MILITARY i860™ XR 32/64-BIT MICROPROCESSOR

- **Parallel Architecture that Supports Up to Three Operations per Clock**
  - **— One Integer or Control Instruction per Clock**
  - **— Up to Two Floating-Point Results per Clock**
- **High Performance Design**
  - **— 80 Peak Single Precision MFLOPs**
  - **— 60 Peak Double Precision MFLOPs**
  - **— 64-Bit External Data Bus**
  - **— 64-Bit Internal Instruction Cache Bus**
  - **— 128-Bit Internal Data Cache Bus**
- **High Level of Integration on One Chip**
  - **— 32-Bit Integer and Control Unit**
  - **— 32/64-Bit Pipelined Floating-Point Adder and Multiplier Units**
  - **— 64-Bit 3-D Graphics Unit**
  - **— Paging Unit with Translation Lookaside Buffer (TLB)**
  - **— 4 Kbyte Instruction Cache**
  - **— 8 Kbyte Data Cache**
- **Compatible with Industry Standards**
  - **— ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
  - **— i386™/i486™ Microprocessor Data Formats and Page Table Entries**
- **Easy to Use**
  - **— On-Chip Debug Register**
  - **— Assembler, Linker, Simulator, Debugger, C, FORTRAN and Ada Compilers, FORTRAN Vectorizer, Scalar and Vector Math Libraries for both VMS* and UNIX* Environments**
- **Military Temperature Range: −55°C to +125°C (T_C)**
- **Available in a 168-pin Ceramic Pin Grid Array Package** (See Package/Module/PC Card Outlines and Dimensions, Order # 231369)
- **Available in 196-Lead Ceramic Quad Flatpack** (See Package/Module/PC Card Outlines and Dimensions, Order # 231369)

The Intel i860™ XR Microprocessor delivers supercomputing performance in a single VLSI component. The 32/64-bit design of the i860 XR Microprocessor balances integer, floating point and graphics performance for applications such as target tracking, acoustic imaging, terrain data mapping and image processing. Its parallel architecture achieves high throughput with RISC design techniques, pipelined processing units, wide data paths, large on-chip caches, million-transistor design, and fast one-micron CHMOS IV silicon technology.

NOTE:
References to devices within this document refer to the Military versions of those devices.

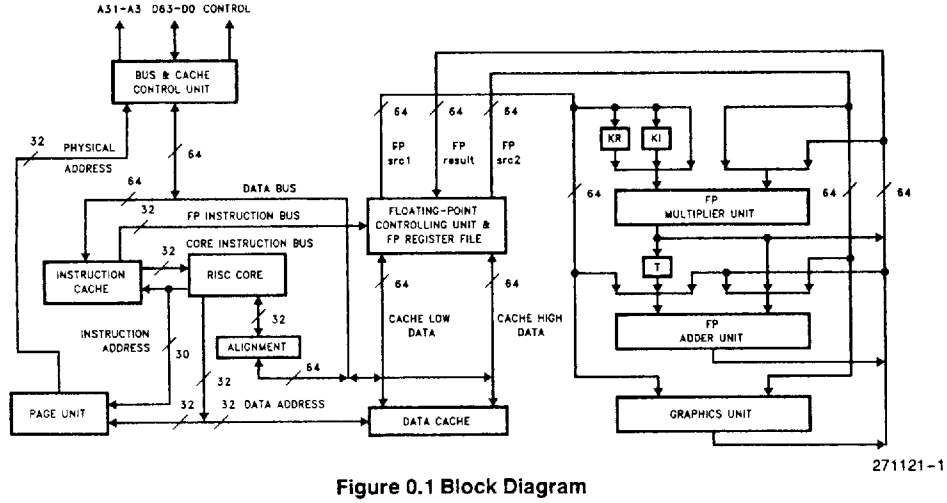


Figure 0.1 Block Diagram

Intel, intel, i386, i486 and i860 are trademarks of Intel Corporation
*UNIX is a registered trademark of AT&T. *VMS is a trademark of Digital Equipment Corporation.

October 1992
Order Number: 271121-005

Military i860™ XR 32/64-Bit Microprocessor

CONTENTS

PAGE

1.0 FUNCTIONAL DESCRIPTION 9-7

2.0 PROGRAMMING INTERFACE 9-7

2.1 Data Types 9-8
2.1.1 Integer 9-8
2.1.2 Ordinal 9-8
2.1.3 Single- and Double-Precision Real 9-8
2.1.4 Pixel 9-9
2.2 Register Set 9-9
2.2.1 Integer Register File 9-10
2.2.2 Floating-Point Register File 9-10
2.2.3 Processor Status Register 9-10
2.2.4 Extended Processor Status Register 9-13
2.2.5 Data Breakpoint Register 9-14
2.2.6 Directory Base Register 9-14
2.2.7 Fault Instruction Register 9-15
2.2.8 Floating-Point Status Register 9-15
2.2.9 KR, KI, T and MERGE Registers 9-16
2.3 Addressing 9-17
2.4 Virtual Addressing 9-17
2.4.1 Page Frame 9-19
2.4.2 Virtual Address 9-19
2.4.3 Page Tables 9-19
2.4.4 Table Entries 9-20
2.4.5 Address Translation Algorithm 9-22
2.4.6 Address Translation Faults 9-23
2.4.7 Page Translation Cache 9-23
2.5 Caching and Cache Flushing 9-23
2.6 Instruction Set 9-24
2.6.1 Pipelined and Scalar Operations 9-24
2.6.2 Dual-Instruction Mode 9-27
2.6.3 Dual-Operation Instructions 9-28

CONTENTS

PAGE

2.0 PROGRAMMING INTERFACE (Continued)

2.7 Addressing Modes 9-28
2.8 Traps and Interrupts 9-29
2.8.1 Trap Handler Invocation 9-29
2.8.2 Instruction Fault 9-30
2.8.3 Floating-Point Fault 9-30
2.8.4 Instruction Access Fault 9-31
2.8.5 Data Access Fault 9-31
2.8.6 Interrupt Trap 9-32
2.8.7 Reset Trap 9-32
2.9 Debugging 9-32

3.0 HARDWARE INTERFACE 9-32
3.1 Signal Description 9-32
3.1.1 Clock (CLK) 9-33
3.1.2 System Reset (RESET) 9-33
3.1.3 Bus Hold (HOLD) and Bus Hold Acknowledge (HLDA) 9-33
3.1.4 Bus Request (BREQ) 9-33
3.1.5 Interrupt/Code-Size (INT/CS8) 9-33
3.1.6 Address Pins (A31-A3) and Byte Enables ($\overline{BE7}$-$\overline{BE0}$) 9-34
3.1.7 Data Pins (D63-D0) 9-35
3.1.8 Bus Lock ($\overline{LOCK}$) 9-35
3.1.9 Write/Read Bus Cycle (W/$\overline{R}$) 9-35
3.1.10 Next Near ($\overline{NENE}$) 9-35
3.1.11 Next Address Request ($\overline{NA}$) 9-35
3.1.12 Transfer Acknowledge ($\overline{READY}$) 9-36
3.1.13 Address Status ($\overline{ADS}$) 9-36
3.1.14 Cache Enable ($\overline{KEN}$) 9-36
3.1.15 Page Table Bit (PTB) 9-36
3.1.16 Boundary Scan Shift Input (SHI) 9-36
3.1.17 Boundary Scan Enable (BSCN) 9-36
3.1.18 Shift Scan Path (SCAN) 9-36
3.1.19 Configuration (CC1-CC0) 9-36
3.1.20 System Power (V_{CC}) and Ground (V_{SS}) 9-36
3.2 Initialization 9-37
3.3 Testability 9-37
3.3.1 Normal Mode 9-38
3.3.2 Shift Mode 9-38

CONTENTS

PAGE

4.0 BUS OPERATION 9-38
4.1 Pipelining 9-39
4.2 Bus State Machine 9-39
4.3 Bus Cycles 9-41
4.3.1 Nonpipelined Read Cycles 9-41
4.3.2 Nonpipelined Write Cycles 9-42
4.3.3 Pipelined Read and Write Cycles 9-44
4.3.4 Locked Cycles 9-46
4.3.5 HOLD and BREQ Arbitration Cycles 9-46
4.4 Bus States During RESET 9-47

5.0 MECHANICAL DATA 9-48

6.0 ELECTRICAL DATA 9-55
6.1 Absolute Maximum Ratings 9-55
6.2 Operating Conditions 9-55
6.3 DC Characteristics 9-55
6.4 AC Characteristics 9-56

7.0 INSTRUCTION SET 9-58
7.1 Instruction Definitions in Alphabetical Order 9-59
7.2 Instruction Format and Encoding 9-67
7.2.1 REG-Format Instructions 9-67
7.2.2 CTRL-Format Instructions 9-70
7.2.3 Floating-Point Instructions 9-71
7.3 Instruction Timings 9-73
7.4 Instruction Characteristics 9-76

FIGURES

Figure 0.1 Block Diagram
Figure 2.1 Real Number Formats
Figure 2.2 Pixel Format Example
Figure 2.3 Registers and Data Paths
Figure 2.4 Processor Status Register (psr)
Figure 2.5 Extended Processor Status Register (epsr)
Figure 2.6 Directory Base Register (dirbase)
Figure 2.7 Floating-Point Status Register (fsr)
Figure 2.8 Little and Big Endian Accesses
Figure 2.9 Format of a Virtual Address
Figure 2.10 Address Translation
Figure 2.11 Format of a Page-Table and Page-Directory Entry
Figure 2.12 FP-Adder Pipelined Instruction Execution
Figure 2.13 Dual-Instruction Mode Transitions
Figure 2.14 Dual-Operation Data Paths
Figure 3.1 Order of Boundary Scan Chain
Figure 4.1 Bus State Machine
Figure 4.2 Fastest Read Cycles
Figure 4.3 Fastest Write Cycles
Figure 4.4 Fastest Read/Write Cycles
Figure 4.5 Pipelined Read Followed by Pipelined Write
Figure 4.6 Pipelined Write Followed by Pipelined Read
Figure 4.7 Pipelining Driven by $\overline{NA}$
Figure 4.8 $\overline{NA}$ Active with No Internal Bus Request
Figure 4.9 Locked Cycles
Figure 4.10 HOLD, HLDA and BREQ
Figure 4.11 Reset Activities
Figure 5.1 168-Pin Ceramic Pin Grid Array. Pin Configuration—View from Top Side
Figure 5.2 168-Pin Ceramic Pin Grid Array. Pin Configuration—View from Pin Side
Figure 5.3 196-Pin Ceramic Quad Flatpack. Pin Configuration—View from Lid Side
Figure 6.1 CLK, Input and Output Timings
Figure 7.1 REG-Format Variations
Figure 7.2 Core Escape Instruction Format
Figure 7.3 CTRL Instruction Format
Figure 7.4 Floating-Point Instruction Encoding

TABLES

Table 2.1 Pixel Formats
Table 2.2 Values of PS
Table 2.3 Values of RB
Table 2.4 Values of RC
Table 2.5 Values of RM
Table 2.6 Combining Directory and Page Protections
Table 2.7 Instruction Set
Table 2.8 Types of Traps
Table 2.9 Register and Cache Values after Reset
Table 3.1 Pin Summary
Table 3.2 Indentifying Instruction Fetches
Table 3.3 Cacheability Based on $\overline{KEN}$ and CD or WT
Table 3.4 Output Pin Status During Reset
Table 3.5 Test Mode Selection
Table 3.6 Test Mode Latches
Table 5.1 168-Pin Ceramic PGA Pin Assignment by Location
Table 5.2 168-Pin Ceramic PGA Pin Assignment by Function
Table 5.3 196-Pin Ceramic Quad Flatpack (CQFP) Pin Assignment by Location
Table 5.4 196-Pin Ceramic Quad Flatpack (CQFP) Pin Assignment by Function
Table 6.1 DC Characteristics
Table 6.2 AC Characteristics
Table 7.1 Precision Specification
Table 7.2 FADDP MERGE Update
Table 7.3 Register Encoding
Table 7.4 REG-Format Opcodes
Table 7.5 Core Escape Opcodes
Table 7.6 CTRL-Format Opcodes
Table 7.7 Floating-Point Opcodes
Table 7.8 DPC Encoding
Table 7.9 Instruction Characteristics

1.0 FUNCTIONAL DESCRIPTION

As shown by the block diagram on the front page, the Military i860 XR microprocessor consists of 9 units:

1. RISC Based Core Execution Unit
2. Floating-Point 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

The core execution unit controls overall operation of the Military i860 XR microprocessor. The core unit executes load, store, integer, bit, and control-transfer operations, and fetches instructions for the floating-point unit as well. A set of 32 x 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 x 64-bit registers, or 32 x 32-bit registers. Special load and store instructions can also access these same registers as 8 x 128-bit registers. All floating-point 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 floating-point 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 and 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 has special integer logic that 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 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 Military i860 XR microprocessor assume that the surface of a solid object is drawn with polygon patches whose shapes approximate 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 Military i860 XR microprocessor directly aid such interpolation.

The paging unit implements a protected, paged, virtual memory system via a 64-entry, four-way set-associative TLB (Translation Lookaside Buffer). The paging unit uses the TLB 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.

The instruction cache is a two-way set-associative memory of four Kbytes, with a 32-byte line size. It transfers up to 64 bits per clock (266 Mbyte/sec at 33 MHz).

The data cache is a two-way set-associative memory of eight Kbytes, with a 32-byte line size. It transfers up to 128 bits per clock (640 Mbyte/sec at 40 MHz). The Military i860 XR microprocessor uses writeback caching, i.e. memory writes update the cache (if applicable) without necessarily updating memory immediately; however, caching can be inhibited by software where necessary.

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 instuction-cache miss processing, controls TLB translation, and provides the interface to the external bus. Its pipelined structure supports up to three outstanding bus cycles.

2.0 PROGRAMMING INTERFACE

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

2.1 Data Types

The Military i860 XR 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 oriented 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 e = 0 and f ≠ 0 or e = 255 then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If 0 < e < 255, then the value is $(-1)^s \times 1.f \times 2^{e-127}$.
3. If e = 0 and 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 e = 0 and f ≠ 0 or e = 2047, then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If 0 < e < 2047, then the value is $(-1)^s \times 1.f \times 2^{e-1023}$.

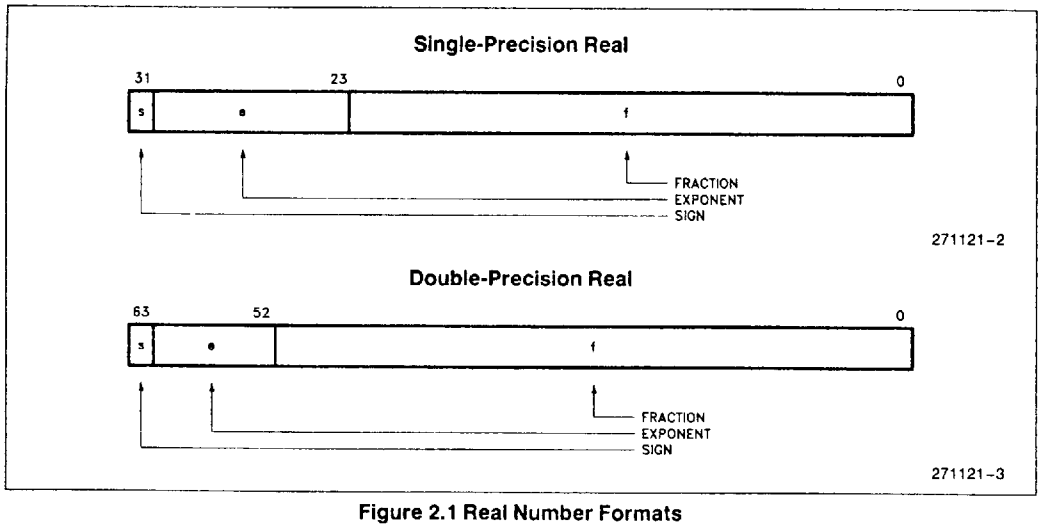


Figure 2.1 Real Number Formats

3. If e = 0 and 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-precision 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 Military i860 XR microprocessor always operates on 64 bits worth of pixels 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 Military i860 XR 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 Military i860 XR microprocessor defines only the field sizes, not the specific use of each field. Other ways of using the fields of pixels are possible.

Table 2.1 Pixel Formats

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

The intensity attribute fields may be assigned to colors in any order convenient to the application.

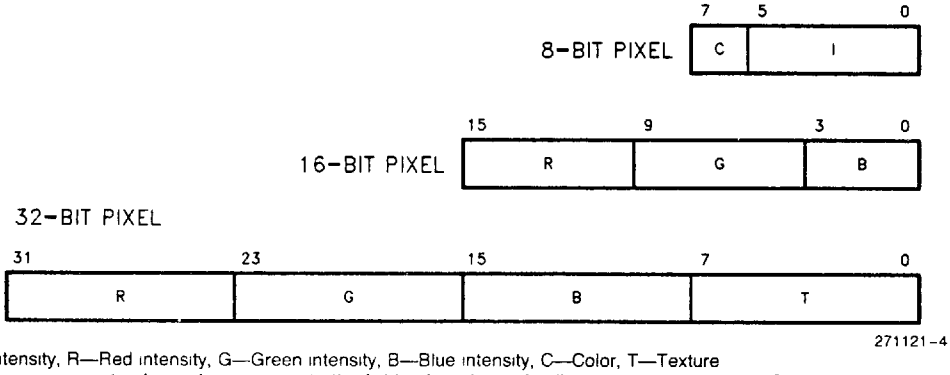
*With 8-bit pixels, up to 8 bits can be used for intensity; the remaining bits can be used for any other attribute, such as color. The intensity bits must be the low-order bits of the pixel.

2.2 Register Set

As Figure 2 3 shows, the Military i860 XR microprocessor has the following registers:

- An integer register file
- A floating-point register file
- Six control registers (**psr, epsr, db, dirbase, fir** and **fsr**)
- Four special-purpose registers (KR, KI, T and MERGE)

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, T and MERGE are used by a few specific instructions.



I—Intensity, R—Red intensity, G—Green intensity, B—Blue intensity, C—Color, T—Texture
These assignments of specific meanings to the fields of pixels are for illustration purposes only. Only the field sizes are defined, not the specific use of each field.

Figure 2.2 Pixel Format Example

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, independently of what is stored in it. This special behavior of **r0** makes it useful for modifying the function of certain instructions.

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, independently of what is stored in them. The floating-point registers are also used by a set of graphics operations, primarily for 3D graphics computations.

When accessing 64-bit floating-point or integer values, the Military i860 XR microprocessor uses an even/odd pair of registers. When accessing 128-bit values, it uses an aligned set of four contiguous registers (**f0, f4, f8, . . . , f28**). The instruction must designate the lowest register number of the set of registers containing 64- or 128-bit values. The register with the lowest number contains the least significant part of the value. For 128-bit values, the register pair with the lower numbers contain the least significant 64 bits while the register pair with the higher numbers contain the most significant 64 bits.

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 the extraordinarily high rate of computation.

2.2.3 PROCESSOR STATUS REGISTER

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

- BR (Break Read) and BW (Break Write) enable a data access trap when the operand address matches the address in the Data Breakpoint (**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 Branch on CC and Add instruction sets and tests LCC (Loop Condition Code).
- IM (Interrupt Mode) enables external interrupts if set; disables interrupts if clear.
- U (User Mode) is set when the Military i860 XR microprocessor is executing in user mode; it is clear when the Military i860 XR microprocessor is executing in supervisor mode. In user mode, writes to some control registers are inhibited. This bit also controls the memory protection mechanism. See section 2.4.4.3 for a description of memory protection in user and supervisor modes.

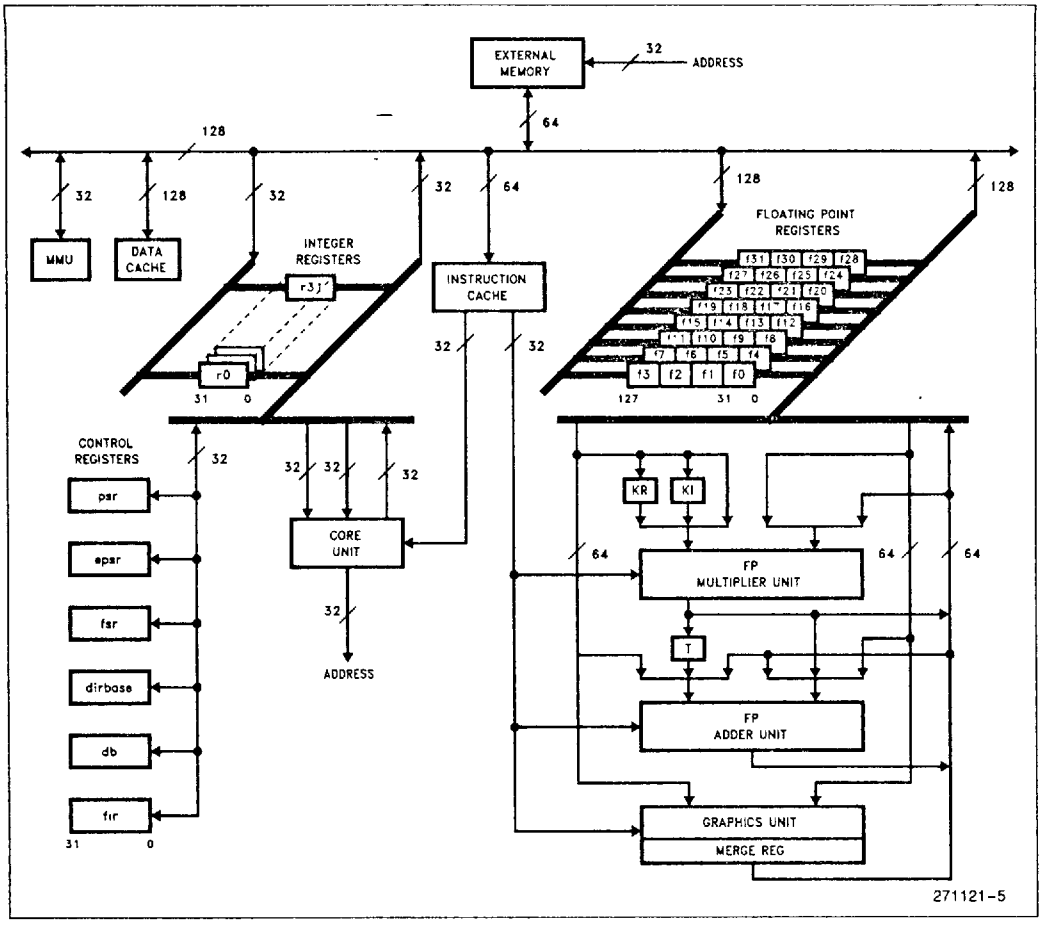


Figure 2.3 Registers and Data Paths

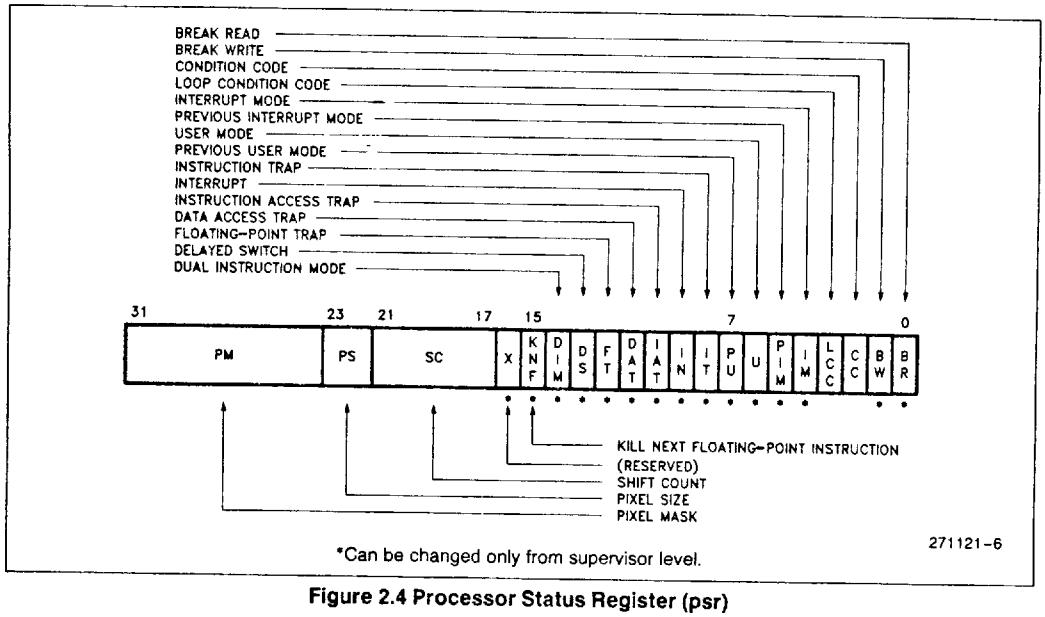


Figure 2.4 Processor Status Register (psr)

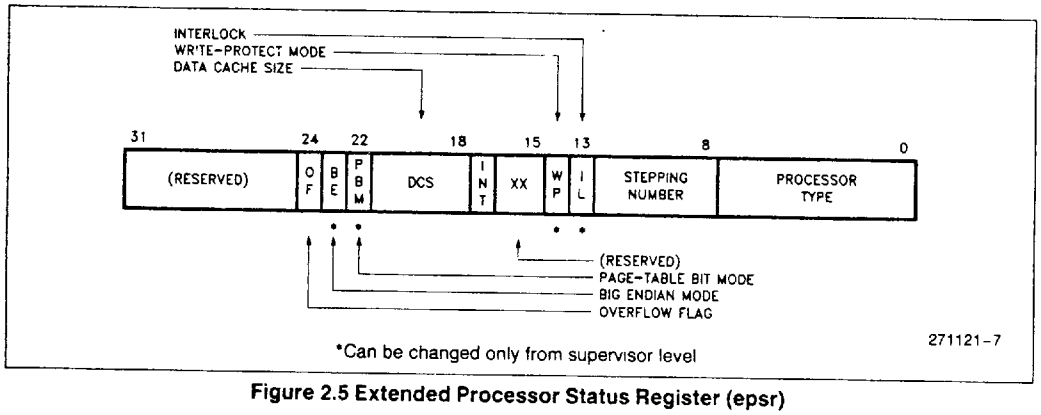


Figure 2.5 Extended Processor Status Register (epsr)

- 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 previous state upon 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 to determine which condition or conditions 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 Military i860 XR microprocessor switches to dual-instruction mode one instruction after returning from the trap handler. If DS and DIM are both set, the Military i860 XR microprocessor switches to single-instruction mode one instruction after returning from the trap handler.
- When a trap occurs, the Military i860 XR 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 Military i860 XR 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 instruction and by the 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 7.

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 **psr**. Figure 2.5 shows the format of **epsr**.

- The processor type is one (1) for the Military i860 XR microprocessor (Read Only field).
- The stepping number has a unique value that distinguishes among different revisions of the processor (Read Only field).
- IL (Interlock) is set if a trap occurs after a **lock** instruction but before 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 progamming error.
- 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. After the value of the WP bit is changed, the TLB must be invalidated by setting the ITI bit of the **dirbase** register, before any stores are performed.
- 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. For the i860 processor, DCS = 1.

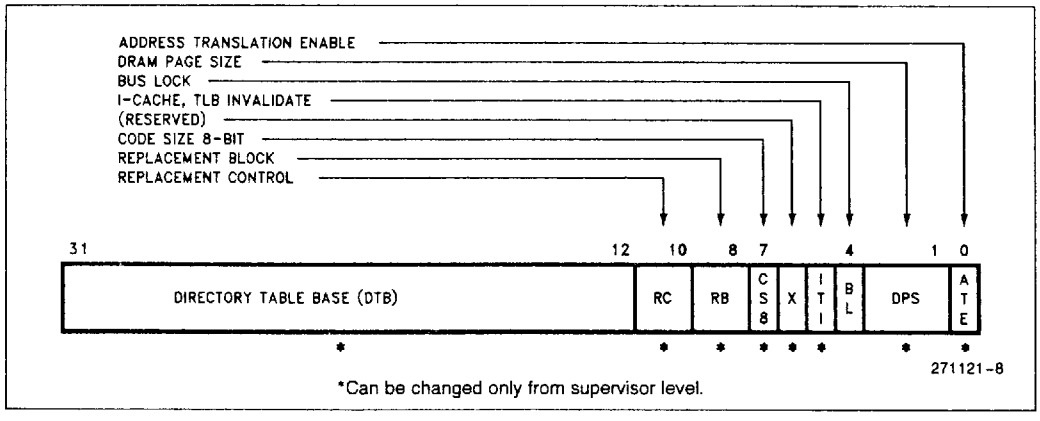


Figure 2.6 Directory Base Register (dirbase)

- PBM (Page-Table Bit Mode) determines which bit of page-table entries is output on the PTB pin. When PBM is clear, the PTB signal reflects bit CD of the page-table entry used for the current cycle. When PBM is set, the PTB signal reflects bit WT of the page-table entry used for the current cycle.
- BE (Big Endian) controls the ordering of bytes within a data item in memory. Normally (i.e. when BE is clear) the Military i860 XR 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 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. Section 2.3 discusses little and big endian addressing.
- 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 controls the function of the **intovr** instruction.

2.2.5 DATA BREAKPOINT REGISTER

The data breakpoint register (**db**) is used to generate a trap when the Military i860 XR microprocessor makes a data-operand access to the address stored in this register. The trap is enabled by BR and BW in **psr**. The **db** register can only be changed from supervisor level. 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 two low-order bits. This ensures that any access that overlaps the address contained in the register will generate a trap. The data access trap occurs before the data is accessed and prevents the load or store from completing.

2.2.6 DIRECTORY BASE REGISTER

The directory base register **dirbase** (shown in Figure 2.6) controls address translation, caching, and bus options. The **dirbase** register can only be changed from the supervisor level. However, the BL bit can be changed from user level with the **lock** and **unlock** instructions.

- ATE (Address Translation Enable), when set, enables the virtual-address translation algorithm. The data cache must be flushed before changing the ATE bit.
- 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 $\overline{\text{NENE}}$ 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, one for 1M × *n* RAMS, etc.
- When BL (Bus Lock) is set, external bus accesses are locked. The $\overline{\text{LOCK}}$ signal is asserted on the next bus cycle whose internal bus request is generated after BL is set. It remains asserted on all subsequent bus cycles as long as BL remains set. The $\overline{\text{LOCK}}$ signal is deasserted on the next bus cycle whose internal bus request is generated after BL is cleared. Traps immediately clear BL. The **lock** and **unlock** instructions control the BL bit.
- ITI (I-Cache, TLB Invalidate), when set in the value being stored into **dirbase**, causes the instruction cache and address-translation cache (TLB) to be flushed. The ITI bit does not remain set in **dirbase**. ITI always appears as zero when reading **dirbase** Section 2.5 discusses flushing the data cache before invalidating the TLB.
- 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 must be aligned on a 64-bit boundary.
- RB (Replacement Block) identifies the cache block to be replaced by cache replacement algorithms. The high-order bit of RB is ignored by the instruction and data caches. RB conditions the cache flush instruction **flush**, which is discussed in Section 7. 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 twelve low-order bits of the address are zeros.

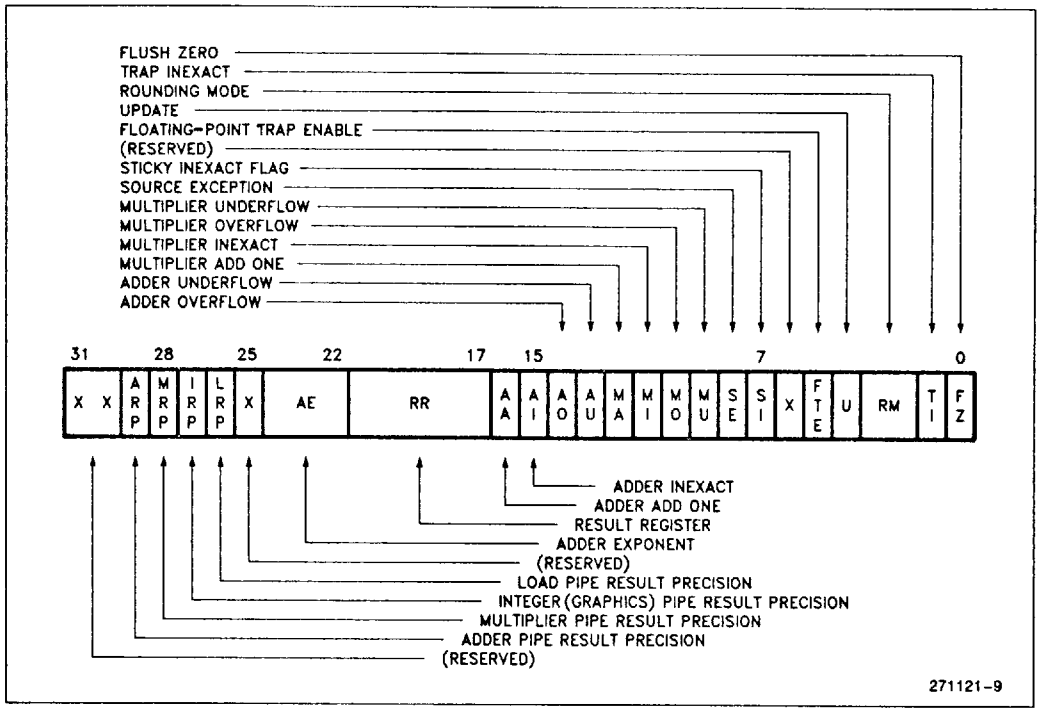


Figure 2.7 Floating-Point Status Register (fsr)

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	0
1 1	3	1

Table 2.4 Values of RC

Value	Meaning
00	Selects the normal 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. The instruction and data caches ignore the high-order bit of RB. This mode is used for instruction cache and TLB testing.
10	Data cache misses replace the block selected by the low-order bit of RB.
11	Disables data cache 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. The address of the **ld.c** instruction used to read the **fir** is returned in *rdest* when reading the **fir** at any time other than the first **ld.c fir** after a trap.

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. The **fsr** is writable in user level.

- 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

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*.

by the target data type, the Military i860 XR microprocessor determines the two representable numbers *a* and *c* that most closely bracket *b* in value ($a < b < c$). The Military i860 XR 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.

- 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. A **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.** 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.
- LRP (Load Pipe Result Precision), 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. These bits cannot be changed by software.

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**, pfmam, pfsm and **pfmsm**, which

initiate both an adder (A-unit) operation and a multiplier (M-unit) 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 2.14.)

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 7 for more information about these registers.

2.3 Addressing

Memory is addressed in byte units within a paged virtual-address space of 2^{32} bytes. Data and instructions can be located anywhere in this address space. Address arithmetic is performed using 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 Military i860 XR microprocessor also offers big endian mode, in which the most significant byte of a data item is at the lowest address. Figure 2.8 shows the difference between the two storage 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 code will appear differently than documented here when accessed as big endian data. Intel recommends that disassemblers running in a big endian system, convert instructions which 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.

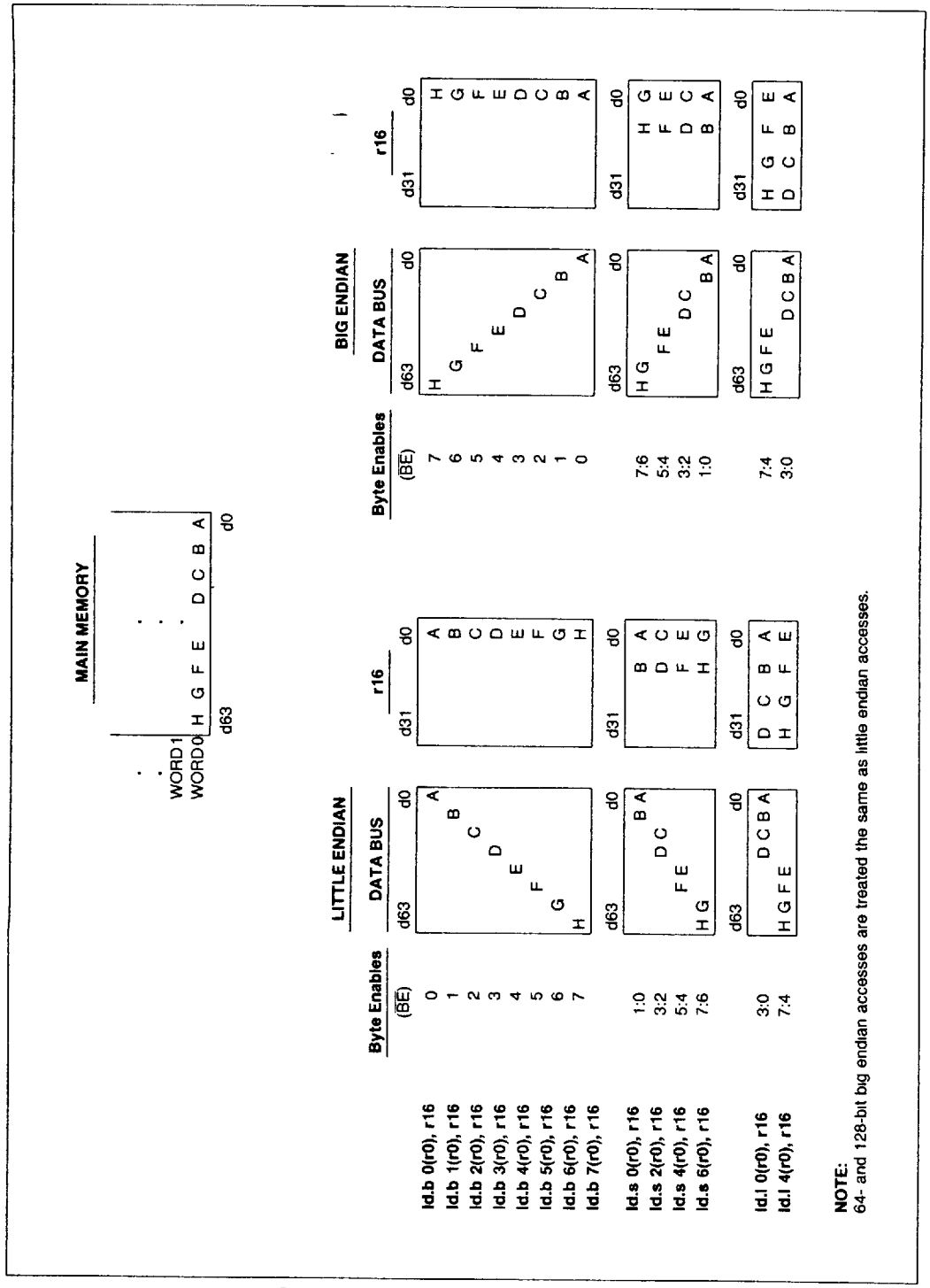
Alignment requirements are as follows (any violation results in a data-access trap):

- 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).
- 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 Military i860 XR microprocessor maps instruction and data virtual addresses into physical addresses before referencing memory. This address transformation is compatible with that of the Military 386 microprocessor and implements the basic features needed for page-oriented virtual-memory systems and page-level protection.

The address translation is optional. Address translation is in effect only when the ATE bit of **dirbase** is set. This bit is typically set by the operating system during software initialization. The ATE bit must be set if the operating system is to implement page-oriented protection or page-oriented virtual memory.



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

Figure 2.8 Little and Big Endian Accesses

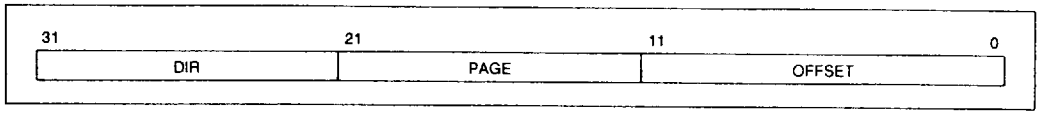


Figure 2.9 Format of a Virtual Address

Address translation is disabled when the processor is reset. It is enabled when a store to **dirbase** sets the ATE bit. It is disabled again when a store clears the ATE bit.

2.4.1 PAGE FRAME

A **page frame** is a 4-Kbyte unit of contiguous addresses of physical main memory. Page frames begin on 4-Kbyte boundaries and are fixed in size. A **page** is the collection of data that occupies a page frame when that data is present in main memory. The data may also occupy some location in secondary storage when there is not sufficient space in main memory.

2.4.2 VIRTUAL ADDRESS

A virtual address refers indirectly to a physical address by specifying a page table, a page within that table, and an offset within that page. Figure 2.9 shows the format of a virtual address.

Figure 2.10 shows how the Military i860 XR microprocessor converts the DIR, PAGE, and OFFSET fields of a virtual address into the physical address by consulting two levels of page tables. The addressing mechanism uses the DIR field as an index into a page directory, 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.

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 therefore contains 4 Kbytes of memory or at most 1K 32-bit entries.

Two levels of tables are used to address a page of memory. At the higher level is a page directory. The page directory addresses up to 1K page tables of the second level. A page table of the second level addresses up to 1K pages. All the tables addressed by one page directory, therefore, can address 1M pages (2^{20}). Because each page contains 4 Kbytes (2^{12} bytes), the tables of one page directory can span the entire physical address space of the Military i860 XR microprocessor ($2^{20} \times 2^{12} = 2^{32}$).

The physical address of the current page directory is stored in 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.

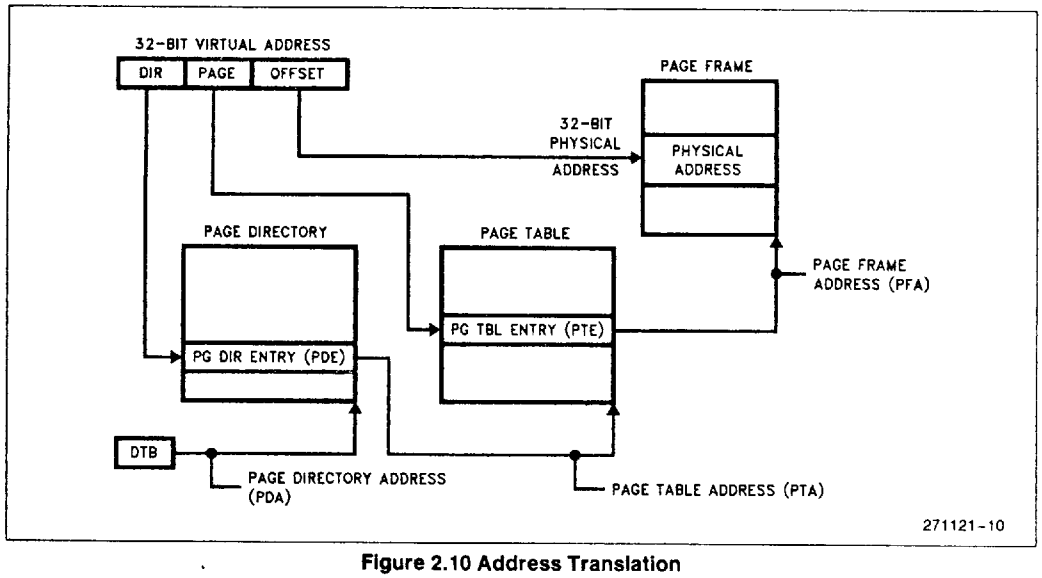


Figure 2.10 Address Translation

2.4.4 TABLE ENTRIES

Table entries in either level of page tables have the same general format. Figure 2.11 illustrates this format for a page-directory entry and a page-table entry.

2.4.4.1 Page Frame Address Generation

The page frame address specifies the physical starting address of a page. Because pages are located on 4K boundaries, the low-order 12 bits are always zero. In a page directory, the page table address is the address of a page table. In a second-level page table, the page frame address is the address of the page frame that contains the desired memory operand.

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 are 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.

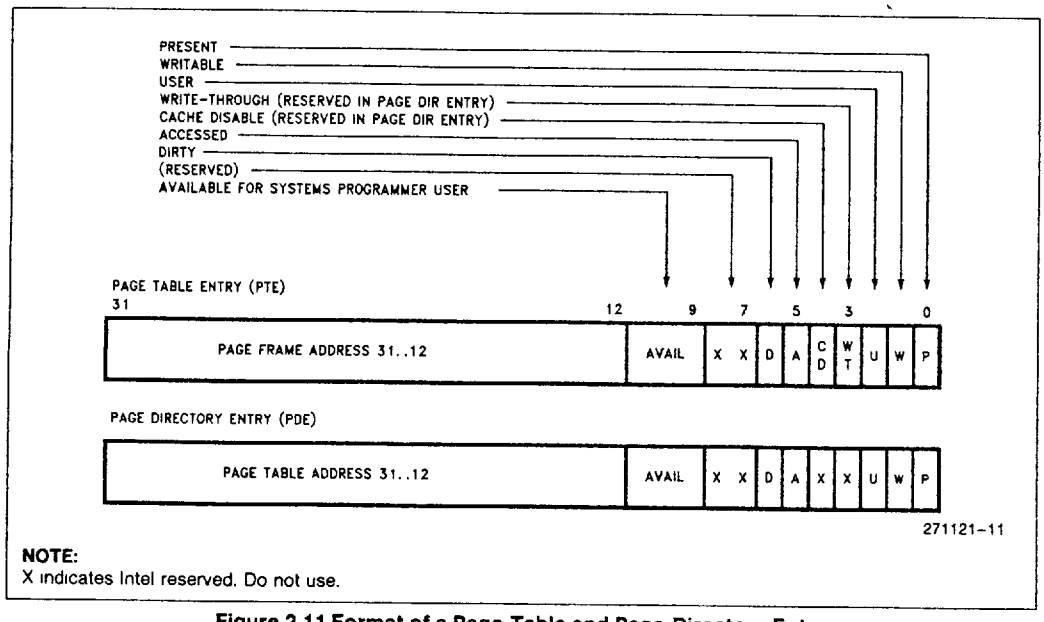
Note that there is no P bit for the page directory itself. The page directory may be not-present while the associated process is suspended, but the operating system must ensure that the page directory indicated by the **dirbase** image associated with the process is present in physical memory before 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 Military i860 XR 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:

1. Supervisor level (U = 0)—for the operating system and other systems software and related data.
2. User level (U = 1)—for applications procedures and data.

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



NOTE:
X indicates Intel reserved. Do not use.

Figure 2.11 Format of a Page-Table and Page-Directory Entry

- The Military i860 XR 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 Military i860 XR 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 Military i860 XR microprocessor enters user level.

With the U bit of **psr** and the W and U bits of the page table entries, the Military i860 XR 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 clear causes a trap.
- When at user level, **st.c** to certain control registers is ignored.

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

WP = 0	All pages are writable.
WP = 1	A write to a page whose W bit is clear causes a trap.

When the Military i860 XR 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 Military i860 XR microprocessor does not implement a write-through caching policy for the on-chip data cache; however, the WT (write-through) bit in the second-level page-table entry does determine internal caching policy. If WT is set in a PTE, on-chip caching of data from the corresponding page is inhibited. The Military i860 CPU may place pages having WT = 1 into the instruction cache Future implementations of the Military i860 architecture may adhere to a write-through data caching policy. Therefore, they may cache pages having the WT bit of the PTE set If WT is clear, the normal write-back policy is applied to data from the page in the on-chip caches. The WT bit of page directory entries is not referenced by the processor, but is **reserved**.

The WT bit is independent of the CD bit; therefore, data may be placed in a second-level coherent cache, but kept out of the on-chip caches.

2.4.4.5 Cache Disable Bit

If the CD (cache disable) bit in the second-level page-table entry is set, data from the associated page is not placed in instruction or data caches. Clearing CD permits the cache hardware to place data from the associated page into caches. The CD bit of page directory entries is not referenced by the processor, but is **reserved.**

To control external caches, the Military i860 XR microprocessor outputs on its PTB pin either the CD or WT bit. The PBM bit of **epsr** determines which bit is output.

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 Military i860 XR microprocessor sets the corresponding accessed bits in both levels of page tables before a read or write operation to a page. The processor tests the dirty bit in the second-level page table before a write to an address covered by that page table entry, and, under certain conditions, causes traps. The trap handler then has the opportunity to maintain appropriate values in the dirty bits. The dirty bit in directory entries is not tested by the Military i860 XR microprocessor. The precise algorithm for using these bits is specified in Section 2.4.5.

An operating system that supports paged virtual memory can use these 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 in the PDE (Page-Directory Entry) or the PTE (Page-Table Entry) 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- or instruction-access fault occurs, the trap handler sets the D bit if an allowable write is being performed, then re-executes 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 Military i860 XR microprocessor automatically asserts the $\overline{\text{LOCK}}$ signal while setting the A bit. If an A-bit of a PTE is found not set during a locked sequence (created by the **lock** instruction), a trap will occur and the processor will not update the A-bit.

2.4.4.7 Combining Protection of Both Levels of Page Tables

For any one page, the protection attributes of its page directory entry may differ from those of its page table entry. The Military i860 XR microprocessor computes the effective protection attributes for a

page by examining the protection attributes in both the directory and the page table. Table 2.6 shows the effective protection provided by the possible combinations of protection attributes.

2.4.5 ADDRESS TRANSLATION ALGORITHM

Referring to Figure 2-10, the algorithm below defines the translation of each virtual address to a physical address. Let DIR, PAGE, and OFFSET be the fields of the virtual address; let PTA be the page table address and PFA be the page frame address 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 PTE is zero, the operation is a write, and either the U-bit of the PSR is set or WP = 1, generate a data or instruction access fault.
4. If the U-bit in the PDE is zero and the U-bit in the **psr** is set, generate a data or instruction access fault.
5. If A in the PDE is zero, and if the TLB miss occurred while the bus was locked, generate a data or instruction access fault. (The trap allows software to set A to one and restart the sequence. This avoids ambiguity in determining what address corresponds to a locked semaphore for external bus hardware use.)
6. If A in the PDE is zero, and if the TLB miss occurred while the bus was not locked, assert $\overline{\text{LOCK}}$. Re-fetch and check the PDE, set A, and store the PDE. Deassert $\overline{\text{LOCK}}$ during the store.
7. Locate the PTE (Page Table Entry) at the physical address formed by PTA:PAGE:00.
8. Perform the P, W, U and A checks as in steps 2 through 6 with the second-level PTE.
9. If D in the PTE is clear and the operation is a write, generate a data or instruction access fault.
10. Form the physical address as PFA:OFFSET.

The Military i860 XR microprocessor looks only in external memory for Page Directories and Page Tables, in the translation process. The data cache is not searched. Therefore, any code which modifies Page Directories or Page Tables must keep them out of the cache. The tables should be kept in non-cacheable memory, or flushed from the cache.

Table 2.6 Combining Directory and Page Protections

Page Directory Entry		Page Table Entry		Combined Protection		
				User Access	Supervisor Access	
U-bit	W-bit	U-bit	W-bit	WP = X	WP = 0	WP = 1
0	0	0	0	N	R/W	R
0	0	0	1	N	R/W	R
0	0	1	0	N	R/W	R
0	0	1	1	N	R/W	R
0	1	0	0	N	R/W	R
0	1	0	1	N	R/W	R/W
0	1	1	0	N	R/W	R
0	1	1	1	N	R/W	R/W
1	0	0	0	N	R/W	R
1	0	0	1	N	R/W	R
1	0	1	0	R	R/W	R
1	0	1	1	R	R/W	R
1	1	0	0	N	R/W	R
1	1	0	1	N	R/W	R/W
1	1	1	0	R	R/W	R
1	1	1	1	R/W	R/W	R/W

NOTES:
N = No access allowed
R = Read access only
R/W = Both reads and writes allowed
X = Don't care

The Military i860 XR microprocessor expects Page Directories and Page Tables to be in little endian format. The operating system must maintain these tables in little endian format by either setting BE = 0 when manipulating the tables or by complementing bit 2 of the address when loading or storing entries.

2.4.6 ADDRESS TRANSLATION FAULTS

The address translation fault is one instance of the data-access fault. The instruction causing the fault can be re-executed upon returning from the trap handler.

2.4.7 PAGE TRANSLATION CACHE

For greatest efficiency in address translation, the Military i860 XR microprocessor stores the most recently used page-table data in an on-chip cache called the TLB (translation lookaside buffer). Only if the necessary paging information is not in the cache must both levels of page tables be referenced.

2.5 Caching and Cache Flushing

The Military i860 XR microprocessor has the ability to cache instruction, data, and address-translation information in on-chip caches. Caching uses virtual-address tags. The effects of mapping two different virtual addresses in the same address space to the same physical address are undefined.

Instruction, data and address-translation caching on the i860 XR microprocessor are not transparent. Because the data cache uses a write-back protocol, writes do not immediately update memory, and writes to memory by other bus devices do not update the cache. Changes to page tables do not automatically update the TLB, and changes to instructions do not automatically update the instruction cache. 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 or to flush the caches. The The i860 XR microprocessor provides the following methods for doing this:

- **Bypassing Instruction and Data Caches.** If deasserted during cache-miss processing, the $\overline{KEN}$ pin disables instruction and data caching of the referenced data. If the CD bit of the associated second-level PTE is set, caching of data and instructions is disabled. The Military i860 CPU may place pages having WT = 1 into the instruction cache. Future implementations of the Military i860 architecture may adhere to a write-through data cache policy. Thus, they may cache pages having the WT bit of the PTE set. The value of the CD bit or the WT bit is output on the PTB pin for use by external caches.
- **Flushing Instruction and Address-Translation Caches.** Storing to the **dirbase** register with the ITI bit set invalidates the contents of the instruction and address-translation caches. This bit should be set when a page table or a page containing code is modified or when changing the DTB field of **dirbase**. Note that in order to make the instruction or address-translation caches consistent with the data cache, the data cache must be flushed *before* invalidating the other caches.

  The mapping of the page containing the currently executing instruction and the next six instructions should not be different in the new page tables when **st.c dirbase** changes DTB or activates ITI. The six instructions following the **st.c** should be **nop**s and should lie in the same page as the **st.c**.
- **Flushing the Data Cache.** The data cache is flushed by a software routine using the **flush** instruction. The data cache must be flushed prior to flushing the instruction or address-translation caches (as controlled by the ITI bit of **dirbase**) or enabling or disabling address translation (via the ATE bit). The data cache does not need flushing if the program is modifying only the P, U, W, A, or D bits of a PDE or PTE (as long as the Page Table or Page Frame Address is not changed and the entry itself was not in the data cache.) The Military i860 CPU does not check these protection bits on cache line writeback. Thus, a trap handler can service a Data Access Fault for D-bit-zero by setting D = 1 and then ITI = 1. In the case of setting the P or A bits active, 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. The Military i860 XR microprocessor searches only external memory for Page Directories and Page Tables in the translation process. The data cache is not searched. Therefore, Page Tables and Directories should be kept in non-cacheable memory, or flushed from the cache by any code which accesses them.

2.6 Instruction Set

Table 2.7 shows the complete set of instructions grouped by function and data type. Refer to Section 7 for an algorithmic definition of each instruction.

The architecture of the Military i860 XR microprocessor uses parallelism to increase the rate at which operations may be introduced into the unit. Parallelism in the Military i860 XR 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 unit 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 2.12 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 columns of the figure represent the three stages of the pipeline. Each stage holds intermediate results and also (when introduced into first stage by software) holds status information pertaining to those results. The figure 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 **I**, **i + 1**, etc. The rows of the figure 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 *src1* and *src2* are transferred to the first stage of the pipeline.

In the Military i860 XR 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. Single precision has three stages and double precision has two 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 Military i860 XR 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 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, for the adder, and the MA, MI, MO and MU bits, for the multiplier. This information arrives at the **fsr** via the pipeline in one of two ways:

Table 2.7 Instruction Set

Core Unit	
Mnemonic	**Description**
Load and Store Instructions	
ld.x	Load integer
st.x	Store integer
fld.y	F-P load
pfld.z	Pipelined F-P load
fst.y	F-P store
pst.d	Pixel store
Register to Register Moves	
ixfr	Transfer integer to F-P register
fxfr	Transfer F-P to integer 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	
trap	Software trap
intovr	Software trap on integer overflow
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
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

Floating-Point Unit	
Mnemonic	**Description**
F-P Multiplier Instruction	
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
frsqr.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 F-P greater-than compare
pfeq.p	Pipelined F-P equal compare
fix.p	F-P to integer conversion
pfix.p	Pipelined F-P to integer conversion
ftrunc.p	F-P to integer truncation
pftrunc.p	Pipelined 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
pfzchks	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

Assembler Pseudo-Operations	
Mnemonic	**Description**
mov	Integer register-register 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

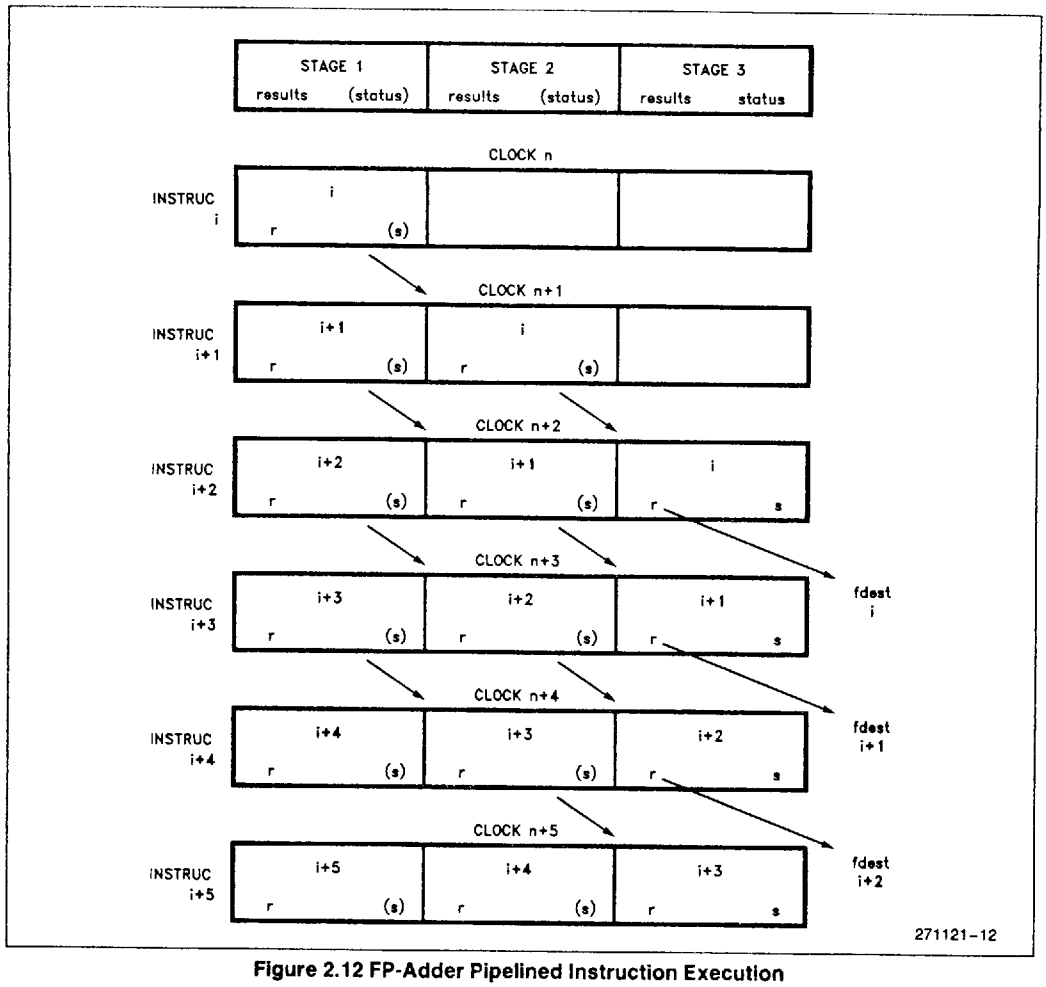


Figure 2.12 FP-Adder Pipelined Instruction Execution

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** with the U bit in the word being written into the **fsr** 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 Military i860 XR microprocessor traps if any of the status bits of the **fsr** indicate exceptions. Note that the instruction that creates the exception 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.

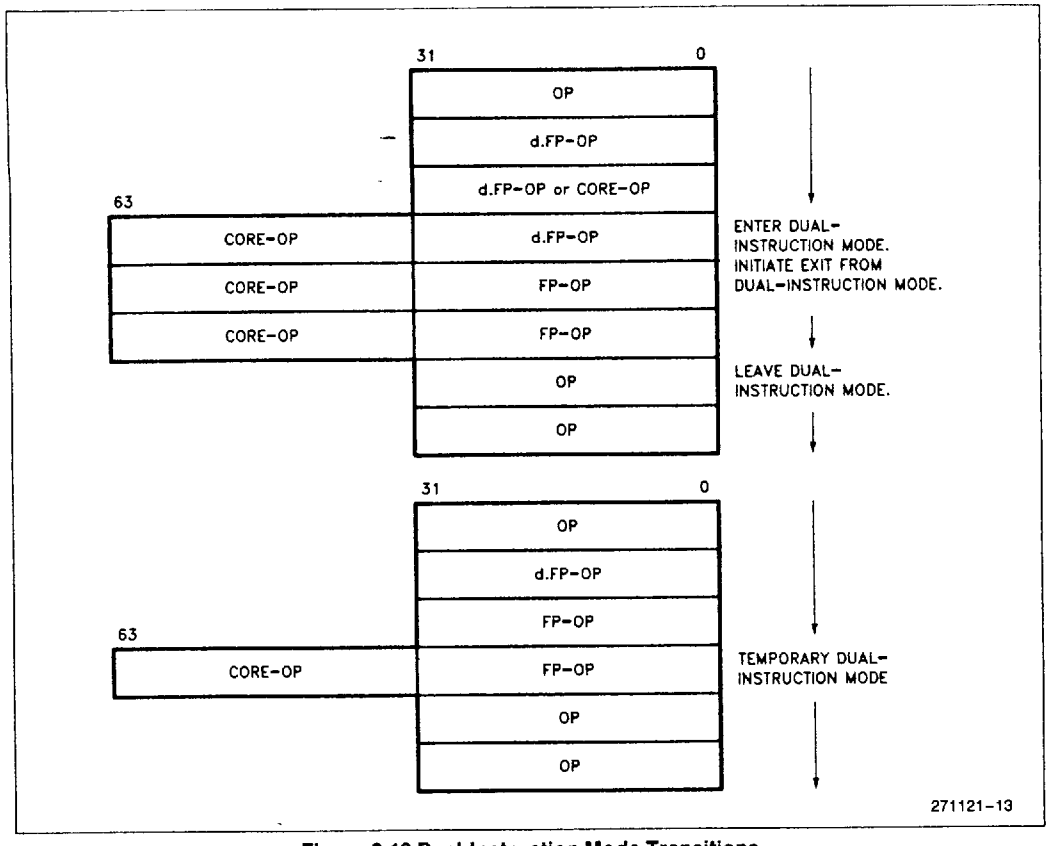


Figure 2.13 Dual-Instruction Mode Transitions

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 operand is double-precision and three stages when the source operand is single-precision. 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.

2.6.2 DUAL-INSTRUCTION MODE

Another form of parallelism results from the fact that the Military i860 XR microprocessor can simultaneously execute both a floating-point and a core instruction. Such parallel execution is called dual-instruction mode. When executing in dual-instruction mode, the instruction sequence consists of 64-bit aligned instructions with a floating-point instruction in the lower 32 bits and a core instruction in the upper 32 bits. Table 2.7 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 Military i860 XR 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 Military i860 XR 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 2.13 illustrates two variations of this sequence of events: one for extended sequences of dual-instructions and one for a single instruction pair.

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 instructions **pfam** *fsrc1*, *fsrc2*, *fdest* (add and multiply), **pfsm** *fsrc1*, *fsrc2*, *fdest* (subtract and multiply), **pfmam** *fscr1*, *fsrc2*, *fdest* (multiply and add) and **pfmsm** *fsrc1*, *fsrc2*, *fdest* (multiply and subtract) 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.
  1. 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*.
  2. 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* or the last stage result of the adder pipeline.
  3. Operand-1 of the adder can be *fsrc1*, the T-register 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 2.14 shows all the possible data paths surrounding the adder and multiplier. A DPC field in these instructions select different data paths. Table 7.8 shows the various encodings of the DPC field.

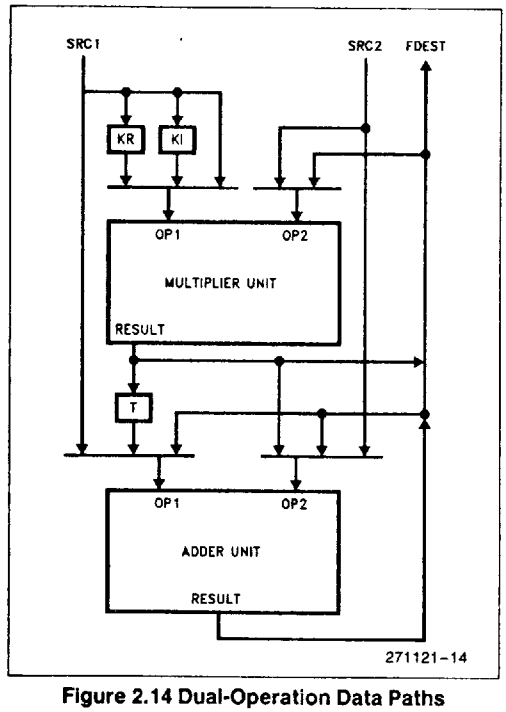


Figure 2.14 Dual-Operation Data Paths

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.

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: *src1* and *src2*.

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.

Table 2.8 Types of Traps

Type	Indication		Caused by	
	PSR,ESPR	FSR	Condition	Instruction
Instruction Fault	IT OF IL		Software traps Missing **unlock**	**trap, intovr** Any
Floating Point Fault	FT	SE AO, MO AU, MU AI, MI	Floating-point source exception Floating-point result exception overflow underflow inexact result	Any M- or A-unit except **fmlow** Any M- or A-unit except **fmlow, pfgt,** and **pfeq**. Reported on any F-P instruction plus **pst, fst** and sometimes **fld, pfld, ixfr**
Instruction Access Fault	IAT		Address translation exception during instruction fetch	Any
Data Access Fault	DAT*		Load/store address translation exception Misaligned operand address Operand address matches **db** register	Any load/store Any load/store Any load/store
Interrupt	IN		External interrupt	
Reset	No trap bits set		Hardware RESET signal	

NOTES: *These cases can be distinguished by examining the operand addresses.
The IL bit of the **epsr** must be checked by the trap handler to tell if the bus is currently in a locked sequence.

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 arbitrary address 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 an 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 2.8. Interrupts and traps start execution in single instruction mode at virtual address 0xFFFFFF00 in supervisor level (U = 0).

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. 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** are set (IT, IN, IAT, DAT, FT). 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.

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) 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 Instruction Trap (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 the Overflow Flag (OF) in **epsr** is set when **intovr** is executed. 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.
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.)

   The lock protocol requires the following sequence of activities:

   1. **lock**
   2. Any load or store instruction that misses the cache.
   3. **unlock**
   4. Any load or store instruction (regardless of whether it misses the cache)

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 the instruction trap occurs. In case of a trap, the Interlock (IL) bit is also set. If the load or store instruction in step 2 hits the cache, the sequence is legal, but the bus is not locked.

2.8.3 FLOATING-POINT FAULT

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

2.8.3.1 Source Exception Faults

When used as inputs to the multiplier or adder, all exceptional operands, including infinities, denormalized numbers and NaNs, cause a floating-point fault and set the Source Exception (SE) bit in the **fsr**. Source exceptions are reported on the instruction that initiates the operation. For pipelined operations, the pipeline is not advanced.

The SE value is undefined for faults on **fld**, **pfld**, **fst**, **pst** and **ixfr** instructions when in single-instruction mode or when in dual-instruction mode and the companion instruction is not a multiplier or adder operation.

2.8.3.2 Result Exception Faults

The class of result exceptions includes any of the following conditions:

- **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, **fst.x** or **pst.x** (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 advanced 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 pipelined floating-point **pfadd** is started, a trap occurs and MO is set.

For scalar operations, the Result Register (RR) field of **fsr** specify the register in which the result was stored. RR is updated when the scalar instruction is initiated. The 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 field is not changed.

For pipelined operations, RR is cleared and 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 then compute the trapping result. In either case, the result has the same fraction as the true result and has an exponent which is the low-order bits of the true result. The trap handler can inspect the result, compute the result appropriate for that instruction (a NaN or an infinity, for example), and store the correct result. The result is either stored in the register specified by RR (if nonzero) or (if RR = 0) the trap handler must reload the pipeline with the saved results and status bits. The trap handler must clear the result status for the last stage and then reexecute the trapping instruction.

Result exceptions may be simultaneously reported for both the adder and multiplier units. In this case, the trap handler must correct the last stage of both pipelines.

2.8.4 INSTRUCTION ACCESS FAULT

The Instruction Access Trap (IAT) 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

The Data Access Trap (DAT) results from an abnormal condition detected during a data operand fetch or store. Such an exception can be caused by only one of the following situations.

- An attempt is being made to write to a page whose (Dirty) D 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 **db** 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 was in a page whose PTE had A = 0, and the access occurred during a locked sequence. (i.e., between **lock** and **unlock**.)
- Write protection (determined by **epsr** bit WP = 1) is violated in supervisor mode.

2.8.6 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 generates an interrupt trap. Vectored interrupts are implemented by interrupt controllers and software.

2.8.7 RESET TRAP

When the Military i860 XR microprocessor is reset, execution begins in single-instruction mode at physical address 0xFFFFFF00. This is the same address as 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. The Code-Size8 (CS8) mode is initialized if the INT/CS8 pin is asserted at the end of reset. The read-only fields of the **espr** 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 2.9 for a summary of these initial settings.

Table 2.9 Register and Cache Values after Reset

Registers	Initial Value
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 = 0; Processor Type, Stepping Number, DCS are read only; others are *undefined*
db	*Undefined*
dirbase	DPS, BL, ATE = 0; others are *undefined*
fir	*Undefined*
fsr	*Undefined*
KR, KI, T, MERGE	*Undefined*
Caches	**Initial Value**
Instruction Cache	Flushed
Data Cache	*Undefined*
TLB	Flushed

The software must ensure that the data cache is flushed and control registers are properly initialized before performing operations that depend on the values of the cache or registers. The data cache has no "validity" bits, so memory accesses before the flush may result in false data cache hits.

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

- **db** (data breakpoint register) which permits specification of a data addresses that the Military i860 XR 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 *src1* and *src2* 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.

3.0 HARDWARE INTERFACE

In the following description of the military i860's hardware interface, the $\overline{\ }$ symbol (overbar) above a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no $\overline{\ }$ is present above the signal name, the signal is asserted when at the high voltage level.

3.1 Signal Description

Table 3.1 identifies functional groupings of the pins, lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output pins are 3-state, except HLDA and

BREQ. All inputs are synchronous with the Clock (CLK), except HOLD and INT.

3.1.1 CLOCK (CLK)

The CLK input determines execution rate and timing of the Military i860 XR microprocessor. Timing of other signals is specified relative to the rising edge of this signal. The internal operating frequency is the same as the external clock. This signal is TTL compatible.

3.1.2 SYSTEM RESET (RESET)

Asserting RESET for at least 16 CLK periods causes initialization of the Military i860 XR microprocessor. Refer to section 3.2 "Initialization" for more details related to RESET.

3.1.3 BUS HOLD (HOLD) AND BUS HOLD ACKNOWLEDGE (HLDA)

These pins are used for Military i860 XR microprocessor bus arbitration. At some clock after the HOLD signal is asserted, the Military i860 XR microprocessor releases control of the local bus and puts all bus interface outputs (except BREQ and HLDA) into a 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 time required to acknowledge a hold request is one clock plus the number of clocks needed to finish any outstanding bus cycles. HOLD is recognized even while RESET or $\overline{\text{LOCK}}$ are asserted.

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

Hold is an asynchronous input.

3.1.4 BUS REQUEST (BREQ)

This signal is asserted when the i860 XR microprocessor has a pending memory request, even when HLDA is asserted. This allows an external bus arbiter to implement an "on demand only" policy for granting the bus to the i860 XR microprocessor. BREQ is asserted the clock after the i860 XR microprocessor realizes an internal request for the bus. In normal operation, BREQ goes low the clock after $\overline{\text{ADS}}$ goes low for the final pending bus cycle. (Refer to Figure 4.10 for timing information.) During data or instruction cache fills, however, BREQ may be deasserted for one or more clocks, due to cache and TLB logic.

3.1.5 INTERRUPT/CODE-SIZE (INT/CS8)

This dual purpose pin signifies either an external interrupt or Code-Size8 when asserted. If INT/CS8 is asserted during the clock before the falling edge of RESET, the eight-bit code-size mode is selected. For more about this mode, refer to Section 3.2 "Initialization".

At any other time it signifies an interruption of the current instruction stream. If interrupts are enabled (IM set in **psr**) when INT/CS8 is asserted, the Military i860 XR microprocessor fetches the next instruction from virtual address 0xFFFFFF00. To assure that an interrupt is recognized, the INT/CS8 pin should remain asserted until the software acknowledges the interrupt (by writing, for example, to a memory-mapped port of an interrupt controller).

When the bus is not locked, the maximum time between the assertion of the INT/CS8 pin and the execution of the first instruction of the trap handler is ten clocks, plus the time for four sets of four pipelined read cycles and two sets of four pipelined writes (instruction- and data-cache misses and write-back cycles to update memory), plus the time for twenty nonpipelined read cycles (six TLB misses, with eight refetches when the A-bit is zero), plus the time for eight non-pipelined writes (updates to the A-bit).

If the bus is locked from a **lock** instruction, the pin is ignored and the INT bit of **epsr** is always zero. The **lock** instruction can only assert $\overline{\text{LOCK}}$ for 30–33 instructions before trapping.

INT/CS8 is an asynchronous input.

Table 3.1 Pin Summary

Pin Name	Function	Active State	Input/ Output
Execution Control Pins			
CLK	Clock		I
RESET	System reset	High	I
HOLD	Bus hold	High	I
HLDA	Bus hold acknowledge	High	O
BREQ	Bus request	High	O
INT/CS8	Interrupt, code-size	High	I
Bus Interface Pins			
A31–A3	Address bus	High	O
$\overline{BE7}$–$\overline{BE0}$	Byte Enables	Low	O
D63–D0	Data bus	High	I/O
$\overline{LOCK}$	Bus lock	Low	O
W/$\overline{R}$	Write/Read bus cycle	Hi/Low	O
$\overline{NENE}$	Next Near	Low	O
$\overline{NA}$	Next Address request	Low	I
$\overline{READY}$	Transfer Acknowledge	Low	I
$\overline{ADS}$	ADdress Status	Low	O
Cache Interface Pins			
$\overline{KEN}$	Cache Enable	Low	I
PTB	Page Table Bit	High	O
Testability Pins			
SHI	Boundary Scan Shift Input	High	I
BSCN	Boundary Scan Enable	High	I
SCAN	Shift Scan Path	High	I
Intel-Reserved Configuration Pins			
CC1–CC0	Configuration	High	I
Power and Ground Pins			
V_{CC}	System power		
V_{SS}	System ground		

A $\overline{\ }$ above a pin name indicates that the signal is active when at the low voltage level.

3.1.6 ADDRESS PINS (A31–A3) AND BYTE ENABLES ($\overline{BE7}$–$\overline{BE0}$)

The 29-bit address bus (A31–A3) identifies addresses to a 64-bit location. Separate byte-enable signals ($\overline{BE7}$–$\overline{BE0}$) identify which bytes are accessed within the 64-bit location. In all noncacheable read cycles ($\overline{KEN}$ deasserted), the byte enables match the length and address of the requested data. Cacheable read cycles ($\overline{KEN}$ asserted), however, result in four 64-bit memory cycles to fill an entire 32-byte cache line. The $\overline{BEn}$ pins activated are those that represent the operand of the load instruction that caused the line fill, and these same $\overline{BEn}$ pins remain activated for all four cycles of the line fill. All 64 bits must be returned for each cycle without regard for the $\overline{BEn}$ signals. In all write cycles (noncacheable writes as well as cache line write-backs) the $\overline{BEn}$ signals indicate the bytes that must be written.

Instruction fetches (W/$\overline{R}$ is low) are distinguished from data accesses by the unique combinations of $\overline{BE7}$–$\overline{BE0}$ defined in Table 3 2. For an eight-bit code fetch in code-size8 (CS8) mode, $\overline{BE2}$–$\overline{BE0}$ are redefined to be A2–A0 of the address. In this case $\overline{BE7}$–$\overline{BE3}$ form the code shown in Table 3.2 that identifies an instruction fetch. The A2 in the table does not represent a physical pin, just a conceptual internal

address line value. The "x" under A2 for CS8 mode means "not applicable", or "don't care". All other combinations of byte enables indicate data accesses.

The address and byte-enable pins are driven until either $\overline{NA}$ or $\overline{READY}$ is asserted.

3.1.7 DATA PINS (D63–D0)

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 write bus cycles, the point at which data is driven onto the bus depends on the type of the preceding cycle. If there was no preceding cycle (i.e. the bus was idle), data is driven with the address. If the preceding cycle was a write, data is driven as soon as $\overline{READY}$ is returned from the previous cycle. If the preceding cycle was a read, data is driven one clock after $\overline{READY}$ is returned from the previous cycle, thereby allowing time for the bus to be turned around.

3.1.8 BUS LOCK ($\overline{LOCK}$)

This signal is used to provide atomic (indivisible) read-modify-write sequences in multiprocessor systems. A multiprocessor bus arbiter must permit only one processor a locked access to the address which is on the bus when $\overline{LOCK}$ first activates. The system must maintain the lock of that location until $\overline{LOCK}$ deactivates.

The Military i860 XR microprocessor coordinates the external $\overline{LOCK}$ signal with the software-controlled Bus Lock (BL) bit of the **dirbase** register. Programmers do not have to be concerned about the fact that bus activity is not always synchronous with instruction execution. $\overline{LOCK}$ is asserted with $\overline{ADS}$ for the address operand of the first load or store instruction executed after the BL bit is set by the **lock** instruction. Pending bus cycles are locked according to the value of the BL bit when the instruction was executed. Even if the BL bit is changed between the time that an instruction generates an internal bus request and the time that the cycle appears on the bus, the i860 XR microprocessor still asserts $\overline{LOCK}$ for that bus cycle.

If $\overline{ADS}$ is active when $\overline{LOCK}$ deactivates, then that request should complete before the hardware relinquishes the lock. If $\overline{ADS}$ is not active, the locking of the location can immediately end when $\overline{LOCK}$ deactivates. Of course the simplest arbitration hardware can just lock the entire bus against all other accesses during $\overline{LOCK}$ assertion.

When the BL bit is deasserted with the **unlock** instruction, $\overline{LOCK}$ is deasserted with the next load or store but after any pending bus cycles. Between locked sequences, at least one cycle of no $\overline{LOCK}$ is guaranteed by the behavior of the **unlock** instruction. $\overline{LOCK}$ deassertion may occur independently of $\overline{ADS}$ for the case of a trap or a cache hit after **unlock**.

The Military i860 XR microprocessor also asserts $\overline{LOCK}$ during TLB miss processing for updates of the accessed bit in page-table entries. The maximum time that $\overline{LOCK}$ can be asserted in this case is five clocks plus the time required to perform a read-modify-write sequence. Instruction fetches do not alter the $\overline{LOCK}$ pin.

Between **lock** and **unlock** instructions, the INT pin is ignored and the INT bit of **epsr** is zero when read by **ld.c epsr**. The time that interrupts are disabled is limited by the lock protocol outlined in Section 2.8.2.

3.1.9 WRITE/READ BUS CYCLE (W/$\overline{R}$)

This pin specifies whether a bus cycle is a write (HIGH) or read (LOW) cycle.

3.1.10 NEXT NEAR ($\overline{NENE}$)

The $\overline{NENE}$ signal allows higher-speed reads and writes in the case of consecutive reads and writes that access static column or page-mode DRAMs. The Military i860 XR microprocessor asserts $\overline{NENE}$ when the current address is in the same DRAM page as the previous bus cycle. The Military i860 XR microprocessor determines the DRAM page size by inspecting the 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 × 1, 256K × 4, and up. $\overline{NENE}$ is never asserted on the next bus cycle after HLDA is deasserted.

3.1.11 NEXT ADDRESS REQUEST ($\overline{NA}$)

The $\overline{NA}$ signal makes address pipelining possible. The system asserts $\overline{NA}$ to indicate that it is ready to accept the next address from the Military i860 XR microprocessor. $\overline{NA}$ may be asserted before the current cycle ends. (If the system does not implement pipelining, $\overline{NA}$ does not have to be activated.) The Military i860 XR microprocessor samples $\overline{NA}$ every clock, starting one clock after the prior activation of $\overline{ADS}$. When $\overline{NA}$ is active, the Military i860 XR microprocessor is free to drive address and bus-cycle definition for the next pending bus cycle. The Military i860 XR microprocessor remembers that $\overline{NA}$ was asserted when no internal request is pending; therefore, $\overline{NA}$ can be deactivated after the next rising edge of the CLK signal. Up to three bus cycles can be outstanding simultaneously.

Table 3.2 Identifying Instruction Fetches

Code Fetch	A2	$\overline{BE7}$	$\overline{BE6}$	$\overline{BE5}$	$\overline{BE4}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
Normal (Non-CS8)	0	1	1	1	1	1	0	1	0
Normal (Non-CS8)	1	1	0	1	0	1	1	1	1
CS8 Mode	x	1	0	1	0	1	Low-order address bits		

3.1.12 TRANSFER ACKNOWLEDGE ($\overline{READY}$)

The system asserts the $\overline{READY}$ signal during read cycles when valid data is on the data pins and during write cycles when the system has accepted data from the data pins. $\overline{READY}$ must be asserted for at least one clock. Sampling of $\overline{READY}$ begins in the clock after an $\overline{ADS}$ or in the second clock after a prior $\overline{READY}$.

3.1.13 ADDRESS STATUS ($\overline{ADS}$)

The Military i860 XR microprocessor asserts $\overline{ADS}$ during the first clock of each bus cycle to identify the clock period during which it begins to assert outputs on the address bus. This signal is held active for one clock.

3.1.14 CACHE ENABLE ($\overline{KEN}$)

The Military i860 XR microprocessor samples $\overline{KEN}$ to determine whether the data being read for the current cache-miss cycle is to be cached. This pin is internally NORed with the CD and WT bits to control cacheability on a page by page basis (refer to Table 3.3).

If the address is one that is permitted to be in the cache, $\overline{KEN}$ must be continuously asserted during the sampling period starting from the second rising clock edge after $\overline{ADS}$ is asserted, through the clock $\overline{NA}$ or $\overline{READY}$ is asserted. The entire 64 bits of the data bus will be used for the read, regardless of the state of the byte-enable pins. Three additional 64-bit bus cycles will be generated to fill the rest of the 32-byte cache block.

If $\overline{KEN}$ is found deasserted at any clock from the clock after $\overline{ADS}$ through the clock of the **first** $\overline{NA}$ or $\overline{READY}$, the data being read will not be cached and two scenarios can occur: 1) if the cycle is due to data-cache miss, no subsequent cache-fill cycles will be generated; 2) if the cycle is due to an instruction-cache miss, additional cycle(s) will be generated until the address reaches a 32-byte boundary. To avoid caching a line, external hardware must deassert $\overline{KEN}$ during or before the first $\overline{NA}$ or $\overline{READY}$.

Table 3.3 Cacheability Based on $\overline{KEN}$ and CD or WT

CD or WT	$\overline{KEN}$	Meaning
0	0	Cacheable access
0	1	Noncacheable access
1	0	Noncacheable page
1	1	Noncacheable page

3.1.15 PAGE TABLE BIT (PTB)

Depending on the setting of the PBM (page-table bit mode) bit of the **epsr**, the PTB reflects the value of either the CD (cache disable) bit or the WT (write through) bit of the page-table entry used for the current cycle. When paging is disabled, PTB remains inactive.

3.1.16 BOUNDARY SCAN SHIFT INPUT (SHI)

This pin is used with the testability features. Refer to section 3.3.

3.1.17 BOUNDARY SCAN ENABLE (BSCN)

This pin is used with the testability features. Refer to section 3.3.

3.1.18 SHIFT SCAN PATH (SCAN)

This pin is used with the testability features Refer to section 3.3.

3.1.19 CONFIGURATION (CC1-CC0)

These two pins are reserved by Intel. Strap both pins LOW.

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

The Military i860 XR microprocessor has 48 pins for power and ground. All pins must be connected to the appropriate low-inductance power and ground signals in the system.

3.2 Initialization

Initialization of the Military i860 XR microprocessor is caused by assertion of the RESET signal for at least 16 clocks. Table 3.4 shows the status of output pins during the time that RESET is asserted. Note that HOLD requests are honored during RESET and that the status of output pins depends on whether a HOLD request is being acknowledged.

Table 3.4 Output Pin Status During Reset

Pin Name	Pin Value	
	HOLD Not Acknowledged	HOLD Acknowledged
$\overline{ADS}$, $\overline{LOCK}$	HIGH	HI-Z
W/$\overline{R}$, PTB	LOW	HI-Z
BREQ	LOW	LOW
HLDA	LOW	HIGH
D63–D0	HI-Z	HI-Z
A31–A3, $\overline{BE7}$–$\overline{BE0}$, $\overline{NENE}$	Undefined	HI-Z

After a reset, the Military i860 XR microprocessor begins executing at physical address 0xFFFFFF00. The program-visible state of the Military i860 XR microprocessor after reset is detailed in section 2.8.7.

Eight-bit code-size mode is selected when INT/CS8 is asserted during the clock before the falling edge of RESET. While in eight-bit code-size mode, instruction cache misses are byte reads (transferred on D7-D0 of the data bus) instead of eight-byte reads. This allows the Military i860 XR microprocessor to be bootstrapped from an eight-bit EPROM. For these code reads, byte enables $\overline{BE2}$–$\overline{BE0}$ are redefined to be the low order three bits of the address, so that a complete byte address is available. These reads update the instruction cache if $\overline{KEN}$ is asserted (refer to section 3.1.14) and are not pipelined even if $\overline{NA}$ is asserted. 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 Military i860 XR microprocessor.

3.3 Testability

The Military i860 XR microprocessor has a *boundary scan mode* that may be used in component- or board-level testing to test the signal traces leading to and from the Military i860 XR microprocessor. Boundary scan mode provides a simple serial interface that makes it possible to test all signal traces with only a few probes. Probes need be connected only to CLK, BSCN, SCAN, SHI, BREQ, RESET, and HOLD.

The pins BSCN and SCAN control the boundary scan mode (refer to Table 3.5). When BSCN is asserted, the Military i860 XR microprocessor enters boundary scan mode on the next rising clock edge. Boundary scan mode can be activated even while RESET is active. When BSCN is deasserted while in boundary scan mode, the Military i860 XR microprocessor leaves boundary scan mode on the next rising clock edge. After leaving boundary scan mode, the internal state is undefined; therefore, RESET should be asserted.

Table 3.5 Test Mode Selection

BSCN	SCAN	Testability Mode
LO	LO	No testability mode selected
LO	HI	(Reserved for Intel)
HI	LO	Boundary scan mode, normal
HI	HI	Boundary scan mode, shift SHI as input; BREQ as output

For testing purposes, each signal pin has associated with it an internal latch. Table 3.6 indentifies these latches by name and classifies them as input, output, or control The input and output latches carry the name of the corresponding pins.

Table 3.6 Test Mode Latches

Input Latch	Output Latch	Associated Control Latch
SHI		—
BSCN		-
SCAN		
RESET		
D0-D63	D0-D63	DATAt
CC1-CC0		
	A31-A3	ADDRt
	$\overline{NENE}$	NENEt
	$\overline{PTB}$	PTBt
	W/$\overline{R}$	W/Rt
	$\overline{ADS}$	ADSt
	HLDA	
	$\overline{LOCK}$	LOCKt
$\overline{READY}$		
$\overline{KEN}$		
$\overline{NA}$		
INT/CS8		
HOLD		
	$\overline{BE7}$-$\overline{BE0}$	BEt
	BREQ	

Within boundary scan mode the Military i860 XR microprocessor operates in one of two submodes: normal mode or shift mode, depending on the value of the SCAN input. A typical test sequence is . . .

1. Enter shift mode to assign values to the latches that correspond with the pins.
2. Enter normal mode. In normal mode the Military i860 XR microprocessor transfers the latched values to the output pins and latches the values that are being driven onto the input pins.
3. Reenter shift mode to read the new values of the input pins.

3.3.1 NORMAL MODE

When SCAN is deasserted, with BSCN asserted the normal mode is selected. For each input pin (RESET, HOLD, INT/CS8, $\overline{NA}$, $\overline{READY}$, $\overline{KEN}$, SHI, BSCN, SCAN, CC1 and CC0), the corresponding latch is loaded with the value that is being driven onto the pin.

The three-state output pins (A31-A3, $\overline{BE7}$-$\overline{BE0}$, W/$\overline{R}$, $\overline{NENE}$, $\overline{ADS}$, $\overline{LOCK}$ and PTB) are enabled by the control latches ADDRt (for A31-A3), BEt, W/Rt, NENEt, ADSt, LOCKt and PTBt. If a control latch is set, the corresponding output latches drive their output pins; otherwise the pins are not driven.

The I/O pins (D63-D0) are enabled by the control latch DATAt, which is similar to the other control latches. In addition, when DATAt is not set, the data pins are treated as input pins and their values are latched.

3.3.2 SHIFT MODE

When SCAN is asserted, with BSCN asserted the shift mode is selected. In shift mode, the pins are organized into a *boundary scan chain.* The scan chain is configured as a shift register that is shifted on the rising edge of CLK. The SHI pin is connected to the input of one end of the boundary scan chain. The value of the most significant bit of the scan chain is output on the BREQ pin. To avoid glitches while the values are being shifted along the chain, the tester should assert both the RESET and HOLD pins. Then all three-state outputs are disabled (placed in a HI-Z state). The order of the pins within the chain is shown in Figure 3.1.

A tester causes entry into this mode for one of two purposes:

1. To assign values to output latches to be driven onto output pins upon subsequent entry into normal mode.
2. To read the values of input pins previously latched in normal mode.

4.0 BUS OPERATION

A bus cycle begins when $\overline{ADS}$ is asserted and ends when $\overline{READY}$ is sampled active. $\overline{READY}$ is sampled one clock after assertion of $\overline{ADS}$ and thereafter until it becomes active. New cycles can start as often as every other clock until three cycles are outstanding. A bus cycle is considered outstanding as long as $\overline{READY}$ has not been asserted to terminate that cycle. After $\overline{READY}$ becomes active, it is not sampled again for the following (outstanding) cycle until the second clock after the one during which it became active. $\overline{READY}$ is assumed to be inactive when it is not sampled.

With regard to how a bus cycle is generated by the Military i860 XR microprocessor, there are two types of cycles: pipelined and nonpipelined. Both types of cycles can be either read or write cycles. A pipelined cycle is one that starts while one or two other bus cycles are outstanding. A nonpipelined cycle is one that starts when no other bus cycles are outstanding.

4.1 Pipelining

A **m-n** read or write cycle is a cycle with a total cycle time of **m** clocks and a cycle-to-cycle time of **n** clocks (**m** ≥ **n**). Total cycle time extends from the clock in which $\overline{\text{ADS}}$ is activated to the clock in which $\overline{\text{READY}}$ becomes active,. whereas, cycle-to-cycle time extends from the time that $\overline{\text{READY}}$ is sampled active for the previous cycle to the time that it is sampled active again for the current cycle. When **m** = **n**, a nonpipelined cycle is implied; **m** > **n** implies a pipelined cycle.

Pipelining may occur for the next bus cycle any time the current bus cycle requires more than two clock periods to finish (**m** > **2**). The next cycle can be initiated when $\overline{\text{NA}}$ (next address) is sampled active, even if the current cycle has not terminated. In this case, pipelining occurs. $\overline{\text{NA}}$ is not recognized until after $\overline{\text{ADS}}$ has become inactive.

To allow high transfer rates in large memory systems, two-level pipelining is supported (i.e., there may be up to three cycles in progress at one time). Pipelining enables a new word of data to be transferred every two clocks, even though the total cycle time may be up to six clocks.

4.2 Bus State Machine

The operation of the bus is described in terms of a bus state machine using a state transition diagram. Figure 4.1 illustrates the Military i860 XR microprocessor bus state machine. A bus cycle is composed of two or more states. Each bus state lasts for one CLK period.

The Military i860 XR microprocessor supports up to two levels of address pipelining. Once it has started the first bus cycle, it can generate up to two more cycles as long as $\overline{\text{READY}}$ remains inactive. To start a new bus cycle while other cycles are still outstanding, $\overline{\text{NA}}$ must be active for at least one clock cycle starting with the clock after the previous $\overline{\text{ADS}}$. $\overline{\text{NA}}$ is latched internally.

States T_j and T_{jk}, for **j** = {1,2,3} and **k** = {1,2}, are used to describe the state of the Military i860 XR microprocessor Bus State Machine. Index **j** indicates the number of outstanding bus cycles while index **k** distinguishes the intermediate states for the **j**-th outstanding cycle.

Therefore there can be up to three outstanding cycles, and there are two possible intermediate states for each level of pipelining. T_{j1} is the next state after T_j, as long as **j** cycles are outstanding. T_{j2} is entered when $\overline{\text{NA}}$ is active but the Military i860 XR microprocessor is not ready to start a new cycle.

Five conditions have to be met to start a new cycle while one or more cycles are already pending:

1. $\overline{\text{READY}}$ inactive
2. $\overline{\text{NA}}$ having been active
3. An internal request pending (BREQ active)
4. HOLD not active
5. Fewer than three cycles outstanding

Note that BREQ is asserted on the clock after the Military i860 XR microprocessor realizes an internal request for the bus.

Upon hardware RESET, the bus control logic enters the idle state T_I and awaits an internal request for a bus cycle. If a bus cycle is requested while there is no hold request from the system, a bus cycle begins, advancing to state T_1. On the next cycle, the state machine automatically advances to state T_{11}. If $\overline{\text{READY}}$ is active in state T_{11}, the bus control logic returns either to T_I, if no new cycle is started, or to T_1, if a new cycle request is pending internally. If an internal bus request is pending each time $\overline{\text{READY}}$ is active, the state machine continues to cycle between T_{11} and T_1.

However, if $\overline{\text{READY}}$ is not active but the next address request is pending (as indicated by an active $\overline{\text{NA}}$), the state machine advances either to state T_2 (if an internal bus request is pending, signifying that two bus cycles are now outstanding), or to state T_{12}

→ 1 SHI	→ 2 BSCN	→ 3 SCAN	→ 4 RESET	→ 5 DATAt	→ 6 D0	→		→ 69 D63	→
70 CC1 →	71 CC0 →	72 A31 →		→ 100 A3 →	101 ADDRt →	102 NENEt →	103 $\overline{\text{NENE}}$ →	104 PTBt →	
105 $\overline{\text{PTB}}$ →	106 W/Rt →	107 W/$\overline{\text{R}}$ →	108 ADSt →	109 $\overline{\text{ADS}}$ →	110 HLDA →	111 LOCKt →	112 $\overline{\text{LOCK}}$ →	113 $\overline{\text{READY}}$ →	
114 $\overline{\text{KEN}}$ →	115 $\overline{\text{NA}}$ →	116 INT/CS8 →	117 HOLD →	118 BEt →	119 $\overline{\text{BE7}}$ →	. →	126 $\overline{\text{BE0}}$ →	127 BREQ →	

Figure 3.1 Order of Boundary Scan Chain

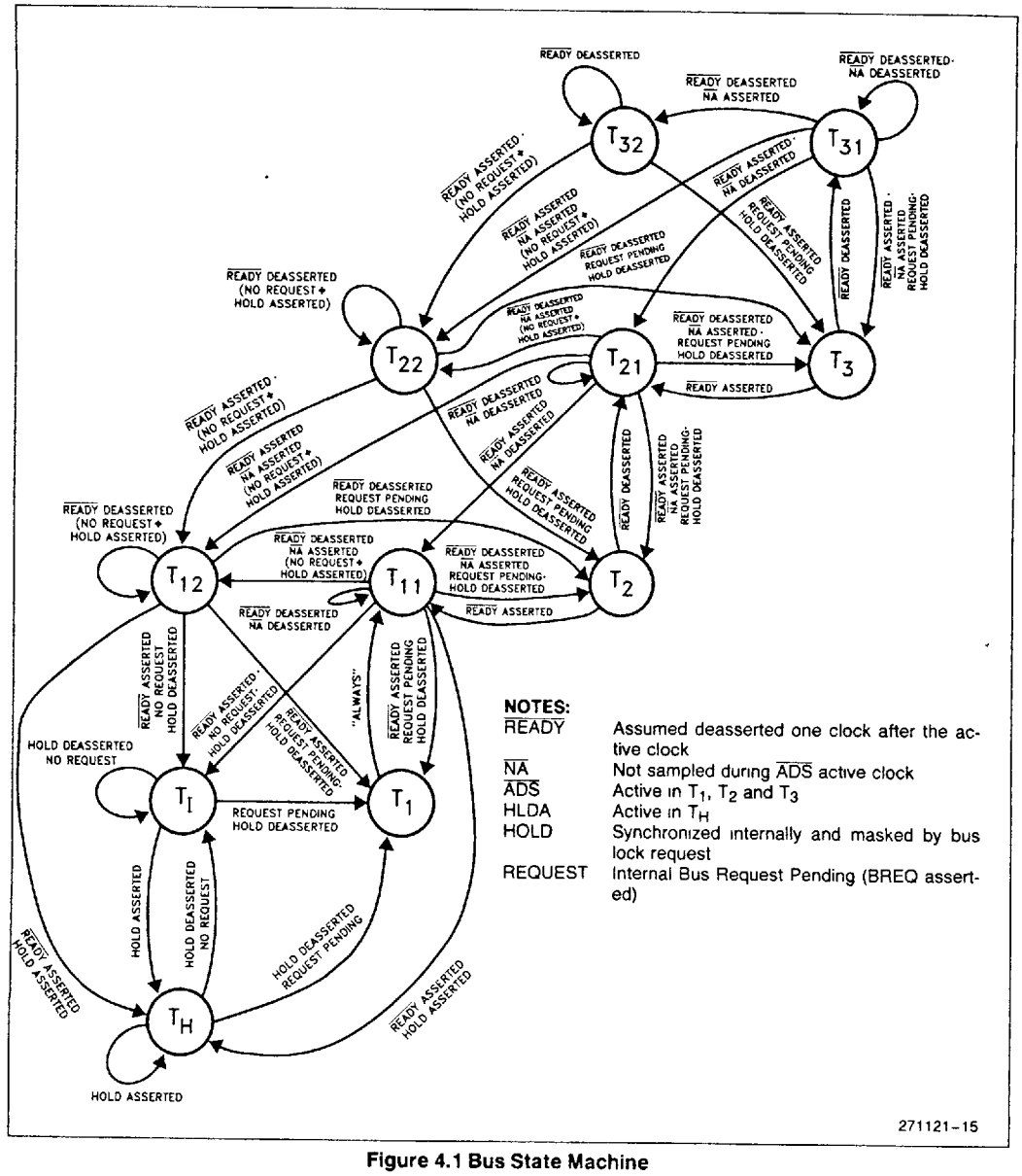


Figure 4.1 Bus State Machine

(if no bus internal request is pending, signifying $\overline{NA}$ has been found active). Transitions from state T_{12} are similar to those from T_{11}.

If there are two outstanding bus cycles (as indicated by T_{2k} for $k = \{1,2\}$) and $\overline{NA}$ is latched active but $\overline{READY}$ is not active, one more bus request causes entry into state T_3. Transitions from this state are similar to those from T_2.

In general, if there is an internal bus request each time both $\overline{READY}$ and $\overline{NA}$ are active, the state machine continues to oscillate between T_{j1} and T_j, for $j = \{2,3\}$.

When $\overline{NA}$ is sampled active while there is a pending bus request, $\overline{ADS}$ is activated in the next clock period (provided no more than two cycles are already outstanding).

Internal pending bus requests start new bus cycles only if no HOLD request has been recognized. T_H is entered from the idle state T_I, T_{I1} and T_{I2}. HLDA is active in this state. There is a one clock delay to synchronize the HOLD input when the signal meets the respective minimum setup and hold time requirements. The state machine uses the synchronized HOLD to move from state to state.

4.3 Bus Cycles

Figures 4.2 through 4.10 illustrate combinations of bus cycles.

4.3.1 NONPIPELINED READ CYCLES

A read cycle begins with the clock in which $\overline{ADS}$ is asserted. The Military i860 XR microprocessor begins driving the address during this clock. It samples $\overline{READY}$ for active state every clock after the first clock. A minimum of two clocks are required per cycle. Data is latched when $\overline{READY}$ is found active when sampled at the end of a clock period. Figure 4.2 illustrates nonpipelined read cycles with zero wait states.

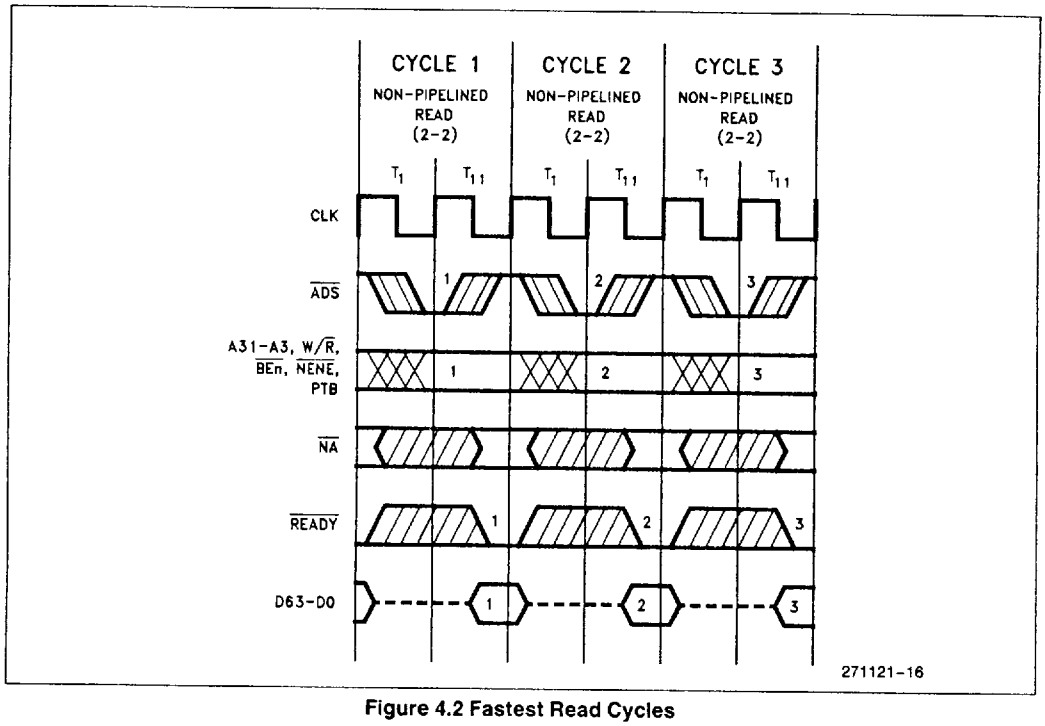


Figure 4.2 Fastest Read Cycles

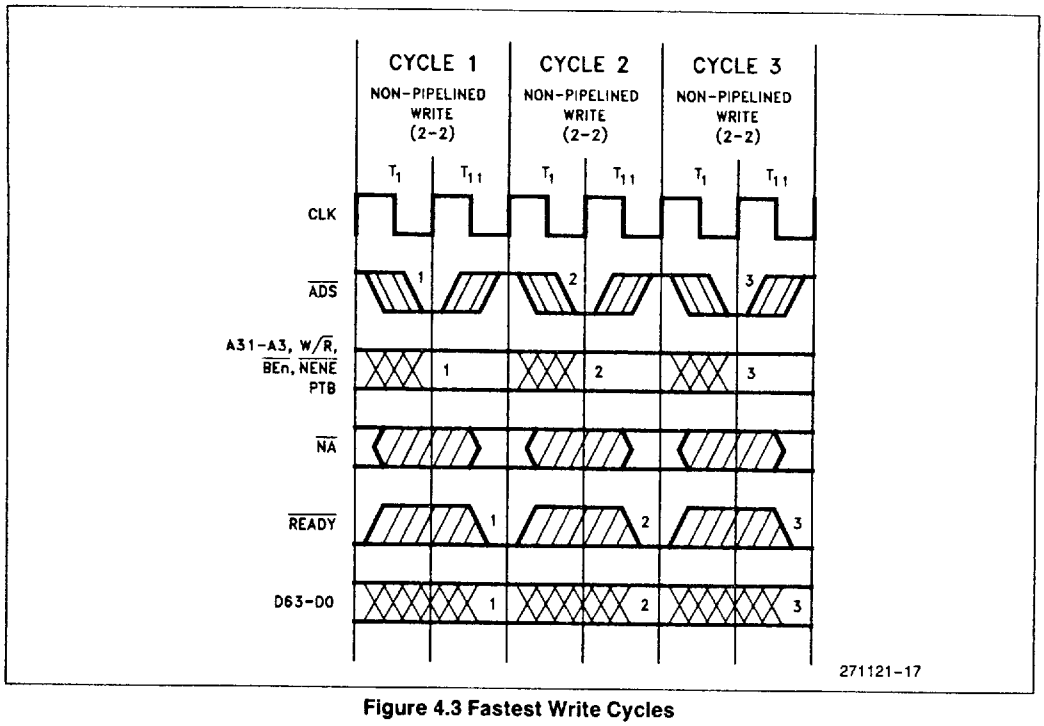


Figure 4.3 Fastest Write Cycles

4.3.2 NONPIPELINED WRITE CYCLES

The ADS and READY activity for write cycles follows the same logic as that for read cycles, as Figure 4.3 illustrates for back-to-back, nonpipelined write cycles with zero wait-states.

The fastest write cycle takes only two clocks to complete. However, when a read cycle immediately precedes a write cycle, the write cycle must contain a wait state, as illustrated in Figure 4.4. Because the device being read might still be driving the data bus during the first clock of the write cycle, there is a potential for bus contention. To help avoid such contention, the Military i860 XR microprocessor does not drive the data bus until the second clock of the write cycle. The wait state is required to provide the additional time necessary to terminate the write cycle. In other read-write combinations, the Military i860 XR microprocessor does not require a wait state.

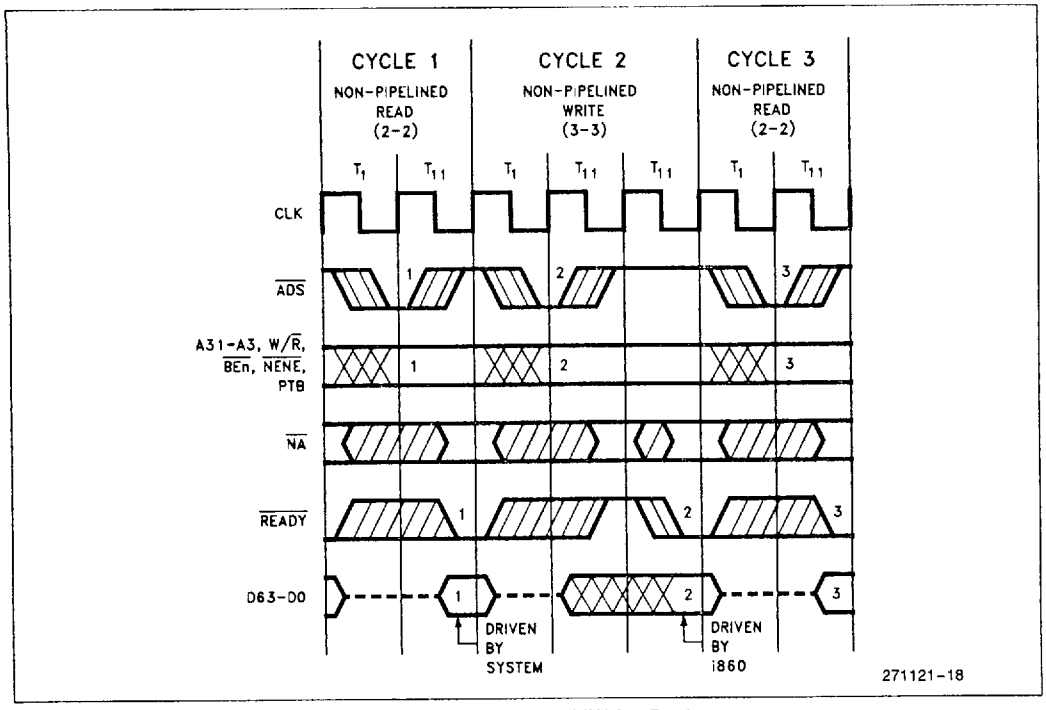


Figure 4.4 Fastest Read/Write Cycles

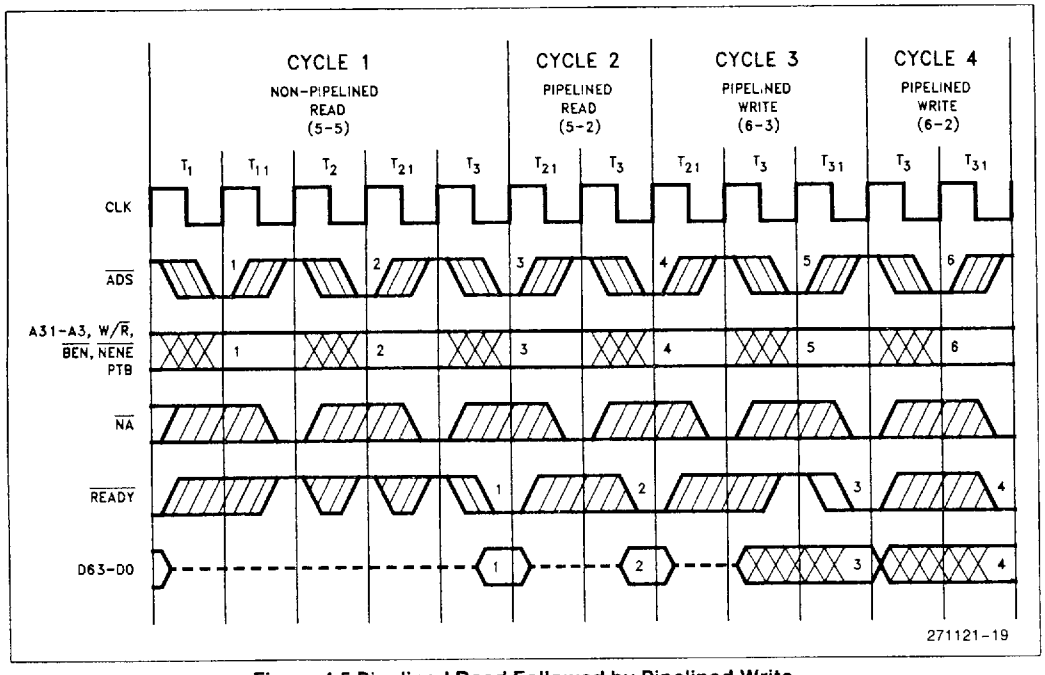


Figure 4.5 Pipelined Read Followed by Pipelined Write

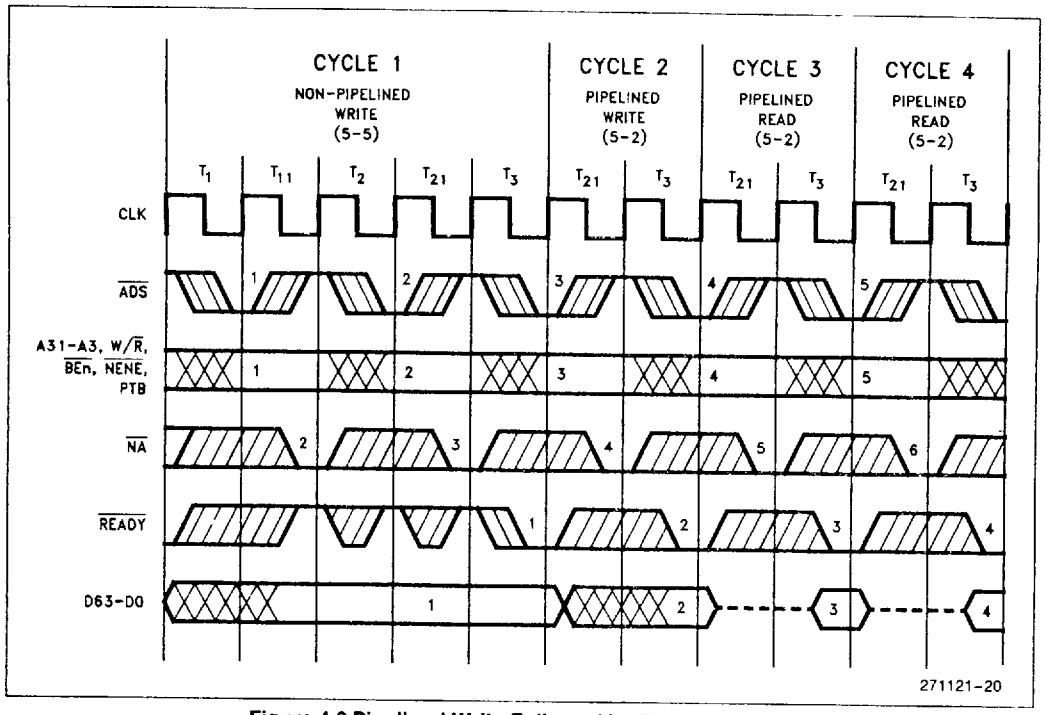


Figure 4.6 Pipelined Write Followed by Pipelined Read

4.3.3 PIPELINED READ AND WRITE CYCLES

Figures 4.5 and 4.6 illustrate combinations of non-pipelined and pipelined read and write cycles. The following description applies to both diagrams. While Cycle 1 is still in progress, two new cycles are initiated. By the time $\overline{READY}$ first becomes active, the state machine has moved through states T_1, T_{11}, T_2, T_{21} and T_3. Cycles 3 and 4 show how activating $\overline{READY}$ terminates the corresponding outstanding cycle, and yet activating $\overline{NA}$ while there is an internal request pending adds a new outstanding cycle.

In Figure 4.5, Cycle 3 is a write cycle following a read cycle; therefore, one wait state must be inserted. The Military i860 XR microprocessor does not drive the data bus until one clock after the read data is returned from the preceding read cycle. During Cycles 3 and 4, the state machine oscillates between states T_3 and T_{31} maintaining full bus capacity (two levels of pipelining, three outstanding cycles). Cycles 2, 3 and 4 in Figure 4.6 are 5-2 cycles, i.e. each requires a total cycle time of five clocks while the throughput rate is one cycle every two clocks.

Figure 4.7 illustrates in a more general manner how the $\overline{NA}$ signal controls pipelining. Cycle 1 is a 2-2 cycle, the fastest possible. The next cycle cannot be started any earlier; therefore, there is no need to activate $\overline{NA}$ to start the next cycle early. Cycle 2, a 3-3 read, is different. Cycle 3 can be started during the third state (a wait state) of Cycle 2, and $\overline{NA}$ is asserted to accomplish this.

$\overline{NA}$ is not activated following the $\overline{ADS}$ clock of Cycle 3, thereby allowing Cycle 3 to terminate before the start of Cycle 4. As a result, Cycle 4 is a nonpipelined cycle.

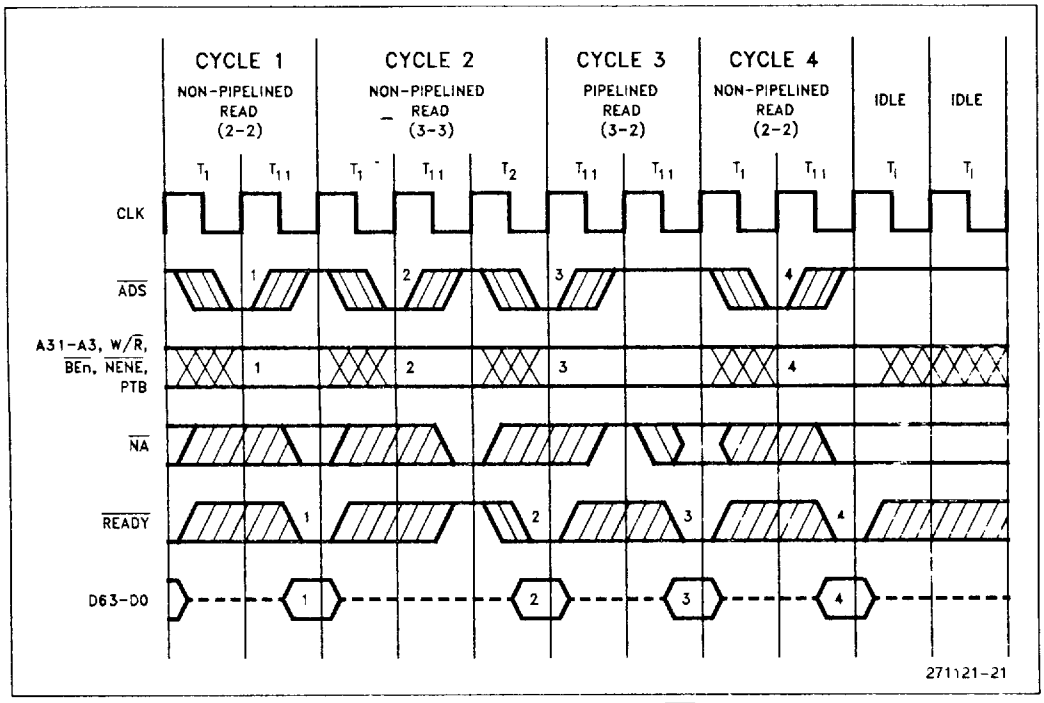


Figure 4.7 Pipelining Driven by $\overline{NA}$

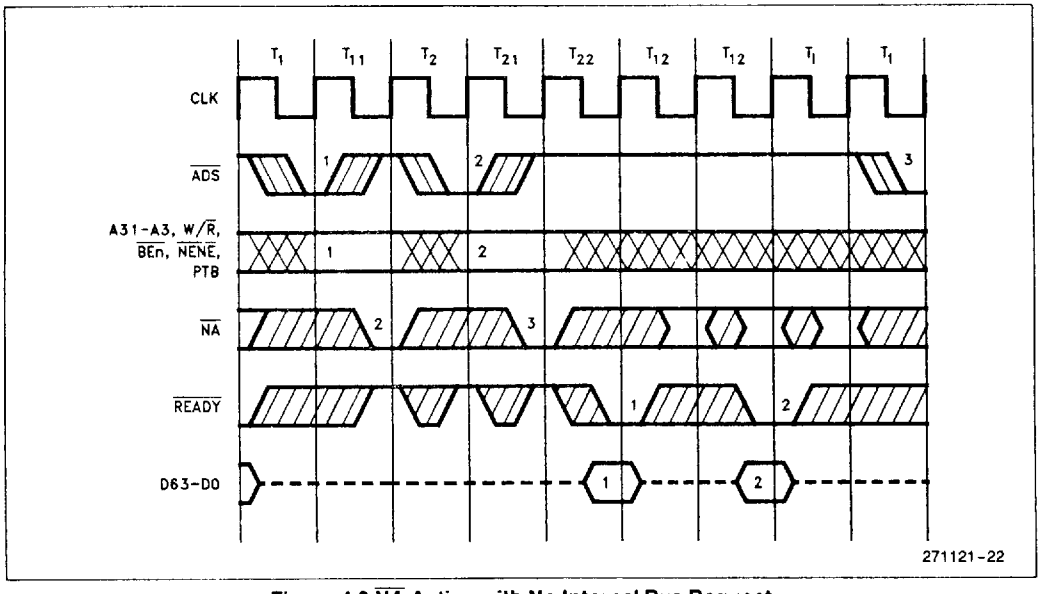


Figure 4.8 $\overline{NA}$ Active with No Internal Bus Request

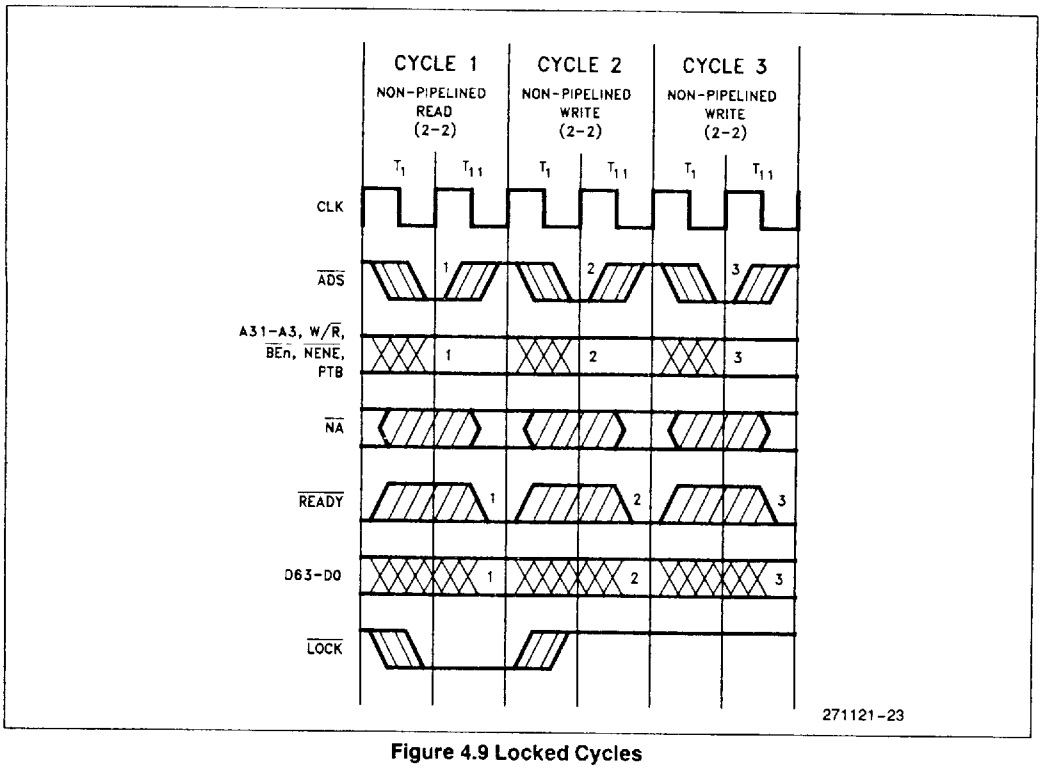


Figure 4.9 Locked Cycles

When there is no internal bus request, activating $\overline{NA}$ does not start a new cycle; the Military i860 XR microprocessor, however, remembers that $\overline{NA}$ has been activated Figure 4.8 illustrates the situation where $\overline{NA}$ is active but no internal bus request is pending. $\overline{NA}$ is activated when two cycles are outstanding. Because there is no internal request pending until after one idle state, no new bus cycle is started during that period.

4.3.4 LOCKED CYCLES

The $\overline{LOCK}$ signal is asserted when the current bus cycle is to be locked with the next bus cycle. Assertion of $\overline{LOCK}$ may be initiated by a program's setting the BL bit of the **dirbase** register using the **lock** instruction (refer to Section 2) or by the Military i860 XR microprocessor itself during page table updates.

In Figure 4.9, the first read cycle is to be locked with the following write cycle. If there were idle states between the cycles, the $\overline{LOCK}$ signal would remain asserted. This is the case for a read/modify/write operation Cycle 3 is not locked because $\overline{LOCK}$ is no longer asserted when Cycle 2 starts.

4.3.5 HOLD AND BREQ ARBITRATION CYCLES

The HOLD, HLDA, and BREQ signals permit bus arbitration between the Military i860 XR microprocessor and another bus master.

See Figure 4.10. When HOLD is asserted, the Military i860 XR microprocessor does not relinquish control of the bus until all outstanding cycles are completed. If HOLD were asserted one clock earlier, the last Military i860 XR microprocessor bus cycle before HLDA would not be started.

The outputs (except HLDA and BREQ) float when HLDA is asserted. HOLD is sampled at the end of the clock in which it is activated. Recommended setup and hold times must be met to guarantee sampling one clock after external HOLD activation. When HOLD is sampled active, a one clock delay for internal synchronization follows. Likewise when HOLD is deasserted, there is a one-clock delay for internal synchronization before HLDA is deasserted.

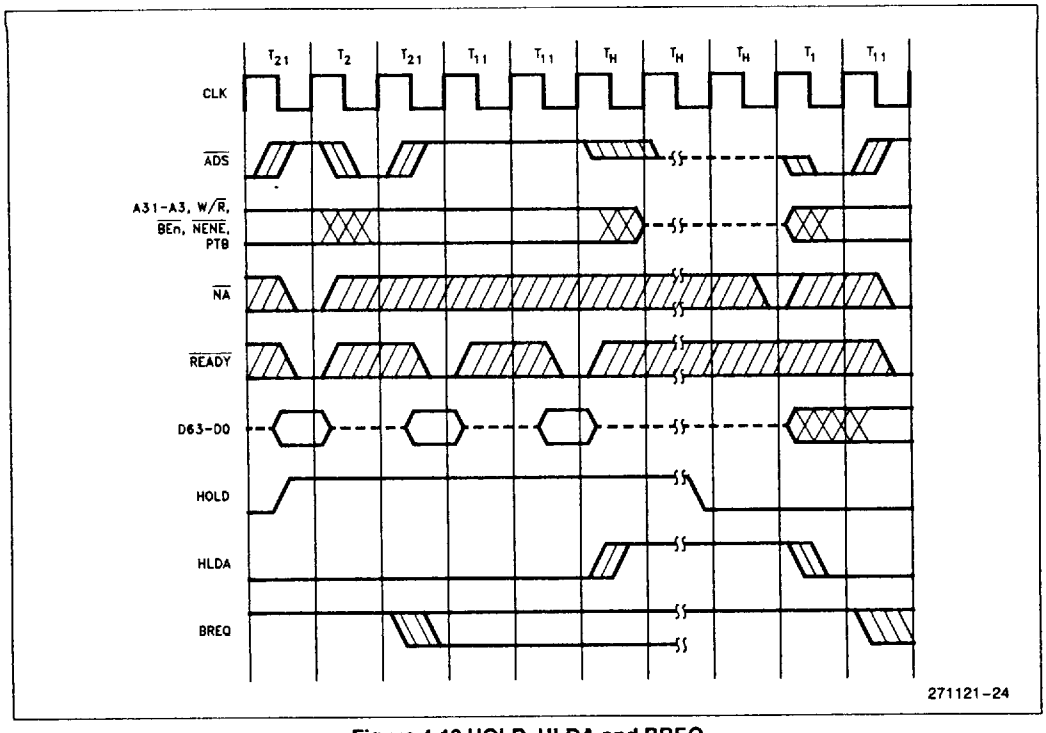


Figure 4.10 HOLD, HLDA and BREQ

If, during a HOLD cycle, an internal bus request is generated, BREQ is activated even though HLDA is asserted. It remains active at least until the clock after ADS is activated for the requested cycle.

4.4 Bus States During RESET

Figure 4.11 shows how INT/CS8 is sampled during the clock period just before the falling edge of RESET. If INT/CS8 is sampled active, the Military i860 XR microprocessor enters CS8 mode. No inputs (except for HOLD and INT/CS8) are sampled during RESET.

Note that, because HOLD is recognized even while RESET is active, the HLDA output signal may also become active during RESET. Refer to Table 3.4 "Output Pin Status During Reset".

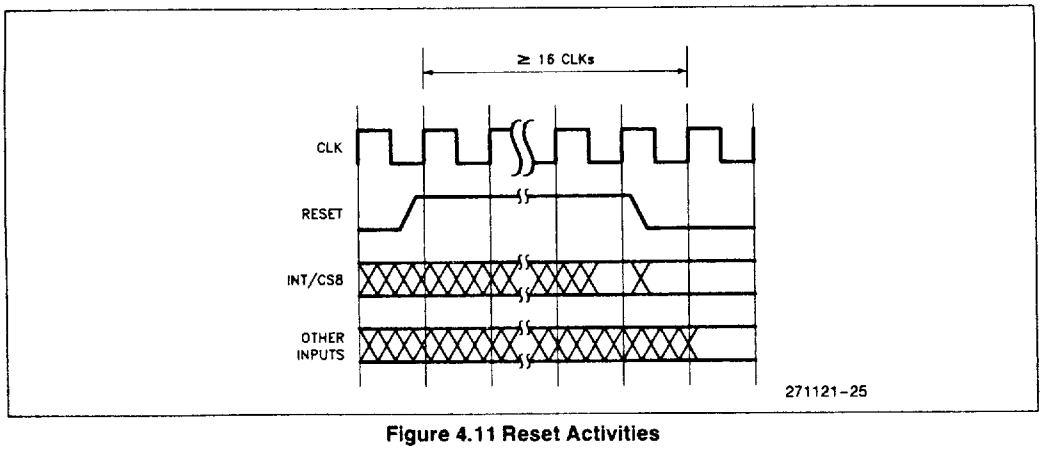


Figure 4.11 Reset Activities

5.0 MECHANICAL DATA

Figures 5.1, 5.2 and 5.3 show the locations of pins; Tables 5.1, 5.2, 5.3 and 5.4 help to locate pin identifiers.

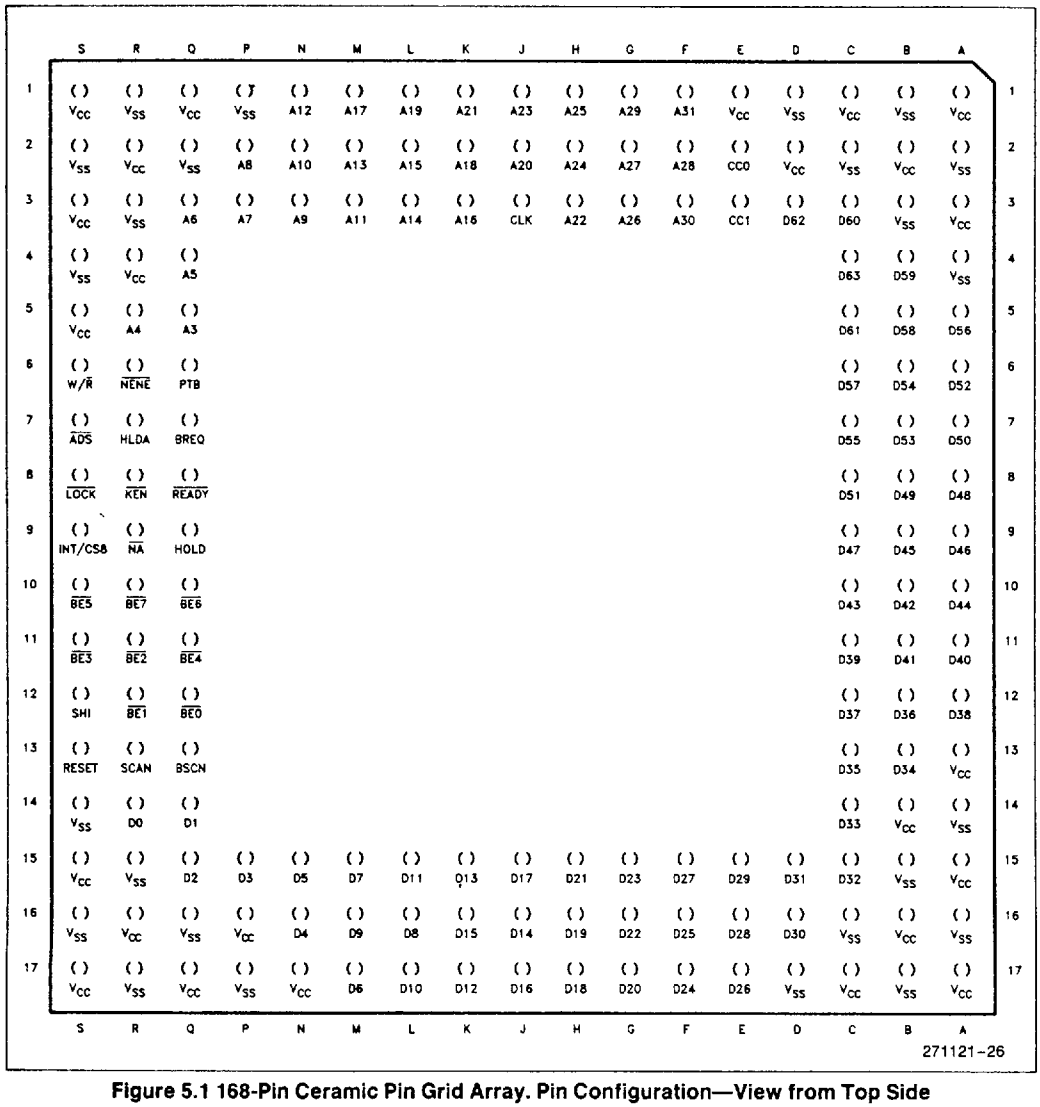


Figure 5.1 168-Pin Ceramic Pin Grid Array. Pin Configuration—View from Top Side

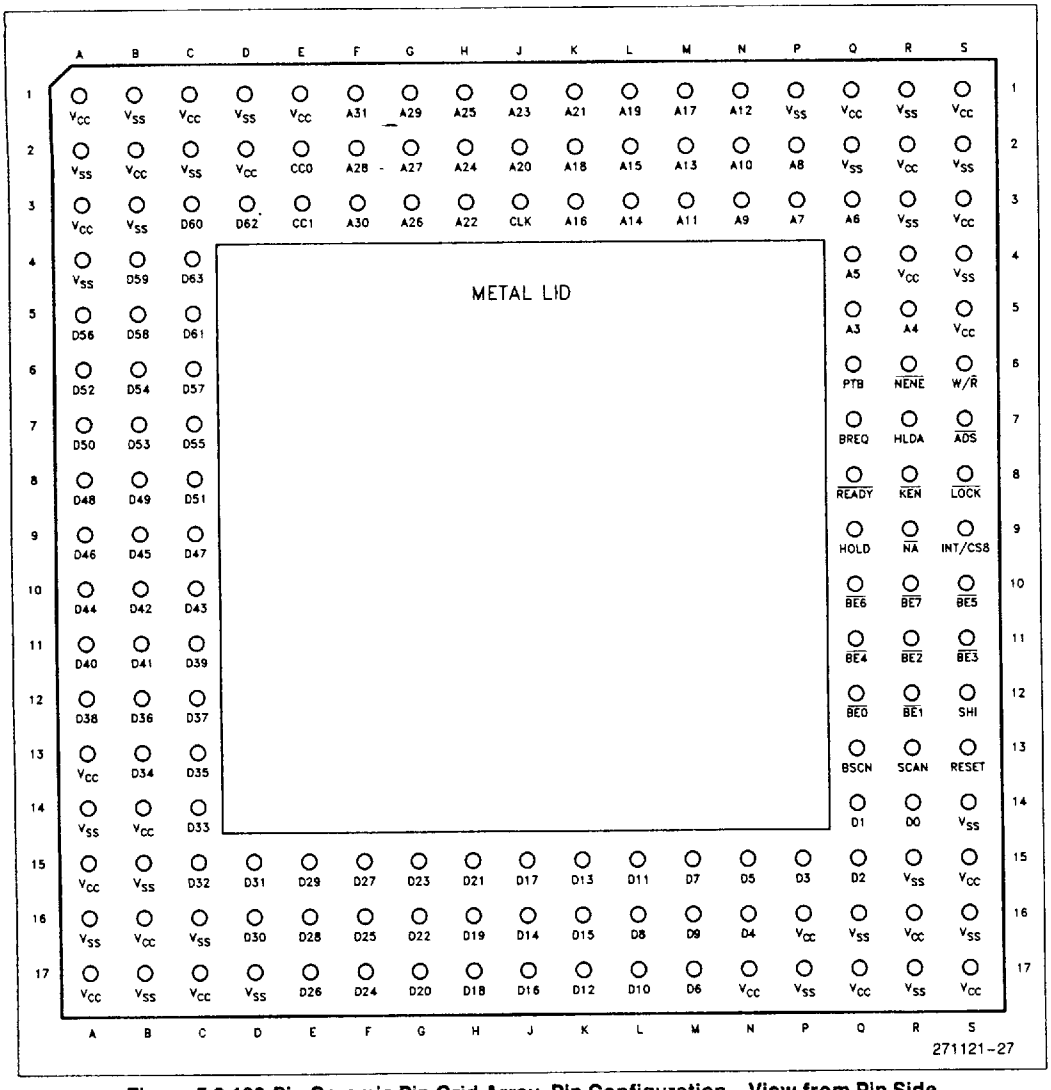


Figure 5.2 168-Pin Ceramic Pin Grid Array. Pin Configuration—View from Pin Side

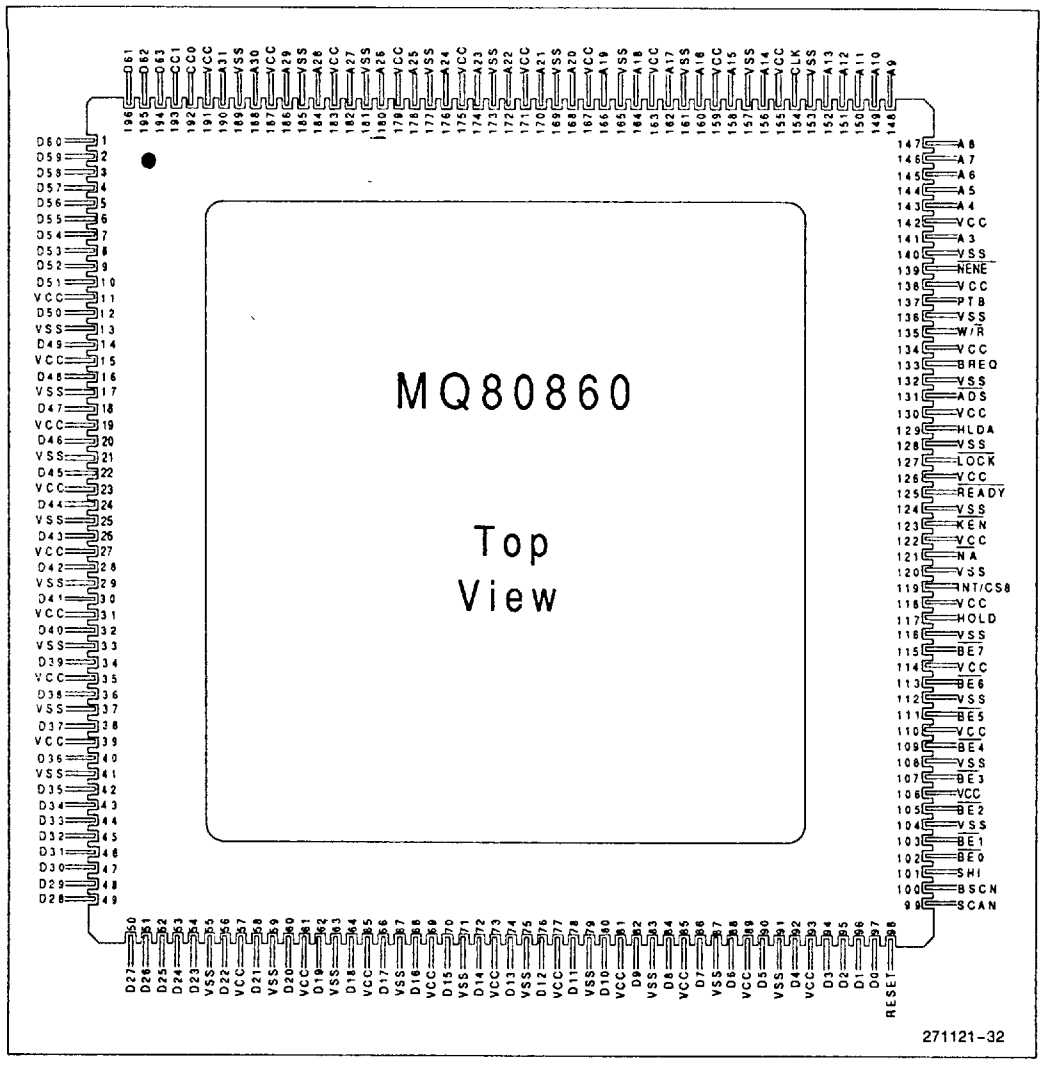


Figure 5.3 196-Pin Ceramic Quad Flatpack (CQFP). Pin Configuration—View from Lid Side

Table 5.1 168-Pin Ceramic PGA Pin Assignment by Location

Location	Signal	Location	Signal	Location	Signal	Location	Signal
A1	V_{CC}	C9	D47	J15	D17	Q10	$\overline{BE6}$
A2	V_{SS}	C10	D43	J16	D14	Q11	$\overline{BE4}$
A3	V_{CC}	C11	D39	J17	D16	Q12	$\overline{BE0}$
A4	V_{SS}	C12	D37	K1	A21	Q13	BSCN
A5	D56	C13	D35	K2	A18	Q14	D1
A6	D52	C14	D33	K3	A16	Q15	D2
A7	D50	C15	D32	K15	D13	Q16	V_{SS}
A8	D48	C16	V_{SS}	K16	D15	Q17	V_{CC}
A9	D46	C17	V_{CC}	K17	D12	R1	V_{SS}
A10	D44	D1	V_{SS}	L1	A19	R2	V_{CC}
A11	D40	D2	V_{CC}	L2	A15	R3	V_{SS}
A12	D38	D3	D62	L3	A14	R4	V_{CC}
A13	V_{CC}	D15	D31	L15	D11	R5	A4
A14	V_{SS}	D16	D30	L16	D8	R6	$\overline{NENE}$
A15	V_{CC}	D17	V_{SS}	L17	D10	R7	HLDA
A16	V_{SS}	E1	V_{CC}	M1	A17	R8	$\overline{KEN}$
A17	V_{CC}	E2	CC0	M2	A13	R9	$\overline{NA}$
B1	V_{SS}	E3	CC1	M3	A11	R10	$\overline{BE7}$
B2	V_{CC}	E15	D29	M15	D7	R11	$\overline{BE2}$
B3	V_{SS}	E16	D28	M16	D9	R12	$\overline{BE1}$
B4	D59	E17	D26	M17	D6	R13	SCAN
B5	D58	F1	A31	N1	A12	R14	D0
B6	D54	F2	A28	N2	A10	R15	V_{SS}
B7	D53	F3	A30	N3	A9	R16	V_{CC}
B8	D49	F15	D27	N15	D5	R17	V_{SS}
B9	D45	F16	D25	N16	D4	S1	V_{CC}
B10	D42	F17	D24	N17	V_{CC}	S2	V_{SS}
B11	D41	G1	A29	P1	V_{SS}	S3	V_{CC}
B12	D36	G2	A27	P2	A8	S4	V_{SS}
B13	D34	G3	A26	P3	A7	S5	V_{CC}
B14	V_{CC}	G15	D23	P15	D3	S6	W/$\overline{R}$
B15	V_{SS}	G16	D22	P16	V_{CC}	S7	$\overline{ADS}$
B16	V_{CC}	G17	D20	P17	V_{SS}	S8	$\overline{LOCK}$
B17	V_{SS}	H1	A25	Q1	V_{CC}	S9	INT/CS8
C1	V_{CC}	H2	A24	Q2	V_{SS}	S10	$\overline{BE5}$
C2	V_{SS}	H3	A22	Q3	A6	S11	$\overline{BE3}$
C3	D60	H15	D21	Q4	A5	S12	SHI
C4	D63	H16	D19	Q5	A3	S13	RESET
C5	D61	H17	D18	Q6	PTB	S14	V_{SS}
C6	D57	J1	A23	Q7	BREQ	S15	V_{CC}
C7	D55	J2	A20	Q8	$\overline{READY}$	S16	V_{SS}
C8	D51	J3	CLK	Q9	HOLD	S17	V_{CC}

Table 5.2 168-Pin Ceramic PGA Pin Assignment by Function

Address		Data		Data		Control		V_{CC}		V_{SS}	
Signal	Location	Signal	Location	Signal	Location	Signal	Location	Signal	Location	Signal	Location
A3	Q5	D0	R14	D32	C15	$\overline{ADS}$	S7	V_{CC}	A1	V_{SS}	A2
A4	R5	D1	Q14	D33	C14	$\overline{BE0}$	Q12	V_{CC}	A3	V_{SS}	A4
A5	Q4	D2	Q15	D34	B13	$\overline{BE1}$	R12	V_{CC}	A13	V_{SS}	A14
A6	Q3	D3	P15	D35	C13	$\overline{BE2}$	R11	V_{CC}	A15	V_{SS}	A16
A7	P3	D4	N16	D36	B12	$\overline{BE3}$	S11	V_{CC}	A17	V_{SS}	B1
A8	P2	D5	N15	D37	C12	$\overline{BE4}$	Q11	V_{CC}	B2	V_{SS}	B3
A9	N3	D6	M17	D38	A12	$\overline{BE5}$	S10	V_{CC}	B14	V_{SS}	B15
A10	N2	D7	M15	D39	C11	$\overline{BE6}$	Q10	V_{CC}	B16	V_{SS}	B17
A11	M3	D8	L16	D40	A11	$\overline{BE7}$	R10	V_{CC}	C1	V_{SS}	C2
A12	N1	D9	M16	D41	B11	BREQ	Q7	V_{CC}	C17	V_{SS}	C16
A13	M2	D10	L17	D42	B10	BSCN	Q13	V_{CC}	D2	V_{SS}	D1
A14	L3	D11	L15	D43	C10	CC0	E2	V_{CC}	E1	V_{SS}	D17
A15	L2	D12	K17	D44	A10	CC1	E3	V_{CC}	N17	V_{SS}	P1
A16	K3	D13	K15	D45	B9	CLK	J3	V_{CC}	P16	V_{SS}	P17
A17	M1	D14	J16	D46	A9	HLDA	R7	V_{CC}	Q1	V_{SS}	Q2
A18	K2	D15	K16	D47	C9	HOLD	Q9	V_{CC}	Q17	V_{SS}	Q16
A19	L1	D16	J17	D48	A8	INT/CS8	S9	V_{CC}	R2	V_{SS}	R1
A20	J2	D17	J15	D49	B8	$\overline{KEN}$	R8	V_{CC}	R4	V_{SS}	R3
A21	K1	D18	H17	D50	A7	$\overline{LOCK}$	S8	V_{CC}	R16	V_{SS}	R15
A22	H3	D19	H16	D51	C8	$\overline{NA}$	R9	V_{CC}	S1	V_{SS}	R17
A23	J1	D20	G17	D52	A6	$\overline{NENE}$	R6	V_{CC}	S3	V_{SS}	S2
A24	H2	D21	H15	D53	B7	PTB	Q6	V_{CC}	S5	V_{SS}	S4
A25	H1	D22	G16	D54	B6	$\overline{READY}$	Q8	V_{CC}	S15	V_{SS}	S14
A26	G3	D23	G15	D55	C7	RESET	S13	V_{CC}	S17	V_{SS}	S16
A27	G2	D24	F17	D56	A5	SCAN	R13				
A28	F2	D25	F16	D57	C6	SHI	S12				
A29	G1	D26	E17	D58	B5	W/$\overline{R}$	S6				
A30	F3	D27	F15	D59	B4						
A31	F1	D28	E16	D60	C3						
		D29	E15	D61	C5						
		D30	D16	D62	D3						
		D31	D15	D63	C4						

Table 5.3 196-Pin Ceramic Quad Flatpack (CQFP) Pin Assignment by Location

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	D60	50	D27	99	SCAN	148	A9
2	D59	51	D26	100	BSCN	149	A10
3	D58	52	D25	101	SHI	150	A11
4	D57	53	D24	102	$\overline{BE0}$	151	A12
5	D56	54	D23	103	$\overline{BE1}$	152	A13
6	D55	55	V_{SS}	104	V_{SS}	153	V_{SS}
7	D54	56	D22	105	$\overline{BE2}$	154	CLK
8	D53	57	V_{CC}	106	V_{CC}	155	V_{CC}
9	D52	58	D21	107	$\overline{BE3}$	156	A14
10	D51	59	V_{SS}	108	V_{SS}	157	V_{SS}
11	V_{CC}	60	D20	109	$\overline{BE4}$	158	A15
12	D50	61	V_{CC}	110	V_{CC}	159	V_{CC}
13	V_{SS}	62	D19	111	$\overline{BE5}$	160	A16
14	D49	63	V_{SS}	112	V_{SS}	161	V_{SS}
15	V_{CC}	64	D18	113	$\overline{BE6}$	162	A17
16	D48	65	V_{CC}	114	V_{CC}	163	V_{CC}
17	V_{SS}	66	D17	115	$\overline{BE7}$	164	A18
18	D47	67	V_{SS}	116	V_{SS}	165	V_{SS}
19	V_{CC}	68	D16	117	HOLD	166	A19
20	D46	69	V_{CC}	118	V_{CC}	167	V_{CC}
21	V_{SS}	70	D15	119	INT/CS8	168	A20
22	D45	71	V_{SS}	120	V_{SS}	169	V_{SS}
23	V_{CC}	72	D14	121	$\overline{NA}$	170	A21
24	D44	73	V_{CC}	122	V_{CC}	171	V_{CC}
25	V_{SS}	74	D13	123	$\overline{KEN}$	172	A22
26	D43	75	V_{SS}	124	V_{SS}	173	V_{SS}
27	V_{CC}	76	D12	125	$\overline{READY}$	174	A23
28	D42	77	V_{CC}	126	V_{CC}	175	V_{CC}
29	V_{SS}	78	D11	127	$\overline{LOCK}$	176	A24
30	D41	79	V_{SS}	128	V_{SS}	177	V_{SS}
31	V_{CC}	80	D10	129	HLDA	178	A25
32	D40	81	V_{CC}	130	V_{CC}	179	V_{CC}
33	V_{SS}	82	D9	131	$\overline{ADS}$	180	A26
34	D39	83	V_{SS}	132	V_{SS}	181	V_{SS}
35	V_{CC}	84	D8	133	BREQ	182	A27
36	D38	85	V_{CC}	134	V_{CC}	183	V_{CC}
37	V_{SS}	86	D7	135	W/$\overline{R}$	184	A28
38	D37	87	V_{SS}	136	V_{SS}	185	V_{SS}
39	V_{CC}	88	D6	137	PTB	186	A29
40	D36	89	V_{CC}	138	V_{CC}	187	V_{CC}
41	V_{SS}	90	D5	139	$\overline{NENE}$	188	A30
42	D35	91	V_{SS}	140	V_{SS}	189	V_{SS}
43	D34	92	D4	141	A3	190	A31
44	D33	93	V_{CC}	142	V_{CC}	191	V_{CC}
45	D32	94	D3	143	A4	192	CC0
46	D31	95	D2	144	A5	193	CC1
47	D30	96	D1	145	A6	194	D63
48	D29	97	D0	146	A7	195	D62
49	D28	98	RESET	147	A8	196	D61

Table 5.4 196-Pin Ceramic Quad Flatpack (CQFP) Pin Assignment by Function

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A3	141	D20	60	$\overline{BE4}$	109	V_{CC}	142
A4	143	D21	58	$\overline{BE5}$	111	V_{CC}	155
A5	144	D22	56	$\overline{BE6}$	113	V_{CC}	159
A6	145	D23	54	$\overline{BE7}$	115	V_{CC}	163
A7	146	D24	53	BREQ	133	V_{CC}	167
A8	147	D25	52	BSCN	100	V_{CC}	171
A9	148	D26	51	CC0	192	V_{CC}	175
A10	149	D27	50	CC1	193	V_{CC}	179
A11	150	D28	49	CLK	154	V_{CC}	183
A12	151	D29	48	HLDA	129	V_{CC}	187
A13	152	D30	47	HOLD	117	V_{CC}	191
A14	156	D31	46	INT/CS8	119	V_{SS}	13
A15	158	D32	45	$\overline{KEN}$	123	V_{SS}	17
A16	160	D33	44	$\overline{LOCK}$	127	V_{SS}	21
A17	162	D34	43	$\overline{NA}$	121	V_{SS}	25
A18	164	D35	42	$\overline{NENE}$	139	V_{SS}	29
A19	166	D36	40	PTB	137	V_{SS}	33
A20	168	D37	38	$\overline{READY}$	125	V_{SS}	37
A21	170	D38	36	RESET	98	V_{SS}	41
A22	172	D39	34	SCAN	99	V_{SS}	55
A23	174	D40	32	SHI	101	V_{SS}	59
A24	176	D41	30	W/$\overline{R}$	135	V_{SS}	63
A25	178	D42	28	V_{CC}	11	V_{SS}	67
A26	180	D43	26	V_{CC}	15	V_{SS}	71
A27	182	D44	24	V_{CC}	19	V_{SS}	75
A28	184	D45	22	V_{CC}	23	V_{SS}	79
A29	186	D46	20	V_{CC}	27	V_{SS}	83
A30	188	D47	18	V_{CC}	31	V_{SS}	87
A31	190	D48	16	V_{CC}	35	V_{SS}	91
D0	97	D49	14	V_{CC}	39	V_{SS}	104
D1	96	D50	12	V_{CC}	57	V_{SS}	108
D2	95	D51	10	V_{CC}	61	V_{SS}	112
D3	94	D52	9	V_{CC}	65	V_{SS}	116
D4	92	D53	8	V_{CC}	69	V_{SS}	120
D5	90	D54	7	V_{CC}	73	V_{SS}	124
D6	88	D55	6	V_{CC}	77	V_{SS}	128
D7	86	D56	5	V_{CC}	81	V_{SS}	132
D8	84	D57	4	V_{CC}	85	V_{SS}	136
D9	82	D58	3	V_{CC}	89	V_{SS}	140
D10	80	D59	2	V_{CC}	93	V_{SS}	153
D11	78	D60	1	V_{CC}	106	V_{SS}	157
D12	76	D61	196	V_{CC}	110	V_{SS}	161
D13	74	D62	195	V_{CC}	114	V_{SS}	165
D14	72	D63	194	V_{CC}	118	V_{SS}	169
D15	70	$\overline{ADS}$	131	V_{CC}	122	V_{SS}	173
D16	68	$\overline{BE0}$	102	V_{CC}	126	V_{SS}	177
D17	66	$\overline{BE1}$	103	V_{CC}	130	V_{SS}	181
D18	64	$\overline{BE2}$	105	V_{CC}	134	V_{SS}	185
D19	62	$\overline{BE3}$	107	V_{CC}	138	V_{SS}	189

6.0 ELECTRICAL DATA

Inputs and outputs are TTL compatible. All input and output timings are specified relative to the 1.5 volt level of the rising edge of CLK and refer to the point that the signals reach 1.5V.

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

**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.*

6.1 Absolute Maximum Ratings

Case Temperature T_C under Bias ..... −55°C to +125°C

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

Voltage on Any Pin with Respect to Ground ..... −0.5 to 6.5V

6.2 Operating Conditions

Symbol	Parameter	Min	Max	Units
T_C	Case Temperature (Instant On)	−55	+125	°C
V_{CC}	Digital Supply Voltage	4.75	5.25	V

6.3 DC Characteristics (Over Specified Operating Conditions)

Table 6.1 DC Characteristics

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Input LOW Voltage	−0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC}+0.3	V	
V_{ILC}	CLK Input LOW Voltage	−0.3	+0.8	V	
V_{IHC}	CLK Input HIGH Voltage	3.0	V_{CC} + 0.3	V	
V_{OL}	Output LOW Voltage		0.45	V	(Note 1)
V_{OH}	Output HIGH Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current CLK = 25.0 MHz, 33.3 MHz CLK = 40.0 MHz		 600 650	 mA mA	 V_{CC} @5V V_{CC} @5V
I_{LI}	Input Leakage Current		±15	μA	No pullup or pulldown
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		15	pF	(Note 3)
C_O	I/O or Output Capacitance		15	pF	(Note 3)
C_{CLK}	Clock Capacitance		20	pF	(Note 3)
θ_{JA}	Thermal Resistance (Junction-to-Ambient) Pin Grid Array Ceramic Quad Flatpack	 NA NA	 17 23	 °C/W °C/W	
θ_{JC}	Thermal Resistance (Junction-to-Case) Pin Grid Array Ceramic Quad Flatpack	 NA NA	 1.5 7	 °C/W °C/W	

NOTES:
1. This parameter is measured at 4.0 mA for A31–A3, D63–D0, $\overline{BE7}$–$\overline{BE0}$; at 5.0 mA for all other outputs.
2. This parameter is measured at 1.0 mA for A31–A3, D63–D0, $\overline{BE7}$–$\overline{BE0}$; at 0.9 mA all other outputs.
3. These parameters are not tested. They are guaranteed by design characterization.

6.4 AC Characteristics (Over Specified Operating Conditions)

Table 6.2 AC Characteristics

All timings measured at CLK = 1.5V unless otherwise specified.

Symbol	Parameter	25 MHz		33 MHz		40 MHz		Notes
		Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	
t1	CLK Period	40	125	30	125	25	125	
t2	CLK High Time	10		7		5		at 3V
t3	CLK Low Time	10		7		5		at 0.8V
t4	CLK Fall Time		7		7		7	3V–0.8V
t5	CLK Rise Time		7		7		7	0.8V–3V
t6a	A31–A3, PTB, W/$\overline{R}$, $\overline{NENE}$ Valid Delay	3.5	43	3.5	23	3.5	21	50 pF load
t6b	$\overline{BEn}$* Valid Delay	3.5	25	3.5	25	3.5	23	50 pF load
t7	Float Time, All Outputs	3.5	40	3.5	30	3.5	25	(Note 1)
t8	$\overline{ADS}$, BREQ, $\overline{LOCK}$, HLDA Valid Delay	3.5	29	3.5	20	3.5	15	50 pF load
t9	D63–D0 Valid Delay	3.5	60	3.5	35	3.5	33	50 pF load
t10a	Setup Time, All Inputs except DATA	16		11		8		(Note 2)
t11a	Hold Time, All Inputs except DATA	7		4		3		(Note 2)
t10b	DATA Setup Time	16		11		10		
t11b	DATA Hold Time	7		4		3		

NOTES:
1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested.
2. INT and HOLD are asynchronous inputs. The setup and hold specifications are given for test purposes or to assure recognition on a specific rising edge of CLK. INT should remain asserted until software acknowledges the interrupt.

*n = 0, 1, . . . , 7

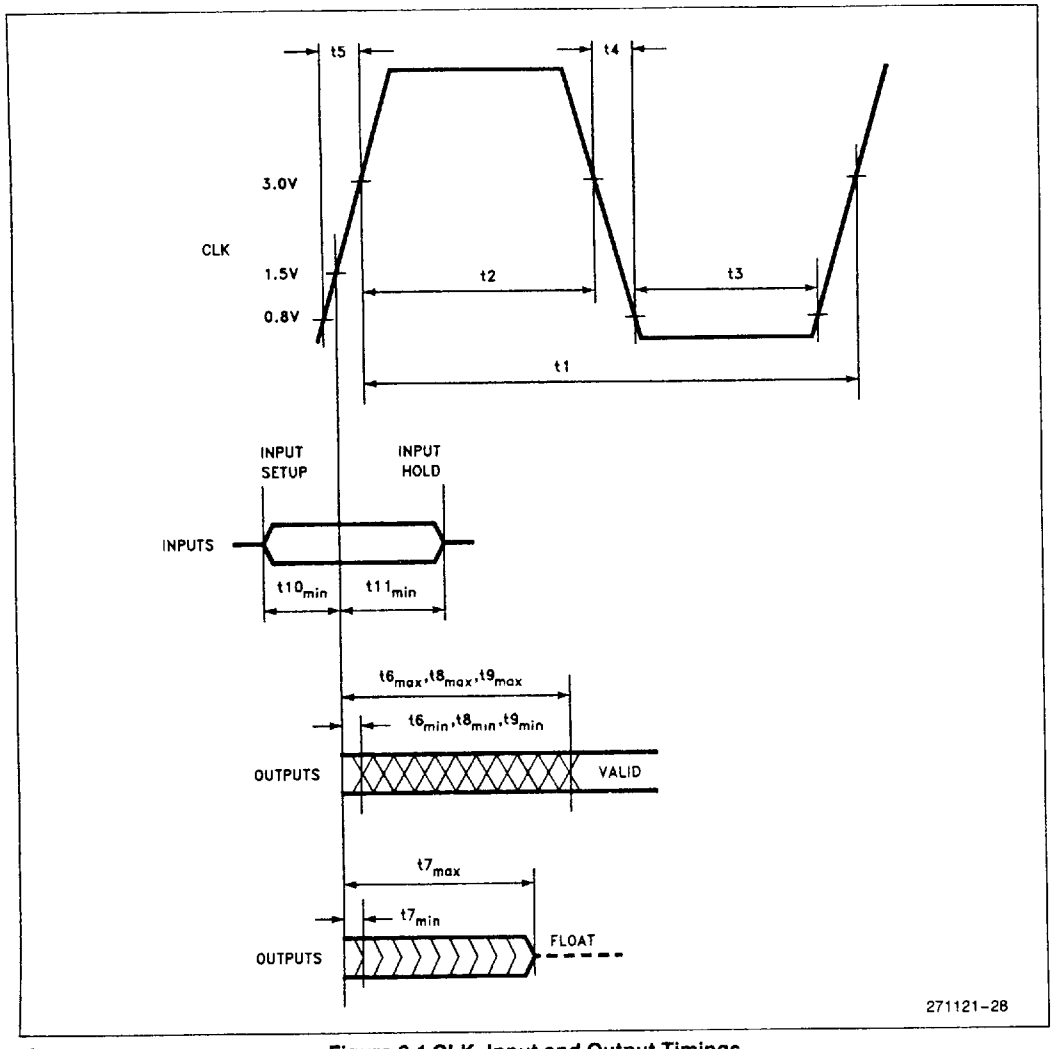


Figure 6.1 CLK, Input and Output Timings

7.0 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** or **fsr**

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 XR 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.

Unless otherwise specified, 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 7.1.
.r	Precision specification **.ss, .sd, .ds** or **.dd**. Refer to Table 7.1.
.v	**.sd** or **.dd**. Refer to Table 7.1.
.w	**.ss** or **.dd.** Refer to Table 7.1.
.x	**.b** (8 bits), **.s** (16 bits), or **.l** (32 bits)
.y	**.l** (32 bits), **.d** (64 bits), or **.q** (128 bits)
.z	**.l** (32 bits), or **.d** (64 bits)
mem.x(address)	The contents of the memory location indicated by *address* with a size of *x*.
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.

Table 7.1 Precision Specification

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

7.1 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* (signed)
CC clear if *isrc2* ≥ −*isrc1* (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 (signed)
        LCC-temp set if isrc2 ≥ −isrc1ni (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 two instructions
                        ELSE    enter single-instruction mode
                                  for next two instructions
                        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 (+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 (+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*
Shift and load MERGE register as defined in Table 7.2

faddz *fsrc1, fsrc2, fdest* ..... **Add with Z Merge**
fdest ← *fsrc1* + *fsrc2*
Shift MERGE right 16 and load fields 31..16 and 63 .48

famov.r *fsrc1, fdest* ..... **Floating-Point Adder Move**
fdest ← *fsrc1*
Send *fsrc1* through the floating-point adder. (Preserves −0 (minus zero) when *fsrc1 is −0. fsrc2* must be coded as **f0** by the assembler.)

fiadd.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Add**
fdest ← *fsrc1* + *fsrc2*

fisub.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Subtract**
fdest ← *fsrc1* − *fsrc2*

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)**
Replace block in data cache with address (*#const* + *isrc2*).
Contents of block 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 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 *fsrc1, fsrc2,* and *fdest* as arrays of two 32-bit
fields *fsrc1*(0)..*fsrc1*(1), *fsrc2*(0)..*fsrc2*(1), and *fdest*(0)..*fdest*(1)
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 *fsrc1, fsrc2,* and *fdest* as arrays of four 16-bit
fields *fsrc1*(0)..*fsrc1*(3), *fsrc2*(0)..*fsrc2*(3), and *fdest*(0)..*fdest*(3)
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 in **epsr** = 1, generate trap with IT set in **psr**.

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*)

lock ..... **Begin Interlocked Sequence**
Set BL in **dirbase**. The next load or store that misses the cache 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
adds *l%const32*, **r0**, *idest*
... when *const32* < 0x8000

orh *h%const32*, **r0**, *idest*
or *l%const32, idest, idest*
... when *const32* ≥ 0x8000

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 result
last stage Graphics result ← *fsrc1* + *fsrc2*
Shift and load MERGE register from last stage Graphics result as defined in Table 7.2

pfaddz *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Z Merge**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* + *fsrc2*
Shift MERGE right 16 and load fields 31..16 and 63..48 from last stage Graphics result

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 Greather-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 result
last stage Graphics result ← *fsrc1* + *fsrc2*

pfisub.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Subtract**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* − *fsrc2*

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.z *isrc1*(*isrc2*), *fdest* ..... **(Normal)**
pfld.z *isrc1*(*isrc2*)++, *fdest* ..... **(Autoincrement)**
fdest ← mem.z (third previous **pfld's** (*isrc1* + *isrc2*))
(where .z is precision of third previous **pfld.z)**
If autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

pfle.p *fsrc1, fsrc2, fdest* ..... **Pipelined F-P Less-Than or Equal Compare**
Assembler pseudo-operation, identical to **pfgt.p** except that
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* ..... **Pipelined Floating-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 result
last stage Graphics 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 *fsrc1, fsrc2,* and *fdest,* as arrays of two 32-bit
fields *fsrc1*(0)..*fsrc1*(1), *fsrc2*(0)..*fsrc2*(1), and *fdest*(0)..*fdest*(1)
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 result
last stage Graphics result ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

pfzchks *fsrc1, fsrc2, fdest* ..... **Pipelined 16-Bit Z-Buffer Check**
Consider *fsrc1, fsrc2,* and *fdest,* as arrays of four 16-bit
fields *fsrc1*(0)..*fsrc1*(3), *fsrc2*(0)..*fsrc2*(3), and *fdest*(0)..*fdest*(3)
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) ← last stage Graphics result
last stage Graphics result ← 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

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 *isrc1, isrc2, idest* — **Shift Right Double**
idest ← low-order 32 bits of *isrc1:isrc2* shifted right by SC bits

st.c *isrc1ni, csrc2* — **Store to Control Register**
csrc2 ← *isrc1ni*

st.x *isrc1ni, #const(isrc2)* — **Store Integer**
mem.x (isrc2 + #const) ← *isrc1ni*

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.
Enable interrupts after bus is unlocked.

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 bit) XOR *isrc2*
CC set if result is zero, cleared otherwise

Table 7.2 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

7.2 Instruction Format and Encoding

All instructions are one word long and begin on a word boundary. When operands are registers, the register encodings shown in Table 7.3 are used. There are two general core-instruction formats, REG-format and CTRL-format, as well as a separate format for floating-point instructions.

7.2.1 REG-FORMAT INSTRUCTIONS

Within the REG-format are several variations as shown in Figure 7.1. Table 7.4 gives the encodings for these instructions. One encoding is an escape code that defines yet another variation: the core escape instructions. Figure 7.2 shows the format of this group, and Table 7.5 shows the encodings.

In these instructions, the *src2* field selects one of the 32 integer registers (most instructions) or five 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.

Table 7.3 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 Process Status	5

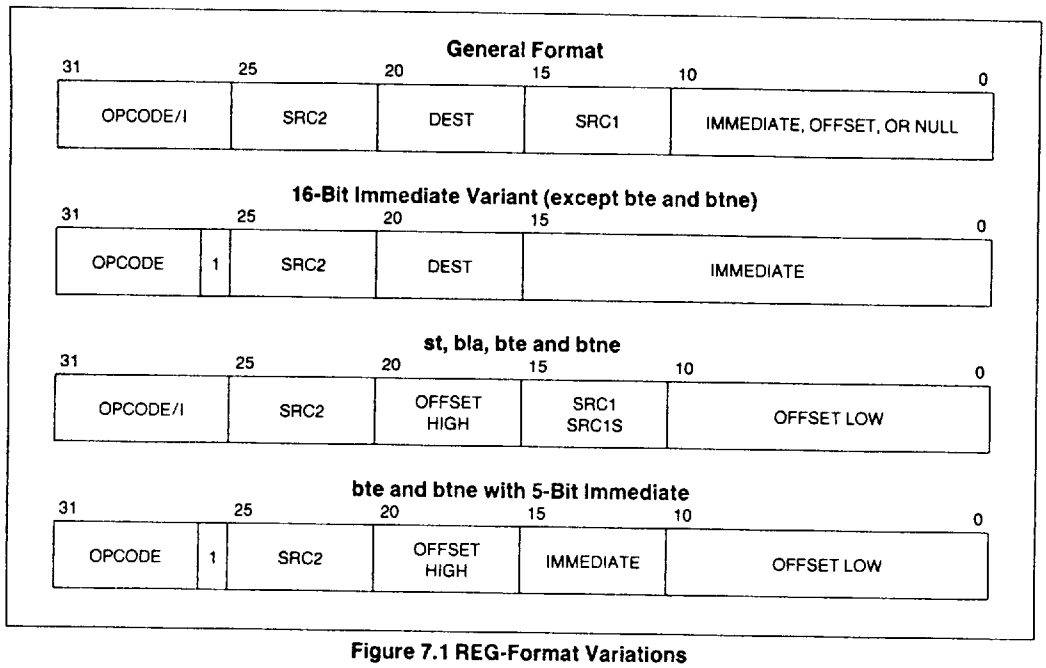


Figure 7.1 REG-Format Variations

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 an 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. 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

When *src1* is an immediate value, 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 **flush**, bits one and two must be zero.

Table 7.4 REG-Format Opcodes

		31					26
ld.x	Load Integer	0	0	0	L	0	I
st.x	Store Integer	0	0	0	L	1	I
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 Not Equal
- 1 —Branch on Equal

I Immediate
- 0 —*src1* is register
- 1 —*src1* is immediate

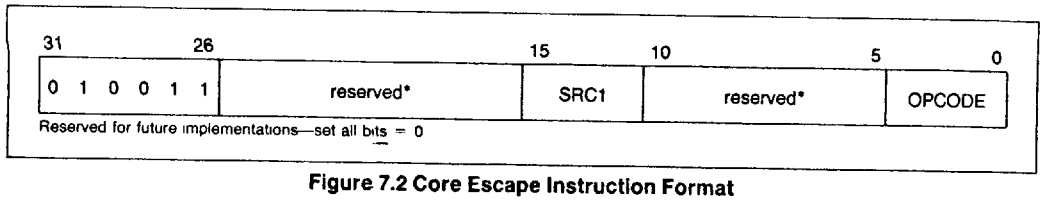


Figure 7.2 Core Escape Instruction Format

Table 7.5 Core Escape Opcodes

		4				0
	(reserved)	0	0	0	0	0
lock	Begin Interlocked Sequence	0	0	0	0	1
calli	Indirect Subroutine Call	0	0	0	1	0
	(reserved)	0	0	0	1	1
intovr	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
	(reserved)	0	1	x	x	x
	(reserved)	1	0	x	x	x
	(reserved)	1	1	x	x	x

7.2.2 CTRL-FORMAT INSTRUCTIONS

The CTRL instructions do not refer to registers, so instead of the register fields, they have a 26-bit relative branch offset. Figure 7.3 shows the format of these instructions and Table 7.6 defines the encodings.

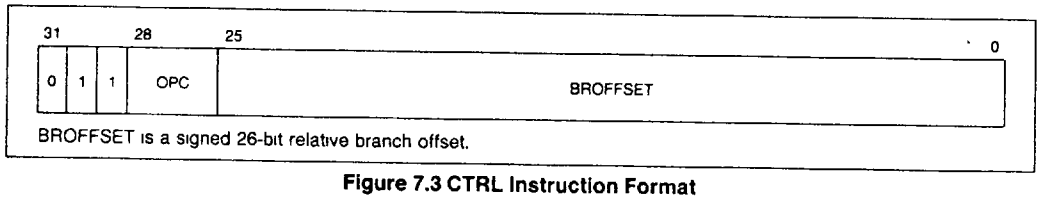


Figure 7.3 CTRL Instruction Format

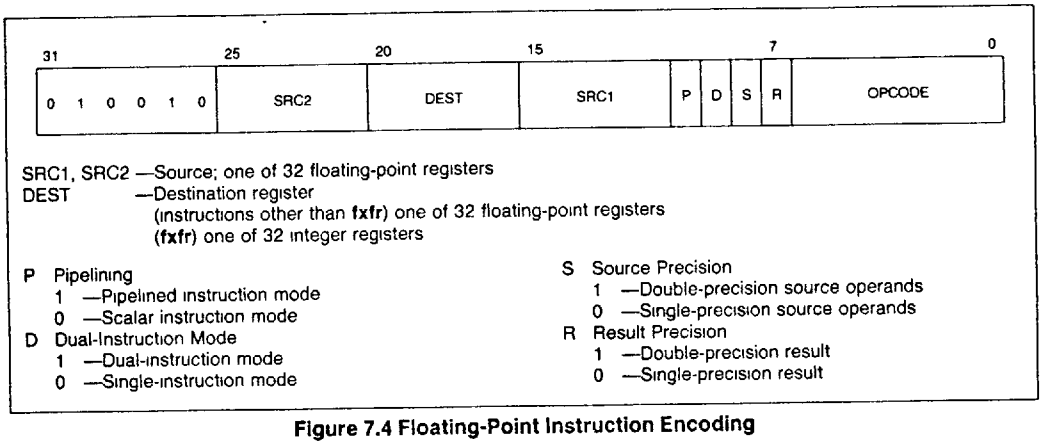
Table 7.6 CTRL-Format Opcodes

		28		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**

7.2.3 FLOATING-POINT INSTRUCTIONS

The floating-point instructions also constitute an escape series. All these instructions begin with the bit sequence 010010. Figure 7.4 shows the format of the floating point instructions, and Table 7.7 gives the encodings. Within the dual-operation instructions is a subcode DPC whose values are given in Table 7.8 along with the mnemonic that corresponds to each.



SRC1, SRC2 —Source; one of 32 floating-point registers
DEST —Destination register
(instructions other than **fxfr**) one of 32 floating-point registers
(**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 7.4 Floating-Point Instruction Encoding

Table 7.7 Floating-Point Opcodes

		6						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

*pfam and pfsm have P-bit set; pfmam and pfmsm have P-bit clear.
pfgt has R bit cleared, **pfle has R bit set.

NOTE:
All opcodes not shown are **reserved**

The following table shows the opcode mnemonics that generate the various encodings of DPC and explains each encoding.

Table 7.8 DPC Encoding

DPC	PFAM Mnemonic	PFSM Mnemonic	M-Unit op1	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	ra1s2	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	A 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							

*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.

7.3 Instruction Timings

Military i860 XR microprocessor instructions take one clock to execute unless a freeze condition is invoked. Freeze conditions and their associated delays are shown in the table below. Freezes due to multiple simultaneous cache misses result in a delay that is the sum of the delays for processing each miss by itself. Other multiple freeze conditions usually add only the delay of the longest individual freeze.

Freeze Condition	Delay
Instruction-cache miss.	Number of clocks to read instruction (from $\overline{ADS}$ clock to first $\overline{READY}$ clock) plus time to last $\overline{READY}$ of block when jump or freeze occurs during miss processing plus two clocks if data-cache being accessed when instruction-cache miss occurs.
Reference to destination of **ld** instruction that misses.	One plus number of clocks to read data (from $\overline{ADS}$ clock to first $\overline{READY}$ clock) minus number of instructions executed since load (not counting instruction that references load destination).
fld miss.	One plus number of clocks until first $\overline{READY}$ returned (for 32- or 64-bit read cycles) or until second $\overline{READY}$ returned (for 128-bit **fld.q** read cycles).
call/calli/ixfr/fxfr/ld.c/st.c and data cache miss processing in progress.	One plus number of clocks until first $\overline{READY}$ returned (for 64-bit read cycles) or until second $\overline{READY}$ returned (for 128-bit **fld.q** read cycles).
ld/st/pfld/fld/fst and data cache miss processing in progress.	One plus number of clocks until last $\overline{READY}$ returned.
Reference to *dest* of **ld, call, calli, fxfr** or **ld.c** in the next instruction. (*Dest* of **call** and **calli** is r1.)	One clock.

Freeze Condition	Delay
Reference to *dest* of **fld/pfld/ixfr** in the next two instructions	Two clocks in the first instruction; one in the second instruction
bc/bnc/bc.t/bnc.t following **fadd/fsub/pfeg/ pfgt**	One clock
Src1 of multiplier operation refers to result of previous operation	One clock
Floating-point or graphics unit instruction or **fst,** and scalar operation in progress other than **frcp** or **frsqr**	If the scalar operation is **fadd, fix, fmlow, fmul.ss, fmul.sd, ftrunc** or **fsub,** two minus the number of instructions (or dual-mode pairs) already executed after the scalar operation. If the scalar operation is **fmul.dd,** three minus the number of instructions (or dual-mode pairs) executed after it. Add one if either or both of these two 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 is different than the source precision of the floating-point operation. 2. The floating-point operation is pipelined and its destination is not **f0.** There is no delay if the result is negative.
Multiplier operation preceded by a double-precision multiply	One clock
Multiplier operation with data pattern requiring extra rounding operation	One clock
TLB miss	Five plus the number of clocks to finish two reads plus the number of clocks to set A-bits (if necessary)
pfld when three **pfld**'s are outstanding	One plus the number of clocks to return data from first **pfld**
pfld hits in the data cache	Two plus the number of clocks to finish all outstanding accesses
st, pst or **fst** miss **ld** miss, or **flush** with modified block when store path full (two stores or one 256-bit write-back internally waiting for bus plus external bus pipeline full)	One plus the number of clocks until $\overline{\text{READY}}$ active on next 64-bit write cycle or second $\overline{\text{READY}}$ of next 128-bit write cycle.
ld, fld, pfld, st, pst or **fst** when address path full (one address internally waiting for bus plus external bus pipeline full)	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).
ld/fld following **st/fst** hit	One clock
Delayed branch not taken	One clock

Freeze Condition	Delay
Nondelayed branch taken: **bc, bnc** **bte, btne**	One clock Two clocks
Indirect branch **bri** or call **calli**	One clock
st.c	Two clocks
Result of graphics-unit instruction (other than **fmov.dd**) used in next instruction when the next instruction is an adder- or multiplier-unit instruction	One clock
Result of graphics-unit instruction used in next instruction when the next instruction is a graphics-unit instruction	One clock
flush followed by **flush**	Three clocks minus the number of instructions between the two **flush** instructions. There is no delay if the result is negative.
fst or **pst** followed by pipelined floating-point operation that overwrites the register being stored	One clock

7.4 Instruction Characteristics

The following table lists some of the characteristics 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. *P* indicates that the instruction is pipelined.
- Whether the instruction is a delayed branch instruction. *D* marks the delayed branches.
- Whether the instruction changes the condition code CC. *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 Military i860 XR 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 *src1*, *src2* should not be the same as the *rdest* of the prior operation.
  8. When the prior operation is scalar, *fsrc1* and *fsrc2* of the current operation should not be the same as *fdest* 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 (except in the case of external interrupts), 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 the multiplier *op1*, KR, or KI, *fsrc1* must not be the same as *fdest*.
  e. If *fdest is not zero, fsrc1* and *fsrc2* must not be the same as *fdest*.
  f. *isrc1* must not be the same as *isrc2* for the autoincrementing form of this instruction.
  g. *isrc1* must not be the same as *isrc2*.

Table 7.9 Instruction Characteristics

Instruction	Execution Unit	Pipelined? Delayed?	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
brl	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.z	G				8	
fisub.z	G				8	
fix.p	A			SE, RE		
fld.y	E			DAT	2, 3	f
flush	E					
fmlow.p	M				4	
fmul.p	M			SE, RE	4	
form	G				8	
frep.p	M			SE, RE		
frsqr.p	M			SE, RE		
fst.y	E			DAT	5	f
fsub.p	A			SE, RE		
ftrunc.v	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	
or	E		CC			
orh	E		CC			
pfadd.p	A	P		SE, RE		
pfaddp	G	P			8	e

Instruction	Execution Unit	Pipelined? Delayed?	Sets CC?	Faults	Performance Notes	Programming Restrictions
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.z	G	P			8	e
pfisub.z	G	P			8	e
pfix.p	A	P		SE, RE		
pfld.z	E	P		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.v	A	P		SE, RE		
pfzchkl	G	P			8	
pfzchks	G	P			8	
pst.d	E			DAT		f
shl	E					
shr	E					
shra	E					
shrd	E					
st.c	E					
st.x	E			DAT		
subs	E		CC		1	
subu	E		CC		1	
trap	E			IT		
xor	E		CC			
xorh	E		CC			

intel

0

DOMESTIC SERVICE OFFICES

ALABAMA

*Intel Corp.
5015 Bradford Dr., Suite 2
Huntsville 35805
Tel: (205) 830-4010

ALASKA

Intel Corp.
c/o TransAlaska Data Systems
300 Old Steese Hwy.
Fairbanks 99701-3120
Tel: (907) 452-4401

Intel Corp.
c/o TransAlaska Data Systems
1551 Lore Road
Anchorage 99507
Tel: (907) 522-1776

ARIZONA

*Intel Corp.
11225 N. 28th Dr.
Suite D-214
Phoenix 85029
Tel: (602) 869-4980

*Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

CALIFORNIA

†Intel Corp.
21515 Vanowen St., Ste. 116
Canoga Park 91303
Tel: (818) 704-8500

*Intel Corp.
2250 E. Imperial Hwy., Ste. 218
El Segundo 90245
Tel: (213) 640-6040

*Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 351-6143
1-800-468-3548

Intel Corp.
9665 Cheasapeake Dr., Suite 325
San Diego 92123-1326
Tel: (619) 292-8086

**Intel Corp.
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642

**†Intel Corp.
San Tomas 4
2700 San Tomas Exp., 2nd Floor
Santa Clara 95051
Tel: (408) 986-8086

COLORADO

*Intel Corp.
650 S. Cherry St., Suite 915
Denver 80222
Tel: (303) 321-8086

CONNECTICUT

*Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130

FLORIDA

**Intel Corp.
6363 N.W. 6th Way, Ste. 100
Ft. Lauderdale 33309
Tel: (305) 771-0600

*Intel Corp.
5850 T.G. Lee Blvd., Ste. 340
Orlando 32822
Tel: (407) 240-8000

GEORGIA

*Intel Corp.
3280 Pointe Pkwy., Ste. 200
Norcross 30092
Tel: (404) 449-0541

HAWAII

*Intel Corp.
U.S.I.S.C. Signal Batt.
Building T-1521
Shafter Flats
Shafter 96856

ILLINOIS

**†Intel Corp.
300 N. Martingale Rd., Ste. 400
Schaumburg 60173
Tel: (312) 605-8031

INDIANA

*Intel Corp.
8777 Purdue Rd., Ste. 125
Indianapolis 46268
Tel: (317) 875-0623

KANSAS

*Intel Corp.
10985 Cody, Suite 140
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

**†Intel Corp.
10010 Junction Dr., Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: 301-206-3677

MASSACHUSETTS

**†Intel Corp.
3 Carlisle Rd., 2nd Floor
Westford 01886
Tel: (508) 692-1060

MICHIGAN

*†Intel Corp.
7071 Orchard Lake Rd., Ste. 100
West Bloomfield 48322
Tel: (313) 851-8905

MINNESOTA

*†Intel Corp.
3500 W. 80th St., Suite 360
Bloomington 55431
Tel: (612) 835-6722

MISSOURI

*Intel Corp.
4203 Earth City Exp., Ste. 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

**Intel Corp.
300 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0821

*Intel Corp.
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

*Intel Corp.
280 Corporate Center
75 Livingston Ave., 1st Floor
Roseland 07068
Tel: (201) 740-0111

NEW YORK

*†Intel Corp.
2950 Expressway Dr. South
Islandia 11722
Tel: (516) 231-3300

*Intel Corp.
Westage Business Center
Bldg. 300, Route 9
Fishkill 12524
Tel: (914) 897-3860

NORTH CAROLINA

*Intel Corp.
5800 Executive Dr., Ste. 105
Charlotte 28212
Tel: (704) 568-8966

**Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

**†Intel Corp.
3401 Park Center Dr., Ste. 220
Dayton 45414
Tel: (513) 890-5350

*†Intel Corp.
25700 Science Park Dr., Ste. 100
Beachwood 44122
Tel: (216) 464-2736

OREGON

Intel Corp.
15254 N.W. Greenbrier Parkway
Building B
Beaverton 97006
Tel: (503) 645-8051

*Intel Corp.
5200 N.E. Elam Young Parkway
Hillsboro 97123
Tel: (503) 681-8080

PENNSYLVANIA

*†Intel Corp.
455 Pennsylvania Ave., Ste. 230
Fort Washington 19034
Tel: (215) 641-1000

†Intel Corp.
400 Penn Center Blvd., Ste. 610
Pittsburgh 15235
Tel: (412) 823-4970

Intel Corp.
1513 Cedar Cliff Dr.
Camp Hill 17011
Tel: (717) 761-0860

PUERTO RICO

Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

Intel Corp.
8815 Dyer St., Suite 225
El Paso 79904
Tel: (915) 751-0186

*Intel Corp.
313 E. Anderson Lane, Suite 314
Austin 78752
Tel: (512) 454-3628

**†Intel Corp.
12000 Ford Rd., Suite 401
Dallas 75234
Tel: (214) 241-8087

*Intel Corp.
7322 S.W. Freeway, Ste. 1490
Houston 77074
Tel: (713) 988-8086

UTAH

Intel Corp.
428 East 6400 South, Ste. 104
Murray 84107
Tel: (801) 263-8051

VIRGINIA

*Intel Corp.
1504 Santa Rosa Rd., Ste. 108
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

*Intel Corp.
155 108th Avenue N.E., Ste. 386
Bellevue 98004
Tel: (206) 453-8086

CANADA

ONTARIO

Intel Semiconductor of
Canada, Ltd.
2650 Queensview Dr., Ste. 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: 613-820-5936

Intel Semiconductor of
Canada, Ltd.
190 Attwell Dr., Ste. 102
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: 416-675-2438

CUSTOMER TRAINING CENTERS

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1700
1-800-421-0386

ILLINOIS

300 N. Martingale Road
Suite 300
Schaumburg 60173
Tel: (708) 706-5700
1-800-421-0386

MASSACHUSETTS

3 Carlisle Road, First Floor
Westford 01886
Tel: (301) 220-3380
1-800-328-0386

MARYLAND

10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
1-800-328-0386

SYSTEMS ENGINEERING MANAGERS OFFICES

MINNESOTA

3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722

NEW YORK

2950 Expressway Dr., South
Islandia 11722
Tel: (506) 231-3300

†System Engineering locations
*Carry-in locations
**Carry-in/mail-in locations

CG/SALE/101789

D-10