Document Revision 2.2
Date: 03/2001

pT-110™ Processor Core Datasheet

1 Introduction

The pT-110™ core is a 32-bit RISC processor optimized for designs that demand high performance, minimal chip area, and low power, such as PDAs, cellular phones, and system-on-chip (SOC) applications. The core is provided with design files and software tools that enable easy integration into popular chip-design environments. The pT-110 processor is designed for use with applications that require ARM® (version 4T) instructions.

The pT-110 processor provides a fully programmable cache design with sizes range from 1 Kbyte to 64 Kbytes in 2x increments. Both instruction and data caches are direct-mapped and support locking on a per-entry basis. The cache line size is 16-bytes, requiring four sequential memory transfers on a cache line fill operation.

The pT-110 cache allows support for up to 8 separate memory regions or 'pages'. Each region can contain specific cacheability attributes for the instruction cache, data cache, and write buffer. In addition, multiple levels of access permissions can be attached to each memory region. Memory prefetching is supported on a per-instruction basis.

In addition to the primary caches, the pT-110 also contains separate instruction and data scratch pad caches that are programmable between 256 and 2 Kbytes. These caches are used to store time-sensitive code or data segments that require a guaranteed amount of bandwidth when executing.

2 Features

- 32-bit RISC architecture
- 300 MHz clock rate (0.18 micron process)
- 5-stage pipeline
- Programmable cache sizes from 1 Kbyte to 64 Kbytes
- Direct mapped instruction and data caches
- Instruction and data cache locking on a per-entry basis
- Data cache writeback/writethrough on a per-entry basis
- Support for up to 8 separate memory regions
- Memory access protection for each region
- 16-byte fixed line size for both caches
- Support for both Supervisor mode and User mode on cache accesses
- 4-deep write buffer for data cache writebacks
- Memory prefetching on a per-instruction basis
- Memory prefetch and lock in one operation
- Cache flush on single lines or the entire cache
- Cache 'clean' mechanism writes all dirty cache lines to memory in one operation
- Burst transfers on cache line fill operations
- Built-In Self Test (BIST) for both caches
- Instruction and Data Scratch Pad caches programmable from 256 bytes to 2 Kbytes

3 Document Organization

This document is organized into the following main sections.

Section	Name	Description	Page Number
1	Introduction	Provides a brief overview and market positioning for the pT-110 processor.	page 1
2	Features	Provides a list of features provided by the pT-110 processor.	page 1
3	Document Organization	Provides an overview of how this datasheet is organized.	page 2
4	Block Diagram	Provides a block diagram of the pT-110 processor and an overview of each block.	page 3
5	Modes of Operation	Discusses the six operating modes of the pT-110 processor.	page 7
6	Pipeline	Describes the 5 stages of the pT-110 processor pipeline.	page 10
7	Memory Management	Discusses the address regions and programmability of the address space.	page 12
8	Primary Caches	Discusses the features and programmability of the primary instruction and data caches.	page 16
9	Scratch Pad Caches	Discusses the features and programmability of the instruction and data scratch pad caches.	page 22
10	Register Set	Provides a list of General Purpose (GP) registers and CP15 Control registers.	page 26
11	Exception Processing	Defines the types of exceptions serviced by the pT-110 processor.	page 29
12	Signal Descriptions	Lists the external signals of the pT-110 processor along with a definition of each signal.	page 32
13	Bus Interface Unit	Describes the types of transactions supported on the external bus.	page 35
14	Instruction Set Overview	Provides a list of all 32-bit and 16-bit instructions executed by the pT-110 processor.	page 39

4 Block Diagram

Figure 1 shows a block diagram of the pT-110 core. The following subsections describe each block in the diagram. For more information on the connectivity of the MMU, refer to Figure 6.

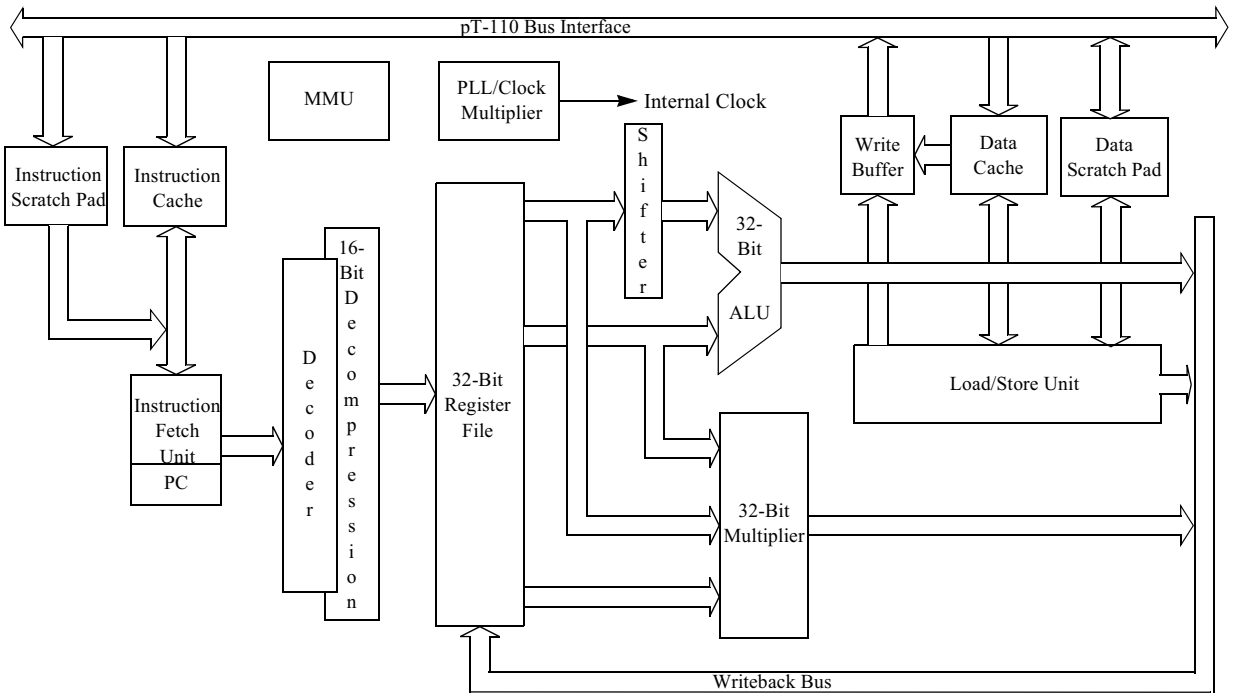

Figure 1. pT-110 Block Diagram

4.1 Instruction Fetch Unit

The *Instruction Fetch Unit* fetches one instruction per clock cycle and contains a 3-deep FIFO for instruction storage during pipeline stalls. If the pipeline is stalled, instruction fetching continues until the FIFO becomes full, at which time fetching stops.

The program counter (PC) is used to increment the address values used to access memory. In 32-bit mode, the counter increments by 4 each time an instruction is fetched. In 16-bit mode, the counter increments by 2 each time an instruction is fetched. All instruction fetching is performed on the physical address value, eliminating the need for internal virtual to physical address translation.

4.2 Primary Instruction Cache

The pT-110 primary cache design includes a fully programmable direct-mapped *Instruction Cache* that ranges between 1 Kbyte and 64 Kbytes in size. In general, application requirements, available die area, and price/performance constraints are some of the factors that determine the size of the cache. As a point of reference, in a 4 Kbyte instruction / 4 Kbyte data pT-110 cache implementation, the caches effectively double the size of the die.

The pT-110 primary instruction cache contains a fixed line size of 128 bits and incorporates a 32-bit wide data SRAM and a Tag SRAM. Refer to Section 8, "Caches" for more information.

4.3 Instruction Scratch Pad Cache

The pT-110 scratch pad cache design includes a programmable direct-mapped *Instruction Scratch Pad Cache* that ranges between 256 bytes and 2 Kbytes in size. This cache can be used to store code segments which must execute in a guaranteed amount of time and which require a guaranteed amount of bandwidth. The cache is typically filled on power up and cannot be written to by the processor core. The cache is flushed whenever the primary instruction cache is flushed. This cache can be used to store interrupt service routines (ISR), or any other time-sensitive code segment. Refer to Section 9, "Scratch Pad Caches" for more information.

4.4 Decoder

The *Decoder* is used to decode instructions prior to being written to the register file. The instructions are decoded and the appropriate signals are sent to the core indicating the type of operation to be performed. Note that 32-bit instructions bypass the *Decompression* block.

4.5 16-bit Decompression

The 16-bit *Decompression* block is used when the processor is operating in 16-bit THUMB® mode. Before being written to the register file, 16-bit instructions are decoded and are decompressed to 32 bits using the Decompression block.

4.6 32-bit Register File

The *32-bit General Purpose (GP) Register File* stores the operands and results of a computation. The pT-110 accesses the GP register file in one of 6 operating modes: User, FIQ, IRQ, Supervisor, Abort, and Undefined Instruction. The following GP registers are contained in each mode. A complete list of GP registers is shown in Table 16.

- User Mode - In user mode the following registers can be accessed: R0 through R14, PC, and the Current Processor Status Register (CPSR). R0 through R14 are temporary storage registers, the PC register contains the program counter value, and the CPSR contains the current processor status.
- FIQ Mode - FIQ mode shares registers R0 through R7 with User mode (and all other operating modes). In addition, this mode contains a separate set of 7 dedicated registers (R8_FIQ through R14_FIQ) that can be accessed only in FIQ mode. R8_FIQ through R12_FIQ are used for to accelerate interrupt processing by providing dedicated registers that help to eliminate push and pop operations during interrupt processing. Register R13_FIQ contains the stack pointer, and register R14_FIQ is the Link register. FIQ mode also contains a dedicated Saved Processor Status Register (SPSR), called SPSR_FIQ, that contains the saved contents of the processor status register.
- IRQ Mode - IRQ mode shares registers R0 through R12 with all other operating modes (except FIQ mode where it shares only registers R0 through R7). In addition, this mode contains a separate set of 2 dedicated registers (R13_IRQ and R14_IRQ) that can be accessed only in IRQ mode. Register R13_IRQ contains the stack pointer, and register R14_IRQ is the Link register. IRQ mode also contains a dedicated SPSR register, called SPSR_IRQ, that contains the saved contents of the processor status register.
- Supervisor Mode - Supervisor mode shares registers R0 through R12 with all other operating modes (except FIQ mode where it shares only registers R0 through R7). In addition, this mode contains a separate set of 2 dedicated registers (R13_SVC and R14_SVC) that can be accessed only in Supervisor mode. Register R13_SVC contains the stack pointer, and register R14_SVC is the Link register. Supervisor mode also contains a dedicated SPSR register, called SPSR_SVC, that contains the saved contents of the processor status register.
- Abort Mode - Abort mode shares registers R0 through R12 with all other operating modes (except FIQ mode where it shares only registers R0 through R7). In addition, this mode contains a separate set of 2 dedicated registers (R13_ABORT and R14_ABORT) that can be accessed only in Abort mode. Register R13_ABORT contains the stack pointer, and register R14_ABORT is the Link register. Abort mode also contains a dedicated SPSR register, called SPSR_ABORT, that contains the saved contents of the processor status register.

- Undefined Instruction Mode - Undefined Instruction mode shares registers R0 through R12 with all other operating modes (except FIQ mode where it shares only registers R0 through R7). In addition, this mode contains a separate set of 2 dedicated registers (R13_UND and R14_UND) that can be accessed only in Abort mode. Register R13_UND contains the stack pointer, and register R14_UND is the Link register. Undefined Instruction mode also contains a dedicated SPSR register, called SPSR_UND, that contains the saved contents of the processor status register.

Refer to Section 10, "Register Set" and Section 5, "Modes of Operation" for more information.

4.7 Shifter

The Shifter performs logical and arithmetic shifting based on the type of instruction being executed. The pT-110 instruction set incorporates certain shift operations into the instructions and hence does not require a separate shift operation to be performed.

4.8 Arithmetic Logic Unit (ALU)

The Arithmetic Logic Unit accepts two operands and associated control signals, one from the register file and one from the shifter. The ALU processes all operations except multiply. These include move, load/store, data processing, and coprocessor operations.

4.9 32-bit Multiplier

The pT-110 processor contains a 32-bit multiplier that performs signed and unsigned multiply and multiply-accumulate operations. The multiplier requires two cycles to perform a multiply-accumulate operation. In the first cycle three operands are provided to the multiplier and the actual multiply operation is performed. In the second cycle the 4th operand is provided and the accumulate operation is performed. The multiplier requires two operands to perform the multiply, and another two operands to perform the accumulate.

The pT-110 processor always generates a 64-bit value on a multiply or multiply-accumulate operation. The lower 32-bits of the result are stored to a GP register whose location is defined in the instruction. The upper 32-bits are stored to the CP15 RdHi register (CP15-11) using the *Move to Coprocessor from Register* (MCR) instruction. CP15-11 is register 11 in the CP15 Control register set. Unlike a multiply long or multiply-accumulate long operation, generating a 64-bit result on multiply and multiply-accumulate operations allows the upper and lower halves of the 64-bit result to be written to nonsequential registers.

For a long multiply or multiply-accumulate operation, the 64-bit result is stored to two sequential General Purpose (GP) registers whose locations are defined in the instruction.

The pT-110 multiplier executes the following operations:

- Multiply: The multiply instruction (MUL) multiplies two signed or unsigned variables to produce a 64-bit result. The lower 32-bits of the result are stored to a GP register whose location is defined in the instruction. The upper 32-bits are stored to the CP15 RdHi register (CP15-11) using the MCR instruction. This allows the 64-bit result to be stored to two nonsequential registers. This instruction is only executed if the condition specified in bits 31:28 of the instruction matches the condition code status.
- Multiply-accumulate: The multiply-accumulate instruction (MLA) multiplies two signed or unsigned operands to produce a 64-bit result, which is added to a third operand and written to the destination register. The lower 32-bits of the result are stored to a GP register whose location is defined in the instruction. The upper 32-bits are stored to the CP15 RdHi register (CP15-11) using the MCR instruction. This allows the 64-bit result to be stored to two nonsequential registers. This instruction is only executed if the condition specified in bits 31:28 of the instruction matches the condition code status.

- Signed multiply long: The signed multiply long instruction (SMULL) multiplies two signed variables to produce a 64-bit result. The result is written to two sequential GP registers defined in the instruction. This instruction is only executed if the condition specified in bits 31:28 of the instruction matches the condition code status.
- Signed multiply-accumulate long: The signed multiply-accumulate long instruction (SMLAL) multiplies two signed variables to produce a 64-bit result, which is then added to another 64-bit value stored in two sequential destination GP registers. This instruction is only executed if the condition specified in bits 31:28 of the instruction matches the condition code status.
- Unsigned multiply long: The unsigned multiply long instruction (UMULL) multiplies two unsigned variables to produce a 64-bit result. The result is written to two sequential GP registers defined in the instruction. This instruction is only executed if the condition specified in bits 31:28 of the instruction matches the condition code status.
- Unsigned multiply-accumulate long: The unsigned multiply-accumulate long instruction (UMLAL) multiplies two unsigned variables to produce a 64-bit result, which is then added to another 64-bit value stored in two sequential destination GP registers. This instruction is only executed if the condition specified in bits 31:28 of the instruction matches the condition code status.

4.10 Data Cache

The pT-110 primary cache design includes a fully programmable direct-mapped data cache that ranges between 1 Kbyte and 64 Kbytes in size. In general, application requirements, available die area, and price/performance constraints are some of the factors that determine the size of the cache. As a point of reference, in a 4 Kbyte instruction / 4 Kbyte data pT-110 cache implementation, the caches effectively double the size of the die.

The pT-110 primary data cache contains a fixed line size of 128 bits and incorporates a 32-bit wide data SRAM, a Tag SRAM, and a 1-bit wide Dirty Bit RAM. Refer to Section 8, "Caches" for more information.

4.11 Data Scratch Pad Cache

The pT-110 scratch pad cache design includes a fully programmable direct-mapped *Data Scratch Pad Cache* that ranges between 256 bytes and 2 Kbytes in size. This cache can be used to store any time-sensitive operation where the data must be fetched and executed in a guaranteed amount of time and which requires a guaranteed amount of bandwidth. The cache is typically filled on power up and can be written to by the processor core. The cache is flushed whenever the primary data cache is flushed. Refer to Section 9, "Scratch Pad Caches" for more information.

4.12 Write Buffer

The pT-110 provides a 4-entry write buffer to maximize memory bus bandwidth. This buffer is used by the data cache to store modified lines to be written out to memory, and by the core to store non-cacheable data that has access to the write buffer. Entries in the cache that are marked as ‘writeback’ are written to the write buffer instead of directly to memory. Writethrough pages are written directly to memory and do not use the write buffer.

Refer to Section 8.2, "Write Buffer" for more information.

4.13 Load Store Unit

The pT-110 processor contains a load/store unit that controls the loading and storing of data between the data cache and the write buffer. During a data cache access, the address is generated by the Load/Store Unit of the core and driven to the cache.

The address and data paths between the load/store unit and the write buffer are also used for non-cacheable stores that have access to the buffer.

4.14 Memory Management

The Memory Management Unit (MMU) provides an interface between the pT-110 core and the caches. The MMU accepts only MCR and MRC instructions from the core. Although the pT-110 processor executes the CDP, LDC, and STC instructions, no coprocessor is implemented that will respond to them. Therefore, execution of these three instruction results in an Undefined Instruction exception.

The MMU decodes the instruction and manipulates the cache accordingly. This includes updating the contents of the caches, flushing the caches, and locking certain lines within either cache.

Refer to Section 7, "Memory Management" for more information.

4.15 PLL/Clock Multiplier

The clock multiplier multiplies the input reference clock by a value of 2 to 16. The RCLK input clock to the pT-110 processor is multiplied within the PLL to derive the CPU clock. The input clock can be multiplied by the following ratios: 2, 4, 6, 8, 10, 12, 14, and 16.

Figure 2 shows a block diagram of the clock generator.

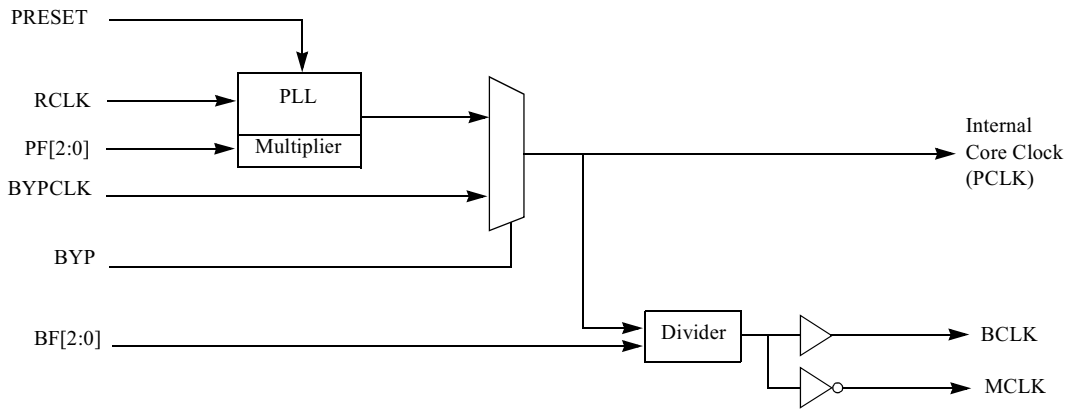

Figure 2. pT-110 Clock Generator

In Figure 2, the RCLK input is multiplied by the ratio determined by the PF[2:0] input pins to produce the internal PCLK used by the core. A separate clock divider is used to divide the PCLK signal by the ratio determined by the BF[2:0] pins to produce the clock outputs MCLK and BCLK used to drive the bus.

Because of skew between RCLK and the bus clock outputs, all devices on the system bus must synchronize to either MCLK or BCLK when the the PLL is used as the clock source.

Alternatively, the BYP input can be asserted to bypass the PLL. In this case, an external clock BYPCLK is used. If the PLL is bypassed, the frequency of BYPCLK is used to drive PCLK directly.

5 Modes of Operation

The pT-110 processor contains seven operating modes controlled by the M[4:0] field in the Current Processor Status Register (CPSR), as shown in Table 1.

Table 1. Processor Modes

M[4:0]	Name	Operating Mode	Description
10000	User	User	Application program
10001	FIQ	Privileged	Fast interrupt request handler
10010	IRQ	Privileged	Normal interrupt request handler
10011	Supervisor	Privileged	Operating system
10111	Abort	Privileged	Memory manager
11011	Undefined Instruction	Privileged	Emulator for instruction set extensions
11111	User	User	Device drivers and other tasks

There are two basic types of operating modes as shown in Table 1, User and Privileged.

5.1 User Mode

User mode is where the application program and other user code such as device drivers resides. The processor operates in this mode during normal operation and only enters one of the privileged modes when an exception or interrupt occurs. User mode is selected when the M[4:0] field contains a value of 0b10000 or 0b11111.

5.2 Privileged Modes

There are five types of privileged modes in Table 1 above which are defined in the following subsections.

5.2.1 IRQ Mode

IRQ mode is a privileged mode that is entered when an external interrupt is generated on the IRQ pin. IRQ contains a dedicated Link register (R14_IRQ) that contains the return address, and a Save Processor Status Register (SPSR_IRQ) that contains the processor state at the time the interrupt was taken. Once the interrupt has been serviced, the contents of R14_IRQ are loaded into the program counter (PC), and the contents of SPSR_IRQ are loaded into the CPSR, allowing the program to resume execution in the mode specified in SPSR_IRQ.

5.2.2 FIQ Mode

FIQ mode is a privileged mode that allows for faster interrupt processing than IRQ mode by providing five additional dedicated general purpose registers (R8_FIQ through R12_FIQ) that the interrupt handler can use for temporary storage. FIQ mode is entered when an external interrupt is generated on the FIQ pin. Like IRQ mode, FIQ mode also contains a dedicated Link register (R14_FIQ) that contains the return address, and a Save Processor Status Register (SPSR_FIQ) that contains the processor state at the time the interrupt was taken. Once the interrupt has been serviced, the contents of R14_FIQ are loaded into the program counter (PC), and the contents of SPSR_FIQ are loaded into the CPSR, allowing the program to resume execution in User mode.

5.2.3 Supervisor Mode

Supervisor mode is a privileged mode entered through execution of the Software Interrupt (SWI) instruction. Certain memory spaces not available in User mode can be accessed while in Supervisor mode. In addition, many core maintenance functions are performed in Supervisor mode. Supervisor mode contains a dedicated Link register (R14_SVC) that contains the return address, and a Save Processor Status Register (SPSR_SVC) that contains the processor state at the time the interrupt was taken. Once the interrupt has been serviced, the contents of R14_SVC are loaded into the program counter (PC), and the contents of SPSR_SVC are loaded into the CPSR, allowing the program to resume execution in User mode.

5.2.4 Abort Mode

Abort mode is a privileged mode that is entered when the processor must abort an operation. There are four types of abort operations.

- External Instruction Abort
- External Data Abort
- Internal Instruction Abort
- Internal Data Abort

An external data or instruction abort is initiated when external logic asserts the ABORT pin to the pT-110. An instruction abort occurs when the processor attempts to fetch instruction from an invalid or restricted address. An instruction operation is indicated by the processor driving the nOPC pin low. A data abort occurs when the processor attempts to store or load instructions to or from an invalid or restricted address. A data operation is indicated by the processor driving the nOPC pin high.

The MMU can also perform an internal instruction or data abort by checking the address generated by the processor against its own access permissions.

Internal and external abort operations are logically OR'd within the pT-110 core to provide a single internal Abort signal as shown in Figure 3. The MMU compares the 32-bit address with the access permissions. If the does match the access permissions provided by the processor, the MMU generates an internal abort. The internal abort signal is then logically OR'd with the external ABORT pin. If either of these signals is asserted, an abort signal is sent to the core.

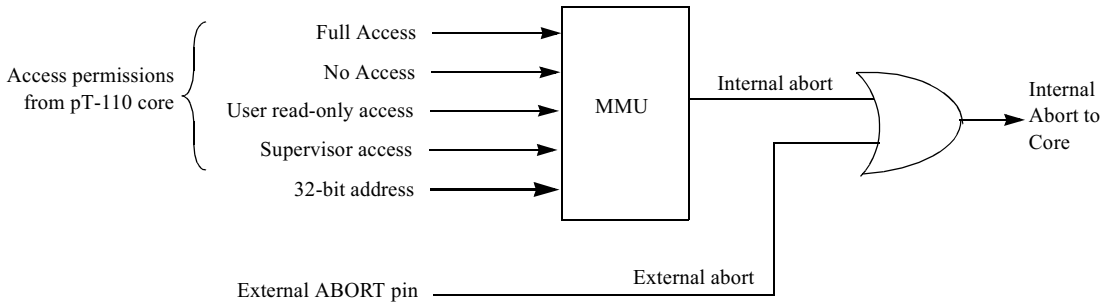

Figure 3. Logical OR of Internal and External Abort Signals

Abort mode contains a dedicated Link register (R14_ABORT) that contains the return address, and a Save Processor Status Register (SPSR_ABORT) that contains the processor state at the time the abort was taken. Once the abort request has been serviced, the contents of R14_ABORT are loaded into the program counter (PC), and the contents of SPSR_ABORT are loaded into the CPSR, allowing the program to resume execution in User mode.

5.2.5 Undefined Instruction Mode

Undefined Instruction mode is a privileged mode that is entered under either of the following two conditions:

- When no coprocessor responds to a coprocessor instruction generated by the processor.
- When bits 27:25 of the 32-bit instruction contain a value of 0b011, and bit 3 is 0b1, indicating an access to undefined instruction space.

Undefined Instruction mode contains a dedicated Link register (R14_UND) that contains the return address, and a Save Processor Status Register (SPSR_UND) that contains the processor state at the time the interrupt was taken. Once the interrupt has been serviced, the contents of R14_UND are loaded into the program counter (PC), and the contents of SPSR_UND are loaded into the CPSR, allowing the program to resume execution in User mode.

5.3 Context Switching Between Operating Modes

When switching from Supervisor mode to User mode, cleaning and flushing of the cache is recommended. This is because certain cache lines in Supervisor mode may not be available in User mode, thereby limiting the total number of available cache lines in User mode. Flushing the cache invalidates all lines in the cache, regardless of their dirty bit status.

When flushing the data cache, there may be lines in the cache that should be written out to memory before the flush operation is performed. In this case the Clean instruction can be executed. There are two ‘clean’ instructions, one of which locates all dirty lines in the data cache and writes them out to memory (the other Clean instruction writes only a single line out to memory and would not be used during a context switch). A flush operation can then be performed.

6 Pipeline

The pT-110 core is a high-performance, single-issue RISC architecture that implements a 5-stage pipeline:

- *Fetch Stage*—instruction prefetch.
- *Decode Stage*—instruction decode and read source registers from multiported register file.
- *Execute Stage*—generate memory read address and perform ALU/MAC operation.
- *Memory Stage*—read data input bus and ALU/MAC result.
- *Writeback Stage*—writeback to register file and load write buffer.

A block diagram of the pipeline is shown in Figure 4.

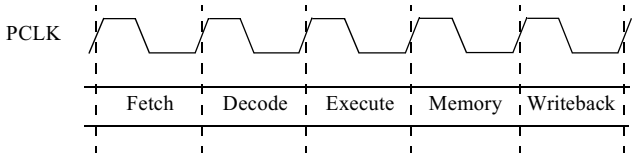

Figure 4. pT-110 Pipeline Block Diagram

6.1 Fetch Stage

During the Fetch stage the Instruction Fetch Unit retrieves the instruction from the Instruction Cache and passes it to the decoder.

6.2 Decode Stage

In the Decode stage 32-bit instructions are decoded and the appropriate internal signals are driven to indicate the type of operation to be performed. If the processor is operating in 16-bit mode, a 16-bit instruction is decompressed into a 32-bit instruction during this stage. The result of the operation is written to the register file.

6.3 Execute Stage

In the Execute stage the instruction operands are read from the register file and passed to the ALU or 32-bit multiplier depending on the type of operation. Most ALU operations require only one PCLK cycle to complete. A multiply

instruction occupies the Execute stage for three PCLK cycles, which stalls the next instruction, as shown in Figure 5. The latency for a multiply is constant, not data-dependent.

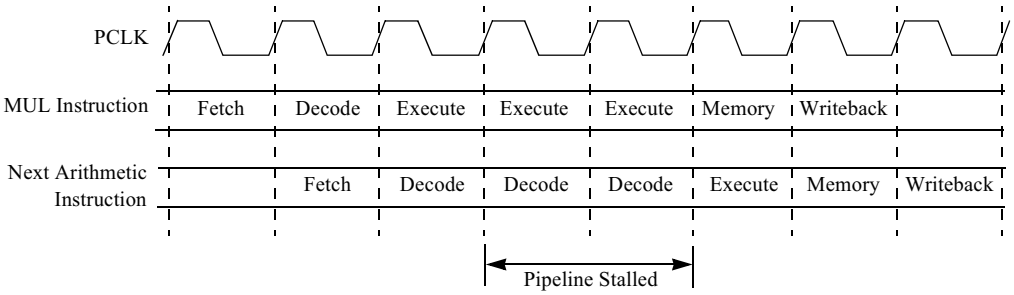

Figure 5. Pipeline Usage During a Multiply Instruction

In Figure 5, the next instruction stalls in the Decode stage for two clocks because the MUL instruction is using the Execute stage. Once the multiplication is complete and the MUL instruction propagates to the Memory stage, the next instruction can move to the Execute stage. This diagram assumes that the next instruction is an arithmetic instruction since only one Execute cycle is required.

6.4 Memory Stage

In the Memory stage the data cache is accessed and store data is written to the write buffer in write back mode. The address and data paths between the Load/Store Unit and the Write Buffer in Figure 6 below are used for non-cacheable stores that have access to the buffer.

6.5 Writeback Stage

During the Writeback stage data from the load/store unit, the ALU, or the 32-bit multiplier is written back to the register file.

6.6 Cycle Timings

Table 2 provides a summary of minimum cycle times for the following operations.

Table 2. pT-110 Cycle Times

Pipeline Operation	Number of PCLK Cycles
Multiply	3
Multiply-Accumulate	3
Load After Store	3
Store After Load	1
Back-to-Back Loads	1
Back-to-Back Stores	1

7 Memory Management

The memory management unit (MMU) provides an interface between the pT-110 core and the caches.

The MMU accepts only MRC and MCR instructions from the core. The MMU decodes the instruction and manipulates the cache accordingly. This includes updating the contents of the primary caches, flushing the caches, locking certain lines within either primary cache, etc. Figure 6 shows a block diagram of how the MMU connects to the other logic blocks in the pT-110 processor.

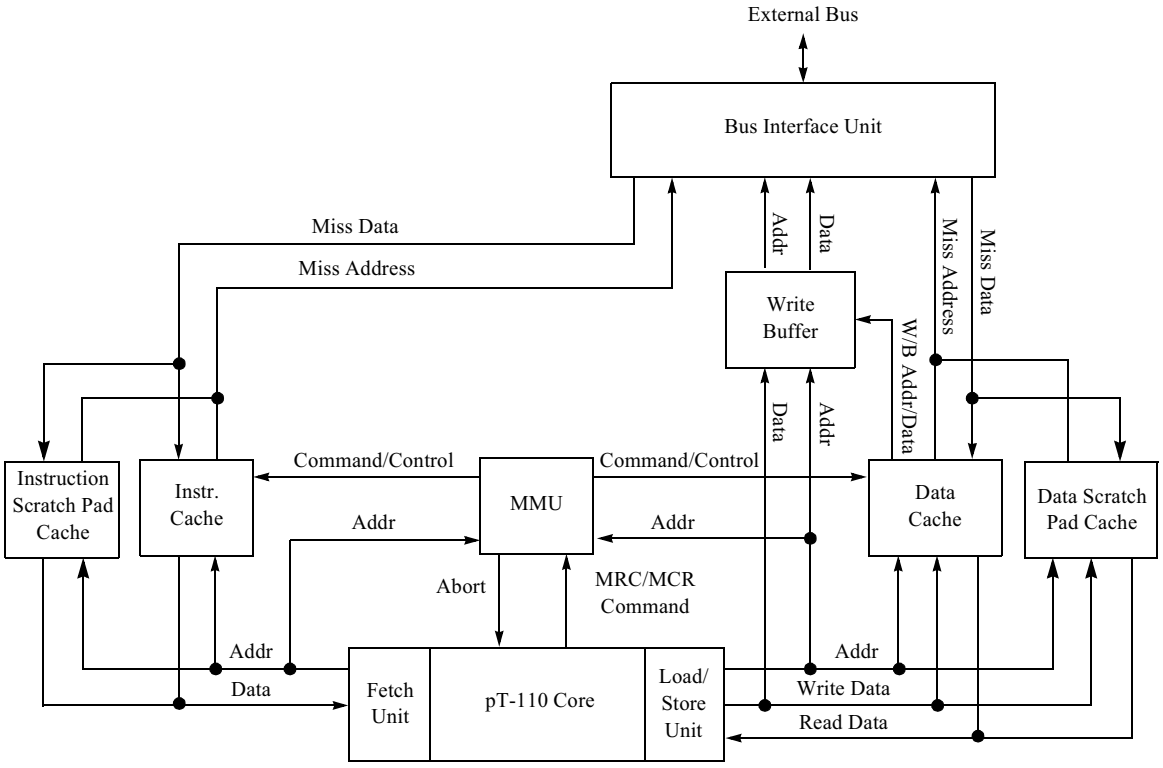

Figure 6. pT-110 MMU Interface Block Diagram

During an instruction cache access, the address is generated by the *Fetch Unit* of the core and driven to the cache. At the same time this address is driven to the *MMU*. The *MMU* checks access permissions for that address. If the address is not permitted access by the processor, the *MMU* asserts the *Abort* signal to the core, thereby aborting the access. If the requested address is located in the cache and is accessible (a cache 'hit'), the instruction cache provides the data to the core and the cycle completes. If the requested address is not located in the cache (a cache 'miss'), the 'miss' address is driven to the *Bus Interface Unit* and a memory access is initiated. If the address is not cacheable, then the MMU signals the cache not to perform a burst read.

During a data cache access, the address is generated by the *Load/Store Unit* of the core and driven to the cache. At the same time this address is driven to the *MMU* as well as the *Write Buffer*. The *MMU* checks access permissions for that address. If the address is not permitted access by the processor, the *MMU* asserts the *Abort* signal to the core, thereby aborting the access. If the requested address is located in the cache and is accessible (a cache 'hit'), the data cache provides the data to the core and the cycle completes. If the requested address is not located in the cache (a cache 'miss'), the 'miss' address is driven to the *Bus Interface Unit* and a memory access is initiated. If the address is not cacheable, then the MMU signals the cache not to perform a burst read. The address and data paths between the *Load/Store Unit* and the *Write Buffer* are used for non-cacheable stores that have access to the buffer.

When a line is modified in the data cache, the address and data for the dirty line are written to the *Write Buffer*. As soon as an entry in the buffer becomes valid, the write buffer requests a memory access and sends the address and data for that entry to the *Bus Interface Unit* for transfer to memory.

7.1 Memory Regions

The pT-110 provides up to 21 different memory region sizes covering a maximum 4 Gbyte address space. Region sizes range from 4 Kbyte to 4 Gbyte in 2x increments. Once the region size is selected, all regions are configured to the same size. The *Region Size* register controls the size of each memory region. Region sizes between 4 Kbytes and 512 Mbytes support 8 different memory regions. For region sizes less than 512 MBytes, the regions are aliased across the address space to fill the entire memory. A 1 Gbyte region size supports 4 different memory regions, a 2 Gbyte region size support 2 different memory regions, and a 4 Gbyte region size supports 1 memory region. Table 3 shows the encoding of the *Region Size* register fields and the corresponding memory region sizes.

Table 3. Region Size Field Encoding

Encoding	Region Size (bytes)
01011	4K
01100	8K
01101	16K
01110	32K
01111	64K
10000	128K
10001	256K
10010	512K
10011	1M
10100	2M
10101	4M
10110	8M
10111	16M
11000	32M
11001	64M
11010	128M
11011	256M
11100	512M
11101	1G
11110	2G
11111	4G

Figure 7 shows how the memory is divided for the commonly used 256 Mbyte, 512 Mbyte, and 1 Gbyte region sizes.

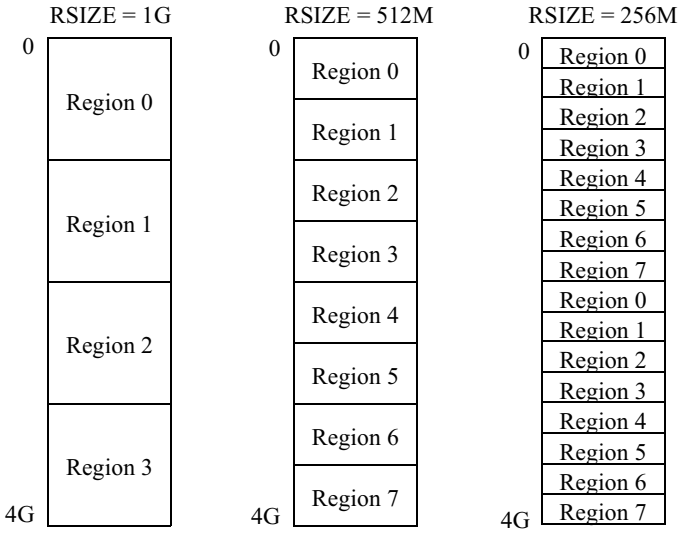

Figure 7. Region Size Configurations

Figure 7 above shows the number of regions available for three common region sizes. Note that for region sizes of 256 MBytes and smaller, the regions are repeated across the entire address space as shown.

The pT-110 provides three registers for configuring the memory regions and attaching access permission attributes to each one.

The *Region Size* register (CP15 - 6) contains two 5-bit fields for programming the region size. One 5-bit field is used for instruction cache memory, and the other 5-bit field is used for data cache memory. Each of these 5-bit fields stores one of the values in Table 3 above. Note that the value in each field must be the same. For example, a value of 0x1C indicates a 512 Mbyte region size. This value must appear in both 5-bit fields of the *Region Size* register.

The *Instruction Space Protection* register (CP15 - 4) attaches access permission attributes to each instruction memory region. These attributes range from ‘unrestricted access in any mode’ to ‘no access’. Refer to Table 4 for a complete list of access permissions.

The *Data Space Protection* register (CP15 - 5) attaches access permission attributes to each data memory region. These attributes range from ‘unrestricted access in any mode’ to ‘no access’. Refer to Table 5 for a complete list of access permissions.

7.2 Memory Region Cacheability

The address regions of the pT-110 processor can be marked as cacheable or noncacheable on a per-page basis. The *Configuration* (CP15 - 2) and *Cache Control* (CP15 - 3) registers are used to select the cacheability attributes for a particular region. In addition, each cacheable page can be marked as writeback or writethough using the *Write Buffer Control* register.

7.3 Setting Address Space Protections

Once the memory address regions have been determined, the address space protections can be set for both instruction and data accesses. In the instruction and data regions the pT-110 offers a number of different read/write accesses in Supervisor and User Mode. The pT-110 limits User mode access to any or all memory regions based on the programming of the *Instruction Space Protection* and *Data Space Protection* registers.

7.3.1 Instruction Space Protection

For instruction spaces, the *Instruction Space Protection* register (CP15 - 4) contains eight 2-bit encoded values (IPR0 - IPR7) that determine the level of access protection for each memory region. For example, if a region is marked ‘no access’, the fact that the same region may have been programmed as instruction-cacheable is ignored. The encoding of each 2-bit field is shown in Table 4 below.

Table 4. IPRx Field Encoding in the Instruction Space Protection Register

Encoding	Description
00	No access
01	Read access in any privileged mode (any mode except User mode)
10	Read/write access in any privileged mode, read-only access in User mode
11	Read/write access in any mode

Figure 8 shows how the access to the various memory regions is affected by programming a value if 0xF3FE into the Instruction Space Protection register.

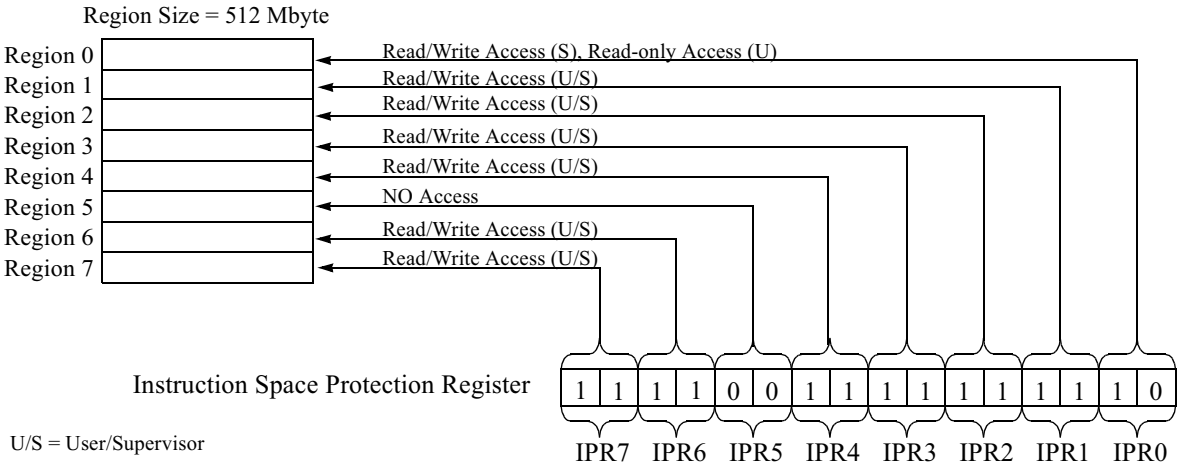

Figure 8. Setting Instruction Space Protections in Memory

7.3.2 Data Space Protection

For data spaces, the *Data Space Protection* register (register 5 in CP15) contains eight 2-bit encoded values (DPR0 - DPR7) that determine the level of access protection for each memory region. These access protections override any other cacheability attributes. For example, if a region is marked as ‘no access’, the fact that the same region may have been programmed as data-cacheable is ignored. The encoding of each 2-bit field is shown in Table 5 below.

Table 5. DPRx Field Encoding in the Data Space Protection Register

Encoding	Description
00	No access
01	Read access in any privileged mode (any mode except User mode)
10	Read/write access in any privileged mode, read-only access in User mode
11	Read/write access in any mode

Figure 9 shows how the access to the various memory regions is affected by programming a value of 0x13FE into the Instruction Space Protection register.

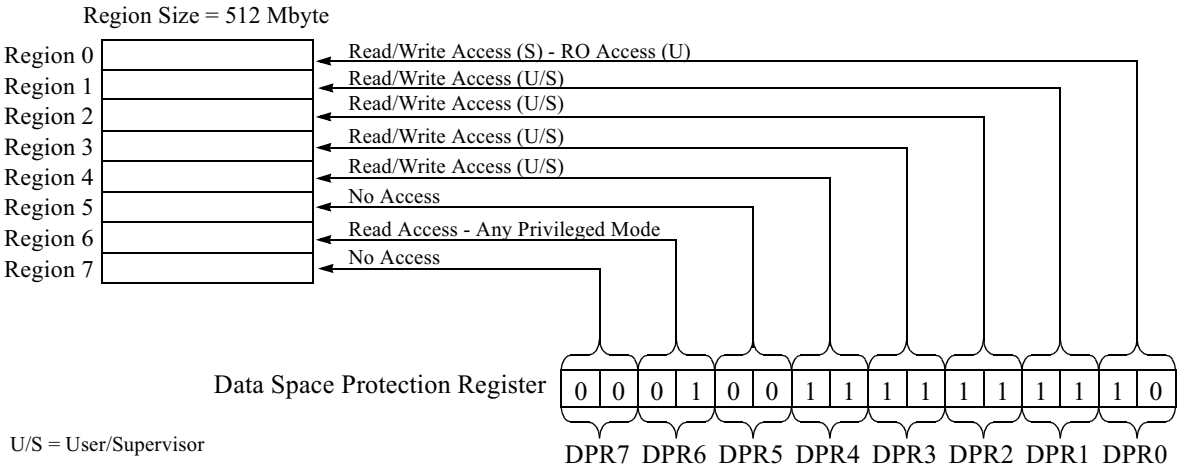

Figure 9. Setting Data Space Protections in Memory

8 Caches

The pT-110 cache design includes direct-mapped primary instruction and data caches that range between 1 Kbyte and 64 Kbytes in size. This section is divided into the following subsections.

- Section 8.1, "Primary Cache Organization"
- Section 8.2, "Write Buffer"
- Section 8.3, "Writeback and Writethrough"
- Section 8.4, "Cache Locking"
- Section 8.5, "Cache Cleaning and Flushing"
- Section 8.6, "Built-In Self Test"

For more information on the instruction and data scratch pad caches, refer to Section 9.

8.1 Primary Cache Organization

The picoTurbo pT-110 cache offers a flexible array of cache sizes and configurations. In general, application requirements, available die area, and price/performance constraints are some of the factors that determine the size of the cache. As a point of reference, in a 4 Kbyte instruction / 4 Kbyte data pT-110 cache implementation, the caches effectively double the size of the die. The pT-110 cache contains a fixed line size of 128 bits. Each cache contains a 32-bit wide data SRAM, and a Tag SRAM. The data cache also contains a 1-bit wide Dirty Bit RAM. Table 6 shows the configuration options for the pT-110 processor.

Table 6. pT-110 Primary Cache Size Options

Cache Size (Kbytes)	Tag Index Width (bits)	Tag Address Width (bits)	Total Tag Width (bits)	Index Width Offset	Tag Width Offset	Number of Tag RAM Entries	Number of Data RAM Entries	Number of Dirty Bit RAM Entries
1	6	22	24	9	10	64	256	64
2	7	21	23	10	11	128	512	128
4	8	20	22	11	12	256	1024	256
8	9	19	21	12	13	512	2048	512
16	10	18	20	13	14	1024	4096	1024
32	11	17	19	14	15	2048	8192	2048
64	12	16	18	15	16	4096	16384	4096

The entries in Table 6, from left to right, are defined as follows.

- *Cache Size:* This number represents the size of the instruction or data cache in kilobytes.
- *Tag Index Width*: This number represents the actual number of bits used to index the Tag RAM, not including the four low-order bits which are always zero. Since each tag RAM entry corresponds to four entries in the data RAM, the lower four bits of the index are zero. Bits 3:2 are used to select one of four words within a cache line, while bits 1:0 are used to select one of four bytes within a given word. This field is referred to as the *Tag RAM Index* field in Figure 10 below.
- *Tag Address Width*: This number represents the actual number of bits in the tag address and is referred to as the *Tag* field in Figure 10 below.
- *Total Tag Width*: Total number of bits in the tag address portion of the tag entry. This value is defined as the *Tag Address Width* + a *Valid* bit + a *Lock* bit (*Tag Address Width* + 2).
- *Index Width Offset*: Highest order bit of the *Tag Index Width* field.
- *Tag Width Offset*: Lowest order bit of the *Tag Address Width* field.
- *Number of Tag RAM Entries*: Number of tag RAM entries required for each cache size.
- *Number of Data RAM Entries*: Number of data RAM entries required for each cache size. Since each tag RAM entry corresponds to four data RAM entries, the number of data RAM entries is always four times the number of tag RAM entries.
- *Number of Dirty Bit RAM Entries*: This number represents the total number of entries in the 1-bit dirty SRAM for the data cache. Since there is one dirty bit per tag entry, this number is always the same as the number of data cache tag RAM entries. This value is only relative to the data cache as the instruction cache does not contain a Dirty Bit RAM.

The cache is indexed using the *Tag RAM Index* portion of the 32-bit address. Figure 10 shows the address breakdown for a 4 Kbyte cache implementation.

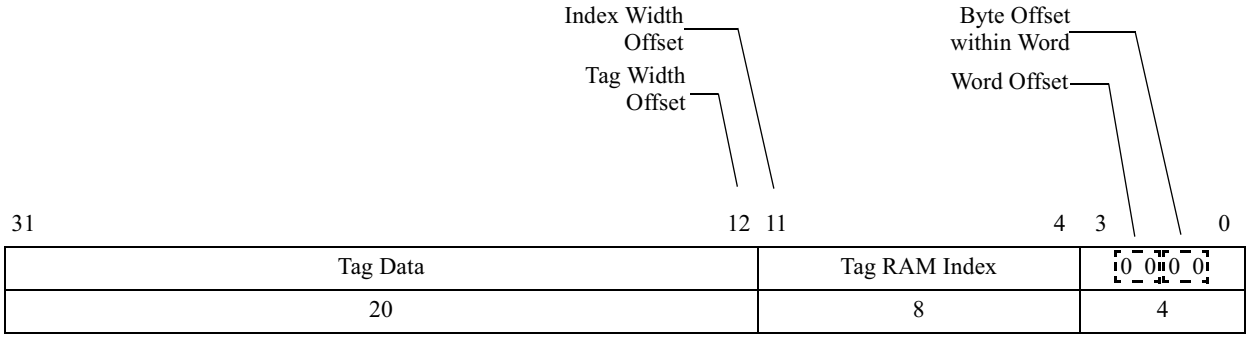

Figure 10. Address for a 4 Kbyte Cache Size

In Figure 10 above, the 8-bit *Tag RAM Index* field is used to index the tag RAM. The data to be copied into the RAM is stored in the *Tag Data* field. There is one Valid (*V*) bit and one Lock (*L*) bit per tag RAM entry.

In addition to indexing the Tag RAM, bits 11:4 and bits 3:2 are also used to index the data RAM. Since there are four data RAM entries for each tag RAM entry, bits 3:2 are used to select one of four words that correspond to a given tag RAM entry. Bits 1:0 are used to select one of four bytes within each 32-bit word.

Figure 11 shows the cache organization in a 4 Kbyte instruction / 4 Kbyte data implementation with 32-bits per data RAM entry.

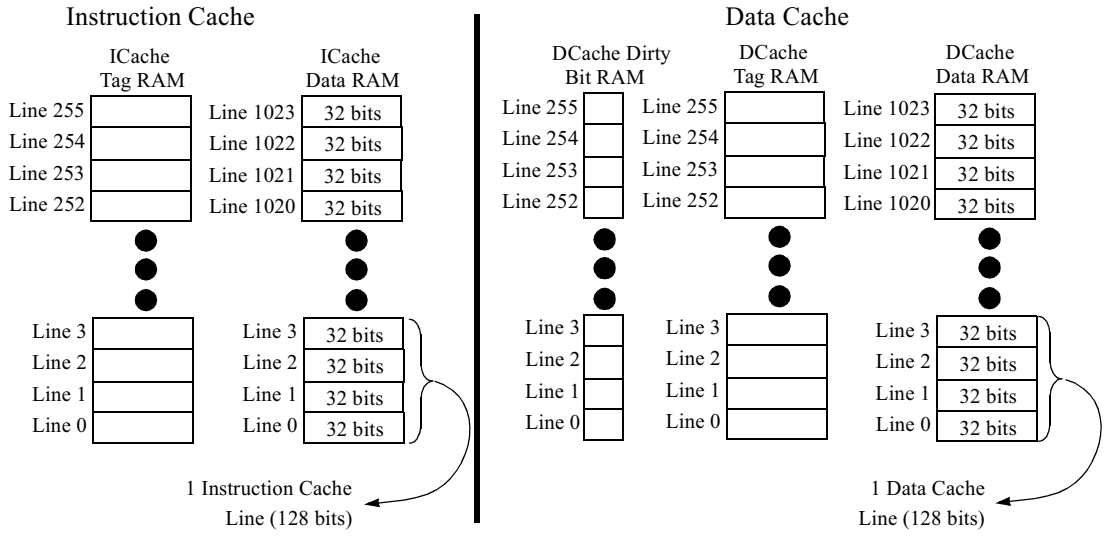

Figure 11. 4 Kbyte Instruction / 4 Kbyte Data Cache Organization Example

8.2 Write Buffer

The pT-110 provides a 4-entry write buffer to maximize memory bus bandwidth. This buffer is used by the data cache to store modified lines to be written out to memory, and by the core to store non-cacheable data that has access to the write buffer. Entries in the cache that are marked as ‘writeback’ are written to the write buffer instead of directly to memory. Writethrough pages are written directly to memory and do not use the write buffer.

When at least one entry in the write buffer is valid, a memory access is initiated. If all four entries in the write buffer are valid, the contents are written out to memory on a first-in, first-out basis. However, these four transactions are not performed atomically. Other memory accesses can occur in between the write operations. For example, there could be

two sequential writes from the buffer to memory, followed by a read, then a non-cacheable write, then another write from the buffer, etc.

8.3 Writeback and Writethrough

Each page in each region can be marked as writeback or writethrough. In order for a page to be marked as writethrough, that page must be data-cacheable. The *W* bit in the *Configuration* register enables the write buffer. In order for this bit to have meaning, the *P* bit in the *Configuration* register must also be set.

8.3.1 Writeback

A page is considered writeback when:

- The *W* bit in the *Configuration* register is set.
- The page is data cacheable.
- The corresponding *WBR* bit in the *Write Buffer Control* register is set.

In order for a page to be marked as writeback, that page must be data-cacheable. The *W* bit in the *Configuration* register enables the write buffer. In order for this bit to have meaning, the *P* bit in the *Configuration* register must also be set.

8.3.2 Writethrough

A page is considered writethrough when:

- The *W* bit in the *Configuration* register is set.
- The page is data cacheable.
- The corresponding *WBR* bit in the *Write Buffer Control* register is clear.

The eight bit *WBR* field in the *Write Buffer Control* register determines which regions of memory have access to the write buffer. Each of the 8 bits in this field corresponds to a given memory region. If a particular bit is set, that regions is marked as bufferable. If the bit is cleared, that region is marked as writethrough. For example, if bit 2 in the *WBR* field is cleared, then memory region 2 is writethrough.

8.4 Cache Locking

The PT-110 cache supports locking for both the primary instruction and data caches on a per-line basis. Each line in the cache contains a lock bit that is set whenever that line is locked. There are two ways to lock a cache line; read in a locked line from memory, or lock a line in the cache by setting the lock bit using the appropriate instruction.

Of the nine registers in coprocessor 15, register 7 is used to store the address contained in the *Rd* field of the instruction. The *Rd* field represents general Core registers r0 through r14. To perform a ‘lock’ operation, the core writes one of the following instructions to CP15 register C7. The MMU decodes this instruction and performs the operation on the appropriate cache. The pT-110 supports the following instructions for locking the cache. For a definition of each instruction described in the tables below, refer to Section 10.2, "CP15 Control Registers".

Table 7. pT-110 Cache Locking Instructions

Operation	Instruction
Instruction Cache Prefetch and Lock	`MCR p15, 0, Rd, C7, C7, 1`
Data Cache Prefetch and Lock	`MCR p15, 0, Rd, C7, C11, 1`
Instruction Cache Unlock	`MCR p15, 0, Rd, C7, C8, 0`

Table 7. pT-110 Cache Locking Instructions (Continued)

Operation	Instruction
Instruction Cache Lock	`MCR p15, 0, Rd, C7, C8, 1`
Data Cache Unlock	`MCR p15, 0, Rd, C7, C12, 0`
Data Cache Lock	`MCR p15, 0, Rd, C7, C12, 1`

- The *'Instruction Cache Prefetch and Lock'* operation performs a memory access and locks the contents of that line in the instruction cache in one operation. The line is fetched from memory and copied into the instruction cache, then locked. The *'Data Cache Prefetch and Lock'* performs the same type of operation on the data cache.
- The *'Instruction Cache Unlock'* operation unlocks a single line in the instruction cache as determined by the index address.
- The *'Instruction Cache Lock'* operation locks a single line in the instruction cache as determined by the index address.
- The *'Data Cache Unlock'* operation unlocks a single line in the data cache as determined by the index address.
- The *'Data Cache Lock'* operation locks a single line in the data cache as determined by the index address.

When a line is locked in the instruction cache and is scheduled for replacement with another locked line, it is not always necessary to unlock the current instruction cache line before copying in the new locked line from memory. In this case simply executing the *'Instruction Cache Prefetch and Lock'* instruction causes the original locked line to be overwritten. However, in the instruction cache, locked lines cannot be replaced with unlocked lines from memory. For example, the *'Instruction Cache Prefetch'* instruction cannot be used to fetch a line in memory and replace a locked line in the instruction cache. In this case the locked line must first be unlocked using the *'Instruction Cache Unlock'* instruction. Unlocked single lines in the instruction cache can be locked using the *'Instruction Cache Lock'* instruction.

In the same manner, when a line is locked in the data cache and is scheduled for replacement with another locked line, it is not necessary to unlock the current cache line before copying in the new locked line from memory. In this case simply executing the *'Data Cache Prefetch and Lock'* operation causes the original locked line to be overwritten.

Data cache stores can be done to locked lines without having to first unlock them, but loads from memory performed using the *'Data Cache Prefetch'* operation cannot replace a locked line in the data cache. In this case the locked line must first be unlocked using the *'Data Cache Unlock'* operation. Unlocked single lines in the data cache can be locked using the *'Data Cache Lock'* operation.

8.5 Cache Cleaning and Flushing

The pT-110 provides a cache flush mechanism that invalidates the contents of the cache, and a cache 'clean' mechanism for writing dirty lines out to memory. To maximize performance, cache cleaning and flushing should be performed when switching from Supervisor mode to User mode as there may be lines in the cache that are not accessible in User mode, thereby limiting the total number of cache lines available.

8.5.1 Cache Cleaning

The pT-110 provides a mechanism for writing dirty data cache lines to memory in the primary caches. When performing a context switch it is recommended that the primary instruction and data caches be flushed. However, flushing the data cache invalidates the entries and all data is lost. To preserve the contents of modified lines in the data cache, the Clean operation is used to write out either a single dirty line to memory, or all dirty lines in sequence. The pT-110 provides the following instructions for performing a cache Clean operation:

Table 8. Cache Clean Instructions

Operation	Instruction
Clean Data Cache (all)	`MCR p15, 0, Rd, C7, C10, 0`
Clean Data Cache (single entry)	`MCR p15, 0, Rd, C7, C10, 1`

To perform a ‘clean’ operation, the core writes one of the above instructions to CP15 register C7. The MMU decodes this instruction and performs the operation on the data cache.

The *‘Clean Data Cache (single)’* operation writes the contents of a single line in the cache out to memory. The locations is determined by the index address contained in the *Rd* field of the instruction.

The *‘Clean Data Cache (all)’* operation writes out all dirty lines to memory in sequence. This operation should always be performed prior to flushing the data cache.

8.5.2 Cache Flushing

The pT-110 provides the following instructions for flushing the cache:

Table 9. pT-110 Cache Flush Instructions

Operation	Instruction
Flush Instruction Cache (all)	`MCR p15, 0, Rd, C7, C5, 0`
Flush Instruction Cache (single entry)	`MCR p15, 0, Rd, C7, C5, 1`
Flush Data Cache (all)	`MCR p15, 0, Rd, C7, C6, 0`
Flush Data Cache (single entry)	`MCR p15, 0, Rd, C7, C6, 1`

To perform a ‘flush’ operation, the core writes one of the above instructions to CP15 register C7. The MMU decodes this instruction and performs the operation on the appropriate cache.

The *‘Flush Instruction Cache (all)’* operation flushes the entire contents of the instruction cache, invalidating all locked and unlocked lines.

The *‘Flush Instruction Cache (single)’* operation flushes a single line in the cache as determined by the index address contained in the *Rd* field of the instruction. Both locked and unlocked lines can be flushed. Is it not necessary to unlock an instruction cache line prior to performing a flush operation.

The *‘Flush Data Cache (all)’* operation flushes the entire contents of the data cache, invalidating all locked and unlocked lines regardless of dirty bit status. Note that certain lines dirty lines in the cache may require being written out to memory prior to performing the flush operation. This is called ‘cleaning the data cache’. When the cache is ‘cleaned’ all cache locations that have their dirty bit set are written out to memory prior to performing the flush.

The *‘Flush Data Cache (single)’* operation flushes a single line in the data cache as determined by the index address. Both locked and unlocked lines can be flushed. Is it not necessary to unlock a data cache line prior to performing a flush operation. As with the *‘Flush Data Cache (all)’* operation explained above, if the single line to be flushed has its dirty bit set, that line may need to be written to memory prior to executing the flush operation.

8.6 Built-In Self Test

The pT-110 processor contains a Built-In Self Test (BIST) mechanism for both the instruction and data caches. The BIST should be performed whenever the system is powered-up and is typically performed as part of the reset handler. The pT-110 provides two instructions for performing the BIST on the caches.

Table 10. pT-110 BIST Instructions

Operation	Instruction
BIST Instruction Cache	`MCR p15, 0, Rd, C7, C13, 0`
BIST Data Cache	`MCR p15, 0, Rd, C7, C13, 1`

The *BIST Instruction Cache* instruction performs the Built-In Self Test on the instruction cache. This instruction is performed by executing an MCR instruction along with the instruction parameters shown in Table 10.

The *BIST Data Cache* instruction performs the Built-In Self Test on the data cache. This instruction is performed by executing an MCR instruction along with the instruction parameters shown in Table 10.

When the BIST for both caches is complete, the results are obtained by reading the *BIST Result* register (CP15 - 8). Only the low-order two bits of this register contain valid information. Bit 0 contains the result of the instruction cache BIST, while bit 1 contains the result of the data cache BIST. If either of these bits are set, the BIST for the corresponding cache has failed.

9 Scratch Pad Caches

The pT-110 cache design includes direct-mapped instruction and data scratch pad caches that range between 256 bytes and 2 Kbytes in size. These caches can be used to store code or data segments which must execute in a guaranteed amount of time and which require a guaranteed amount of bandwidth. The caches are filled on power up. The scratch pad caches can be configured as follows.

Table 11. Instruction and Data Scratch Pad Cache Configuration Options

Size	Base Address	Offset Address
256 bytes	24 bits	6 bits
512 bytes	23 bits	7 bits
1 Kbytes	22 bits	8 bits
2 Kbytes	21 bits	9 bits

Each base and offset address in the above table comprises the lower 30 bits of the 32 bit address. The upper two bits are used for control as shown in Table 12 below. This table applies to both the instrucion and data scratch pad caches.

Table 12. Encoding of Address Bits 31:30

Bit 31	Bit 30	Action
0	0	Cache disabled
0	1	Cache disabled
1	0	Cache enabled, no fill operation
1	1	Cache enabled, fill operation

If bit 31 is zero, the cache is disabled and the state of bit 30 has no meaning. If bits 31 is one and bit 30 is zero, the cache is enabled but the fill operation does not occur. In order to fill the cache, bits 31:30 must be 0b11.

9.1 Instruction and Data Scratch Pad Cache Organization

The pT-110 scratch pad cache design includes a programmable direct-mapped instruction and data scratch pad caches that range between 256 bytes and 2 Kbytes in size. These caches can be used to store code or data segments which must execute in a guaranteed amount of time and which require a guaranteed amount of bandwidth.

Table 13 shows the configuration options for the pT-110 processor instruction and data scratch pad caches.

Table 13. pT-110 Scratch Pad Cache Size Options

Cache Size (bytes)	Base Address Width (bits)	Offset Address Width (bits)	Number of Tag RAM Entries	Number of Data RAM Entries
256	24	6	1	64
512	23	7	1	128
1024	22	8	1	256
2048	21	9	1	512

The entries in Table 13, from left to right, are defined as follows.

- *Cache Size:* This number represents the size of the instruction or data cache in bytes.
- *Base Address Width*: This number represents the actual number of bits used to index the Tag RAM. If the ‘base address’ portion of the address on the bus matches that stored in the tag RAM, there is a ‘hit’ to the scratch pad cache, and the offset address is then used to select one of the entries in the data RAM.
- *Offset Address Width*: Each value within the offset address corresponds to one location in the data RAM. For example, in a 256 byte scratch pad implementation, the offset address is 6 bits as shown in the above table. These bits corresponds to one of 64 entries in the cache ($2^6 = 64$).
- *Number of Tag RAM Entries*: Each scratch pad cache contains only one tag RAM entry which corresponds to the base address where the cache is located.
- *Number of Data RAM Entries*: Each scratch pad cache contains a data RAM that stores the information. The number of data RAM entries depends on the size of the cache as shown in the table above.

The cache is indexed using the 32-bit address, which is divided into base and offset addresses. Figure 12 shows the address breakdown for a 1 Kbyte cache implementation.

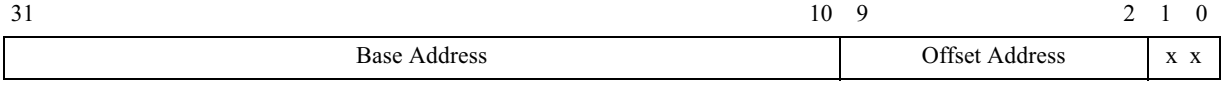

Figure 12. Address Format for a 1 Kbyte Scratch Pad Cache Implementaion

Bits 1:0 support byte and halfword accesses to the caches. On a word access, both bits are zero. On a byte access, bits 1:0 can be any combination of ones and zeros in order to access one of the four bytes the 32-bit word entry.

Figure 13 shows the cache organization in a 2 Kbyte instruction / 2 Kbyte data scratch pad cache implementation with 32-bits per data RAM entry.

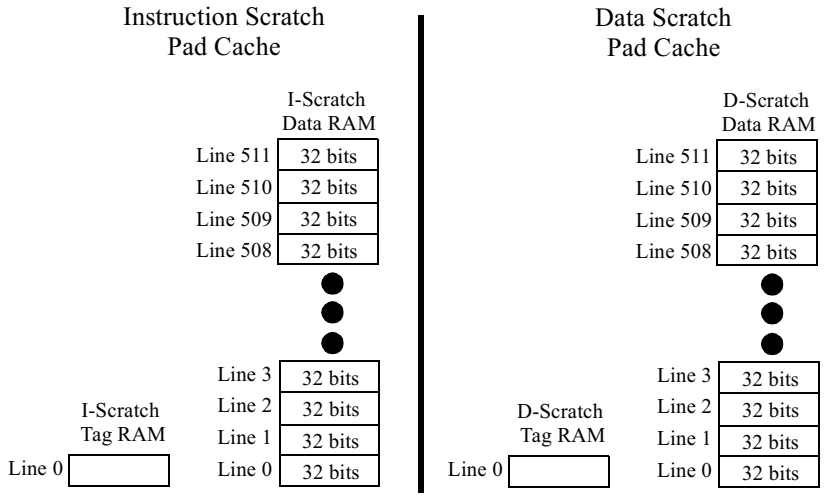

Figure 13. 2 Kbyte Instruction / 2 Kbyte Data Scratch Pad Cache Organization Example

9.2 Accessing the Scratch Pad Caches

The instruction and data scratch pad caches are accessed in the same manner. The instruction scratch pad cache is read-only, while the data scratch pad cache is read/write. There is one tag for each cache, meaning that all entries in the cache reside at a single base address. All entries are filled at the same time during a fill operation. After the caches have been filled, the data scatch pad cache can read or write to single lines in the cache. The instruction scratch pad is read only.

The 32 bit address used to access the cache is embedded in the instruction and contains the offset followed by the base address. The 32-bit address is transferred into register r0 using the MOV instruction as shown in the following example:

```
MOV  r0, 0xC0010000
```

In this instruction the lower 20 bits contains a value of 0x10000, which indicate the base address where the scratch pad cache resides. This is the value stored in the single-entry tag RAM. The upper 12 bits contain a value of 0xC00. The ‘C’ represents bits 31:28 of the address and translates to a binary value or 0b1100. The ‘1’ on bits 31:30 indicate that the cache is enabled and a fill operation is enabled. The instruction and data scratch pad caches can be enabled and disabled by driving a ‘1’ or a ‘0’ onto bit 31 of the base address where each cache resides. The remaining bits indicate the particular entry within the scratch cache being accessed.

The following is an example of how the address is moved into r0 using a 2 Kbyte cache implementation. The E indicates the enable bit, and the F indicates the Fill bit.

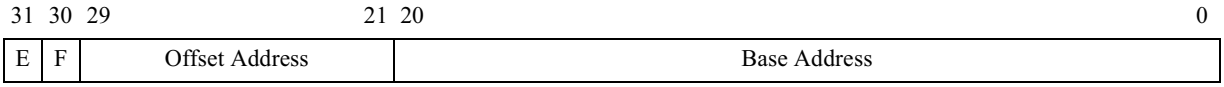

Figure 14. Instruction Format for 2 Kbyte Cache Implementaion

Once the above address is moved into r0, the address format is translated by hardware and driven onto the address bus in the format shown below:

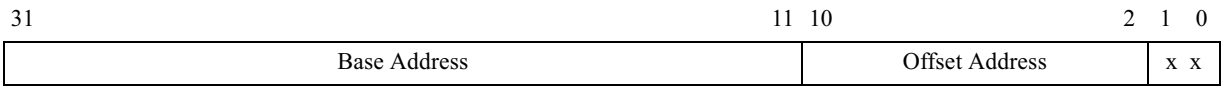

Figure 15. Physical Address Format for 2 Kbyte Cache Implementaion

In Figure 15 above, bits 1:0 are driven based on the size of the transfer. For example, on a byte write bits 1:0 would contain a value between 00 and 11 to indicate which of the four bytes of the 32-bit wide cache entry is being written. For word writes, both bits must be zero.

In Figure 15 the offset address is 9 bits wide for a 2 Kbyte cache implementation. Since each cache entry is 32-bits wide, this means that 512 cache entries are present as shown in Table 11.

The MRC and MCR instructions are used in conjunction with the MOV instruction to access the scratch pad caches. The MOV instruction moves the target address value into register r0. The MCR instruction is used to move the value in r0 into CP15 register 9 or 10, depending on whether the target is the instruction scratch pad of the data scratch pad, respectively.

The following instruction is used to write the value in r0 to CP15-9 and enable the instruction scratch pad RAM.

```
MCR p15, 0, r0, c9, c0, 0
```

The following instruction is used to write the value in r0 to CP15-10 and enable the data scratch pad RAM.

```
MCR p15, 0, r0, c10, c0, 0
```

Under certain conditions it may be necessary to read the address stored in the corresponding CP15 register in order to perform a compare operation with the current address. The following two instructions are used to read the CP15-9 instruction scratch pad and CP15-10 data scratch pad registers.

```
MRC p15, 0, r0, c9, c0, 1  #read base address of instruction scratch RAM
MRC p15, 0, r0, c10, c0, 1 #read base address of data scratch RAM
```

9.3 Data Scratch Pad Cache Write Protocol

The data scratch pad cache is a read/write cache that is filled on power-up. After the cache has been filled, read and write operation can be performed on single entries in the scratch pad cache. The data scratch pad cache conforms to the following write protocols as shown in Table 14.

Table 14. Cache and Memory Update Protocol

Data Scratch Pad	Data Cache	Action
Hit	Miss	Update data scratch pad and also write through data to memory.
Miss	Hit	Update data cache. Do not writethrough to memory.
Hit	Hit	Update data scratch pad and data cache.
Miss	Miss	Write data to memory.

Two internal signals, ‘write_half’ and ‘write_byte’, are used along with the lower 2 bits of address to determine the size and location of the write operation. The pT-110 supports byte, halfword, and word writes as shown in Table 15 below.

Table 15. Determining Write Operation Size

write_half	write_byte	Address 1:0	Write Size
0	0	00	32 bit word write
0	1	00, 01, 10, or 11	8 bit byte write
1	0	00 or 10	16 bit halfword write
1	1		Invalid. Should never occur.

9.4 Flushing the Scratch Pad Caches

The scratch pad caches are flushed whenever the corresponding cache is flushed. The scratch pad caches do not support ‘clean’ operations, meaning that modified lines can not be written to memory.

There are 3 ways to flush the instruction and data scratch pad caches.

1. Asserting Reset: By default, a reset operation flushes and disables the caches. The appropriate MCR command is used to enable the caches.
2. Cache Flush - All: Execution of the *Instruction Cache Flush All* instruction causes both the primary instruction cache and the instruction scratch pad cache to be flushed. In the primary instruction cache this means that all valid bits are reset. Since the scratch pad cache does not have valid bits, the hardware automatically disables the instruction scratch pad whenever the *Instruction Cache Flush All* instruction is executed.
   The same basic protocol also applies to the data scratch pad cache. Execution of the *Data Cache Flush All* instruction causes both the primary data cache and the data scratch pad cache to be flushed. In the primary data cache this means that all valid bits are reset. Since the scratch pad cache does not have valid bits, the hardware automatically disables the data scratch pad whenever the *Data Cache Flush All* instruction is executed.
3. Cache Flush - Single Entry: In the primary instruction cache, execution of the *Instruction Cache Flush Single* instruction causes a single line in the primary instruction cache to be flushed by resetting its valid bit. When this instruction is executed, hardware compares the base address in the instruction to the base address where the instruction scratch pad cache is located. If the compare is valid, the scratch pad is flushed.
   The same applies to the data scratch pad. execution of the *Data Cache Flush Single* instruction causes a single line in the primary data cache to be flushed by resetting its valid bit. When this instruction is executed, hardware compares the base address in the instruction to the base address where the data scratch pad cache is located. If the compare is valid, the scratch pad is flushed.

10 Register Set

The pT-110 register set consists of both General Purpose (GP) registers and CP15 Control registers. General Purpose registers are used to store the operands and results of a computation. A 4-bit value contained in each operand of an instruction indicates which one of the 16 General Purpose registers the information is stored to or read from.

In addition to the GP registers, the pT-110 contains 16 Control registers, called Coprocessor 15 (CP15). These registers are accessed using the MRC and MCR instructions. Distinct fields within these instructions indicate which CP15 register is used as the destination of the operation.

10.1 General Purpose Registers

The pT-110 contains 30 general purpose registers, 6 status registers, and a program counter (PC). The general purpose registers can be accessed in any operating mode. However, only certain registers are available in certain modes. Table 16 shows a diagram of the general purpose register set.

Table 16. pT-110 General Purpose Register File

User	FIQ	IRQ	Supervisor	Abort	Undefined Instruction
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5

Table 16. pT-110 General Purpose Register File

User	FIQ	IRQ	Supervisor	Abort	Undefined Instruction
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_FIQ	R8	R8	R8	R8
R9	R9_FIQ	R9	R9	R9	R9
R10	R10_FIQ	R10	R10	R10	R10
R11	R11_FIQ	R11	R11	R11	R11
R12	R12_FIQ	R12	R12	R12	R12
R13	R13_FIQ	R13_IRQ	R13_SVC	R13_ABORT	R13_UND
R14	R14_FIQ	R14_IRQ	R14_SVC	R14_ABORT	R14_UND
PC					
CSPR					
	SPSR_FIQ	SPSR_IRQ	SPSR_SVC	SPSR_ABORT	SPSR_UND

In Table 16, the non-shaded areas indicate registers that are accessible in any mode. These registers are shared by all modes. For example, there is only one register R1 that is shared by all modes. Shaded areas indicate registers which are only available in that mode. For example, FIQ mode has access to common registers R0 - R7. In addition, this mode contains 7 dedicated registers (R8_FIQ - R14_FIQ) that are only available in FIQ mode, as well as a dedicated SPSR_FIQ register.

IRQ mode has access to common registers R0 - R12. In addition, this mode contains 2 dedicated registers (R13_IRQ - R14_IRQ) that are only available in IRQ mode, as well as a dedicated SPSR_IRQ register.

10.1.1 Link Register

In Table 16 above, each mode also has its own general-purpose register R14. This register has a dedicated function as the Link register. On interrupts and exceptions, the corresponding R14 register is loaded with a return address. The treatment of the return address varies with the type of interrupt or exception, as shown in Table 17.

Table 17. Return Address Loaded in the Link Register

Event Type	Return Address	Return Instruction
FIQ	Address of next instruction + 4	SUBS PC, R14, #4
IRQ	Address of next instruction + 4	SUBS PC, R14, #4
Software Interrupt (32-bit instruction set)	Address of SWI instruction + 4	MOVS PC, R14
Software Interrupt (16-bit instruction set)	Address of SWI instruction + 2	MOVS PC, R14
Instruction Abort	Address of aborted instruction + 4	SUBS PC, R14, #4
Data Abort	Address of aborted instruction + 8	SUBS PC, R14, #8
Undefined Instruction (32-bit instruction set)	Address of undefined instruction + 4	MOVS PC, R14
Undefined Instruction (16-bit instruction set)	Address of undefined instruction + 2	MOVS PC, R14

Table 17 also shows the instruction executed to return from the handler for each type of event. For FIQ and IRQ interrupts, the handler returns to the instruction following the last instruction that was executed before the handler was called. For software interrupts (SWI instruction) and undefined instructions, the handler returns to the instruction following the instruction which raised the exception. For instruction and data aborts, the handler returns to the instruction which raised the exception, which is re-executed after the aborting condition has been removed.

10.1.2 Program Counter

Register R15 is the program counter (PC). While executing the 32-bit instruction set, bits 1:0 of this register read as zero and are unaffected by writes. While executing the 16-bit subset, bit 0 reads as 0 and is unaffected by writes.

10.1.3 CPSR and SPSR

The Saved Processor Status Register (SPSR) is loaded with the contents of the Current Processor Status Register (CPSR) when an interrupt or exception handler is called. Every mode except User mode has its own dedicated SPSR.

The data processing instruction encoding contains a bit that usually controls whether the condition codes are written. However, when the destination register is the PC, this bit controls whether the CPSR is simultaneously loaded from the SPSR. The format of the CPSR and SPSR is shown in Figure 16.

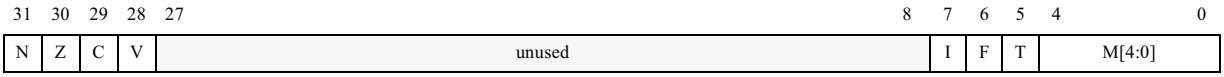

Figure 16. CPSR/SPSR Format

The upper four bits of the CPSR and SPSR registers contain the condition codes:

- *N*—Sign of the result (i.e. bit 31).
- *Z*—Set on zero result, clear on non-zero result.
- *C*—Set on carry, otherwise clear. In the case of move and logical instructions, carry comes from the calculation of the second source operand.
- *V*—Set on overflow, otherwise clear. Unaffected by move or logical instructions.

The lower eight bits hold various mode control bits:

- *I*—Set when IRQ interrupts are disabled, otherwise clear.
- *F*—Set when FIQ interrupts are disabled, otherwise clear.
- *T*—Set when executing the 16-bit instruction set, clear when executing the 32-bit instruction set.
- *M[4:0]*—Processor mode. Refer to Table 1 for the encoding of this field.

Attempts to modify the mode control bits from User mode are ignored. The unused bits read as zero, and must only be written with values that preserve their state. Future implementations may define uses for these bits, so software that forces them to a particular state might create compatibility problems.

10.2 CP15 Control Registers

In the pT-110 processor, the CP15 Control register set consists of 16 registers accessed with the MCR and MRC coprocessor register transfer instructions. CP15 is the only coprocessor supported by the pT-110 processor. Accesses to all other coprocessors result in an exception.

The CP15 registers are summarized in Table 18.

Table 18. CP15 Register Map

Number	Name	Description
0	ID Code	Code to identify the processor type and revision level.
1	Configuration	Control bits to enable/disable caches, protection, and write buffers. Also controls endianness.
2	Cache Control	Control bits to enable/disable cacheability of eight regions of the address space.
3	Write Buffer Control	Control bits to enable/disable write buffering for eight regions of the address space.
4	Instruction Space Protection	Protection bits for instruction access to eight regions of the address space.
5	Data Space Protection	Protection bits for data access to eight regions of the address space.
6	Region Size	Selects the granularity of control over eight regions of the address space, in power of 2 increments.
7	Data Cache Address	Stores the address provided by the Rd field of the instruction.
8	BIST Result	Indicates the result of the most recent BIST operation.
9	Instruction Scratch Pad	Indicates the address used to access the instruction scratch pad RAM.
10	Data Scratch Pad	Indicates the address used to access the data scratch pad RAM.
11	RdHi	Stores the upper 32 bits of the 64-bit result from a multiply or multiply-accumulate operation.
12-15	Reserved	Reserved.

11 Exception Processing

The pT-110 processor core supports seven exceptions types. Each exception is identified with a particular privileged operating mode. When an exception is generated by either an external or internal source, the contents of the CPSR are moved to the corresponding SPSR.

For example, if the external FIQ pin is asserted, indicating a request for fast interrupt servicing, the contents of the CPSR are moved to SPSR_FIQ. The corresponding Link register (R14_FIQ) contains the return address that is moved into the PC once the exception has been processed. These values are shown in Table 17. Once the exception has been serviced, the contents of R14_FIQ are moved into the PC, and the contents of SPSR_FIQ are moved into the CPSR and the program resumes execution.

When an exception is taken, the processor vectors to one of the addresses shown in Table 19.

Table 19. pT-110 Exception Types

Exception	Mode	Vector Address
Reset	SVC	0x00000000
Undefined Instruction	UND	0x00000004
Software Interrupt (SWI)	SVC	0x00000008
Prefetch Abort (instruction fetch)	ABORT	0x0000000C
Data Abort (data fetch)	ABORT	0x00000010
Reserved	Reserved	0x00000014
Interrupt	IRQ	0x00000018
Fast Interrupt	FIQ	0x0000001C

11.1 Reset Exception

The Reset exception is taken whenever the processor's reset pin is asserted. In this case the processor immediately stops instruction execution and vectors to address 0x00000000. This exception is handled in Supervisor mode.

When the nRESET signal is deasserted, the processor defaults to Supervisor mode and begins execution at address 0x00000000 with all interrupts disabled.

11.2 Undefined Instruction Exception

The Undefined Instruction exception is taken under the following conditions:

- Whenever the processor executes a coprocessor instruction and no coprocessor responds.
- Whenever an attempt is made to execute an instruction that is undefined. Undefined instructions occur when bits 27:25 of the instruction contain a value of 0b011, and bit 4 contains a value of 0b1.

In either case the processor allows the instruction currently in the W stage to complete. The address corresponding to the instruction in the M state at the time of this exception is copied to the corresponding Link register (R14_UND) and the processor vectors to address 0x00000004. This exception is handled in Undefined Instruction mode. Once the exception has been serviced, the contents of R14_UND are copied to the program counter (PC) and the processor begins execution at the instruction previously in the M stage when the exception was taken.

When the Undefined Instruction exception is taken, the contents of the PC are copied to R14_UND and the contents of the CPSR are copied to SPSR_UND. Once the exception has been serviced, the contents of R14_UND are incremented by 4 (in 32-bit mode) and copied to the PC. The contents of SPSR_UND are copied back to the CPSR and execution resumes.

11.3 Software Interrupt Exception

Execution of the SWI instruction causes the processor to enter Supervisor mode to perform a particular operating system function. The processor allows the instruction currently in the W stage to complete and then vectors to address 0x00000008.

When the Software Interrupt exception is taken, the contents of the PC are copied to R14_SVC and the contents of the CPSR are copied to SPSR_SVC. Once the exception has been serviced, the contents of R14_SVC are incremented by 4 (in 32-bit mode) and copied to the PC. The contents of SPSR_SVC are copied back to the CPSR and execution resumes.

11.4 Prefetch Abort Exception

A Prefetch Abort exception is taken whenever the processor attempts to execute an invalid instruction. The processor allows the instruction currently in the W stage to complete and then vectors to address 0x0000000C.

The memory system can signal an abort to the processor on any instruction fetch to memory. In doing so, the fetched instruction is marked as invalid. If the processor attempts to execute this invalid instruction, a Prefetch Abort exception occurs. Note that if the fetched instruction is not executed due to a branch being taken or an unconditional jump, no Prefetch Abort exception is taken.

When the Prefetch Abort exception is taken, the contents of the PC are copied to R14_ABORT and the contents of the CPSR are copied to SPSR_ABORT. Once the exception has been serviced, the contents of R14_ABORT are incremented by 4 (in 32-bit mode) and copied to the PC. The contents of SPSR_ABORT are copied back to the CPSR and execution resumes.

11.5 Data Abort Exception

A Data Abort exception is taken whenever the processor attempts to execute invalid data. The processor allows the instruction currently in the W stage to complete and then vectors to address 0x00000010.

The memory system can signal an abort to the processor on any data fetch from memory. In doing so, the fetched data (load or store) is marked as invalid. If this occurs the exception is taken before any subsequent instruction are executed.

When the Data Abort exception is taken, the contents of the PC are copied to R14_ABORT and the contents of the CPSR are copied to SPSR_ABORT. Once the exception has been serviced, the contents of R14_ABORT are incremented by 8 (in 32-bit mode) and copied to the PC. The contents of SPSR_ABORT are copied back to the CPSR and execution resumes.

11.6 Interrupt Exception

The Interrupt exception is taken in response to external logic asserting the IRQ input pin whenever bit 7 of the CPSR is cleared to 0b0. When this bit is set to 0b1, normal interrupts are disabled and the state of the IRQ pin is ignored. Note bit 7 can only be modified in one of the privileged modes. User mode cannot modify the state of this bit. When this exception is taken the processor allows the instruction currently in the W stage to complete and then vectors to address 0x00000018.

When an Interrupt exception is taken, the contents of the PC are copied to R14_IRQ and the contents of the CPSR are copied to SPSR_IRQ. Once the exception has been serviced, the contents of R14_IRQ are incremented by 4 based on the PC value of the next instruction (in 32-bit mode) and copied to the PC. The contents of SPSR_IRQ are copied back to the CPSR and execution resumes.

11.7 Fast Interrupt Exception

The Fast Interrupt exception is taken in response to external logic asserting the FIQ input pin whenever bit 6 of the CPSR is cleared to 0b0. When this bit is set to 0b1, fast interrupts are disabled and the state of the FIQ pin is ignored. Note bit 6 can only be modified in one of the privileged modes. User mode cannot modify the state of this bit. Fast interrupt mode (FIQ) differs from normal interrupt mode (IRQ) in that FIQ mode contains additional dedicated registers that minimize the need for register saving and hence context switching. When this exception is taken the processor allows the instruction currently in the W stage to complete and then vectors to address 0x0000001C.

When an Fast Interrupt exception is taken, the contents of the PC are copied to R14_FIQ and the contents of the CPSR are copied to SPSR_FIQ. Once the exception has been serviced, the contents of R14_FIQ are incremented by 4 (in 32-bit mode) based on the PC value of the next instruction and copied to the PC. The contents of SPSR_FIQ are copied back to the CPSR and execution resumes.

11.8 Exception Priorities

The pT-110 processor prioritizes the above exceptions as shown in Table 20.

Table 20. pT-110 Exception Priorities

Exception Type	Priority Level
Reset	1 (highest)
Data Abort	2
FIQ	3
IRQ	4

Table 20. pT-110 Exception Priorities

Exception Type	Priority Level
Prefetch Abort	5
Undefined Instruction, SWI	6 (lowest)

12 Signal Descriptions

Table 21 lists the pT-110 processor signals.

Table 21. pT-110 Signal Descriptions

Name	Type	Description
A[31:0]	O	*Address Bus*—a 32-bit byte address driven by the processor. On halfword transactions, A[0] is undefined. On word transactions, A[1:0] is undefined. If non-pipelined operation is selected (APE low), the address is driven valid after the beginning of the current cycle and remains valid past the end of the current cycle. If pipelined operation is selected (APE high), the address is driven valid before the end of the previous cycle and remains valid past the middle of the current cycle.
ABORT	I	*Abort Cycle*—when sampled asserted at the end of the cycle, terminates the current instruction, restores the processor state to before execution of the instruction began, and raises an exception. Either a data abort or an instruction abort is raised, depending on the type of access that triggered the abort. Can be used with external logic to implement virtual memory.
ACK	O	*Acknowledge*—asserted by the processor to indicate that an external bus master has control of the bus. In response to receiving an assertion of the HOLDREQ input, the processor quits driving the bus at the end of the current bus cycle and asserts the ACK output. The external bus master then has control of the bus, and it retains control until it deasserts HOLDREQ and the processor deasserts ACK.
APE	I	*Address Pipelining Enable*—when sampled asserted in the middle of the current cycle, selects pipelined operation for the following cycle. If pipelined operation is selected, A[31:0], LOCK, MAS[1:0], nOPC, nRW, and nTRANS are driven valid before the end of the current cycle and are sustained until the middle of the following cycle. If non-pipelined operation is selected, these signals are driven valid after the end of the current cycle and sustained until after the end of the following cycle.
BCLK	O	*Bus Clock*—all devices on the system bus must be synchronized to either BCLK or its complement, MCLK. In the pT-110, the RCLK input clock is multiplied by the value determined by the state of the PF[2:0] pins to derive the internal processor clock frequency (PCLK). BCLK is derived from dividing the internal processor clock (PCLK) signal by the value determined by the state of the BF[2:0] pins. MCLK is an inverted form of BCLK.
BF[2:0]	I	*Bus Clock Divisor*—tied to static logic levels for selecting the clock divisor for the bus clock, which is generated from the internal processor clock. These state of these bits cannot be changed dynamically. BF[2:0] / Divide By: 000 / 2; 001 / 4; 010 / 6; 011 / 8; 100 / 10; 101 / 12; 110 / 14; 111 / 16

Table 21. pT-110 Signal Descriptions (Continued)

Name	Type	Description
BIGEND	O	*Big-Endian Enable*—driven high to indicate the processor is using big-endian byte order, or low to indicate little-endian. Byte order is controlled by the B bit in the CP15 Configuration register.
BL[3:0]	I	*Byte Latch Enable*—when sampled asserted on a falling edge of MCLK, enables the corresponding latches on the D[31:0] bus. Each BL[3:0] signal controls one of the byte lanes, e.g. BL[3] controls the latch on D[31:24]. Used with the nWAIT signal to assemble data from memory and peripherals that are narrower than the transaction requested by the processor. For example, to assemble a 32-bit instruction fetch or data load from four 8-bit memory reads, nWAIT is asserted to add at least three additional falling edges of MCLK to the cycle. On each edge, the 8-bit data is driven on one of the four byte lanes, and the corresponding BL[3:0] signal is asserted. The bytes may be assembled in any order. The latches may be loaded more than once before the end of the cycle, as long as the last data to be loaded is the correct data. If the processor is requesting less than a word of data, the unused bytes are masked off. Latches for masked bytes may be loaded with any data, or not loaded at all.
BYP	I	*Bypass*—asserted to bypass the PLL and drive the processor clock directly from the external BYPCLK clock input.
BYPCLK	I	*Bypass Clock*—external clock used when BYP is asserted.
D[31:0]	I/O	*Data Bus*—a bidirectional 32-bit bus for transferring data between the processor and the system.
nFIQ	I	*Fast Interrupt Request*—asserted to raise the FIQ exception after execution of the current instruction is completed. This exception processing mode has a partial register file not shared with the other processor modes, to minimize context-switch overhead for frequently called or time-critical interrupts. The nFIQ input is synchronized, so it may be asserted asynchronously to MCLK.
HOLDREQ	I	*Hold Request*—Asserted by an external bus master to request ownership of the bus. The pT-110 relinquishes ownership of the bus by asserting ACK.
nIRQ	I	*Interrupt Request*—asserted to raise the IRQ exception after execution of the current instruction is completed. The nIRQ input is synchronized, so it may be asserted asynchronously to MCLK.
LOCK	O	*Locked Cycles*—indicates that the processor is performing an atomic load/store operation, which only occurs during execution of a swap instruction (SWP or SWPB). The LOCK signal is valid simultaneously with A[31:0].
nM[4:0]	O	*Processor Mode*—indicates the current processor mode. Driven valid after the falling edge of MCLK.
MAS[1:0]	O	*Memory Access Size*—indicates the transfer size of a bus transaction. 00 indicates a byte transfer, 01 a halfword transfer, and 10 a word transfer. 11 is a reserved combination. The MAS[1:0] signals are valid simultaneously with A[31:0].
MCLK	O	*Master Clock*—all devices on the system bus must be synchronized to either MCLK or its complement, BCLK. In the pT-110, the RCLK input clock is multiplied by the value determined by the state of the PF[2:0] pins to derive the internal processor clock frequency (PCLK). BCLK is derived from dividing the internal processor clock (PCLK) signal by the value determined by the state of the BF[2:0] pins. MCLK is an inverted form of BCLK.
nMREQ	O	*Memory Request*—indicates that the processor will transfer data on the next cycle. Driven valid in the middle of the cycle before the data transfer, and sustained until after the beginning of the cycle in which the data is transferred.

Table 21. pT-110 Signal Descriptions (Continued)

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>nOPC</td><td>O</td><td>Opcode Fetch—indicates that the processor is performing an opcode fetch. The nOPC signal is valid simultaneously with A[31:0].</td></tr>
<tr><td>PF[2:0]</td><td>I</td><td>Processor Clock Multiplier—tied to static logic levels for selecting the clock multiplier for the internal processor clock, which is generated from the external clock input RCLK. The state of these bits are determined at reset and cannot be changed dynamically.
<table>
<tr><th>PF[2:0]</th><th>Multiply By</th></tr>
<tr><td>000</td><td>2</td></tr>
<tr><td>001</td><td>4</td></tr>
<tr><td>010</td><td>6</td></tr>
<tr><td>011</td><td>8</td></tr>
<tr><td>100</td><td>10</td></tr>
<tr><td>101</td><td>12</td></tr>
<tr><td>110</td><td>14</td></tr>
<tr><td>111</td><td>16</td></tr>
</table></td></tr>
<tr><td>PRESET</td><td>I</td><td>PLL Reset—asserted for at least 0.5 milliseconds to initialize the PLL.</td></tr>
<tr><td>RCLK</td><td>I</td><td>External Clock Input—the internal processor clock and the two bus clock outputs (MCLK and BCLK) are generated from RCLK. All devices on the system bus must synchronize to MCLK or BCLK, because the PLL introduces skew between RCLK and the operation of the bus.</td></tr>
<tr><td>nRESET</td><td>I</td><td>Reset—asserted for at least four periods of the internal processor clock after PRESET goes low to initialize the processor. When nRESET goes high, the processor then begins execution at address 0.</td></tr>
<tr><td>nRW</td><td>O</td><td>Read/Write—a low indicates that the processor is performing a read cycle, while a high indicates a write cycle. The nRW signal is valid simultaneously with A[31:0].</td></tr>
<tr><td>SEQ</td><td>O</td><td>Sequential Cycle—indicates that the processor will transfer data to a sequential address on the next cycle. A sequential address is either the same address used for the current cycle, an address that is greater by 2 if the TBIT signal is high, or an address that is greater by 4 if the TBIT signal is low. Driven valid in the middle of the cycle before the data transfer, and sustained until after the beginning of the cycle in which the data is transferred.</td></tr>
<tr><td>TBIT</td><td>O</td><td>T Bit Status—indicates the current value of the T bit in the CPSR. The T bit is 1 (i.e. the TBIT signal is high) when the processor is executing the 16-bit code-density instruction set. The T bit is 0 (i.e. TBIT low) when the processor is executing the full 32-bit instruction set.</td></tr>
<tr><td>nTRANS</td><td>O</td><td>Translation Enable—a low indicates that the processor is in user mode, which may be used by external memory mapping logic to enable address translation. The nTRANS signal is valid simultaneously with A[31:0].</td></tr>
<tr><td>nWAIT</td><td>I</td><td>Wait—asserted to stall the processor for an integral number of MCLK periods. If this capability is not needed in a system, tie nWAIT high. The nWAIT signal may change only when MCLK is low.</td></tr>
<tr><td>nTRST</td><td>I</td><td>Test Reset—Active low master reset for the JTAG Test Access Port (TAP). At power-up the assertion of nTRST causes the TAP controller to be reset.</td></tr>
<tr><td>TCK</td><td>I</td><td>Test Clock—Test clock input for the JTAG TAP.</td></tr>
<tr><td>TMS</td><td>I</td><td>Test Mode Select—Test mode select input for the JTAG TAP.</td></tr>
<tr><td>TDI</td><td>I</td><td>Test Data In—Test data input for the JTAG TAP.</td></tr>
<tr><td>TDO</td><td>O</td><td>Test Data Out—Test data output for the JTAG TAP.</td></tr>
</table>

Table 21. pT-110 Signal Descriptions (Continued)

Name	Type	Description
DBGRQ	I	*Debug Request*—This is a level sensitive input which, when HIGH, causes the pT-110 processor core to enter debug state after executing the current instruction. This allows external hardware to force the pT-110 processor core into the debug state.
DBGACK	O	*Debug Acknowledge*—This signal is an output from the pT-110 processor core which, when HIGH, indicates that the core is in the debug state.

13 Bus Interface Unit

The Bus Interface Unit manages the flow of data between the processor core and all external logic.

13.1 Bus Cycle Types

Two signals, nMREQ and SEQ, indicate whether a bus transfer occurs on the following MCLK cycle. There are three cycle types indicated by these signals, as shown in Table 22.

Table 22. Bus Cycle Types

Type	nMREQ	SEQ	Description
N-cycle	Low	Low	*Non-sequential cycle*—the target of the bus transfer is an address which is not sequential from the address driven during the previous cycle.
S-cycle	Low	High	*Sequential cycle*—the target of the bus transfer is either the same address driven on the bus during the previous cycle, or an increment of that address. The increment is either 2 for halfwords or 4 for words.
I-cycle	High	Low	*Idle cycle*—no bus transfer is occurring.

The timing of the these signals is shown in Figure 17.

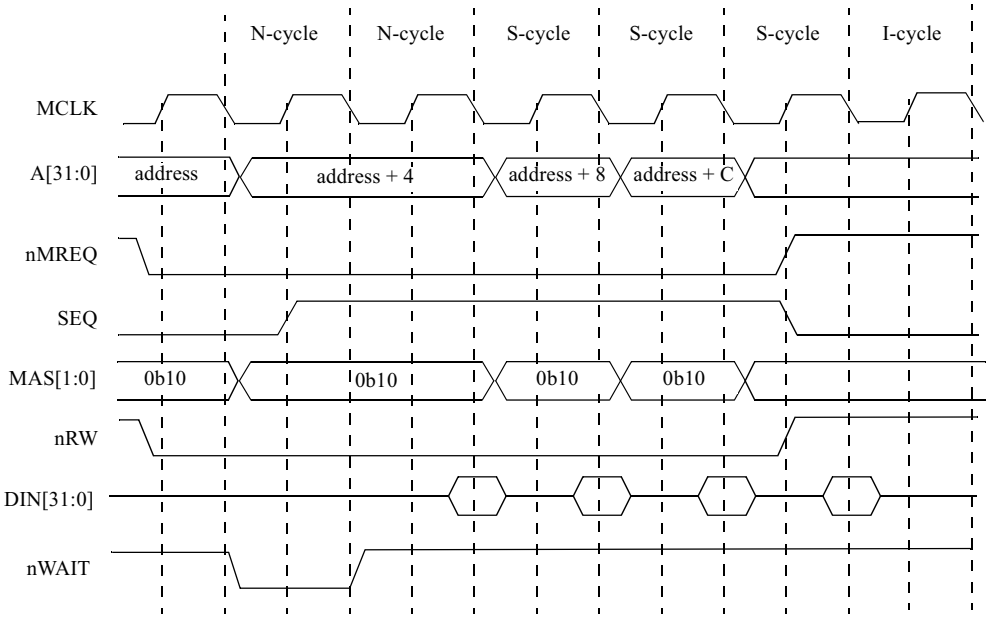

Figure 17. Relationship of MRQ and SEQ During a 32-bit Read Operation

In Figure 17 the processor asserts nMREQ and deasserts SEQ at the start of the cycle indicate a non-sequential cycle (N-cycle) as shown in Table 22 above. The processor drives a value of 0b10 onto MAS[1:0] throughout the cycle to indicate a 32 bit transfer. The processor drives nRW low to indicate a read operation. In this example the memory returns data in two clocks after address first becomes valid on the falling edge of MCLK. The assertion of nWAIT for one clock causes the ‘address + 4’ value to be driven on the address bus for two clocks.

Since MAS[1:0] contains a value of 0b10, indicating a 32-bit transfer, the processor increments the address by 4 SEQ, indicating a sequential cycle (S-cycle) in Table 22 above. The state of nMREQ and SEQ remains stable throughout the remainder of the cycle. Both signals are deasserted on the next rising edge of MCLK after the last address is driven onto the bus (on the previous negative edge of MCLK). The deassertion of these signals indicates an idle cycle (I-cycle) condition as shown in Table 22 above.

13.2 Address Pipelining

Figure 18 and Figure 19 show the behavior of the address bus with address pipelining disabled and enabled. The memory controller drives the APE signal high or low depending on the type of memory being accessed. When APE is low, as illustrated in Figure 18, address pipelining is disabled and the address is driven on the falling edge of clock 2 below. This mode is typically used to interface to an SRAM memory array. Data can be returned on the following negative edge of MCLK as shown in clock 3.

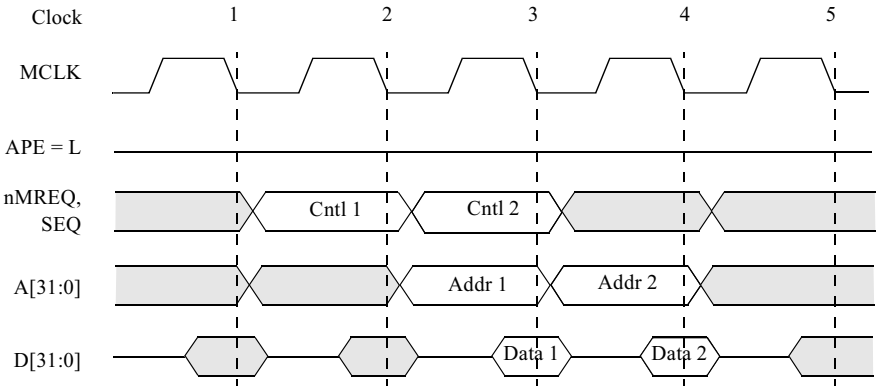

Figure 18. Address Pipelining Disabled

In Figure 19 the memory controller drives the APE signal high, enabling address pipelining. In this mode address is driven one half clock earlier than when pipelining is disabled, at the rising edge of clock 2. This facilitates access to devices with longer initial access times and is typically used to interface to a DRAM memory array. When APE is high, the processor drives address on the rising edge of clock 2. The memory samples the address on the falling edge of clock 2, then drives data on the falling edge of clock 3.

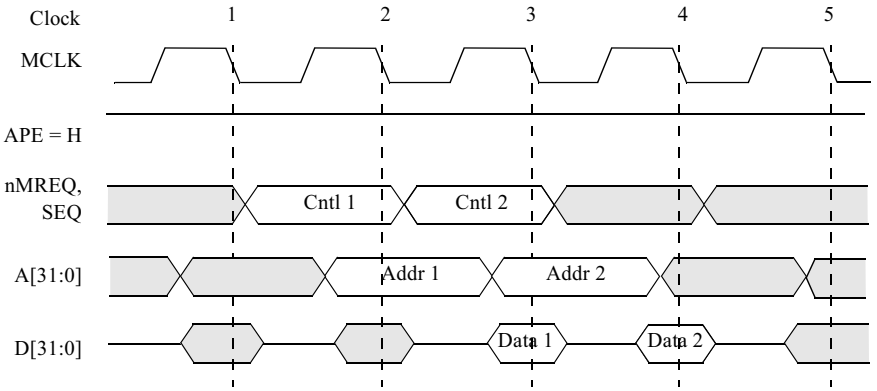

Figure 19. Address Pipelining Enabled

13.3 Data Bus Latches

To support memory and peripherals that are 8- or 16-bits wide, the data bus latches allow assembling a 32-bit word from bytes or 16-bit halfwords. Figure 20 shows an example of assembling a word from two halfwords.

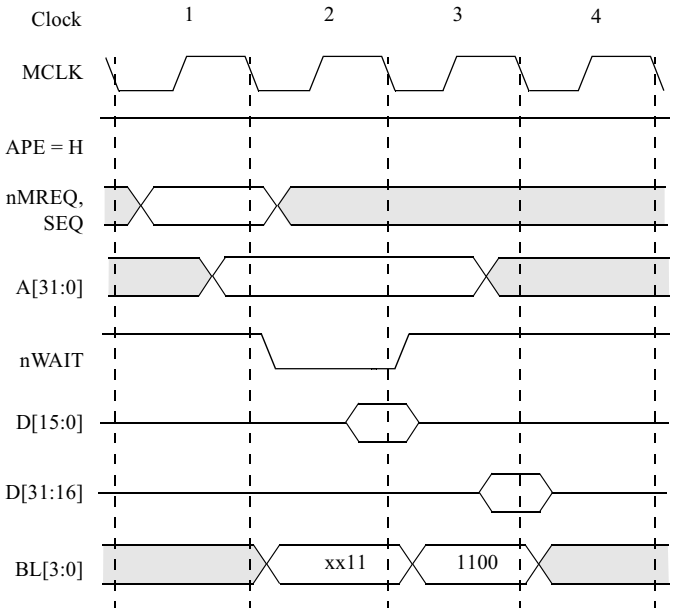

Figure 20. Assembling a Word from Two Halfwords

In the example, the memory controller asserts nWAIT at the falling edge of clock 1 to stall the processor for one additional MCLK cycle. The assertion of nWAIT does not affect the clock for the D[31:0] latches, so data can be loaded into these latches while nWAIT is asserted.

At the falling edge of clock 2, the first halfword is transferred on D[15:0]. The memory controller drives a value of 0bxx11 onto the byte latch signals BL[1:0], indicating that the lower two bytes of the 32-bit processor data bus are valid. BL[1] controls the latch on D[15:8], and BL[0] controls the latch on D[7:0]. Since the data corresponding to BL[3:2] will be loaded in the following clock, the value on these signals is ignored by the processor.

The memory controller drives the second halfword of data at the rising edge of clock 3. The controller asserts the BL[3:2] signals to load the second halfword into the data bus latches. In this cycle, the memory controller must drive a value of 0b00 onto BL[1:0] so that the halfword loaded during the first cycle is preserved.

13.4 Locked Cycles

Locked cycles are generated when the swap (SWP) or swap byte (SWPB) instructions are executed. These instructions provide a mechanism for atomic load and store operations (locked cycles). Figure 21 shows an example of locked cycles with pipelined addressing. The timing of the LOCK signal is the same as A[31:0].

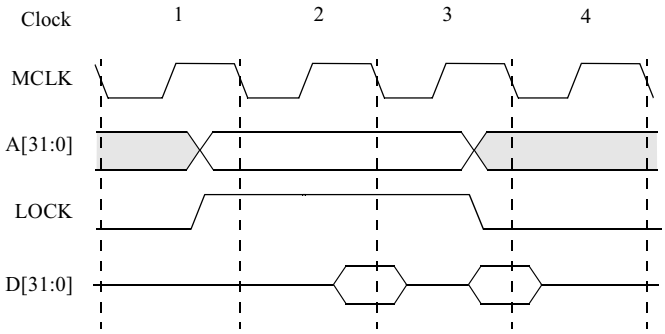

Figure 21. Locked Read and Write Cycles

13.5 Memory Prefetch

Memory prefetching is performed on a per-instruction basis. Each prefetch instruction fetches four words, or one cache line, from memory. Where data is prefetched from is determined by the *Rd* field of the instruction, which contains the memory address.

13.6 Burst Transactions

The pT-110 performs sequential read operations during a line fill operation. The processor places a new address on the bus for each 32-bits of data to be retrieved and asserts the SEQ signal, indicating to the memory controller that this address is part of a burst sequence and should not be interrupted. With a line size of 16-bytes, four read operations are required to perform a cache line fill.

During a cache line fill operation, the pT-110 processor does not fetch the critical word first. Therefore, the CPU must wait until the critical word is returned from memory. Cache lines are fetched in-order with the progression of the two low-order address bits always being 00, 01, 10, 11. For example, if the critical word that caused the cache miss is word 2, the memory transfers word 0, followed by word 1, 2, and 3. In this case the processor must wait for words 0 and 1 to be copied to the cache before gaining access to word 2. However, the CPU does not have to wait for the critical word to actually be written to the cache before gaining access to it. Rather, a bypass mechanism allows the critical word to be written to the cache and forwarded to the CPU simultaneously.

The pT-110 processor does not support burst write operations. If four sequential write addresses appear on the bus, this is a coincidence and simply means that the write buffer was able to empty without other transactions being inserted between the write transactions.

14 Instruction Set Overview

The pT-110 instruction set is divided into 32-bit and 16-bit instructions.

14.1 32-bit Instructions

Table 23 lists the 32-bit instructions.

Table 23. pT-110 32-bit Instruction Set

Mnemonic	Name	Description
Branch Instructions		
B	Branch	Performs an unconditional branch. The program flow is interrupted and begins execution at the target address.
BL	Branch and Link	Performs a conditional branch. The program flow is interrupted if the condition is met. If the condition is met, program flow resumes at the target address.
BX	Branch and Exchange Instruction Set	This instruction is used to branch between 32-bit instruction mode and 16-bit instruction mode.
Data Processing Instructions		
ADC	Add with Carry	Adds the value of the shifter operand and the value of the carry flag to the value stored in register Rn, then stores the result to register Rd.
ADD	Add	Adds the value of the shifter operand to the value stored in register Rn, then stores the result to register Rd.
AND	Logical AND	Performs a logical AND function on the value stored in register Rn with the value of the shifter operand. The result is then stored to register Rd.
BIC	Logical Bit Clear	Performs a logical AND function on the value stored in register Rn with the complement of the value of the shifter operand. The result is then stored to register Rd.
CMN	Compare Negative	Compares one arithmetic value with the negative of another arithmetic value and sets the appropriate condition flags.
CMP	Compare	Compares two arithmetic values and sets the appropriate condition flags.
EOR	Logical Exclusive OR	Performs a logical Exclusive-OR function between the value stored in register Rn, and the value of the shifter operand. The result is stored to register Rd.
MOV	Move	Moves a value from one register to another.
MNV	Move Negative	Moves the logical 1's complement of the value of the shifter operand to register Rd.
ORR	Logical OR	Performs a logical OR function between the value in register Rn with the value of the shifter operand. The result is stored to register Rd.
RSB	Reverse Subtract	Subtracts the value in register Rn with the value of the shifter_operand field. The result is stored to register Rd.
RSC	Reverse Subtract with Carry	Subtracts the value in register Rn and the value of the NOT (carry flag) with the value of the shifter_operand field. The result is stored to register Rd.
SBC	Subtract with Carry	Subtracts the value of the shifter_operand field and the value of the NOT (carry flag) from the value stored in register Rn. The result is written to register Rd.
SUB	Subtract	Subtracts the value of the shifter_operand field from the value stored in register Rn. The result is written to register Rd.

Table 23. pT-110 32-bit Instruction Set (Continued)

Mnemonic	Name	Description
TEQ	Test Equivalence	Performs a logical Exclusive-OR of two operands to determine if they have the same sign. The result does not affect the V-flag.
TST	Test	Used to determine the state of each bit in a register. The condition code flags are updated if at least one bit in the register being tested is set.
Multiply Instructions		
MLA	Multiply Accumulate	Multiplies signed or unsigned operands to produce a 32-bit result. This result is then added to a third operand and the result is written to register Rd.
MUL	Multiply	Multiplies signed or unsigned operands to produce a 32-bit result.
SMLAL	Signed Multiply Accumulate Long	Multiplies two signed values (stored in registers Rm and Rs) to produce a 64-bit result. This result is then added to a 64-bit value stored in two general purpose registers. The result is then stored to two general purpose registers. Bits 31:0 are stored to one register (RdLo), while bits 63:32 are stored to another register (RdHi).
SMULL	Signed Multiply Long	Multiplies two signed values (stored in registers Rm and Rs) to produce a 64-bit result. The result is then stored to two general purpose registers. Bits 31:0 are stored to one register (RdLo), while bits 63:32 are stored to another register (RdHi).
UMLAL	Unsigned Multiply Accumulate Long	Multiplies two unsigned values (stored in registers Rm and Rs) to produce a 64-bit result. This result is then added to a 64-bit value stored in two general purpose registers. The result is then stored to two general purpose registers. Bits 31:0 are stored to one register (RdLo), while bits 63:32 are stored to another register (RdHi).
UMULL	Unsigned Multiply Long	Multiplies two unsigned values (stored in registers Rm and Rs) to produce a 64-bit result. The result is then stored to two general purpose registers. Bits 31:0 are stored to one register (RdLo), while bits 63:32 are stored to another register (RdHi).
Status Register Access Instructions		
MRS	Move SR to General Purpose Register	Moves the value of the CSPR register (or the appropriate SPSR register) into the general purpose register file.
MSR	Move General Purpose Register to SR	Moves the value of the general purpose register file to the CSPR register (or the appropriate SPSR register).
Load / Store Instructions		
LDM(1)	Load Multiple	Loads some of all of the general purpose registers from sequential memory locations determined by the starting and ending addresses of the operation. This instruction can be used for block load operations.
LDM(2)	User Registers Load Multiple	Loads some of all of the general purpose registers from sequential memory locations determined by the starting and ending addresses of the operation. This instruction allows for the loading of User mode registers while the processor is in Privileged mode.
LDM(3)	Load Multiple and Restore CPSR	Loads some of all of the general purpose registers from sequential memory locations determined by the starting and ending addresses of the operation. In this instruction the contents of the SPSR register for the current mode are copied to the CSPR register. This instruction can be used as an exception return vehicle, of for restoring saved registers.
LDR	Load Word	Allows 32-bit memory data to be loaded into the general purpose register file.

Table 23. pT-110 32-bit Instruction Set (Continued)

Mnemonic	Name	Description
LDRB	Load Byte	Allows 8-bit memory data to be loaded into the general purpose register file.
LDRBT	Load Byte with User Mode Privilege	This instruction is used by the exception handler in Privileged mode to emulate a memory access instruction that would normally be executed in User mode. The instruction loads a byte from memory, zero-extends the byte to 32-bits, and writes the result to register Rd.
LDRH	Load Unsigned Halfword	Allows 16-bit unsigned memory data to be loaded into the general purpose register file.
LDRSB	Load Signed Byte	Allows 8-bit memory data to be loaded into the general purpose register file.
LDRSH	Load Signed Halfword	Allows 16-bit signed memory data to be loaded into the general purpose register file.
LDRT	Load Word with User Mode Privilege	This instruction is used by the exception handler in Privileged mode to emulate a memory access instruction that would normally be executed in User mode. The instruction loads a 32-bit word from memory and writes the result to register Rd.
STM(1)	Store Multiple	Stores a subset of the general purpose register file to sequential location in memory determined by the start_address and end_address fields. This instruction is typically used in User mode.
STM(2)	User Registers Store Multiple	Stores a subset of the general purpose register file to sequential location in memory determined by the start_address and end_address fields. This instruction is typically used to store User mode registers to memory when the processor is operating in a Privileged mode.
STR	Store Word	Stores a 32-bit data word from the general purpose register file to memory.
STRB	Store Byte	Stores an 8-bit data byte from the general purpose register file to memory.
STRBT	Store Byte with User Mode Privilege	Stores an 8-bit data byte from the general purpose register file to memory. This instruction is typically used by an exception handler in Privileged mode to emulate a memory access that would normally occur in User mode.
STRH	Store Halfword	Stores a 16-bit data value from the general purpose register file to memory.
STRT	Store Word with User Mode Privilege	Stores a 32-bit data word from the general purpose register file to memory. This instruction is typically used by an exception handler in Privileged mode to emulate a memory access that would normally occur in User mode.
Coprocessor Instructions		
CDP	Coprocessor Data Operations	The pT-110 does not support any coprocessor that responds to this instruction. The CP15 coprocessor is accessed using only the MRC and MCR instructions listed below. Execution of this instruction results in an Undefined Instruction exception.
LDC	Load Coprocessor Register	The pT-110 does not support any coprocessor that responds to this instruction. The CP15 coprocessor is accessed using only the MRC and MCR instructions listed below. Execution of this instruction results in an Undefined Instruction exception.
MCR	Move From Register to Coprocessor	Moves data from a general purpose register to a coprocessor register.
MRC	Move from Coprocessor to Register	Moves data from a coprocessor register to a general purpose register.

Table 23. pT-110 32-bit Instruction Set (Continued)

Mnemonic	Name	Description
STC	Store Coprocessor Register	The pT-110 does not support any coprocessor that responds to this instruction. The CP15 coprocessor is accessed using only the MRC and MCR instructions listed above. Execution of this instruction results in an Undefined Instruction exception.
Software Interrupt Instruction		
SWI	Software Interrupt	Execution of this instruction causes a software interrupt.
Data Swap Instruction		
SWP	Swap	Swaps a 32-bit word between a general purpose register and a specified location in memory.
SWPB	Swap Byte	Swaps an 8-bit byte between a general purpose register and a specified location in memory.

14.2 16-bit Instructions

Table 24 lists the 16-bit instructions.

Table 24. pT-110 16-bit Instruction Set

Mnemonic	Name	Description
Branch Instructions		
B(1)	Branch	Performs a conditional branch to the target address. The target address is loaded into the PC only if the condition is met.
B(2)	Unconditional Branch	Performs an unconditional branch to the target address.
BL	Branch and Link	Performs an unconditional subroutine call by loading the value of LR into the program counter (PC).
BX	Branch and Exchange Instruction Set	This instruction is used to branch between 16-bit instruction mode and 32-bit instruction mode.
Data Processing Instructions		
ADC	Add with Carry	Adds the value of register Rd and the carry flag, to the value stored in register Rm. The result is then written to register Rd.
ADD(1)	Add (Immediate)	Adds the value stored in register Rn with a 3-bit immediate value. The result is stored to register Rd.
ADD(2)	Add (Large Immediate)	Adds the value stored in register Rn with an 8-bit immediate value. The result is stored to register Rd.
ADD(3)	Add (Register)	Adds the value stored in register Rn to the value stored in register Rm. The result is stored to register Rd.
ADD(4)	Add (High Registers)	This instruction adds the value of a low register to the value of a high register, or the value of a high register to the value of a low register, or the value of a high register to the value of another high register.
ADD(5)	Add (Immediate to Program Counter)	This instruction clears two low-order bits of the program counter (PC) and adds the value to an 8-bit immediate value. The result is stored to register Rd.
ADD(6)	Add (Immediate to Stack Pointer)	Adds the value of the stack pointer with an 8-bit immediate value. The result is stored to register Rd.
ADD(7)	Increment Stack Pointer	Adds the value of the stack pointer with a 7-bit immediate value. The result is stored to register Rd.
AND	Logical AND	Performs a logical AND function between the contents of register Rm and the contents of register Rd, then stores the result back to register Rd.
ASR(1)	Arithmetic Shift Right (Immediate)	Performs an arithmetic shift right on the value stored in register Rm by an immeditate value ranging between 1 and 32. The result is stored to register Rd.
ASR(2)	Arithmetic Shift Right (Register)	Performs an arithmetic shift right on the value stored in register Rd by the least-significant byte of register Rs. The result is stored to register Rd.
BIC	Logical Bit Clear	Used to clear selected bits in a register.
CMN	Compare Negative	Compares one arithmetic value with the negative of another arithmetic value and sets the appropriate condition flags.
CMP(1)	Compare (Immediate)	Compares two arithmetic values and sets the appropriate condition flags. In this instruction an 8-bit immediate value is subtracted from the value in register Rd and the appropriate condition flags are set.
CMP(2)	Compare (Register)	Compares two arithmetic values and sets the appropriate condition flags. In this instruction the value of register Rm is subtracted from the value in register Rd and the appropriate condition flags are set.

Table 24. pT-110 16-bit Instruction Set (Continued)

Mnemonic	Name	Description
CMP(3)	Compare (High Registers)	This instruction compares the value of a low register to the value of a high register, or the value of a high register to the value of a low register, or the value of a high register to the value of another high register.
EOR	Logical Exclusive OR	Performs an Exclusive OR function between the value in register Rm and the value in register Rd. The result is then written back to register Rd and the appropriate conditions flags are set.
LSL(1)	Logical Shift Left (Immediate)	Performs a logical shift left between the value stored in register Rm with an immediate value between 0 and 31.
LSL(2)	Logical Shift Left (Register)	Performs a logical shift left between the value stored in register Rm and the least-significant byte of register Rs. The resulting value is then stored to register Rd.
LSR(1)	Logical Shift Left (Immediate)	Performs a logical shift right between the value stored in register Rm with an immediate value between 1 and 32.
LSR(2)	Logical Shift Left (Register)	Performs a logical shift right between the value stored in register Rm and the least-significant byte of register Rs. The resulting value is then stored to register Rd.
MOV(1)	Move (Immediate)	Moves an 8-bit immediate value to register Rd.
MOV(2)	Move (High Registers)	Moves the value in a low register to a high register, or the value in a high register to a low register, or the value in a high register to another high register.
MUL	Multiply	Multiplies signed or unsigned values to produce a 32-bit result.
MNV	Move NOT (Register)	Moves the logical 1's compliment of register Rn to register Rd.
NEG	Negative (Register)	Subtracts the value stored in register Rn from zero and stores the result to register Rd.
ORR	Logical OR	Performs a logical OR operation between the contents of register Rm and the contents of register Rd. The result is then stored back to register Rd and the appropriate condition flags are set.
ROR	Rotate Right (Register)	Performs a rotate right function on the contents of register Rd with the least-significant byte of register Rs. The result is then stored to register Rd.
SBC	Subtract with Carry (Register)	Subtracts the contents of register Rm and the NOT (carry flag) from the contents of register Rd. The result is then stored back to register Rd.
SUB(1)	Subtract (Immediate)	Subtracts a 3-bit immediate value from the contents of register Rn.
SUB(2)	Subtract (Large Immediate)	Subtracts an 8-bit immediate value from the contents of register Rn.
SUB(3)	Subtract (Register)	Subtracts the value in one general purpose register from the value of another general purpose register. The result is then stored to a third general purpose register.
SUB(4)	Decrement Stack Pointer	Subtracts an 7-bit immediate value from the contents of the stack pointer. The result is then written back to the stack pointer.
TST	Test	Used to determine the state of each bit in a register. The condition code flags are updated if at least one bit in the register being tested is set.
Load / Store Instructions		
LDM	Load Multiple	Allows for the block load of instructions or data from memory.

Table 24. pT-110 16-bit Instruction Set (Continued)

Mnemonic	Name	Description
LDR(1)	Load Word (Immediate Offset)	Loads a 32-bit memory data value into general purpose register Rd. This instruction always contains an offset of 0.
LDR(2)	Load Word (Register Offset)	Loads a 32-bit memory data value into general purpose register Rd. The memory location to be accessed is determined by adding the contents in register Rm to the contents of register Rn.
LDR(3)	Load Word (PC-Relative)	Loads a 32-bit memory data value into general purpose register Rd. The memory location to be accessed is determined by adding a specific value to the program counter.
LDR(4)	Load Word (SP-Relative)	Loads a 32-bit memory data value into general purpose register Rd. The memory location to be accessed is determined by adding a specific value to the stack pointer.
LDRB(1)	Load Unsigned Byte (Immediate Offset)	Loads an 8-bit unsigned memory data value into general purpose register Rd. This instruction always contains an offset of 0.
LDRB(2)	Load Unsigned Byte (Register Offset)	Loads an 8-bit unsigned memory data value into general purpose register Rd. The memory location to be accessed is determined by adding the contents in register Rm to the contents of register Rn.
LDRH(1)	Load Unsigned Halfword (Immediate Offset)	Loads a 16-bit unsigned memory data value into general purpose register Rd. This instruction always contains an offset of 0.
LDRH(2)	Load Unsigned Halfword (Immediate Offset)	Loads a 16-bit unsigned memory data value into general purpose register Rd. The memory location to be accessed is determined by adding the contents in register Rm to the contents of register Rn.
LDRSB	Load Signed Byte (Register Offset)	Loads an 8-bit signed memory data value into general purpose register Rd. The memory location to be accessed is determined by adding the contents in register Rm to the contents of register Rn.
LDRSH	Load Signed Halfword (Register Offset)	Loads a 16-bit unsigned memory data value into general purpose register Rd. The memory location to be accessed is determined by adding the contents in register Rm to the contents of register Rn.
POP	Pop Multiple	Loads a subset of the general purpose register file from sequential locations in memory.
PUSH	Push Multiple	Stores a subset of the general purpose register file from sequential locations in memory.
STM	Store Multiple	Performs a block store operation.
STR(1)	Store Word (Immediate Offset)	Stores the contents of a 32-bit general purpose register to memory. This instruction always contains an offset of 0.
STR(2)	Store Word (Register Offset)	Stores the contents of a 32-bit general purpose register to memory. The memory location to be accessed is determined by adding the contents in register Rm to the contents of register Rn.
STR(3)	Store Word (SP-Relative)	Stores the contents of a 32-bit general purpose register to memory. The memory location to be accessed is determined by adding a specific value to the stack pointer.
STRB(1)	Store Byte (Immediate Offset)	Stores the contents of a single byte in a general purpose register to memory. This instruction always contains an offset of 0.
STRB(2)	Store Byte (Register Offset)	Stores the contents of a single byte in a general purpose register to memory. The memory location to be accessed is determined by adding the contents in register Rm to the contents of register Rn.
STRH(1)	Store Halfword (Immediate Offset)	Stores the contents of a 16-bit value in a general purpose register to memory. This instruction always contains an offset of 0.

Table 24. pT-110 16-bit Instruction Set (Continued)

Mnemonic	Name	Description
STRH(2)	Store Halfword (Register Offset)	Stores the contents of a 16-bit value in a general purpose register to memory. The memory location to be accessed is determined by adding the contents in register Rm to the contents of register Rn.
Software Interrupt Instruction		
SWI	Software Interrupt	Execution of this instruction causes a software interrupt.

picoTurbo, Inc.
860 Hillview Ct. Suite 160
Milpitas, CA. 95035
(408) 586-8801
(408) 586-8802 (fax)
www.picoTurbo.com