltage (TTL)	2.0	V_{CC}+0.3	V	
V_{OL}	Output LOW voltage (TTL)		0.45	V	1
V_{OH}	Output HIGH voltage (TTL)	2.4		V	2
I_{CC}	Power supply current (@ 50 MHz)		1.2	Amp	3
I_{CC}	Power supply current (@40 MHz)		1.0	Amp	3
I_{LI}	Input leakage current		±15	μA	4
I_{LIP}	Input leakage current (pull-up)		−400	μA	5
I_{LO}	Output leakage current		±15	μA	6
C_{IN}	Input capacitance		11.5	pF	7
C_O	I/O or output capacitance		14	pF	7
C_{CLK}	Clock capacitance		20	pF	7

NOTES:
1. This parameter is measured with current load of 5 mA.
2. This parameter is measured with current load of 1 mA. Typical value is V_{CC} − 0.45V.
3. Measured at 50 MHz and V_{CC} = 5V.
4. This parameter is for inputs without pullups. V_{CC} is on, and 0.45V ≤ V_{IN} ≤ V_{CC} + 0.45V.
5. This parameter is for inputs with pullups and V_{IL} = 0.45V. Note that if the pull-ups are put in high-impedance state via the DCTL boundary scan cell that also tri-states the data outputs, then the leakage is ±15 μA.
6. 0.45V ≤ V_{IN} ≤ V_{CC} − 0.45V.
7. These parameters are not tested; they are guaranteed by design characterization.

9.3 A.C. Characteristics

Table 9.2. 50 MHz A.C. Characteristics Operating Conditions: V_{CC} = 5V ±5%; T_C = 0°C to 85°C

Symbol	Parameter	Fig	Load (pF)(a)	TTL Level	
				Min (ns)	Max (ns)
tc	CLK period	9.1		20	40
ttc	TCK period	9.2		40	1000
	CLK stability	9.1			0.1%
tch	CLK high time	9.1		7	
tcl	CLK low time	9.1		7	
tr	CLK rise time	9.1			3
tf	CLK fall time	9.1			3
ts	TCK to CLK skew	9.3			±1
ttch	TCK high time	9.2		10	
ttcl	TCK low time	9.2		10	
ttcr	TCK rise time	9.2			4
ttcf	TCK fall time	9.2			4
tsu.1	RESET, HOLD, BERR, PEN#, INT/CS8 setup time	9.1		7	
tsu.2	BOFF#, AHOLD, FLINE# KEN#, NA#, EWBE# INV, WB/WT# setup time	9.1		8	
tsu.3	EADS# setup time	9.1		9	
tsu.5	BRDY# setup time	9.1		7.5	
tsu.6	D63–D0, DP7–DP0 setup time	9.1		7.5	
tsu.7	D63–D0, DP7–DP0 setup time (Processor in Late Backoff mode)	9.1		6	
tsu.8	A31–A5 setup time	9.1		11	

Table 9.2. 50 MHz A.C. Characteristics (Continued)
Operating Conditions: V_{CC} = 5V ±5%; T_C = 0°C to 85°C

Symbol	Parameter	Fig	Load (pF)(a)	TTL Level	
				Min (ns)	Max (ns)
ttsu	TDI, TMS, TRST# setup time	9.2		7	
tth	TDI, TMS, TRST# hold time(b)	9.2		1	
th	Hold time all other inputs(c)	9.1		1	
ttco	TDO valid delay and all outputs valid delay in EXTEST mode(f)	9.2	50	2	20
tco.1	A31–A22 valid delay	9.1	50	2	14
tco.2	A21–A3 valid delay	9.1	150(g)	2	14
tco.3	D63–D0, DP7–DP0 valid delay (for 1st write after float)(d)	9.1	50	3	16
tco.4	D63–D0, DP7–DP0 valid delay for 2nd thru 4th writes of burst and for pipelined write-after-write)(d)	9.1	50	3	15
tco.5	BREQ, HLDA, PCHK#, NENE#, KB0, KB1 valid delay	9.1	50	2	15
tco.6	ADS#, W/R#, HITM# valid delay	9.1	150(g)	2	14
tco.7	PWT, PCD, HIT#, CTYP D/C#, M/IO#, PCYC, LOCK# CACHE#, LEN valid delay	9.1	50	2	14
tco.8	BE0#–BE7# valid delay	9.1	100(g)	2	14
tz	Float time all outputs(e)	9.1		2	18
tzt	Float time during boundary scan EXTEST(f)				20

NOTES:
a. Minimum delays are for 20 pF load.
b. These hold times are referenced to the falling edge of TCK.
c. These hold times are referenced to the rising edge of CLK.
d. Output delay for D63–D0, DP7–DP0 is from the CLK after ADS# activation.
e. Float time = delay until maximum output current is less than $\pm I_{LO}$. Float time is not tested.
f. Delay from falling edge of TCK.
g. These pins have high-current buffers designed for interface with cache memory.
In systems that do not use external cache, series resistor termination (located near the output pins of the i860 XP microprocessor) may be required to prevent overshoot and dampen reflections. A resistor of approximately Z–15Ω is recommended to match the impedance of the driver to the board net, where Z is the impedance of the net, including loads, trace, and parallel termination.

Table 9.3. 40 MHz A.C. Characteristics Operating Conditions: V_{CC} = 5V ±5%; T_C = 0°C to 85°C

Symbol	Parameter	Fig	Load (pF)[a]	TTL Level	
				Min (ns)	Max (ns)
tc	CLK Period	9.1		25	40
ttc	TCK Period CLK Stability	9.2 9.1		50	1000 0.1%
tch	CLK High Time	9.1		7	
tcl	CLK Low Time	9.1		7	
tr	CLK Rise Time	9.1			3
tf	CLK Fall Time	9.1			3
ts	TCK to CLK Skew	9.3			±1
ttch	TCK high time	9.2		10	
ttcl	TCK low time	9.2		10	
ttcr	TCK rise time	9.2			4
ttcf	TCK fall time	9.2			4
tsu.1	RESET, HOLD, BERR, PEN#, INT/CS8 Setup Time	9.1		8	
tsu.2	BOFF#, AHOLD, FLINE#, KEN#, NA#, EWBE#, INV, WB/WT# Setup Time	9.1		8	
tsu.3	EADS# Setup Time	9.1		11	
tsu.5	BRDY# Setup Time	9.1		8	
tsu.6	D63–D0, DP7–DP0 Setup Time	9.1		10	
tsu.7	D63–D0, DP7–DP0 Setup Time (Processor in Late Backoff Mode)	9.1		8.5	
tsu.8	A31–A15 Setup Time	9.1		11	
ttsu	TDI, TMS, TRST# Setup Time	9.2		8	
tth	TDI, TMS, TRST# Hold Time[b]	9.2		3	
th	Hold Time, All Other Inputs[c]	9.1		3	
ttco	TDO Valid Delay and All Outputs Valid Delay in EXTEST Mode[f]	9.2	50	2	20
tco.1	A31–A22 Valid Delay	9.1	50	2	14
tco.2	A21–A3 Valid Delay	9.1	150[g]	2	14
tco.3	D63–D0, DP7–DP0 Valid Delay (for 1st Write after Float)	9.1	50	2	16
tco.4	D63–D0, DP7–DP0 Valid Delay (for 2nd through 4th Writes of Burst and for Pipelined Write-after-Write)[4]	9.1	50	2	15
tco.5	BREQ, HLDA, PCHK#, NENE#, KB0, KB1 Valid Delay	9.1	50	2	15
tco.6	ADS#, W/R#, HITM# Valid Delay	9.1	150[g]	2	14

Table 9.3. 40 MHz A.C. Characteristics (Continued)
Operating Conditions: V_{CC} = 5V ±5%; T_C = 0°C to 85°C

Symbol	Parameter	Fig	Load (pF)(a)	TTL Level	
				Min (ns)	Max (ns)
tco.7	PWT, PCD, HIT#, CTYP, D/C#, M/IO#, PCYC, LOCK#, CACHE#, LEN Valid Delay	9.1	50	2	14
tco.8	BE0#–BE7# Valid Delay	9.1	100(g)	2	14
tz	Float Time, All Outputs(e)	9.1		2	18
tzt	Float Time during Boundary Scan EXTEST(f)				20

NOTES:
a. Minimum delays are for 20 pF load.
b. These hold times are referenced to the falling edge of TCK.
c. These hold times are referenced to the rising edge of CLK.
d. Output delay for D63–D0, DP7–DP0 is from the CLK after ADS# activation.
e. Float time = delay until maximum output current is less than $\pm I_{LO}$. Float time is not tested.
f. Delay from falling edge of TCK.
g. These pins have high-current buffers designed for interface with cache memory.
In systems that do not use external cache, series resistor termination (located near the output pins of the i860 XP microprocessor) may be required to prevent overshoot and dampen reflections. A resistor of approximately Z–15Ω is recommended to match the impedance of the driver to the board net, where Z is the impedance of the net, including loads, trace, and parallel termination.

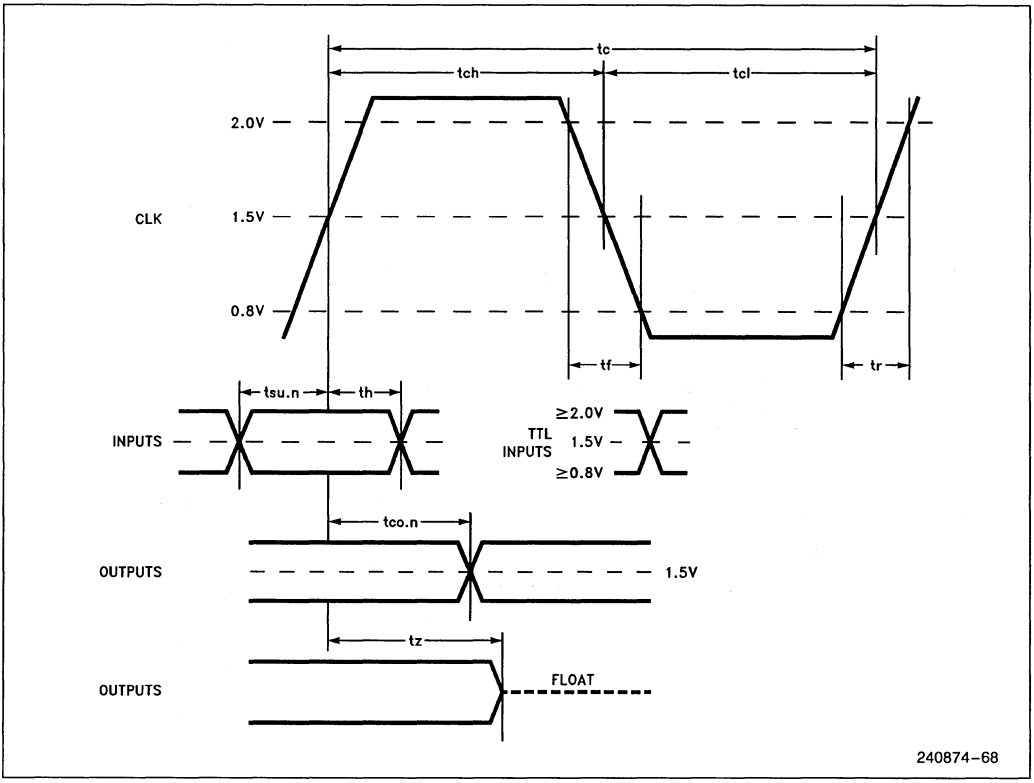


Figure 9.1. CLK, Input, and Output Timings

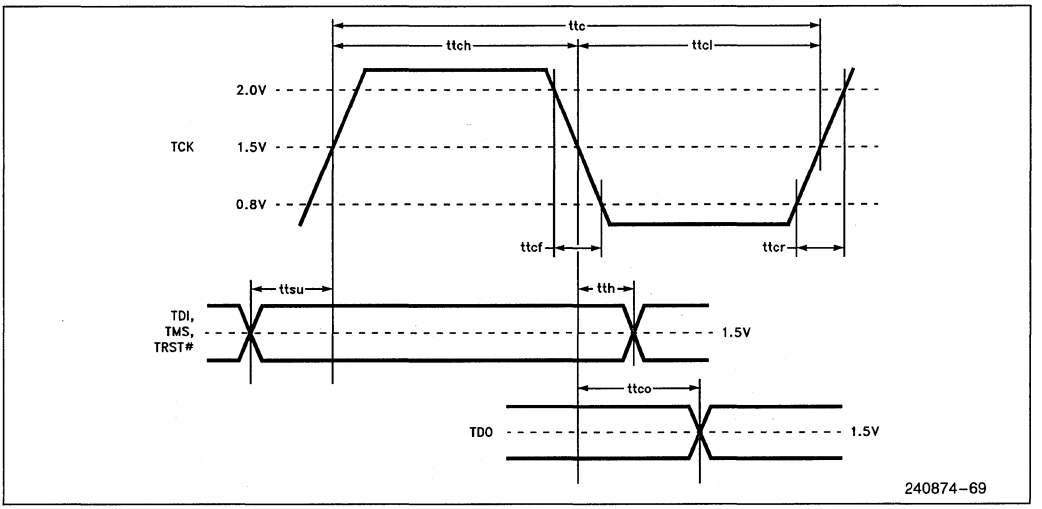


Figure 9.2. TAP Signal Timings

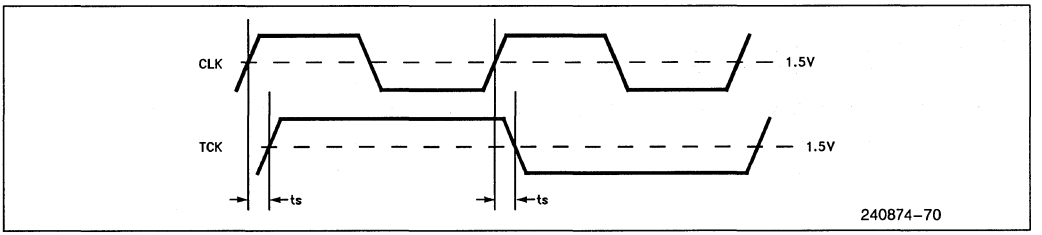
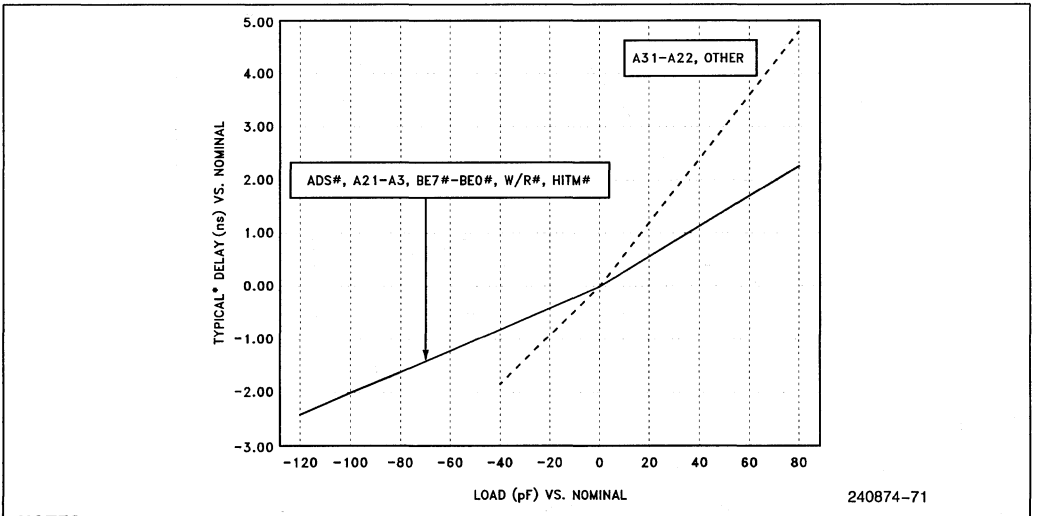
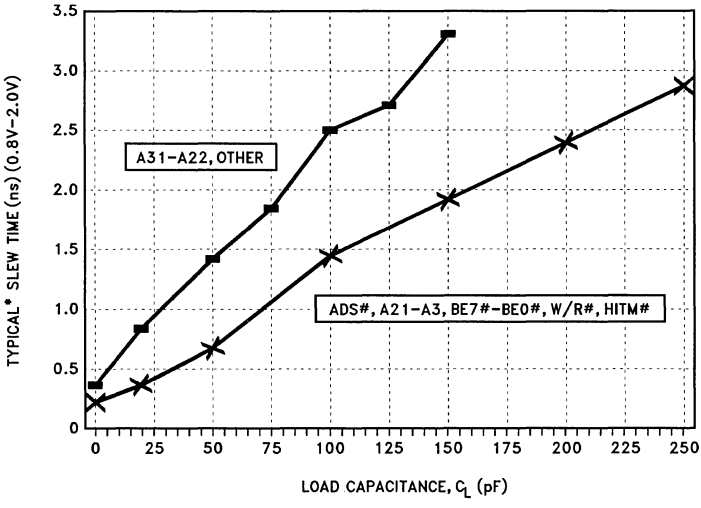


Figure 9.3. TCK to CLK Skew



NOTES:
Graphs are not linear outside the C_L range shown.
NOMINAL = Nominal value given in the AC Timings table.
*Typical part under worst-case conditions.

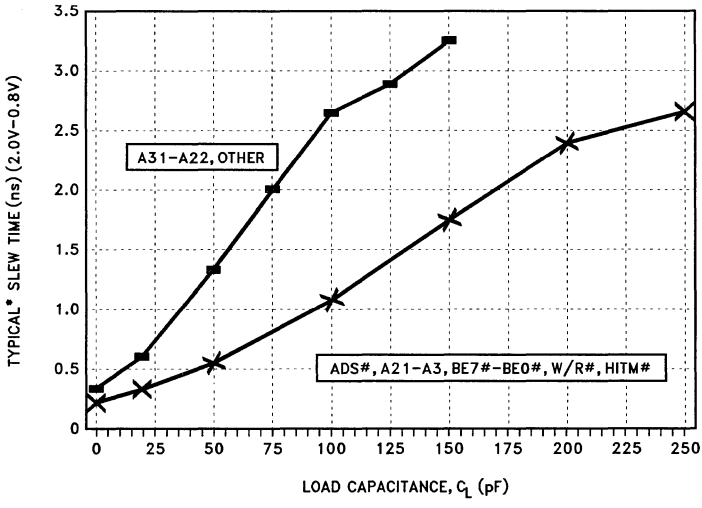
Figure 9.4. Typical Output Delay vs. Load Capacitance under Worst-Case Conditions



240874-81

NOTES:
Graphs are not linear outside the C_L range shown.
*Typical part under worst-case conditions.

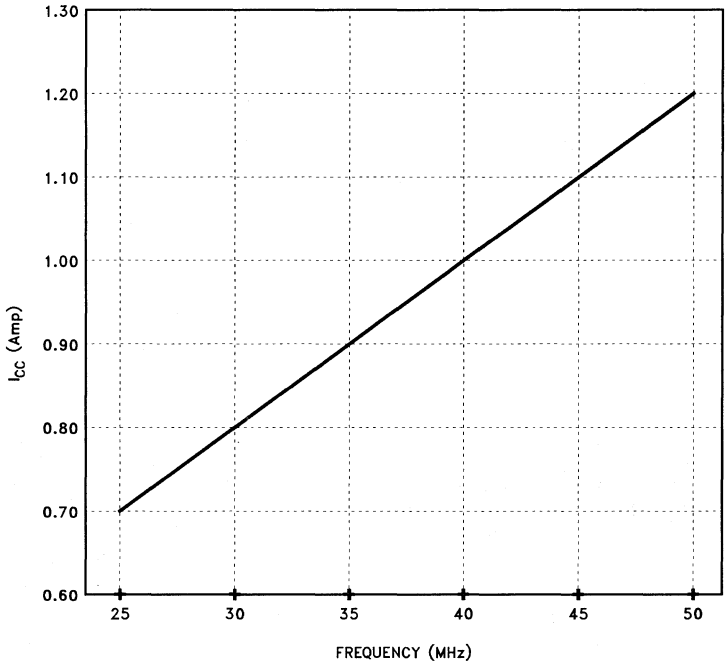
Figure 9.5a. Typical Slew Time vs. Load Capacitance under Worst-Case Conditions (Rising Voltage)



240874-82

NOTES:
Graphs are not linear outside the C_L range shown.
*Typical part under worst-case conditions.

Figure 9.5b. Typical Slew Time vs. Load Capacitance under Worst-Case Conditions (Falling Voltage)



NOTES:
Graph is not linear outside the frequency range shown.
*Worst-case supply current at 5V.

Figure 9.6. Typical I_{CC} vs. Frequency

10.0 INSTRUCTION SET

Key to abbreviations:

For register operands, the abbreviations that describe the operands are composed of two parts. The first part describes the type of register:

c One of the control registers **fir**, **psr**, **epsr**, **dirbase**, **db**, **fsr**, **bear**, **ccr**, **p0**, **p1**, **p2**, or **p3**

f One of the floating-point registers: **f0** through **f31**

i One of the integer registers: **r0** through **r31**

The second part identifies the field of the machine instruction into which the operand is to be placed:

src1 The first of the two source-register designators, which may be either a register or a 16-bit immediate constant or address offset. The immediate value is zero-extended for logical operations and is sign-extended for add and subtract operations (including **addu** and **subu**) and for all addressing calculations.

src1ni Same as *src1* except that no immediate constant or address offset value is permitted.

src1s Same as *src1* except that the immediate constant is a 5-bit value that is zero-extended to 32 bits.

src2 The second of the two source-register designators.

dest The destination register designator.

Thus, the operand specifier *isrc2*, for example, means that an integer register is used and that the encoding of that register must be placed in the *src2* field of the machine instruction.

Other (nonregister) operands are specified by a one-part abbreviation that represents both the type of operand required and the instruction field into which the value of the operand is placed:

#const A 16-bit immediate constant or address offset that the i860 XP microprocessor sign-extends to 32 bits when computing the effective address.

lbroff A signed, 26-bit, immediate, relative branch offset.

sbroff A signed, 16-bit, immediate, relative branch offset.

brx A function that computes the target address by shifting the offset (either *lbroff* or *sbroff*) left by two bits, sign-extending it to 32 bits, and adding the result to the current instruction pointer plus four. The resulting target address may lie anywhere within the address space.

Table 10.1. Precision Specification

Suffix	Source Precision	Result Precision
.ss	single	single
.sd	single	double
.dd	double	double
.ds	double	single

Unless otherwise specificed, floating-point operations accept single- or double-precision source operands and produce a result of equal or greater precision. Both input operands must have the same precision. The source and result precision are specified by a two-letter suffix to the mnemonic of the operation.

Other abbreviations include:

.p Precision specification **.ss**, **.sd**, or **.dd** (**.ds** not permitted). Refer to Table 10.1.

.r Precision specification **.ss**, **.sd**, **.ds**, or **.dd**. Refer to Table 10.1.

.v **.sd** or **.dd** Refer to Table 10.1.

.w **.ss** or **.dd**. Refer to Table 10.1.

.x **.b** (8 bits), **.s** (16 bits), or **.l** (32 bits)

.y **.l** (32 bits), **.d** (64 bits), or **.q** (128 bits)

mem.x(address) The memory location indicated by *address* with a size of *x*.

port.x(address) The I/O port indicated by *address* with a size of *x*.

int_vector.x(address) The interrupt vector with a size of *x* returned from I/O port *address*.

PM The pixel mask, which is considered as an array of eight bits PM(7)..PM(0), where PM(0) is the least-significant bit.

10.1 Instruction Definitions in Alphabetical Order

adds *isrc1, isrc2, idest* ..... **Add Signed**
idest ← *isrc1* + *isrc2*
OF ← (bit 31 carry ≠ bit 30 carry)
CC set if *isrc2* + *isrc1* < 0 (signed)
CC clear if *isrc2* + *isrc1* ≥ 0 (signed)

addu *isrc1, isrc2, idest* ..... **Add Unsigned**
idest ← *isrc1* + *isrc2*
OF ← bit 31 carry
CC ← bit 31 carry

and *isrc1, isrc2, idest* ..... **Logical AND**
idest ← *isrc1* and *isrc2*
CC set if result is zero, cleared otherwise

andh *#const, isrc2, idest* ..... **Logical AND High**
idest ← (*#const* shifted left 16 bits) and *isrc2*
CC set if result is zero, cleared otherwise

andnot *isrc1, isrc2, idest* ..... **Logical AND NOT**
idest ← (not *isrc1)* and *isrc2*
CC set if result is zero, cleared otherwise

andnoth *#const, isrc2, idest* ..... **Logical AND NOT High**
idest ← (not (*#const* shifted left 16 bits)) and *isrc2*
CC set if result is zero, cleared otherwise

bc *lbroff* ..... **Branch on CC**

```
IF          CC = 1
THEN        continue execution at brx(lbroff)
FI
```

bc.t *lbroff* ..... **Branch on CC, Taken**

```
IF          CC = 1
THEN        execute one more sequential instruction
            continue execution at brx(lbroff)
ELSE        skip next sequential instruction
FI
```

bla *isrc1ni, isrc2, sbroff* ..... **Branch on LCC and Add**

```
LCC-temp clear if isrc2 + isrc1ni < 0 (signed)
LCC-temp set if isrc2 + isrc1ni ≥ 0 (signed)
isrc2 ← isrc1ni + isrc2
Execute one more sequential instruction
IF          LCC
THEN        LCC ← LCC-temp
            continue execution at brx(sbroff)
ELSE        LCC ← LCC-temp
FI
```

bnc *lbroff* ..... **Branch on Not CC**

```
IF          CC = 0
THEN        continue execution at brx(lbroff)
FI
```

bnc.t *lbroff* ..... **Branch on Not CC, Taken**

```
IF      CC = 0
THEN    execute one more sequential instruction
        continue execution at brx(lbroff)
ELSE    skip next sequential instruction
FI
```

br *lbroff* ..... **Branch Direct Unconditionally**

Execute one more sequential instruction.
Continue execution at *brx(lbroff)*.

bri [*isrc1ni*] ..... **Branch Indirect Unconditionally**

```
Execute one more sequential instruction
IF      any trap bit in psr is set
THEN    copy PU to U, PIM to IM in psr
        clear trap bits
        IF      DS is set and DIM is reset
        THEN    enter dual-instruction mode after executing one
                    instruction in single-instruction mode
        ELSE    IF      DS is set and DIM is set
                THEN    enter single-instruction mode after executing one
                            instruction in dual-instruction mode
                ELSE    IF      DIM is set
                        THEN    enter dual-instruction mode
                                  for next instruction pair
                        ELSE    enter single-instruction mode
                                  for next instruction pair
                        FI
                FI
        FI
FI
Continue execution at address in isrc1ni
    (The original contents of isrc1ni is used even if the next instruction
    modifies isrc1ni. Does not trap if isrc1ni is misaligned.)
```

bte *isrc1s, isrc2, sbroff* ..... **Branch If Equal**

```
IF      isrc1s = isrc2
THEN    continue execution at brx(sbroff)
FI
```

btne *isrc1s, isrc2, sbroff* ..... **Branch If Not Equal**

```
IF      isrc1s ≠ isrc2
THEN    continue execution at brx(sbroff)
FI
```

call *lbroff* ..... **Subroutine Call**

r1 ← address of next sequential instruction + 4 (or + 8 in dual mode)
Execute one more sequential instruction
Continue execution at *brx(lbroff)*

calli [*isrc1ni*] ..... **Indirect Subroutine Call**

r1 ← address of next sequential instruction + 4 (or + 8 in dual mode)
Execute one more sequential instruction
Continue execution at address in *isrc1ni*
(The original contents of *isrc1ni* is used even if the next instruction modifies *isrc1ni*. Does not trap if *isrc1ni* is misaligned. The register *isrc1ni* must not be **r1**.)

fadd.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Add**

fdest ← *fsrc1* + *fsrc2*

faddp *fsrc1, fsrc2, fdest* ..... **Add with Pixel Merge**
fdest ← *fsrc1* + *fsrc2* (using integer arithmetic; 8-byte operands and destination)
Shift and load MERGE register from *fsrc1* + *fsrc2* as defined in Table 10.2

faddz *fsrc1, fsrc2, fdest* ..... **Add with Z Merge**
fdest ← *fsrc1* + *fsrc2* (using integer arithmetic; 8-byte operands and destination)
Shift MERGE right 16 and load fields 31..16 and 63..48 from *fsrc1* + *fsrc2*

famov.r *fsrc1, fdest* ..... **Floating-Point Adder Move**
fdest ← *fsrc1*

fiadd.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Add**
fdest ← *fsrc1* + *fsrc2* (2's complement integer arithmetic)

fisub.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Subtract**
frdest ← *fsrc1* − *fsrc2* (2's complement integer arithmetic)

fix.v *fsrc1, fdest* ..... **Floating-Point to Integer Conversion**
fdest ← 64-bit value with low-order 32 bits equal to integer part of *fsrc1* rounded

Floating-Point Load
fld.y *isrc1(isrc2), fdest* ..... **(Normal)**
fld.y *isrc1(isrc2)*++, *fdest* ..... **(Autoincrement)**
fdest ← mem.y (*isrc1* + *isrc2*)
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

Cache Flush
flush *#const(isrc2)* ..... **(Normal)**
flush *#const(isrc2)*++ ..... **(Autoincrement)**
Write back (if modified) the line in data cache that has address (*#const* + *isrc2*)
80860XR: and set tag value to (*#const* + *isrc2*).
80860XP: and invalidate its virtual and physical tags.
Contents of line undefined.
IF autoincrement
THEN *isrc2* ← *#const* + *isrc2*
FI

fmlow.dd *fsrc1, fsrc2, fdest* ..... **Floating-Point Multiply Low**
fdest ← low-order 53 bits of (*fsrc1* mantissa × *fsrc2* mantissa)
fdest bit 53 ← most significant bit of (*fsrc1*mantissa × *fsrc2* mantissa)

fmov.r *fsrc1, fdest* ..... **Floating-Point Reg-Reg Move**
Assembler pseudo-operation

fmov.ss *fsrc1, fdest*	= **fiadd.ss** *fsrc1*, **f0**, *fdest*
fmov.dd *fsrc1, fdest*	= **fiadd.dd** *fsrc1*, **f0**, *fdest*
fmov.sd *fsrc1, fdest*	= **famov.sd** *fsrc1, fdest*
fmov.ds *fsrc1, fdest*	= **famov.ds** *fsrc1, fdest*

fmul.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Multiply**
fdest ← *fsrc1* × *fsrc2*

fnop ..... **Floating-Point No Operation**
Assembler pseudo-operation
fnop = **shrd r0, r0, r0**

form *fsrc1, fdest* ..... **OR with MERGE Register**
fdest ← *fsrc1* OR MERGE
MERGE ← 0

frcp.p *fsrc2, fdest* ..... **Floating-Point Reciprocal**
fdest ← 1 / *fsrc2* with maximum mantissa error < 2^{-7}

frsqr.p *fsrc2, fdest* ..... **Floating-Point Reciprocal Square Root**
fdest ← 1 / $\sqrt{fsrc2}$ with maximum mantissa error < 2^{-7}

Floating-Point Store
fst.y *fdest, isrc1(isrc2)* ..... **(Normal)**
fst.y *fdest, isrc1(isrc2)+ +* ..... **(Autoincrement)**
mem.y (*isrc2* + *isrc1*) ← *fdest*
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

fsub.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Subtract**
fdest ← *fsrc1* – *fsrc2*

ftrunc.v *fsrc1, fdest* ..... **Floating-Point to Integer Conversion**
fdest ← 64-bit value with low-order 32 bits equal to integer part of *fsrc1*

fxfr *fsrc1, idest* ..... **Transfer F-P to Integer Register**

idest ← *fsrc1*

fzchkl *fsrc1, fsrc2, fdest* ..... **32-Bit Z-Buffer Check**
Consider the 64-bit operands as arrays of two 32-bit
fields *fsrc1*(1)..*fsrc1*(0), *fsrc2*(1)..*fsrc2*(0), and *fdest*(1)..*fdest*(0)
where zero denotes the least-significant field.
PM ← PM shifted right by 2 bits
FOR i = 0 to 1
DO
PM [i + 6] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

fzchks *fsrc1, fsrc2, fdest* ..... **16-Bit Z-Buffer Check**
Consider the 64-bit operands as arrays of four 16-bit
fields *fsrc1*(3)..*fsrc1*(0), *fsrc2*(3)..*fsrc2*(0), and *fdest*(3)..*fdest*(0)
where zero denotes the least-significant field.
PM ← PM shifted right by 4 bits
FOR i = 0 to 3
DO
PM [i + 4] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

intovr ..... **Software Trap on Integer Overflow**
IF OF = 1
THEN generate trap with IT set in **psr**
FI

ixfr *isrc1ni, fdest* ..... **Transfer Integer to F-P Register**
fdest ← *isrc1ni*

ld.c *csrc2, idest* ..... **Load from Control Register**
idest ← *csrc2*

ld.x *isrc1*(*isrc2*), *idest* ..... **Load Integer**
idest ← *mem.x* (*isrc1* + *isrc2*)

ldint.x *isrc2, idest* ..... **Load Interrupt Vector**
idest ← *int_vector.x (isrc2)*
NOTE: Not available with the i860 XR CPU

ldio.x *isrc2, idest* ..... **Load I/O**
idest ← *port.x (isrc2)*
NOTE: Not available with the i860 XR CPU

lock ..... **Begin Interlocked Sequence**
Set BL in **dirbase**.
The next load or store that appears on the bus locks that location.
Disable interrupts until the bus is unlocked.

mov *isrc2, idest* ..... **Register-Register Move**
Assembler pseudo-operation
mov *isrc2, idest* = **shl r0**, *isrc2, idest*

mov *const32, idest* ..... **Constant-to-Register Move**
Assembler pseudo-operation
when 0xFFFF8000 ≤ *const32* < 0x8000 . . .
adds *l%const32*, **r0**, *idest*
otherwise . . .
orh *h%const32*, **r0**, *idest*
or *l%const32, idest, idest*

nop ..... **Core-Unit No Operation**
Assembler pseudo-operation
nop = **shl r0, r0, r0**

or *isrc1, isrc2, idest* ..... **Logical OR**
idest ← *isrc1* OR *isrc2*
CC set if result is zero, cleared otherwise

orh *#const, isrc2, idest* ..... **Logical OR high**
idest ← (*#const* shifted left 16 bits) OR *isrc2*
CC set if result is zero, cleared otherwise

pfadd.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1* + *fsrc2*

pfaddp *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Pixel Merge**
fdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* + *fsrc2*
(using integer arithmetic; 8-byte operands and destination)
Shift, then load MERGE register from *fsrc1* + *fsrc2* as defined in Table 10.2

pfaddz *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Z Merge**
frdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* + *fsrc2*
(using integer arithmetic; 8-byte operands and destination)
Shift MERGE right 16, then load fields 31..16 and 63..48 from*fsrc1* + *fsrc2*

pfam.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add and Multiply**
fdest ← last stage adder result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 + A-op2
M pipeline first stage ← M-op1 × M-op2

pfamov.r *fsrc1, fdest* ..... **Pipelined Floating-Point Adder Move**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1*

pfeq.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Equal Compare**
fdest ← last stage adder result
CC set if *fsrc1* = *fsrc2*, else cleared
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfgt.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Greater-Than Compare**
(Assembler clears R-bit of instruction)
fdest ← last stage adder result
CC set if *fsrc1* > *fsrc2*, else cleared
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfiadd.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Add**
fdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* + *fsrc2* (2's complement integer arithmetic)

pfisub.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Subtract**
fdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* − *fsrc2* (2's complement integer arithmetic)

pfix.v *fsrc1, fdest* ..... **Pipelined Floating-Point to Integer Conversion**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← 64-bit value with low-order 32 bits
equal to integer part of *fsrc1* rounded

Pipelined Floating-Point Load

pfld.y *isrc1*(*isrc2*), *fdest* ..... **(Normal)**
pfld.y *isrc1*(*isrc2*)++, *fdest* ..... **(Autoincrement)**
fdest ← mem.y (third previous **pfld**'s (*isrc1* + *isrc2*))
(where .y is precision of third previous **pfld.y**)
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI
NOTE: pfld.q is not available with the i860 XR CPU

pfle.p *fsrc1, fsrc2, fdest* ..... **Pipelined F-P Less-Than or Equal Compare**
Assembler sets R-bit of instruction
fdest ← last stage adder result
CC clear if *fsrc1* ≤ *fsrc2*, else set
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfmam.p *fsrc1, fsrc2, fdest* ..... **PipelinedFloating-Point Add and Multiply**
fdest ← last stage multiplier result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 + A-op2
M pipeline first stage ← M-op1 × M-op2

pfmov.r *fsrc1, fdest* ..... **Pipelined Floating-Point Reg-Reg Move**
Assembler pseudo-operation
pfmov.ss *fsrc1, fdest* = **pfiadd.ss** *fsrc1*, **f0**, *fdest*
pfmov.dd *fsrc1, fdest* = **pfiadd.dd** *fsrc1*, **f0**, *fdest*
pfmov.sd *fsrc1, fdest* = **pfamov.sd** *fsrc1*, *fdest*
pfmov.ds *fsrc1, fdest* = **pfamov.ds** *fsrc1*, *fdest*

pfmsm.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract and Multiply**
fdest ← last stage multiplier result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfmul.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Multiply**
fdest ← last stage multiplier result
Advance M pipeline one stage
M pipeline first stage ← *fsrc1* × *fsrc2*

pfmul3.dd *fsrc1, fsrc2, fdest* ..... **Three-Stage Pipelined Multiply**
fdest ← last stage multiplier result
Advance 3-Stage M pipeline one stage
M pipeline first stage ← *fsrc1* × *fsrc2*

pform *fsrc1, fdest* ..... **Pipelined OR to MERGE Register**
fdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* OR MERGE
MERGE ← 0

pfsm.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract and Multiply**
fdest ← last stage adder result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfsub.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1* − *fsrc2*

pftrunc.v *fsrc1, fdest* ..... **Pipelined Floating-Point to Integer Conversion**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← 64-bit value with low-order 32 bits
equal to integer part of *fsrc1*

pfzchkl *fsrc1, fsrc2, fdest* ..... **Pipelined 32-Bit Z-Buffer Check**

Consider the 64-bit operands as arrays of two 32-bit
fields *fsrc1*(1)..*fsrc1*(0), *fsrc2*(1)..*fsrc2*(0), and *fdest*(1)..*fdest*(0)
where zero denotes the least-significant field.
PM ← PM shifted right by 2 bits
FOR i = 0 to 1
DO
PM [i + 6] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← last-stage graphics-unit result
last-stage graphics-unit result ← smaller of *fsrc2*(i) and *fsrc1*
OD
MERGE ← 0

pfzchks *fsrc1, fsrc2, fdest* ..... **Pipelined 16-Bit Z-Buffer Check**

Consider the 64-bit operands as arrays of four 16-bit
fields *fsrc1*(3)..*fsrc1*(0), *fsrc2*(3)..*fsrc2*(0), and *fdest*(3)..*fdest*(0)
where zero denotes the least-significant field.
PM ← PM shifted right by 4 bits
FOR i = 0 to 3
DO
PM [i + 4] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest ← last-stage graphics-unit result
last-stage graphics-unit result(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

pst.d *fdest, #const(isrc2)* ..... **Pixel Store**
pst.d *fdest, #const(isrc2)++* ..... **Pixel Store Autoincrement**

Pixels enabled by PM in *mem.d* (*isrc2* + *#const*) ← *fdest*
Shift PM right by 8/pixel size (in bytes) bits
IF autoincrement
THEN *isrc2* ← *#const* + *isrc2*
FI

scyc.x *isrc2* ..... **Special Cycles**

Generate a special bus cycle (D/C# = 0, W/R# = 1, M/IO# = 0) and
set BE7# – BE0# according to the value contained in the register *isrc2*
NOTE: Not available with the i860 XR CPU

shl *isrc1, isrc2, idest* ..... **Shift Left**

idest ← *isrc2* shifted left by *isrc1* bits

shr *isrc1, isrc2, idest* ..... **Shift Right**

SC (in **psr**) ← *isrc1*
idest ← *isrc2* shifted right by *isrc1* bits

shra *isrc1, isrc2, idest* ..... **Shift Right Arithmetic**

idest ← *isrc2* arithmetically shifted right by *isrc1* bits

shrd *isrc1ni, isrc2, idest* ..... **Shift Right Double**

idest ← low-order 32 bits of *isrc1ni:isrc2* shifted right by SC bits

st.c *isrc1ni, csrc2* ..... **Store to Control Register**

csrc2 ← *src1ni*

st.x *isrc1ni, #const(isrc2)* ..... **Store Integer**

mem.x (*isrc2* + *#const*) ← *isrc1ni*

stio.x *isrc1ni, isrc2* ..... **Store I/O**
port.x (isrc2) ← *isrc1ni*
NOTE: Not available with the i860 XR CPU

subs *isrc1, isrc2, idest* ..... **Subtract Signed**
idest ← *isrc1* − *isrc2*
OF ← (bit 31 carry ≠ bit 30 carry)
CC set if *isrc2* > *isrc1* (signed)
CC clear if *isrc2* ≤ *isrc1* (signed)

subu *isrc1, isrc2, idest* ..... **Subtract Unsigned**
idest ← *isrc1* − *isrc2*
OF ← NOT (bit 31 carry)
CC ← bit 31 carry
(i.e. CC set if *isrc2* ≤ *isrc1* (unsigned)
CC clear if *isrc2* > *isrc1* (unsigned))

trap *isrc1ni, isrc2, idest* ..... **Software Trap**
Generate trap with IT set in **psr**

unlock ..... **End Interlocked Sequence**
Clear BL in **dirbase**. The next load or store unlocks the bus. Interrupts are enabled.

xor *isrc1, isrc2, idest* ..... **Logical Exclusive OR**
idest ← *isrc1* XOR *isrc2*
CC set if result is zero, cleared otherwise

xorh *#const, isrc2, idest* ..... **Logical Exclusive OR High**
idest ← (*#const* shifted left 16 bits) XOR *isrc2*
CC set if result is zero, cleared otherwise

Table 10.2. FADDP MERGE Update

Pixel Size (from PS)	Fields Loaded from Result into MERGE				Right Shift Amount (Field Size)
8	63..56,	47..40,	31..24,	15..8	8
16	63..58,	47..42,	31..26,	15..10	6
32	63..56,		31..24		8

10.2 Instruction Format and Encoding

All instructions are 32 bits long and begin on a four-byte boundary. When operands are registers, the encodings shown in Table 10.3 are used.

There are two general core-instruction formats (REG-format and CTRL-format) and a separate format for floating-point instructions.

Table 10.3. Register Encoding

Register	Encoding
r0	0
.	.
.	.
.	.
r31	31
f0	0
.	.
.	.
.	.
f31	31
Fault Instruction	0
Processor Status	1
Directory Base	2
Data Breakpoint	3
Floating-Point Status	4
Extended Processor Status	5
Bus Error Address*	6
Concurrency Control*	7
p0*	8
p1*	9
p2*	10
p3*	11

NOTE:
*Available only with i860 XP CPU. Using these encodings with the i860 XR CPU produces undefined results.

10.2.1 REG-FORMAT INSTRUCTIONS

Within the REG-format are several variations as shown in Figure 10.1. Table 10.4 gives the encodings for these instructions. One encoding is an escape code that defines yet another variation: the core escape instructions. Figure 10.2 shows the format of this group, and Table 10.5 shows the encodings.

In these instructions, the *src2* field selects one of the 32 integer registers (most instructions) or one of the control registers (**st.c** and **ld.c**). *Dest* selects one of the 32 integer registers (most instructions) or floating-point registers (**fld**, **fst**, **pfld**, **pst**, **ixfr**). For instructions where *src1* is optionally an immediate value, bit 26 of the opcode (I-bit) indicates whether *src1* is an immediate. If bit 26 is clear, an integer register is used; if bit 26 is set, *src1* is contained in the low-order 16 bits, except for **bte** and **btne** instructions. For **bte** and **btne**, the five-bit immediate value is contained in the *src1* field. For **st**, **bte**, **btne**, and **bla**, the upper five bits of the *offset* or *broffset* are contained in the *dest* field instead of *src1*, and the lower 11 bits of *offset* are the lower 11 bits of the instruction.

For **ld** and **st**, bits 28 and zero determine operand size as follows:

Bit 28	Bit 0	Operand Size
0	0	8-bits
0	1	8-bits
1	0	16-bits
1	1	32-bits

When *src1* is immediate and bit 28 is set, bit zero of the immediate value is forced to zero.

For **fld**, **fst**, **pfld**, **pst**, and **flush**, bit 0 selects autoincrement addressing if set. For **fld, fst, pfld,** and **pst,** bits one and two select the operand size as follows:

Bit 1	Bit 2	Operand Size
0	0	64-bits
0	1	128-bits
1	0	32-bits
1	1	32-bits

For **flush**, bits one and two must be zero.

When *src1* is immediate, bits zero and one of the immediate value are forced to zero to maintain alignment. When bit one of the immediate value is clear, bit two is also forced to zero.

For the instructions **ldio, stio, ldint,** and **scyc**, the operand size is encoded by bits 9 and 10 as follows. For other instructions, these bits are *reserved* and should be set to zero.

Operand Size	Bit 10	Bit 9
8 Bits (.b)	0	0
16 Bits (.s)	0	1
32 Bits (.l)	1	0
reserved	1	1

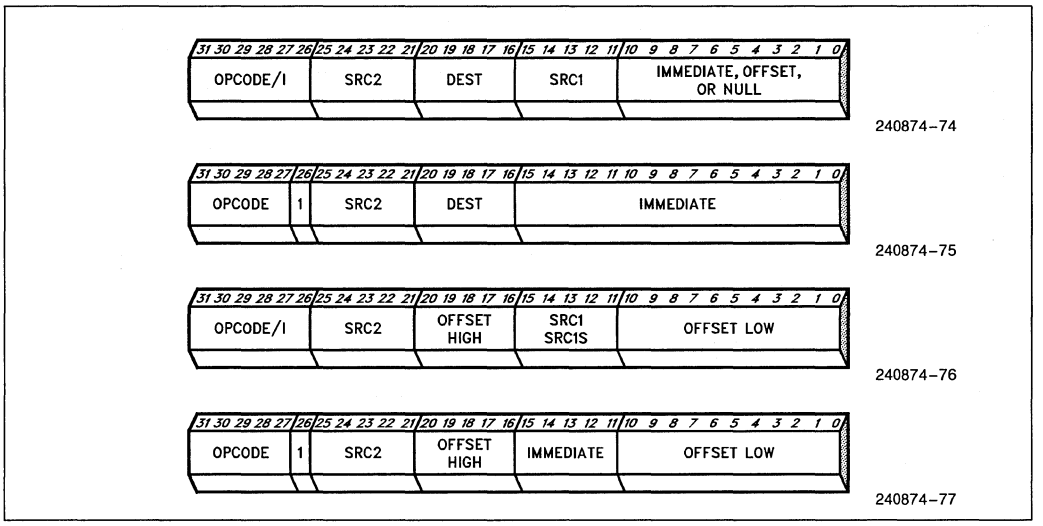


Figure 10.1. REG-Format Variations

Table 10.4. REG-Format Opcodes

		31	30	29	28	27	26
ld.x	Load Integer	0	0	0	L	0	I
st.x	Store Integer	0	0	0	L	1	1
ixfr	Integer to F-P Reg Transfer	0	0	0	0	1	0
—	*(reserved)*	0	0	0	1	1	0
fld.x, fst.x	Load/Store F-P	0	0	1	0	LS	I
flush	Flush	0	0	1	1	0	1
pst.d	Pixel Store	0	0	1	1	1	1
ld.c, st.c	Load/Store Control Register	0	0	1	1	LS	0
bri	Branch Indirect	0	1	0	0	0	0
trap	Trap	0	1	0	0	0	1
—	(Escape for F-P Unit)	0	1	0	0	1	0
—	(Escape for Core Unit)	0	1	0	0	1	1
bte, btne	Branch Equal or Not Equal	0	1	0	1	E	I
pfld.y	Pipelined F-P Load	0	1	1	0	0	I
—	(CTRL-Format Instructions)	0	1	1	x	x	x
addu, -s, subu, -s	Add/Subtract	1	0	0	SO	AS	I
shl, shr	Logical Shift	1	0	1	0	LR	I
shrd	Double Shift	1	0	1	1	0	0
bla	Branch LCC Set and Add	1	0	1	1	0	1
shra	Arithmetic Shift	1	0	1	1	1	I
and(h)	AND	1	1	0	0	H	I
andnot(h)	ANDNOT	1	1	0	1	H	I
or(h)	OR	1	1	1	0	H	I
xor(h)	XOR	1	1	1	1	H	I
—	*(reserved)*	1	1	x	x	1	0

L Integer Length
- 0 —8 bits
- 1 —16 or 32 bits (selected by bit 0)

LS Load/Store
- 0 —Load
- 1 —Store

SO Signed/Ordinal
- 0 —Ordinal
- 1 —Signed

H High
- 0 **—and, or, andnot, xor**
- 1 **—andh, orh, andnoth, xorh**

AS Add/Subtract
- 0 —Add
- 1 —Subtract

LR Left/Right
- 0 —Left Shift
- 1 —Right Shift

E Equal
- 0 —Branch on Unequal
- 1 —Branch on Equal

I Immediate
- 0 —*src1* is register
- 1 —*src1* is immediate

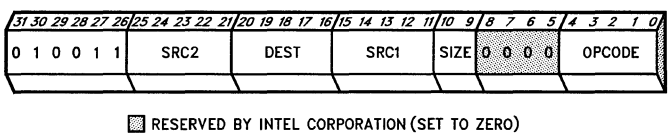


Figure 10.2. Core Escape Instructions

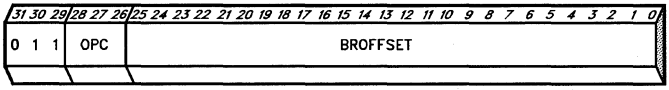
Table 10.5. Core Escape Opcodes

		4	3	2	1	0
—	*(reserved)*	0	0	0	0	0
lock	Begin Interloacked Sequence	0	0	0	0	1
calli	Indirect Subroutine Call	0	0	0	1	0
—	*(reserved)*	0	0	0	1	1
introvr	Trap on Integer Overflow	0	0	1	0	0
—	*(reserved)*	0	0	1	0	1
—	*(reserved)*	0	0	1	1	0
unlock	End Interlocked Sequence	0	0	1	1	1
ldio*	Load I/O	0	1	0	0	0
stio*	Store I/O	0	1	0	0	1
ldint*	Load Interrupt Vector	0	1	0	1	0
scyc*	Special Cycles	0	1	0	1	1
—	*(reserved)*	0	1	1	x	x
—	*(reserved)*	1	0	x	x	x
—	*(reserved)*	1	1	x	x	x

NOTE:
*Available only with i860 XP CPU, not with i860 XR CPU

10.2.2 CTRL-FORMAT INSTRUCTIONS

The CTRL-Format instructions do not refer to registers; so, instead of the register fields, they have a 26-bit relative branch offset. Figure 10.3 shows the format of these instructions and Table 10.6 defines the encodings.



240874-79

NOTE:
BROFFSET is a signed 26-bit relative branch offset

Figure 10.3. CTRL-Format Instructions

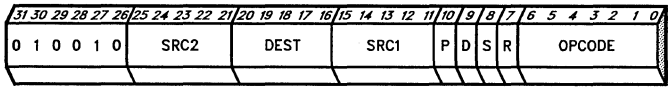
Table 10.6. CTRL-Format Opcodes

		28	27	26
—	*(reserved)*	0	0	0
—	*(reserved)*	0	0	1
br	Branch Direct	0	1	0
call	Call	0	1	1
bc(.t)	Branch on CC Set	1	0	T
bnc(.t)	Branch on CC Clear	1	1	T

T Taken
0 —**bc** or **bnc**
1 —**bc.t** or **bnc.t**

10.2.3 FLOATING-POINT INSTRUCTION ENCODING

The floating-point instructions also constitute an escape series. All these instructions begin with the bit sequence 010010. Figure 10.4 shows the format of the floating-point instructions, and Table 10.7 gives the encodings. Within the dual-operation instructions is a subcode DPC whose values are given in Table 10.9 along with the mnemonic that corresponds to each.



240874-80

SRC1, SRC2 Source; one of 32 floating-point registers
DEST Destination; one of 32 floating-point registers (except **fxfr**; one of 32 integer registers)

P Pipelining
- 1 Pipelined instruction mode
- 0 Scalar instruction mode

D Dual-Instruction Mode
- 1 Dual-instruction mode
- 0 Single-instruction mode

S Source Precision
- 1 Double-precision source operands
- 0 Single-precision source operands

R Result Precision
- 1 Double-precision result
- 0 Single-precision result

Figure 10.4. Floating-Point Instruction Encoding

Table 10.7. Floating-Point Opcodes

		6	5	4	3	2	1	0
pfam	Add and Multiply*	0	0	0	DPC			
pfmam	Multiply with Add*							
pfsm	Subtract and Multiply*	0	0	1	DPC			
pfmsm	Multiply with Subtract*							
(p)fmul	Multiply	0	1	0	0	0	0	0
fmlow	Multiply Low	0	1	0	0	0	0	1
frcp	Reciprocal	0	1	0	0	0	1	0
frsqr	Reciprocal Square Root	0	1	0	0	0	1	1
pfmul3.dd	3-Stage Pipelined Multiply	0	1	0	0	1	0	0
(p)fadd	Add	0	1	1	0	0	0	0
(p)fsub	Subtract	0	1	1	0	0	0	1
(p)fix	Fix	0	1	1	0	0	1	0
(p)famov	Adder Move	0	1	1	0	0	1	1
pfgt/pfle**	Greater Than	0	1	1	0	1	0	0
pfeq	Equal	0	1	1	0	1	0	1
(p)ftrunc	Truncate	0	1	1	1	0	1	0
fxfr	Transfer to Integer Register	1	0	0	0	0	0	0
(p)fiadd	Long-Integer Add	1	0	0	1	0	0	1
(p)fisub	Long-Integer Subtract	1	0	0	1	1	0	1
(p)fzchkl	Z-Check Long	1	0	1	0	1	1	1
(p)fzchks	Z-Check Short	1	0	1	1	1	1	1
(p)faddp	Add with Pixel Merge	1	0	1	0	0	0	0
(p)faddz	Add with Z Merge	1	0	1	0	0	0	1
(p)form	OR with MERGE Register	1	0	1	1	0	1	0

NOTE:
All opcodes not shown are *reserved*.
* **pfam** and **pfsm** have P-bit set; **pfmam** and **pfmsm** have P-bit clear.
** **pfgt** has R bit cleared; **pfle** has R bit set.

Table 10.8. DPC Encoding

DPC	PFAM Mnemonic	PFSM Mnemonic	M-Unit o1	M-Unit op2	A-Unit op1	A-Unit op2	T Load	K Load*
0000	**r2p1**	r2s1	KR	src2	src1	M result	No	No
0001	**r2pt**	r2st	KR	src2	T	M result	No	Yes
0010	**r2ap1**	r2as1	KR	src2	src1	A result	Yes	No
0011	**r2apt**	r2ast	KR	src2	T	A result	Yes	Yes
0100	**i2p1**	i2s1	KI	src2	src1	M result	No	No
0101	**i2pt**	i2st	KI	src2	T	M result	No	Yes
0110	**i2ap1**	i2as1	KI	src2	src1	A result	Yes	No
0111	**i2apt**	i2ast	KI	src2	T	A result	Yes	Yes
1000	**rat1p2**	rat1s2	KR	A result	src1	src2	Yes	No
1001	**m12apm**	m12asm	src1	src2	A result	M result	No	No
1010	**ra1p2**	ra2s2	KR	A result	src1	src2	No	No
1011	**m12ttpa**	m12ttsa	src1	src2	T	A result	Yes	No
1100	**iat1p2**	iat1s2	KI	A result	src1	src2	Yes	No
1101	**m12tpm**	m12tsm	src1	src2	T	M result	No	No
1110	**ia1p2**	ia1s2	KI	A result	src1	src2	No	No
1111	**m12tpa**	m12tsa	src1	src2	T	A result	No	No
DPC	**PFMAM Mnemonic**	**PFMSM Mnemonic**	**M-Unit op1**	**M-Unit op2**	**A-Unit op1**	**A-Unit op2**	**T Load**	**K Load***
0000	**mr2p1**	mr2s1	KR	src2	src1	M result	No	No
0001	**mr2pt**	mr2st	KR	src2	T	M result	No	Yes
0010	**mr2mp1**	mr2ms1	KR	src2	src1	M result	Yes	No
0011	**mr2mpt**	mr2mst	KR	src2	T	M result	Yes	Yes
0100	**mi2p1**	mi2s1	KI	src2	src1	M result	No	No
0101	**mi2pt**	mi2st	KI	src2	T	M result	No	Yes
0110	**mi2mp1**	mi2ms1	KI	src2	src1	M result	Yes	No
0111	**mi2mpt**	mi2mst	KI	src2	T	M result	Yes	Yes
1000	**mrmt1p2**	mrmt1s2	KR	M result	src1	src2	Yes	No
1001	**mm12mpm**	mm12msm	src1	src2	M result	M result	No	No
1010	**mrm1p2**	mrm1s2	KR	M result	src1	src2	No	No
1011	**mm12ttpm**	mm12ttsm	src1	src2	T	M result	Yes	No
1100	**mimt1p2**	mimt1s2	KI	M result	src1	src2	Yes	No
1101	**mm12tpm**	mm12tsm	src1	src2	T	M result	No	No
1110	**mim1p2**	mim1s2	KI	M result	src1	src2	No	No
1111	Intel Reserved							

NOTE:
* If K-load is set, KR is loaded when operand-1 of the multiplier is KR; KI is loaded when operand-1 of the multiplier is KI.

10.3 Instruction Timings

Generally, i860 XP microprocessor instructions take one clock to execute unless a freeze condition is invoked. Detailed times, along with freeze conditions and their associated delays, are shown in the table on the following pages. The following symbols are used for brevity in the timing table:

$+n$	*n* clocks must be added to the execution time if the stated conditions apply.
$\longleftrightarrow n$	The processor requires at least *n* clocks between the indicated instructions. The actual delay will be *n* minus the number of clocks for executing intervening instructions (or dual-mode pairs). If the time for intervening instructions is $\geq n$, there is no delay.
n..m	Indicates a range of clocks. These cases are accompanied by a reference to a note where further explanation is available.
XR:	Applies to i860 XR microprocessors only.
XP:	Applies to i860 XP microprocessors only.
OA	The number of clocks to finish all outstanding accesses.
R1	The number of clocks from ADS# through the first READY# (80860XR) or BRDY# (80860XP) of the indicated bus activity.
R2	The number of clocks from ADS# through the second READY# or BRDY#.
RL	The number of clocks from ADS# through the last READY# or BRDY#.
RL1	XL: The number of clocks through last BRDY# of first access.
RN	XR: The number of clocks until next nonrepeated address can be issued (i.e., an address that is not the 2nd–4th cycle of a cache fill, the 2nd–8th cycle of a CS8 mode instruction fetch, nor the 2nd cycle of a 128-bit write).
RX	The number of clocks through READY# or BRDY# for the next 64-bit-or-less write cycle or second READY# or BRDY# for the next 128-bit write cycle.

NOTES:

a. "Address path full" means one address internally waiting for bus while external bus pipeline full.

b. "Store path full" means two stores or one 256-bit write-back internally waiting for bus plus external bus pipeline full.

c. If a floating-point instruction, graphics-unit instruction, **fst**, or **pst** is executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress, the scalar operation must complete first: two additional clocks for **fadd, fix, fmlow, fmul.ss, fmul.sd, ftrunc,** and **fsub**; three additional clocks for **fmul.dd**. Add one if either or both of these situations occur:

1. There is an overlap between the result register of the previous scalar operation and the source of the floating-point operation, and the destination precision of the scalar operation differs from the source precision of the floating-point operation.
2. The floating-point operation is pipelined and its destination is not **f0**.

TLB TLB miss. Five clocks plus the number of clocks to finish two reads plus the number of clocks to set A-bits (if necessary).

In addition, any instruction may be delayed due to an instruction cache miss or TLB miss during the instruction fetch. The time for a TLB miss is shown above in note **TLB**. An instruction cache miss adds the following delays:

- The number of clocks to get the next instruction from the bus (ADS# clock to first READY# or BRDY# clock, inclusive).
- XR: When any of the instructions in the new instruction-cache line is a branch or call or causes a freeze, the time through the last READY# for the new line.
- If the data cache is being accessed when the instruction-cache miss occurs, two clocks for data cache miss; one clock for hit.

Not included in the table is the delay caused by a trap. This depends on the trap handler.

In dual instruction mode, each pair of instructions requires the maximum of the times required by each individual instruction.

Instruction	Execution Clocks	Condition
adds	1	
addu	1	
and	1	
andh	1	
andnot	1	
andnoth	1	
bc	1	If branch not taken.
	2	If branch taken.
	+	If the prior instruction is **addu, adds, subu, subs, pfeq,** or **pfgt**.
bc.t	1	If branch taken.
	2	If branch not taken.
	+1	If the prior instruction is **addu, adds, subu, subs, pfeq,** or **pfgt**.
bla	1	If branch taken.
	2	If branch not taken.
bnc	(same as **bc**)	
bnc.t	(same as **bc.t**)	
br	1	
brl	2	
bte	1	If branch not taken.
	3	If branch taken.
btne	(same as **bte**)	
call	1	
	+1	If **r1** referenced in next instruction.
	+1+R1	If data cache load miss in progress for a read of less than 128 bits.
	+1+R2	If data cache load miss in progress for 128-bit read.
calli	2	
	+1	If **r1** referenced in next instruction.
	+1+R1	If data cache load miss in progress for a read of less than 128 bits.
	+1+R2	If data cache load miss in progress for 128-bit read.
fadd.p	1	(. . . and all other A-unit instructions except dual operations)
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)

Instruction	Execution Clocks	Condition
faddp	1	(. . . and all other G-unit instructions except **fiadd.w, fxfr**)
	+1	If *fdest* is used by next instruction and next instruction is G-, M- or A-unit instruction
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
faddz	(same as **faddp**)	
famov.r	(same as **fadd.p**)	
fiadd.w	1	
	+1	If *fdest* is used by next instruction and next instruction is M- or A-unit instruction (except when **fiadd** is used for **fmov.dd** or **fmov.ss**).
	+1	If *fdest* is used by next instruction and next instruction is G-unit instruction.
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
fisub.w		(same as **faddp**)
fix.v		(same as **fadd.p**)
fld.y	1	
	+1	If this is the instruction after a **st, fst** or **pst** that hits the data cache.
	⟷ 2	If *fdest* is referenced in the next two instructions.
	+1+R1	If 32-bit **fld.l** or 64-bit **fld.d** misses the data cache.
	+1+R2	If 128-bit **fld.q** misses the data cache.
	+1+RL	If data cache load miss in progress (except in the following case).
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+TLB	If TLB miss.
flush	1	
	⟷ 3	XR: If preceded by another **flush**.
	⟷ 2	XP: If preceded by another **flush**.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+1+RX	If flush to modified line when store path full.(b)
	+TLB	If TLB miss.
fmlow.dd	1	(. . . and all other M-unit instruction except dual operations)
	+1	If *fsrc1* refers to result of the prior operation (either scalar or pipelined).
	+1	If the prior operation is a double-precision multiply.
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
fmov.r		**fmov.ss** and **fmov.dd** same as **fiadd.w** **fmov.sd** and **fmov.ds** same as **fadd.p**
fmul.p		(same as **fmlow.dd**)

Instruction	Execution Clocks	Condition
fnop	1	
form	(same as **faddp**)	
frcp.p	(same as **fmlow.dd**)	
frsqr.p	(same as **fmlow.dd**)	
fst.y	1	
	+1	If followed by pipelined floating-point operation that overwrites the register being stored.
	+1+RL	If data cache load miss in progress.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+1+RX	If cache miss when store path full.(b)
	+TLB	If TLB miss.
fsub.p	(same as **fadd.p**)	
ftrunc.v	(same as **fadd.p**)	
fxfr	1	
	+1	If *idest* referenced in next instruction.
	+1+R1	If data cache load miss in progress for 64-bit read.
	+1+R2	If data cache load miss in progress for 128-bit read.
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
fzchkl	(same as **faddp**)	
fzchks	(same as **faddp**)	
intovr	1	
ixfr	1	
	+1+R1	If data cache load miss in progress for 64-bit read.
	+1+R2	If data cache load miss in progress for 128-bit read.
	⟷ 2	If *fdest* is referenced in the next two instructions.
ld.c	1	
	+1	If *idest* referenced in next instruction.
	+1+R1	If data cache load miss in progress for 64-bit read.
	+1+R2	If data cache load miss in progress for 128-bit read.

Instruction	Execution Clocks	Condition
ld.x	1	
	+1	If *idest* referenced in next instruction.
	+1	If this is the instruction after a **st, fst** or **pst** that hits the data cache.
	+1+RL	If data cache load miss in progress.
	⟷ 1+R1	If **ld.x** misses the data cache and a subsequent instruction references the *idest* of the **ld.x** (except for following case).
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+1+RX	If cache miss when store path full.(b)
	+TLB	If TLB miss.
ldint.x	1	+ OA
ldio.x	1	+ OA
lock	1	
mov	1	
nop	1	
or	1	
orh	1	
pfadd.p	(same as **fadd.p**)	
pfaddp	(same as **faddp**)	
pfaddz	(same as **faddp**)	
pfam.p	1	(. . . and all other dual operations)
	+1	If *fsrc1* refers to result of the prior operation (either scalar or pipelined).
	+1	If the prior operation is a double-precision multiply.
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
pfamov.r	(same as **fadd.p**)	
pfeq.p	(same as **fadd.p**)	
pfgt.p	(same as **fadd.p**)	
pfiadd.w	(same as **faddp**)	
pfisub.w	(same as **faddp**)	
pfix.v	(same as **fadd.p**)	

Instruction	Execution Clocks	Condition
pfld.y	1	
	+1+RL	If data cache load miss in progress.
	⟷ 2	If *fdest* is referenced in the next two instructions.
	+1+RL1	If three **pfld's** are outstanding.
	+2+OA	XR: If **pfld** hits data cache.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+TLB	If TLB miss.
pfle.p	1	
pfmam.p		(same as **pfam.p**)
pfmov.r		**pfmov.ss** and **pfmov.dd** same as **faddp** **pfmov.sd** and **pfmov.ds** same as **fadd.p**
pfmsm.p		(same as **pfam.dd**)
pfmul.p		(same as **fmlow.dd**)
pfmul3.dd		(same as **fmlow.dd**)
pform		(same as **faddp**)
pfsm.p		(same as **pfam.dd**)
pfsub.p		(same as **fadd.p**)
pftrunc.v		(same as **fadd.p**)
pfzchkl		(same as **faddp**)
pfzchks		(same as **faddp**)
pst.d		(same as **fst.d**)
scyc.x	1 + OA	
shl	1	
shr	1	
shra	1	
shrd	1	
st.c	3	
	+1+R1	If data cache load miss in progress for a read of less than 128 bits.
	+1+R2	If data cache load miss in progress for 128-bit read.

Instruction	Execution Clocks	Condition
st.x	1	
	+1+RL	If data cache load miss in progress.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+1+RX	If cache miss when store path full.(b)
	+TLB	If TLB miss.
stio.x	1 + OA	
subs	1	
subu	1	
trap	1	
unlock	1	
xor	1	
xorh	1	

10.4 Instruction Characteristics

The following table lists some of the characterisics of each instruction. The characteristics are:

- What processing unit executes the instruction. The codes for processing units are:
  - **A** Floating-point adder unit
  - **E** Core execution unit
  - **G** Graphics unit
  - **M** Floating-point multiplier unit
- Whether the instruction is pipelined or not. A *P* indicates that the instruction is pipelined.
- Whether the instruction is a delayed branch instruction. A *D* marks the delayed branches.
- Whether execution is suppressed in user mode. An *SU* marks supervisor-only instructions.
- Whether the instruction is available on both the i860 XR and i860 XP microprocessors. An *XL* marks instructions that are available only on the i860 XP microprocessor.
- Whether the instruction changes the condition code CC. A *CC* marks those instructions that change CC.
- Which faults can be caused by the instruction. The codes used for exceptions are:
  - **IT** Instruction Fault
  - **SE** Floating-Point Source Exception
  - **RE** Floating-Point Result Exception, including overflow, underflow, inexact result
  - **DAT** Data Access Fault

Note that this is not the same as specifying at which instructions faults may be reported. A result exception is reported on the subsequent floating-point instruction, **pst**, **fst**, or sometimes **fld**, **pfld**, and **ixfr**.

The instruction access fault IAT and the interrupt trap IN are not shown in the table because they can occur for any instruction.

- Performance notes. These comments regarding optimum performance are recommendations only. If these recommendations are not followed, the i860 XP microprocessor automatically waits the necessary number of clocks to satisfy internal hardware requirements. The following notes define the numeric codes that appear in the instruction table:
  1. The following instruction should not be a conditional branch (**bc**, **bnc**, **bc.t**, or **bnc.t**).
  2. The destination should not be a source operand of the next two instructions.
  3. A load should not directly follow a store that is expected to hit in the data cache.
  4. When the prior instruction is scalar, *fsrc1* should not be the same as the *fdest* of the prior operation.

5. The *fdest* should not reference the destination of the next instruction if that instruction is a pipelined floating-point operation.
6. The destination should not be a source operand of the next instruction. (For **call** and **calli**, the destination is **r1**.)
7. When the prior operation is scalar and multiplier *op1* is *fsrc1, fsrc2* should not be the same as the *fdest* of the prior operation.
8. When the prior operation is scalar, *src1* and *src2* of the current operation should not be the same as *dest* of the prior operation.
9. A **pfld** should not immediately follow a **pfld**

- Programming restrictions. These indicate combinations of conditions that must be avoided by programmers, assemblers, and compilers. The following notes define the alphabetic codes that appear in the instruction table:

a. The sequential instruction following a delayed control-transfer instruction may not be another control-transfer instruction, nor a **trap** instruction, nor the target of a control-transfer instruction.

b. When using a **bri** to return from a trap handler, programmers should take care to prevent traps from occurring on that or on the next sequential instruction. IM should be zero (interrupts disabled) when the **bri** is executed.

c. If *fdest* is not zero, *fsrc1* must not be the same as *fdest*.

d. When *fsrc1* goes to multiplier *op1* or to KR or KI, *fsrc1* must not be the same as *fdest*.

e. If *dest* is not zero, *src1* and *src2* must not be the same as *dest*.

f. *Isrc1* must not be the same register as *isrc2* for the autoincrementing form of this instruction.

g. *Isrc1* must not be the same register as *isrc2*.

Instruction	Execution Unit	Pipelined? Delayed? Supervisor? i860™ XP Only?	Sets CC?	Faults	Performance Notes	Programming Restrictions
adds	E		CC		1	
addu	E		CC		1	
and	E		CC			
andh	E		CC			
andnot	E		CC			
andnoth	E		CC			
bc	E					
bc.t	E	D				a
bla	E	D				a,g
bnc	E					
bnc.t	E	D				a
br	E	D				a
bri	E	D				a,b
bte	E					
btne	E					
call	E	D			6	a
calli	E	D			6	a
fadd.p	A			SE,RE		
faddp	G				8	
faddz	G				8	
famov.r	A			SE,RE		
fiadd.w	G				8	
fisub.w	G				8	
fix.p	A			SE,RE		
fld.y	E			DAT	2,3	f

NOTES:
* On the i860 XP microprocessor, the pipelined instructions can generate ITR with PI.
** On the i860 XP micropocessor, the 128-bit **pfld.q** is not available. If used it causes an instruction trap.

Instruction	Execution Unit	Pipelined? Delayed? Supervisor? i860™ XP Only?	Sets CC?	Faults	Performance Notes	Programming Restrictins
flush	E					
fmlow.dd	M				4	
fmul.p	M			SE,RE	4	
form	G				8	
frcp.p	M			SE,RE		
frsqr.p	M			SE,RE		
fst.y	E			DAT	5	f
fsub.p	A			SE,RE		
ftrunc.p	A			SE,RE		
fxfr	G				6,8	
fzchkl	G				8	
fzchks	G				8	
Intovr	E			IT		
ixfr	E				2	
ld.c	E					
ld.x	E			DAT	6	
ldint.x	E	SU,XL		DAT		
ldio.x	E	SU,XL		DAT		
lock	E					
or	E		CC			
orh	E		CC			
pfadd.p	A	P		SE,RE*		
pfaddp	G	P		*	8	e
pfaddz	G	P		*	8	e
pfam.p	A&M	P		SE,RE*	7	d
pfamov.r	A	P		SE,RE*		
pfeq.p	A	P	CC	SE*	1	
pfgt.p	A	P	CC	SE*	1	
pfiadd.w	G	P		*	8	e
pfisub.w	G	P		*	8	e
pfix.p	A	P		SE,RE*		
pfld.y	E	P,(XL)**		DAT*	2,9	f
pfmam.p	A&M	P		SE,RE*	7	d
pfmsm.p	A&M	P		SE,RE*	7	d
pfmul.p	M	P		SE,RE*	4	c
pfmul3.dd	M	P		SE,RE*	4	c
pform	G	P		*	8	e
pfsm.p	A&M	P		SE,RE*	7	d
pfsub.p	A	P		SE,RE*		
pftrunc.p	A	P		SE,RE*		
pfzchkl	G	P		*	8	

NOTES:
* On the i860 XP microprocessor, the pipelined instructions can generate ITR with PI.
** On the i860 XP micropocessor, the 128-bit **pfld.q** is not available. If used it causes an instruction trap.

Instruction	Execution Unit	Pipelined? Delayed? Supervisor? i860™ XP Only?	Sets CC?	Faults	Performance Notes	Programming Restrictions
pfzchks	G	P		*	8	
pst.d	E			DAT	5	f
scyc.x	E	SU,XL		DAT		
shl	E					
shr	E					
shra	E					
shrd	E					
st.c	E					
st.x	E			DAT		
stio.x	E	SU,XL		DAT		
subs	E		CC		1	
subu	E		CC		1	
trap	E			IT		
unlock	E					
xor	E		CC			
xorh	E		CC			

NOTES:
*On the i860 XP microprocessor, the pipelined instructions can generate ITR with PI.
On the i860 XP micropocessor, the 128-bit **pfld.q is not available. If used it causes an instruction trap.

10.5 Software Compatibility

10.5.1 REQUIRED CHANGES

To port existing systems software from the i860 XR microprocessor to the i860 XP microprocessor, the following changes may be required. Applications software does not require changes.

1. Data cache flush. All four ways of the data cache must be flushed on the i860 XP microprocessor. The cache flush routine can be modified to check processor type in **epsr** or the DCS field of **dirbase** and flush the appropriate number of ways.
2. Parity and bus error traps. If the i860 XP system signals these errors, the trap handler must be extended to handle them. Software must avoid testing the BEF and PEF bits unless executing on the i860 XP microprocessor.
3. LOCK# deactivation. On the i860 XP microprocessor, traps do not automatically deactivate the LOCK# signal, so the trap handler must do a data access to deactivate LOCK#. Trap handlers that already access data soon after invocation do not require this modification.
4. Load pipe precision. The precision of the last stage of the load pipeline is specified by the LRP bit on the i860 XR microprocessor but by the LRP0 and LRP1 bits on the i860 XP microprocessor. The procedure that restores the load pipe must check the processor type, use the appropriate bits, and restore the correct precision. Pipe restoration code for the i860 XR microprocessor will work correctly on the i860 XP microprocessor if **pfld.q** is not used.
5. Pre-accessed trap handler pages. Page-directory and page-table entries for the instruction pages of the trap handler and for the first data page accessed by the trap handler must always have A = 1. Software modified to allocate page tables this way works on both i860 XR and i860 XP microprocessors.
6. Page directory entry bit 7 must be zero. This is the bit that selects four Mbyte or four Kbyte page size. On the i860 XR microprocessor, it is *reserved* and should be set to zero. It must be set to zero for four Kbyte pages to work on the i860 XP microprocessor.

10.5.2 PERFORMANCE OPTIMIZATIONS

Software developers may wish to make the following performance enhancements in systems software for the i860 XP microprocessor. Systems software that must execute on both i860 XP and i860 XR systems can contain code both with and without the optimizations. By testing the processor type, the appropriate instruction path can be determined.

1. Data cache flush. On the i860 XP microprocessor, a complete flushing of the data cache is not needed when changing context or marking a page not present.
2. The **epsr** bits AI, DI, PI, and PT can be used on the i860 XP microprocessor to make trap handlers more efficient.
3. Four-Mbyte pages can be allocated to frame buffers and the operating-system kernel, thereby reducing the cost of TLB misses.

10.5.3 NEW FEATURES

Software that uses the new features available only on the i860 XP microprocessor will not be compatible with the i860 XR microprocessor unless alternate instruction paths are provided.

Systems software features:

1. New instructions **ldio, stio, ldint,** and **scyc**.
2. Four-Mbyte pages.
3. Privileged Registers **p0, p1, p2,** and **p3**.
4. Concurrency control unit.
5. 128-bit load instruction **pfld.q**.
6. Support for virtual address aliases.

Applications software features:

1. Concurrency control unit.
2. 128-bit load instruction **pfld.q**. The i860 XR microprocessor traps on **pfld.q**; therefore, software has the opportunity to emulate a **pfld.q** with two **pfld.d** instructions. However, this strategy does not yield optimal performance on the i860 XR microprocessor.

10.5.4 NOTES

On the i860 XP microprocessor, pages with WT = 1 are cached with the write-through policy; whereas, on the i860 XR microprocessor, they are not cached at all. Because this change in the function of WT was anticipated in the i860 XR microprocessor documentation, no incompatibility should arise.

I

S

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010
FAX: (205) 837-2640

ARIZONA

†Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) 231-0386
FAX: (602) 244-0446

Intel Corp.
7225 N. Mona Lisa Rd.
Suite 215
Tucson 85741
Tel: (602) 544-0227
FAX: (602) 544-0232

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500
FAX: (818) 340-1144

†Intel Corp.
300 N. Continental Blvd.
Suite 100
El Segundo 90245
Tel: (213) 640-6040
FAX: (213) 640-7133

Intel Corp.
1 Sierra Gate Plaza
Suite 280C
Roseville 95678
Tel: (916) 782-8086
FAX: (916) 782-8153

†Intel Corp.
9665 Chesapeake Dr.
Suite 325
San Diego 92123
Tel: (619) 292-8086
FAX: (619) 292-0628

†Intel Corp.*
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114
FAX: (714) 541-9157

†Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 910-338-0255
FAX: (408) 727-2620

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (719) 594-6622
FAX: (303) 594-0720

†Intel Corp.*
600 S. Cherry St.
Suite 700
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289
FAX: (303) 322-8670

CONNECTICUT

†Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130
FAX: (203) 794-0339

FLORIDA

†Intel Corp.
800 Fairway Drive
Suite 160
Deerfield Beach 33441
Tel: (305) 421-0506
FAX: (305) 421-2444

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (407) 240-8000
FAX: (407) 240-8097

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33716
Tel: (813) 577-2413
FAX: (813) 578-1607

GEORGIA

†Intel Corp.
20 Technology Parkway
Suite 150
Norcross 30092
Tel: (404) 449-0541
FAX: (404) 605-9762

ILLINOIS

†Intel Corp.*
Woodfield Corp. Center III
300 N. Martingale Road
Suite 400
Schaumburg 60173
Tel: (708) 605-8031
FAX: (708) 706-9762

INDIANA

†Intel Corp.
8910 Purdue Road
Suite 350
Indianapolis 46268
Tel: (317) 875-0623
FAX: (317) 875-8938

IOWA

Intel Corp.
1930 St. Andrews Drive N.E.
2nd Floor
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

†Intel Corp.
10985 Cody St.
Suite 140
Overland Park 66210
Tel: (913) 345-2727
FAX: (913) 345-2076

MARYLAND

†Intel Corp.*
10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: (301) 206-3677
(301) 206-3678

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01886
Tel: (508) 692-0960
TWX: 710-343-6333
FAX: (508) 692-7867

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48322
Tel: (313) 851-8096
FAX: (313) 851-8770

MINNESOTA

†Intel Corp.
3500 W. 80th St.
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867
FAX: (612) 831-6497.

MISSOURI

†Intel Corp.
3300 Rider Trail South
Suite 170
Earth City 63045
Tel: (314) 291-1990
FAX: (314) 291-4341

NEW JERSEY

Intel Corp.
Arbor Circle South
8 Campus Drive
Parsippany 07054
Tel: (201) 455-1868
FAX: (201) 644-0680

†Intel Corp.*
Lincroft Office Center
125 Half Mile Road
Red Bank 07701
Tel: (908) 747-2233
FAX: (908) 747-0983

NEW YORK

Intel Corp.*
850 Crosskeys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391
FAX: (716) 223-2561

†Intel Corp.*
2950 Express Dr., South
Suite 130
Islandia 11722
Tel: (516) 231-3300
TWX: 510-227-6236
FAX: (516) 348-7939

†Intel Corp.
300 Westage Business Center
Suite 230
Fishkill 12524
Tel: (914) 897-3860
FAX: (914) 897-3125

Intel Corp.
Seventeen State Street
14th Floor
New York 10004
Tel: (212) 248-8086
FAX: (212) 248-0888

NORTH CAROLINA

†Intel Corp.
5800 Executive Center Dr.
Suite 105
Charlotte 28212
Tel: (704) 568-8966
FAX: (704) 535-2236

†Intel Corp.
5540 Centerview Dr.
Suite 215
Raleigh 27606
Tel: (919) 851-9537
FAX: (919) 851-8974

OHIO

†Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528
FAX: (513) 890-8658

†Intel Corp.*
25700 Science Park Dr.
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298
FAX: (804) 282-0673

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73162
Tel: (405) 848-8086
FAX: (405) 840-9819

OREGON

†Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741
FAX: (503) 645-8181

PENNSYLVANIA

†Intel Corp.*
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
FAX: (215) 641-0785

†Intel Corp.*
400 Penn Center Blvd.
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970
FAX: (412) 829-7578

PUERTO RICO

†Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

†Intel Corp.
8911 N. Capital of Texas Hwy.
Suite 4230
Austin 78759
Tel: (512) 794-8086
FAX: (512) 338-9335

†Intel Corp.*
12000 Ford Road
Suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: (214) 484-1180

†Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490
FAX: (713) 988-3660

UTAH

†Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

VIRGINIA

†Intel Corp.
9030 Stony Point Pkwy.
Suite 360
Richmond 23235
Tel: (804) 330-9393
FAX: (804) 330-3019

WASHINGTON

†Intel Corp.
155 108th Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002
FAX: (206) 451-9556

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086
FAX: (509) 928-9467

WISCONSIN

Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

CANADA

BRITISH COLUMBIA

Intel Semiconductor of
Canada, Ltd.
4585 Canada Way
Suite 202
Burnaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of
Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: (613) 820-5936

†Intel Semiconductor of
Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of
Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

†Sales and Service Office
*Field Application Location

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955
FAX: 205-751-1581

Hamilton/Avnet Computer
4930 I Corporate Drive
Huntsville 35805

Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel: (205) 837-7210
FAX: 205-721-0356

MTI Systems Sales
4950 Corporate Drive
Suite 120
Huntsville 35806
Tel: (205) 830-9526
FAX: (205) 830-9557

Pioneer/Technologies Group, Inc.
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
FAX: 205-837-9358

ALASKA

Hamilton/Avnet Computer
1400 W. Benson Blvd., Suite 400
Anchorage 99503

ARIZONA

†Arrow Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Hamilton/Avnet Computer
30 South McKemy Avenue
Chandler 85226

Hamilton/Avnet Computer
90 South McKemy Road
Chandler 85226

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5140
TWX: 910-950-0077

Hamilton/Avnet Electronics
30 South McKemy
Chandler 85226
Tel: (602) 961-6669
FAX: 602-961-4073

Wyle Distribution Group
4141 E. Raymond
Phoenix 85040
Tel: (602) 249-2232
TWX: 910-371-2871

CALIFORNIA

Arrow Commercial System Group
1502 Crocker Avenue
Hayward 94544
Tel: (415) 489-5371
FAX: (415) 489-9393

Arrow Commercial System Group
14242 Chambers Road
Tustin 92680
Tel: (714) 544-0200
FAX: (714) 731-8438

†Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (213) 701-7500
TWX: 910-493-2086

†Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
FAX: 619-279-8062

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

†Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

Hamilton/Avnet Computer
3170 Pullman Street
Costa Mesa 92626

Hamilton/Avnet Computer
1361B West 190th Street
Gardena 90248

Hamilton/Avnet Computer
4103 Northgate Blvd.
Sacramento 95834

Hamilton/Avnet Computer
4545 Viewridge Avenue
San Diego 92123

Hamilton/Avnet Computer
1175 Bordeaux Drive
Sunnyvale 94089

Hamilton/Avnet Electronics
21150 Califa Street
Woodland Hills 91367

†Hamilton/Avnet Electronics
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

†Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Ridgeview Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
21150 Califa St.
Woodland Hills 91376
Tel: (818) 594-0404
FAX: 818-594-8233

†Hamilton/Avnet Electronics
10950 W. Washington Blvd.
Culver City 20230
Tel: (213) 558-2458
TWX: 910-340-6364

†Hamilton/Avnet Electronics
1361B West 190th Street
Gardena 90248
Tel: (213) 217-6700
TWX: 910-340-6364

†Hamilton/Avnet Electronics
4103 Northgate Blvd.
Sacramento 95834
Tel: (916) 920-3150

Pioneer/Technologies Group, Inc.
134 Rio Robles
San Jose 95134
Tel: (408) 954-9100
FAX: 408-954-9113

Wyle Distribution Group
124 Maryland Street
El Segundo 90254
Tel: (213) 322-8100

Wyle Distribution Group
7431 Chapman Ave.
Garden Grove 92641
Tel: (714) 891-1717
FAX: 714-891-1621

†Wyle Distribution Group
2951 Sunrise Blvd., Suite 175
Rancho Cordova 95742
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 408-988-2747

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
TWX: 910-371-7127

†Wyle Distribution Group
26677 W. Agoura Rd.
Calabasas 91302
Tel: (818) 880-9000
TWX: 372-0232

COLORADO

Arrow Electronics, Inc.
7060 South Tucson Way
Englewood 80112
Tel: (303) 790-4444

Hamilton/Avnet Computer
9605 Maroon Circle, Ste. 200
Engelwood 80112

†Hamilton/Avnet Electronics
9605 Maroon Circle
Suite 200
Englewood 80112
Tel: (303) 799-0663
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

Hamilton/Avnet Computer
Commerce Industrial Park
Commerce Drive
Danbury 06810

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Pioneer/Standard Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
FAX: 203-838-9901

FLORIDA

†Arrow Electronics, Inc.
400 Fairway Drive
Suite 102
Deerfield Beach 33441
Tel: (305) 429-8200
FAX: 305-428-3991

†Arrow Electronics, Inc.
37 Skyline Drive
Suite 3101
Lake Marv 32746
Tel: (407) 323-0252
FAX: 407-323-3189

Hamilton/Avnet Computer
6801 N.W. 15th Way
Ft. Lauderdale 33309

Hamilton/Avnet Computer
3247 Spring Forest Road
St. Petersburg 33702

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
FAX: 305-971-5420

†Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg 33702
Tel: (813) 573-3930
FAX: 813-572-4329

†Hamilton/Avnet Electronics
6947 University Boulevard
Winter Park 32792
Tel: (407) 628-3888
FAX: 407-678-1878

†Pioneer/Technologies Group, Inc.
337 Northlake Blvd., Suite 1000
Alta Monte Springs 32701
Tel: (407) 834-9090
FAX: 407-834-0865

Pioneer/Technologies Group, Inc.
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
FAX: 305-481-2950

GEORGIA

Arrow Commercial System Group
3400 C. Corporate Way
Deluth 30139
Tel: (404) 623-8825
FAX: (404) 623-8802

†Arrow Electronics, Inc.
4250 E. Rivergreen Parkway
Deluth 30136
Tel: (404) 497-1300
TWX: 810-766-0439

Hamilton/Avnet Computer
5825 D. Peachtree Corners E.
Norcross 30092

†Hamilton/Avnet Electronics
5825 D Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer/Technologies Group, Inc.
3100 F Northwoods Place
Norcross 30071
Tel: (404) 448-1711
FAX: 404-446-8270

ILLINOIS

†Arrow Electronics, Inc.
1140 W. Thorndale
Itasca 60143
Tel: (708) 250-0500
TWX: 708-250-0916

Hamilton/Avnet Computer
1130 Thorndale Avenue
Bensenville 60106

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (708) 860-7780
TWX: 708-860-8530

MTI Systems Sales
1100 W. Thorndale
Itasca 60143
Tel: (708) 773-2300

†Pioneer/Standard Electronics
2171 Executive Dr., Suite 200
Addison 60101
Tel: (708) 495-9680
FAX: 708-495-9831

INDIANA

†Arrow Electronics, Inc.
7108 Lakeview Parkway West Drive
Indianapolis 46268
Tel: (317) 299-2071
FAX: 317-299-0255

Hamilton/Avnet Computer
485 Gradle Drive
Carmel 46032

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
FAX: 317-844-5921

†Pioneer/Standard Electronics
9350 Priority Way
West Drive
Indianapolis 46250
Tel: (317) 573-0880
FAX: 317-573-0979

†Certified Technical Distributor

DOMESTIC DISTRIBUTORS (Contd.)

IOWA

Hamilton/Avnet Computer
915 33rd Avenue SW
Cedar Rapids 52404

Hamilton/Avnet Electronics
915 33rd Avenue, S.W.
Cedar Rapids 52404
Tel: (319) 362-4757

KANSAS

Arrow Electronics, Inc.
8208 Melrose Dr., Suite 210
Lenexa 66214
Tel: (913) 541-9542
FAX: 913-541-0328

Hamilton/Avnet Computer
15313 W. 95th Street
Lenexa 61219

†Hamilton/Avnet Electronics
15313 W. 95th
Overland Park 66215
Tel: (913) 888-8900
FAX: 913-541-7951

KENTUCKY

Hamilton/Avnet Electronics
805 A. Newtown Circle
Lexington 40511
Tel: (606) 259-1475

MARYLAND

†Arrow Electronics, Inc.
8300 Guilford Drive
Suite H, River Center
Columbia 21046
Tel: (301) 995-6002
FAX: 301-381-3854

Hamilton/Avnet Computer
6822 Oak Hall Lane
Columbia 21045

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
FAX: 301-995-3593

†Mesa Technology Corp.
9720 Patuxent Woods Dr.
Columbia 21046
Tel: (301) 290-8150
FAX: 301-290-6474

†Pioneer/Technologies Group, Inc.
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
FAX: 301-921-4255

MASSACHUSETTS

Arrow Electronics, Inc.
25 Upton Dr.
Wilmington 01887
Tel: (508) 658-0900
TWX: 710-393-6770

Hamilton/Avnet Computer
10 D Centennial Drive
Peabody 01960

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (508) 532-9838
FAX: 508-596-7802

†Pioneer/Standard Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
FAX: 617-863-1547

Wyle Distribution Group
15 Third Avenue
Burlington 01803
Tel: (617) 272-7300
FAX: 617-272-6809

MICHIGAN

†Arrow Electronics, Inc.
19880 Haggerty Road
Livonia 48152
Tel: (313) 665-4100
TWX: 810-223-6020

Hamilton/Avnet Computer
2215 S.E. A-5
Grand Rapids 49508

Hamilton/Avnet Computer
41650 Garden Rd., Ste. 100
Novi 48050

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
FAX: 616-698-1831

Hamilton/Avnet Electronics
41650 Garden Brook
Novi 48050
Tel: (313) 347-4271
FAX: 313-347-4021

†Pioneer/Standard Electronics
4505 Broadmoor S.E.
Grand Rapids 49508
Tel: (616) 698-1800
FAX: 616-698-1831

†Pioneer/Standard Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
FAX: 313-427-3720

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

Hamilton/Avnet Computer
12400 Whitewater Drive
Minnetonka 55343

†Hamilton/Avnet Electronics
12400 Whitewater Drive
Minnetonka 55434
Tel: (612) 932-0600
TWX: 910-576-2720

†Pioneer/Standard Electronics
7625 Golden Triange Dr.
Suite G
Eden Prairie 55343
Tel: (612) 944-3355
FAX: 612-944-3794

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
FAX: 314-567-1164

Hamilton/Avnet Computer
739 Goddard Avenue
Chesterfield 63005

†Hamilton/Avnet Electronics
741 Goddard
Chesterfield 63005
Tel: (314) 537-1600
FAX: 314-537-4248

NEW HAMPSHIRE

Hamilton/Avnet Computer
2 Executive Park Drive
Bedford 03102

Hamilton/Avnet Computer
444 East Industrial Park Dr.
Manchester 03103

NEW JERSEY

†Arrow Electronics, Inc.
4 East Stow Road
Unit 11
Marlton 08053
Tel: (609) 596-8000
FAX: 609-596-9632

†Arrow Electronics
6 Century Drive
Parsipanny 07054
Tel: (201) 538-0900
FAX: 201-538-0900

Hamilton/Avnet Computer
1 Keystone Ave., Bldg. 36
Cherry Hill 08003

Hamilton/Avnet Computer
10 Industrial Road
Fairfield 07006

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
FAX: 609-751-2552

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
FAX: 201-575-5839

†MTI Systems Sales
9 Law Drive
Fairfield 07006
Tel: (201) 227-5552
FAX: 201-575-6336

†Pioneer/Standard Electronics
14-A Madison Rd.
Fairfield 07006
Tel: (201) 575-3510
FAX: 201-575-3454

NEW MEXICO

Alliance Electronics Inc.
10510 Research Avenue
Albuquerque 87123
Tel: (505) 292-3360
FAX: 505-292-6537

Hamilton/Avnet Computer
5659 Jefferson, N.E. Suites A & B
Albuquerque 87109

†Hamilton/Avnet Electronics
5659A Jefferson N.E.
Albuquerque 87109
Tel: (505) 765-1500
FAX: 505-243-1395

NEW YORK

†Arrow Electronics, Inc.
3375 Brighton Henrietta Townline Rd.
Rochester 14623
Tel: (716) 427-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet Computer
933 Motor Parkway
Haupauge 11788

Hamilton/Avnet Computer
2060 Townline
Rochester 14623

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†Hamilton/Avnet Electronics
2060 Townline Rd.
Rochester 14623
Tel: (716) 272-2744
TWX: 510-253-5470

Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-0288
TWX: 710-541-1560

†MTI Systems Sales
38 Harbor Park Drive
Port Washington 11050
Tel: (516) 621-6200
FAX: 510-223-0846

Pioneer/Standard Electronics
68 Corporate Drive
Binghamton 13904
Tel: (607) 722-9300
FAX: 607-722-9562

Pioneer/Standard Electronics
40 Oser Avenue
Hauppauge 11787
Tel: (516) 231-9200
FAX: 510-227-9869

†Pioneer/Standard Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
FAX: 516-921-2143

†Pioneer/Standard Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
FAX: 716-381-5955

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greensdairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

Hamilton/Avnet Computer
3510 Spring Forest Road
Raleigh 27604

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer/Technologies Group, Inc.
9401 L-Southern Pine Blvd.
Charlotte 28210
Tel: (919) 527-8188
FAX: 704-522-8564

Pioneer Technologies Group, Inc.
2810 Meridian Parkway
Suite 148
Durham 27713
Tel: (919) 544-5400
FAX: 919-544-5885

OHIO

Arrow Commercial System Group
284 Cramer Creek Court
Dublin 43017
Tel: (614) 889-9347
FAX: (614) 889-9680

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

Hamilton/Avnet Computer
7764 Washington Village Dr.
Dayton 45459

Hamilton/Avnet Computer
30325 Bainbridge Rd., Bldg. A
Solon 44139

†Hamilton/Avnet Electronics
7760 Washington Village Dr.
Dayton 45459
Tel: (513) 439-6733
FAX: 513-439-6711

†Hamilton/Avnet Electronics
30325 Bainbridge
Solon 44139
Tel: (216) 349-5100
TWX: 810-427-9452

Hamilton/Avnet Computer
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004
FAX: 614-882-8650

Hamilton/Avnet Electronics
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004

MTI Systems Sales
23400 Commerce Park Road
Beachwood 44122
Tel: (216) 464-6688

†Pioneer/Standard Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
FAX: 513-236-8133

†Pioneer/Standard Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
FAX: 216-663-1004

†Certified Technical Distributor

DOMESTIC DISTRIBUTORS (Contd.)

OKLAHOMA

Arrow Electronics, Inc.
4719 South Memorial Dr.
Tulsa 74145

†Hamilton/Avnet Electronics
12121 E. 51st St., Suite 102A
Tulsa 74146
Tel: (918) 252-7297

OREGON

†Almac Electronics Corp.
1885 N.W. 169th Place
Beaverton 97005
Tel: (503) 629-8090
FAX: 503-645-0611

Hamilton/Avnet Computer
9409 Southwest Nimbus Ave.
Beaverton 97005

†Hamilton/Avnet Electronics
9409 S.W. Nimbus Ave.
Beaverton 97005
Tel: (503) 627-0201
FAX: 503-641-4012

Wyle
9640 Sunshine Court
Bldg. G, Suite 200
Beaverton 97005
Tel: (503) 643-7900
FAX: 503-646-5466

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Hamilton/Avnet Computer
2800 Liberty Ave., Bldg. E
Pittsburgh 15222

Hamilton/Avnet Electronics
2800 Liberty Ave.
Pittsburgh 15238
Tel: (412) 281-4150

Pioneer/Standard Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
FAX: 412-963-8255

†Pioneer/Technologies Group, Inc.
Delaware Valley
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
FAX: 215-674-3107

TENNESSEE

Arrow Commercial System Group
3635 Knight Road
Suite 7
Memphis 38118
Tel: (901) 367-0540
FAX: (901) 367-2081

TEXAS

Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
FAX: (214) 248-7208

Hamilton/Avnet Computer
1807A West Braker Lane
Austin 78758

Hamilton/Avnet Computer
Forum 2
4004 Beltline, Suite 200
Dallas 75244

Hamilton/Avnet Computer
4850 Wright Rd., Suite 190
Stafford 77477

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
4004 Beltline, Suite 200
Dallas 75234
Tel: (214) 308-8111
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Rd., Suite 190
Stafford 77477
Tel: (713) 240-7733
TWX: 910-881-5523

†Pioneer/Standard Electronics
1826-D Kramer
Austin 78758
Tel: (512) 835-4000
FAX: 512-835-9829

†Pioneer/Standard Electronics
13710 Omega Road
Dallas 75244
Tel: (214) 386-7300
FAX: 214-490-6419

†Pioneer/Standard Electronics
10530 Rockley Road
Houston 77099
Tel: (713) 495-4700
FAX: 713-495-5642

†Wyle Distribution Group
1810 Greenville Avenue
Richardson 75081
Tel: (214) 235-9953
FAX: 214-644-5064

UTAH

Hamilton/Avnet Computer
1585 West 2100 South
Salt Lake City 84119

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

†Wyle Distribution Group
1325 West 2200 South
Suite E
West Valley 84119
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
FAX: 206-643-9709

Hamilton/Avnet Computer
17761 Northeast 78th Place
Redmond 98052

†Hamilton/Avnet Electronics
17761 N.E. 78th Place
Redmond 98052
Tel: (206) 881-6697
FAX: 206-867-0159

Wyle Distribution Group
15385 N.E. 90th Street
Redmond 98052
Tel: (206) 881-1150
FAX: 206-881-1567

WISCONSIN

Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 792-0150
FAX: 414-792-0156

Hamilton/Avnet Computer
20875 Crossroads Circle
Suite 400
Waukesha 53186

†Hamilton/Avnet Electronics
28875 Crossroads Circle
Suite 400
Waukesha 53186
Tel: (414) 784-4510
FAX: 414-784-9509

CANADA

ALBERTA

Hamilton/Avnet Computer
2816 21st Street Northeast
Calgary T2E 6Z2

Hamilton/Avnet Electronics
2816 21st Street N.E. #3
Calgary T2E 6Z3
Tel: (403) 230-3586
FAX: 403-250-1591

Zentronics
6815 #8 Street N.E.
Suite 100
Calgary T2E 7H
Tel: (403) 295-8818
FAX: 403-295-8714

BRITISH COLUMBIA

†Hamilton/Avnet Electronics
8610 Commerce Ct.
Burnaby V5A 4N6
Tel: (604) 420-4101
FAX: 604-437-4712

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
FAX: 604-273-2413

ONTARIO

Arrow Electronics, Inc.
36 Antares Dr., Unit 100
Nepean K2E 7W5
Tel: (613) 226-6903
FAX: 613-723-2018

†Arrow Electronics, Inc.
1093 Meyerside, Unit 2
Mississauga L5T 1M4
Tel: (416) 673-7769
FAX: 416-672-0849

Hamilton/Avnet Computer
Canada System Engineering Group
3688 Nashua Drive
Units 7 & 8
Mississuaga L4V 1M5

Hamilton/Avnet Computer
3688 Nashua Drive
Units 9 & 10
Mississuaga L4V 1M5

Hamilton/Avnet Computer
6845 Rexwood Road
Units 7, 8, & 9
Mississuaga L4V 1R2

Hamilton/Avnet Computer
190 Colonade Road
Nepean K2E 7J5

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units 3-4-5
Mississauga L4T 1R2
Tel: (416) 677-7432
FAX: 416-677-0940

†Hamilton/Avnet Electronics
190 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
FAX: 613-226-1184

†Zentronics
1355 Meyerside Drive
Mississauga L5T 1C9
Tel: (416) 564-9600
FAX: 416-564-8320

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840
FAX: 613-226-6352

QUEBEC

Arrow Electronics Inc.
1100 St. Regis
Dorval H9P 2T5
Tel: (514) 421-7411
FAX: 514-421-7430

Arrow Electronics, Inc.
500 Boul. St-Jean-Baptiste
Suite 280
Quebec G2E 5R9
Tel: (418) 871-7500
FAX: 418-871-6816

Hamilton/Avnet Computer
2795 Rue Halpern
St. Laurent H4S 1P8

†Hamilton/Avnet Electronics
2795 Halpern
St. Laurent H2E 7K1
Tel: (514) 335-1000
FAX: 514-335-2481

†Zentronics
520 McCaffrey
St. Laurent H4T 1N3
Tel: (514) 737-9700
FAX: 514-737-5212

†Certified Technical Distributor

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.
Unit 13
Allambie Grove Business Park
25 Frenchs Forest Road East
Frenchs Forest, NSW, 2086
Tel: 61-2975-3300
FAX: 61-2975-3375

BRAZIL

Intel Semiconductores do Brazil LTDA
Avenida Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 11-37-557-ISDB
FAX: 55-11-287-5119

CHINA/HONG KONG

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

Intel Semiconductor Ltd.*
10/F East Tower
Bond Center
Queensway, Central
Hong Kong
Tel: (852) 844-4555
FAX: (852) 868-1989

INDIA

Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001
Tel: 91-812-215773
TLX: 953-845-2646 INTEL IN
FAX: 091-812-215067

JAPAN

Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26
Tel: 0298-47-8511
TLX: 3656-160
FAX: 0298-47-8450

Intel Japan K.K.*
Hachioji ON Bldg.
4-7-14 Myojin-machi
Hachioji-shi, Tokyo 192
Tel: 0426-48-8770
FAX: 0426-48-8775

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360
Tel: 0485-24-6871
FAX: 0485-24-7518

Intel Japan K.K.*
Kawa-asa Bldg.
2-11-5 Shin-Yokohama
Kohoku-ku, Yokohama-shi
Kanagawa, 222
Tel: 045-474-7661
FAX: 045-471-4394

Intel Japan K.K.*
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560
Tel: 06-863-1091
FAX: 06-863-1084

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-3201-3621
FAX: 03-3201-6850

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 450
Tel: 052-204-1261
FAX: 052-204-1285

KOREA

Intel Korea, Ltd.
16th Floor, Life Bldg.
61 Yoido-dong, Youngdeungpo-Ku
Seoul 150-010
Tel: (2) 784-8186
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #08-03/06
United Square
Singapore 1130
Tel: (65) 250-7811
FAX: (65) 250-9256

TAIWAN

Intel Technology Far East Ltd.
Taiwan Branch Office
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei
Tel: 886-2-716-9660
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

Dafsys S.R.L.
Chacabuco, 90-6 Piso
1069-Buenos Aires
Tel: 54-1-34-7726
FAX: 54-1-34-1871

AUSTRALIA

Email Electronics
15-17 Hume Street
Huntingdale, 3166
Tel: 011-61-3-544-8244
TLX: AA 30895
FAX: 011-61-3-543-8179

NSD-Australia
205 Middleborough Rd.
Box Hill, Victoria 3128
Tel: 03 8900970
FAX: 03 8990819

BRAZIL

Elebra Componentes
Rua Geraldo Flausina Gomes, 78
7 Andar
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9641
TLX: 55-11-54593/54591
FAX: 55-11-534-9424

CHINA/HONG KONG

Novel Precision Machinery Co., Ltd.
Room 728 Trade Square
681 Cheung Sha Wan Road
Kowloon, Hong Kong
Tel: (852) 360-8999
TWX: 32032 NVTNL HX
FAX: (852) 725-3695

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 011-91-812-600-631
011-91-812-611-365
TLX: 9538458332 MDBG

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Trombay Road
Chembur
Bombay 400 071
TLX: 9531 171447 MDEV

Micronic Devices
25/8, 1st Floor
Bada Bazaar Marg
Old Rajinder Nagar
New Delhi 110 060
Tel: 011-91-11-5723509
011-91-11-589771
TLX: 031-63253 MDND IN

Micronic Devices
6-3-348/12A Dwarakapuri Colony
Hyderabad 500 482
Tel: 011-91-842-226748

S&S Corporation
1587 Kooser Road
San Jose, CA 95118
Tel: (408) 978-6216
TLX: 820281
FAX: (408) 978-8635

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

CTC Components Systems Co., Ltd.
4-8-1 Dobashi, Miyamae-ku
Kawasaki-shi, Kanagawa 213
Tel: 044-852-5121
FAX: 044-877-4268

Dia Semicon Systems, Inc.
Flower Hill Shinmachi Higashi-kan
1-23-9 Shinmachi, Setagaya-ku
Tokyo 154
Tel: 03-3439-1600
FAX: 03-3439-1601

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-2916
FAX: 052-204-2901

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-3546-5011
FAX: 03-3546-5044

KOREA

J-Tek Corporation
Dong Sung Bldg. 9/F
158-24, Samsung-Dong, Kangnam-Ku
Seoul 135-090
Tel: (822) 557-8039
FAX: (822) 557-8304

Samsung Electronics
Samsung Main Bldg.
150 Taepyung-Ro-2KA, Chung-Ku
Seoul 100-102
C.P.O. Box 8780
Tel: (822) 751-3680
TWX: KORSST K 27970
FAX: (822) 753-9065

MEXICO

SSB Electronics, Inc.
675 Palomar Street, Bldg. 4, Suite A
Chula Vista, CA 92011
Tel: (619) 585-3253
TLX: 287751 CBALL UR
FAX: (619) 585-8322

Dicopel S.A.
Tochtli 368 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 52-5-561-3211
TLX: 177 3790 Dicome
FAX: 52-5-561-1279

PSI S.A. de C.V.
Fco. Villa esq. Ajusco s/n
Cuernavaca – Morelos
Tel: 52-73-13-9412
FAX: 52-73-17-5333

NEW ZEALAND

Email Electronics
36 Olive Road
Penrose, Auckland
Tel: 011-64-9-591-155
FAX: 011-64-9-592-681

SAUDI ARABIA

AAE Systems, Inc.
642 N. Pastoria Ave.
Sunnyvale, CA 94086
U.S.A.
Tel: (408) 732-1710
FAX: (408) 732-3095
TLX: 494-3405 AAE SYS

SINGAPORE

Electronic Resources Pte, Ltd.
17 Harvey Road
#03-01 Singapore 1336
Tel: (65) 283-0888
TWX: RS 56541 ERS
FAX: (65) 289-5327

SOUTH AFRICA

Electronic Building Elements
178 Erasmus St. (off Watermeyet St.)
Meyerspark, Pretoria, 0184
Tel: 011-2712-803-7680
FAX: 011-2712-803-8294

TAIWAN

Micro Electronics Corporation
12th Floor, Section 3
285 Nanking East Road
Taipei, R.O.C.
Tel: (886) 2-7198419
FAX: (886) 2-7197916

Acer Sertek Inc.
15th Floor, Section 2
Chien Kuo North Rd.
Taipei 18479 R.O.C.
Tel: 886-2-501-0055
TWX: 23756 SERTEK
FAX: (886) 2-5012521

*Field Application Location

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26

FRANCE
Intel Corporation S.A.R.L.
1, Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen

HONG KONG
Intel Semiconductor Ltd.
10/F East Tower
Bond Center
Queensway, Central

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Order Number: 240874-001
Printed in U.S.A./X51400.A/10K/0591/BL GC
MICROPROCESSORS