Document Revision 1.1
Date: 03/2001

pT-100™ Processor Core Datasheet

1 Introduction

The pT-100™ core is a 32-bit RISC processor optimized for applications 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-100 processor is designed for use with applications that require ARM® (version 4T) instructions.

The pT-100 processor core implements a five stage pipeline and operates at up to 200 MHz. In the pT-100 processor, execution of the Multiply (MUL) and Multiply-Accumulate (MLA) instruction always generates a 64-bit result. The lower 32 bits are written to a General Purpose register. The upper 32 bits are written to the RdHi Coprocessor 15 (CP15) register. This feature allows a 64-bit result to be written to non-sequential registers. The RdHi register is accessed using the MCR and MRC instructions.

The pT-100 processor contains a single-cycle Multiple/Accumulate (MAC) unit that requires only one cycle to execute the Multiply (MUL) and Multiply-Accumulate (MLA) instructions when operating at frequencies up to 100 MHz.

To optimize performance, the pT-100 supports burst transfers during instruction fetch operations from memory, allowing a steady stream of instructions to be transferred from the memory to the processor. On-chip power management logic continually monitors the flow of instructions through the pipeline. If the pipeline is empty for more than 5 clock cycles, the power management logic automtically places the pT-100 processor in a low power state.

The pT-100 hardware interface is compatible with the interface used on the ARM7TDMI core.

2 Features

- 32-bit RISC architecture
- Up to 200 MHz clock rate (0.18 micron process)
- 5-stage pipeline with full support of AMBA and TDMI busses
- Single-cycle MAC at frequencies up to 100 MHz
- Single-cycle MAC uses clock gating to improve performance and reduce power
- Support for both Supervisor mode and User mode
- Burst transfers on instruction fetch operations
- On-chip Power Management unit disables the core when there is no instruction in the pipeline for more than 5 clock cycles
- Fully testable design with built-in JTAG debug capability
- Ability to provide control to other peripherals

3 Block Diagram

Figure 1 shows a block diagram of the pT-100 core. The following subsections describe each block in the diagram.

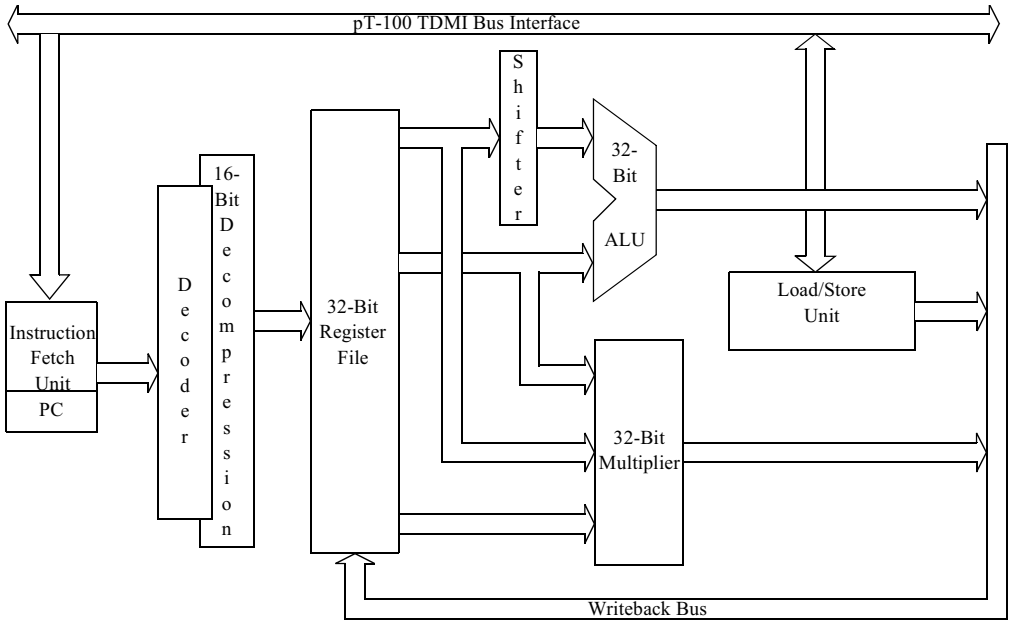

Figure 1. pT-100 Block Diagram

3.1 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.

3.2 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.

3.3 16-bit Decompression

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

3.4 32-bit Register File

The *32-bit General Purpose (GP) Register File* stores the operands and results of a computation. The pT-100 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 3.

- 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.

3.5 Shifter

The Shifter performs logical and arithmetic shifting based on the type of instruction being executed. The pT-100 instruction set incorporates certain shift operations into the instructions and hence does not require a separate shift operation to be performed.

3.6 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.

3.7 32-bit Multiplier

The pT-100 processor contains a 32-bit multiplier that performs signed and unsigned multiply and multiply-accumulate operations. The pT-100 requires only a single cycle to execute the MUL and MLA instructions.

The pT-100 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-100 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.

3.8 Load Store Unit

The pT-100 processor contains a load/store unit that controls the loading and storing of data between the Bus Interface Unit (BIU) and the processor core.

4 Modes of Operation

The pT-100 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.

4.1 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.

4.2 Privileged Modes

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

4.2.1 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.

4.2.2 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.

4.2.3 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.

4.2.4 Abort Mode

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

- External Instruction Abort
- External Data Abort

An external data or instruction abort is initiated when external logic asserts the ABORT pin to the pT-100. 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.

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.

4.2.5 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 Pipeline

The pT-100 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 2.

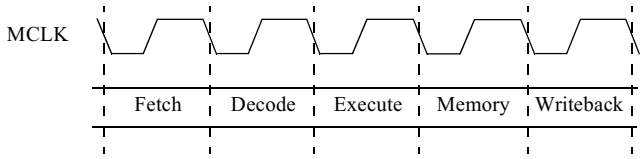

Figure 2. pT-100 Pipeline Block Diagram

5.1 Fetch Stage

During the Fetch stage the Instruction Fetch Unit retrieves the instructions from memory and passes them to the decoder block.

5.2 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.

5.3 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 processor cycle to complete.

5.4 Memory Stage

In the Memory stage external memory is accessed and store data is written to the Load/Store Unit.

5.5 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.

5.6 Cycle Timings

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

Table 2. pT-100 Cycle Times

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

6 Register Set

The pT-100 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 3 shows a diagram of the general purpose register set.

Table 3. pT-100 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
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 3, 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.

6.1 Link Register

In Table 3 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 4.

Table 4. 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

Table 4. Return Address Loaded in the Link Register (Continued)

Event Type	Return Address	Return Instruction
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 4 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.

6.2 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.

6.3 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 3.

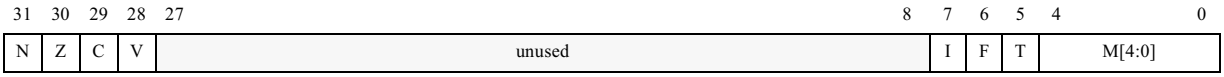

Figure 3. CPSR/SPSR Format

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.

6.4 CP15 Control Registers

In the pT-100 processor, the CP15 Control register set consists of 1 register that stores the upper 64 bits of a Multiply (MUL) or Multiply-Accumulate (MLA) instruction. The CP15 coprocessor is accessed using only the MCR and MRC coprocessor register transfer instructions.

Note that CP15 is the only coprocessor supported by the pT-100 processor. Accesses to all other coprocessors result in an exception.

The CP15 registers are summarized in Table 5.

Table 5. CP15 Register Map

Number	Name	Description
0 - 10	Reserved	Reserved.
11	RdHi	Stores the upper 32 bits of the 64-bit result from a multiply or multiply-accumulate operation.
12-15	Reserved	Reserved.

7 Exception Processing

The pT-100 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 4. 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 6.

Table 6. pT-100 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

7.1 Reset Exception

The Reset exception is taken whenever the 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.

7.2 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.

7.3 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.

7.4 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.

7.5 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.

7.6 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.

7.7 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.

7.8 Exception Priorities

The pT-100 processor prioritizes the above exceptions as shown in Table 7.

Table 7. pT-100 Exception Priorities

Exception Type	Priority Level
Reset	1 (highest)
Data Abort	2
FIQ	3
IRQ	4
Prefetch Abort	5
Undefined Instruction, SWI	6 (lowest)

8 Signal Descriptions

Table 8 lists the pT-100 processor signals.

Table 8. pT-100 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.
BIGEND	I	*Big-Endian Enable*—driven high by external logic to select the Big-Endian mode, or low to indicate Little-Endian.
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.
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.

Table 8. pT-100 Signal Descriptions (Continued)

Name	Type	Description
HOLDREQ	I	*Hold Request*—Asserted by an external bus master to request ownership of the bus. The pT-100 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. These signals are 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	I	*External Clock Input*—this signal is used to drive the internal processor clock.
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.
nOPC	O	*Opcode Fetch*—indicates that the processor is performing an opcode fetch. The nOPC signal is valid simultaneously with A[31:0].
nRESET	I	*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.
nRW	O	*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].
SEQ	O	*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.
TBIT	O	*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.
nTRANS	O	*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].
nWAIT	I	*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.
nTRST	I	*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.
TCK	I	*Test Clock*—Test clock input for the JTAG TAP.
TMS	I	*Test Mode Select*—Test mode select input for the JTAG TAP.
TDI	I	*Test Data In*—Test data input for the JTAG TAP.
TDO	O	*Test Data Out*—Test data output for the JTAG TAP.
DBGRQ	I	*Debug Request*—This is a level sensitive input which, when HIGH, causes the pT-100 processor core to enter debug state after executing the current instruction. This allows external hardware to force the pT-100 processor core into the debug state.
DBGACK	O	*Debug Acknowledge*—This signal is an output from the pT-100 processor core which, when HIGH, indicates that the core is in the debug state.

9 Bus Interface Unit

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

9.1 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 9.

Table 9. 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 4.

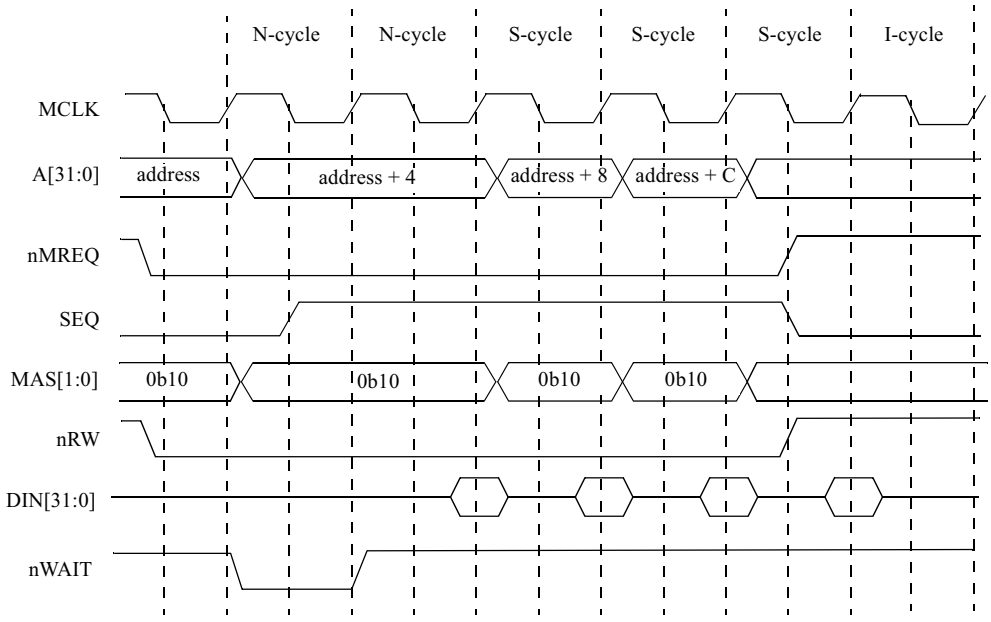

Figure 4. Relationship of MRQ and SEQ During a 32-bit Read Operation

In Figure 4 the processor asserts nMREQ and deasserts SEQ at the start of the cycle indicate a non-sequential cycle (N-cycle) as shown in Table 9 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 9 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 9 above.

9.2 Address Pipelining

Figure 5 and Figure 6 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 5, 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.

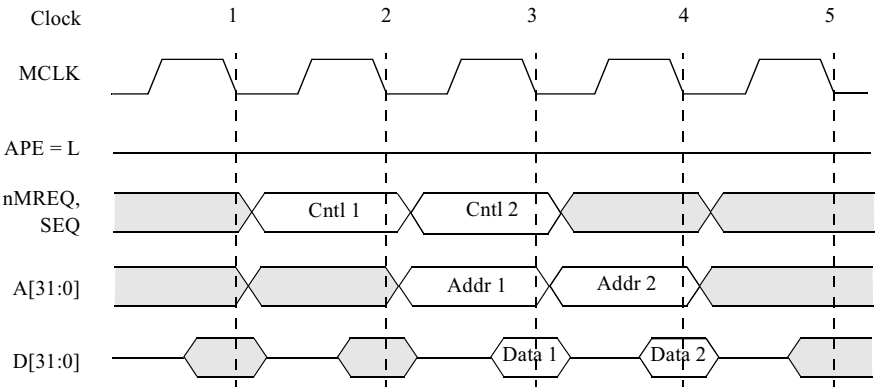

Figure 5. Address Pipelining Disabled

In Figure 6 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.

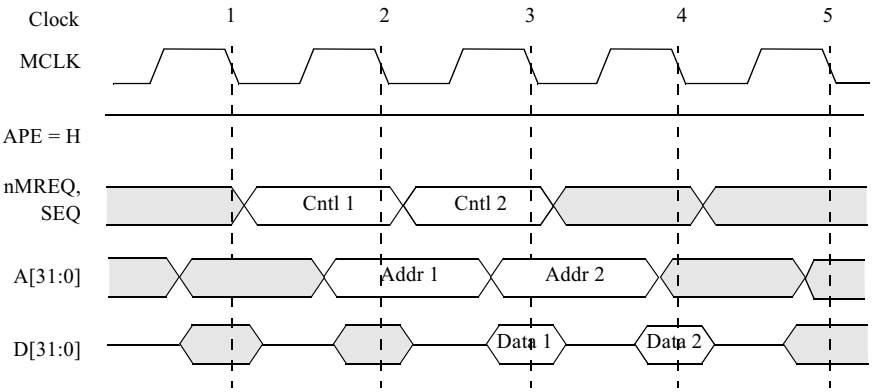

Figure 6. Address Pipelining Enabled

9.3 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 7 shows an example of assembling a word from two halfwords.

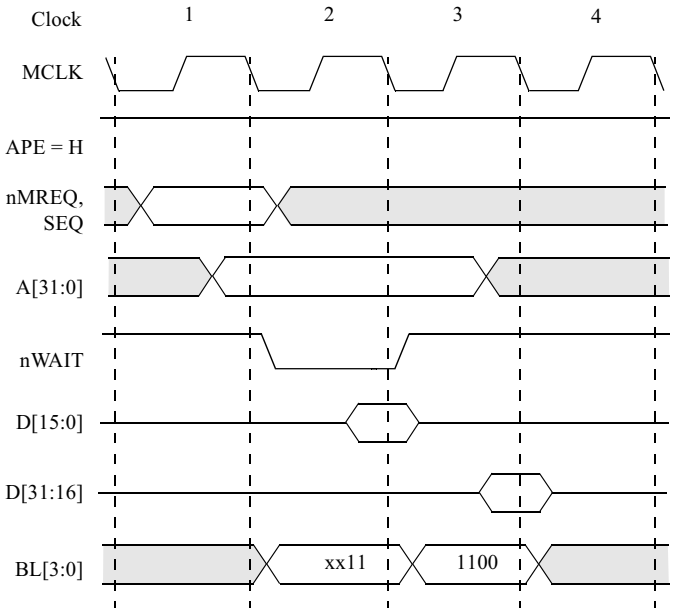

Figure 7. Assembling a Word from Two Halfwords

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.

9.4 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 8 shows an example of locked cycles with pipelined addressing. The timing of the LOCK signal is the same as A[31:0].

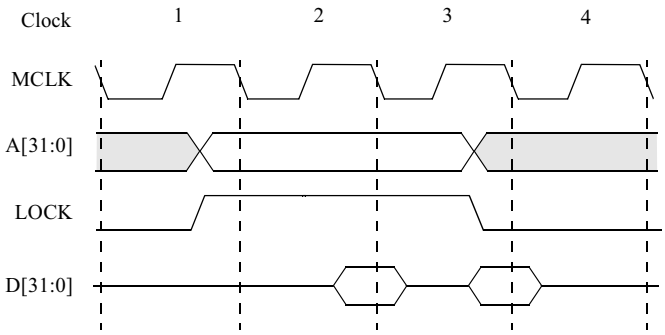

Figure 8. Locked Read and Write Cycles

9.5 Memory Prefetching and Burst Transactions

The pT-100 performs sequential read operations to instruction memory. The processor places a new address on the bus for each 32-bits of data to be retrieved and asserts the nOPC and SEQ signals, indicating to the memory controller that this address is part of a instruction fetch burst sequence and should not be interrupted. The pT-100 processor supports only instruction memory burst read operations. Data memory burst reads and burst writes are not supported.

10 Instruction Set Overview

The pT-100 instruction set is divided into 32-bit and 16-bit instructions.

10.1 32-bit Instructions

Table 10 lists the 32-bit instructions.

Table 10. pT-100 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.

Table 10. pT-100 32-bit Instruction Set

Mnemonic	Name	Description
SUB	Subtract	Subtracts the value of the shifter_operand field from the value stored in register Rn. The result is written to register Rd.
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 10. pT-100 32-bit Instruction Set

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-100 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-100 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.

Table 10. pT-100 32-bit Instruction Set

Mnemonic	Name	Description
MRC	Move from Coprocessor to Register	Moves data from a coprocessor register to a general purpose register.
STC	Store Coprocessor Register	The pT-100 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.

10.2 16-bit Instructions

Table 11 lists the 16-bit instructions.

Table 11. pT-100 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.

Table 11. pT-100 16-bit Instruction Set

Mnemonic	Name	Description
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.
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 11. pT-100 16-bit Instruction Set

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 11. pT-100 16-bit Instruction Set

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.

© 2001 picoTurbo Inc.
Printed in U.S.A
All Rights Reserved

This publication is provided as is. picoTurbo Inc. (the “Company”) does make any warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Information in this document is provided solely to enable system developers to use the Companies products. Unless specifically set forth herein, there are no express or implied patent, copyright, or any other intellectual property rights or licenses granted hereunder to design or fabricate integrated circuits based on the information in this document. The Company does not warrant that the contents of this publication, whether individually or as one or more groups, meets anyone’s requirements, or that the document is error-free. This publication may include technical inaccuracies or typographical errors. Changes may be made to the information herein, and these changes may be incorporated in new editions of this publication.

picoTurbo™ is a trademark of picoTurbo, Inc.
pT-100™, pT-110™, pT-110Ax™, and pT-120™ are trademarks of picoTurbo, Inc.

ARM® and THUMB® are registered trademarks of Advanced RISC Machines, Inc.
All other trademarks and registered trademarks are the property of their respective companies.

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