PDtrace™ Interface Specification

Document Number: MD00136
Revision 3.01
May 14, 2003

MIPS Technologies, Inc.
1225 Charleston Road
Mountain View, CA 94043-1353

Template: B1.08, Built with tags: 2B

Table of Contents

Chapter 1 About This Book 1
1.1 Typographical Conventions 1
1.1.1 Italic Text 1
1.1.2 Bold Text 1
1.1.3 Courier Text 1
1.2 UNPREDICTABLE and UNDEFINED 1
1.2.1 UNPREDICTABLE 1
1.2.2 UNDEFINED 2
1.3 Special Symbols in Pseudocode Notation 2
1.4 For More Information 4

Chapter 2 Overview 5
2.1 Processor Modes 6
2.2 Subsetting 6

Chapter 3 The PDtrace Interface Signals 7
3.1 PDtrace Interface Signal List 7

Chapter 4 PDtrace Interface Description 17
4.1 Trace Output Signals 17
4.1.1 The Instruction Completion Status Signal 17
4.1.2 Start of Tracing 19
4.1.3 Trace Synchronization 20
4.1.4 Trace Bus 20
4.1.5 Trace Overflow and Restart 20
4.1.6 Trace Type and an Example Code Fragment 21
4.1.7 Trace Mode 24
4.1.8 Data Order Signal 24
4.1.9 Instruction-Data Map Signal 26
4.1.10 Trace Timing Example 26
4.2 Trace Input Signals 27
4.3 Tracing Multi-Issue and High-Performance Processors 28
4.3.1 Background on High Performance Processors 28
4.3.2 The Basic Tracing Methodology 28
4.3.3 Coordinating the Instruction Completion Trace with the Address/Data Trace 30
4.3.4 Out-of-Order Loads and Stores in the Multi-Pipe Core 31
4.3.5 Tagging Instructions that Issue Together 31
4.3.6 Miscellaneous 31
4.4 Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints 32
4.4.1 The *TraceBPC* Register (CP0 Register 23, Select 4) 32
4.5 Software Trace Control 33
4.5.1 Coprocessor 0 Trace Registers 34
4.6 Trace Enabling/Disabling Condition 39
4.7 Tracing During Processor Mode Changes 40
4.8 Tracing Store Conditionals 40
4.9 Tracing MIPS16e Macro Instructions 41
4.10 Tracing MIPS16e Extend Instructions 41

Chapter 5 Trace Compression 43
5.1 PC tracing 43
5.2 Load or Store Address Tracing 43
5.3 Load or Store Data Tracing 43

5.4 Using Early PDO_TEnd Assertion 44

Appendix A Revision History 45

A.1 Revision History 45

List of Figures

Figure 2-1: Illustration of a PC and Data Trace Flow 5
Figure 2-2: Config3 Register Format 6
Figure 4-1: A Sample Pipeline And The PDO_InsComp Trace Point 18
Figure 4-2: Illustration of a Pipeline and Trace Tap Points 19
Figure 4-3: A TMOAS Trace Record 23
Figure 4-4: An Example of Load Data Bypassing an Earlier Store 25
Figure 4-5: PDtrace interface timing example 27
Figure 4-6: An Example Showing the Coordination of Instructions and their Data 30
Figure 4-7: *TraceBPC* Register Format 33
Figure 4-8: *TraceControl* Register Format 34
Figure 4-9: *TraceControl2* Register Format 37
Figure 4-10: *UserTraceData* Register Format 38

List of Tables

Table 1-1: Symbols Used in Instruction Operation Statements 2
Table 2-1: Config3 Register Field Descriptions 6
Table 3-1: PDtrace Interface Signals 7
Table 4-1: Example Code Fragment With Some PDtrace Signal Value 22
Table 4-2: A TMOAS Trace Record Field Descriptions 23
Table 4-3: Load Order Example 24
Table 4-4: Data (Load/Store) Order Example 26
Table 4-5: Example Code Fragment Showing the Graduation Cycle and Trace Bus Number 29
Table 4-6: TraceBPC Register Field Descriptions 33
Table 4-7: A List of Coprocessor 0 Trace Registers 34
Table 4-8: TraceControl Register Field Descriptions 34
Table 4-9: TraceControl2 Register Field Descriptions 37
Table 4-10: UserTraceData Register Field Descriptions 39
Table A-1: Revision History 45

About This Book

1.1 Typographical Conventions

This section describes the use of *italic*, **bold** and `courier` fonts in this book.

1.1.1 Italic Text

- is used for *emphasis*
- is used for *bits*, *fields*, *registers*, that are important from a software perspective (for instance, address bits used by software, and programmable fields and registers), and various *floating point instruction formats*, such as *S*, *D*, and *PS*
- is used for the memory access types, such as *cached* and *uncached*

1.1.2 Bold Text

- represents a term that is being **defined**
- is used for **bits** and **fields** that are important from a hardware perspective (for instance, **register** bits, which are not programmable but accessible only to hardware)
- is used for ranges of numbers; the range is indicated by an ellipsis. For instance, **5..1** indicates numbers 5 through 1
- is used to emphasize **UNPREDICTABLE** and **UNDEFINED** behavior, as defined below.

1.1.3 Courier Text

`Courier` fixed-width font is used for text that is displayed on the screen, and for examples of code and instruction pseudocode.

1.2 UNPREDICTABLE and UNDEFINED

The terms **UNPREDICTABLE** and **UNDEFINED** are used throughout this book to describe the behavior of the processor in certain cases. **UNDEFINED** behavior or operations can occur only as the result of executing instructions in a privileged mode (i.e., in Kernel Mode or Debug Mode, or with the CP0 usable bit set in the Status register). Unprivileged software can never cause **UNDEFINED** behavior or operations. Conversely, both privileged and unprivileged software can cause **UNPREDICTABLE** results or operations.

1.2.1 UNPREDICTABLE

UNPREDICTABLE results may vary from processor implementation to implementation, instruction to instruction, or as a function of time on the same implementation or instruction. Software can never depend on results that are **UNPREDICTABLE**. **UNPREDICTABLE** operations may cause a result to be generated or not. If a result is generated, it is **UNPREDICTABLE**. **UNPREDICTABLE** operations may cause arbitrary exceptions.

UNPREDICTABLE results or operations have several implementation restrictions:

- Implementations of operations generating **UNPREDICTABLE** results must not depend on any data source (memory or internal state) which is inaccessible in the current processor mode
- **UNPREDICTABLE** operations must not read, write, or modify the contents of memory or internal state which is inaccessible in the current processor mode. For example, **UNPREDICTABLE** operations executed in user mode must not access memory or internal state that is only accessible in Kernel Mode or Debug Mode or in another process
- **UNPREDICTABLE** operations must not halt or hang the processor

1.2.2 UNDEFINED

UNDEFINED operations or behavior may vary from processor implementation to implementation, instruction to instruction, or as a function of time on the same implementation or instruction. **UNDEFINED** operations or behavior may vary from nothing to creating an environment in which execution can no longer continue. **UNDEFINED** operations or behavior may cause data loss.

UNDEFINED operations or behavior has one implementation restriction:

- **UNDEFINED** operations or behavior must not cause the processor to hang (that is, enter a state from which there is no exit other than powering down the processor). The assertion of any of the reset signals must restore the processor to an operational state

1.3 Special Symbols in Pseudocode Notation

In this book, algorithmic descriptions of an operation are described as pseudocode in a high-level language notation resembling Pascal. Special symbols used in the pseudocode notation are listed in Table 1-1.

Table 1-1 Symbols Used in Instruction Operation Statements

Symbol	Meaning
←	Assignment
=, ≠	Tests for equality and inequality
||	Bit string concatenation
x^y	A *y*-bit string formed by *y* copies of the single-bit value *x*
b#n	A constant value *n* in base *b*. For instance 10#100 represents the decimal value 100, 2#100 represents the binary value 100 (decimal 4), and 16#100 represents the hexadecimal value 100 (decimal 256). If the "b#" prefix is omitted, the default base is 10.
$x_{y..z}$	Selection of bits *y* through *z* of bit string *x*. Little-endian bit notation (rightmost bit is 0) is used. If *y* is less than *z*, this expression is an empty (zero length) bit string.
+, −	2's complement or floating point arithmetic: addition, subtraction
*, ×	2's complement or floating point multiplication (both used for either)
div	2's complement integer division
mod	2's complement modulo
/	Floating point division
<	2's complement less-than comparison
>	2's complement greater-than comparison
≤	2's complement less-than or equal comparison

Table 1-1 Symbols Used in Instruction Operation Statements

Symbol	Meaning
≥	2's complement greater-than or equal comparison
nor	Bitwise logical NOR
xor	Bitwise logical XOR
and	Bitwise logical AND
or	Bitwise logical OR
GPRLEN	The length in bits (32 or 64) of the CPU general-purpose registers
GPR[x]	CPU general-purpose register *x*. The content of *GPR[0]* is always zero.
FPR[x]	Floating Point operand register *x*
FCC[CC]	Floating Point condition code CC. *FCC[0]* has the same value as *COC[1]*.
FPR[x]	Floating Point (Coprocessor unit 1), general register *x*
CPR[z,x,s]	Coprocessor unit *z*, general register *x*, select *s*
CCR[z,x]	Coprocessor unit *z*, control register *x*
COC[z]	Coprocessor unit *z* condition signal
Xlat[x]	Translation of the MIPS16 GPR number *x* into the corresponding 32-bit GPR number
BigEndianMem	Endian mode as configured at chip reset (0 →Little-Endian, 1 → Big-Endian). Specifies the endianness of the memory interface (see LoadMemory and StoreMemory pseudocode function descriptions), and the endianness of Kernel and Supervisor mode execution.
BigEndianCPU	The endianness for load and store instructions (0 → Little-Endian, 1 → Big-Endian). In User mode, this endianness may be switched by setting the *RE* bit in the *Status* register. Thus, BigEndianCPU may be computed as (BigEndianMem XOR ReverseEndian).
ReverseEndian	Signal to reverse the endianness of load and store instructions. This feature is available in User mode only, and is implemented by setting the *RE* bit of the *Status* register. Thus, ReverseEndian may be computed as (SR_{RE} and User mode).
LLbit	Bit of **virtual** state used to specify operation for instructions that provide atomic read-modify-write. *LLbit* is set when a linked load occurs; it is tested and cleared by the conditional store. It is cleared, during other CPU operation, when a store to the location would no longer be atomic. In particular, it is cleared by exception return instructions.
I:, **I+n**:, **I-n**:	This occurs as a prefix to *Operation* description lines and functions as a label. It indicates the instruction time during which the pseudocode appears to "execute." Unless otherwise indicated, all effects of the current instruction appear to occur during the instruction time of the current instruction. No label is equivalent to a time label of **I**. Sometimes effects of an instruction appear to occur either earlier or later — that is, during the instruction time of another instruction. When this happens, the instruction operation is written in sections labeled with the instruction time, relative to the current instruction **I**, in which the effect of that pseudocode appears to occur. For example, an instruction may have a result that is not available until after the next instruction. Such an instruction has the portion of the instruction operation description that writes the result register in a section labeled **I+1**. The effect of pseudocode statements for the current instruction labelled **I+1** appears to occur "at the same time" as the effect of pseudocode statements labeled **I** for the following instruction. Within one pseudocode sequence, the effects of the statements take place in order. However, between sequences of statements for different instructions that occur "at the same time," there is no defined order. Programs must not depend on a particular order of evaluation between such sections.

Table 1-1 Symbols Used in Instruction Operation Statements

Symbol	Meaning
PC	The *Program Counter* value. During the instruction time of an instruction, this is the address of the instruction word. The address of the instruction that occurs during the next instruction time is determined by assigning a value to *PC* during an instruction time. If no value is assigned to *PC* during an instruction time by any pseudocode statement, it is automatically incremented by either 2 (in the case of a 16-bit MIPS16 instruction) or 4 before the next instruction time. A taken branch assigns the target address to the *PC* during the instruction time of the instruction in the branch delay slot.
PABITS	The number of physical address bits implemented is represented by the symbol PABITS. As such, if 36 physical address bits were implemented, the size of the physical address space would be $2^{PABITS} = 2^{36}$ bytes.
FP32RegistersMode	Indicates whether the FPU has 32-bit or 64-bit floating point registers (FPRs). In MIPS32, the FPU has 32 32-bit FPRs in which 64-bit data types are stored in even-odd pairs of FPRs. In MIPS64, the FPU has 32 64-bit FPRs in which 64-bit data types are stored in any FPR. In MIPS32 implementations, **FP32RegistersMode** is always a 0. MIPS64 implementations have a compatibility mode in which the processor references the FPRs as if it were a MIPS32 implementation. In such a case **FP32RegisterMode** is computed from the FR bit in the *Status* register. If this bit is a 0, the processor operates as if it had 32 32-bit FPRs. If this bit is a 1, the processor operates with 32 64-bit FPRs. The value of **FP32RegistersMode** is computed from the FR bit in the *Status* register.
InstructionInBranchDelaySlot	Indicates whether the instruction at the Program Counter address was executed in the delay slot of a branch or jump. This condition reflects the *dynamic* state of the instruction, not the *static* state. That is, the value is false if a branch or jump occurs to an instruction whose PC immediately follows a branch or jump, but which is not executed in the delay slot of a branch or jump.
SignalException(exception, argument)	Causes an exception to be signaled, using the exception parameter as the type of exception and the argument parameter as an exception-specific argument). Control does not return from this pseudocode function - the exception is signaled at the point of the call.

1.4 For More Information

Various MIPS RISC processor manuals and additional information about MIPS products can be found at the MIPS URL:

http://www.mips.com

Comments or questions on the MIPS™ Architecture or this document should be directed to

Director of MIPS Architecture
MIPS Technologies, Inc.
1225 Charleston Road
Mountain View, CA 94043

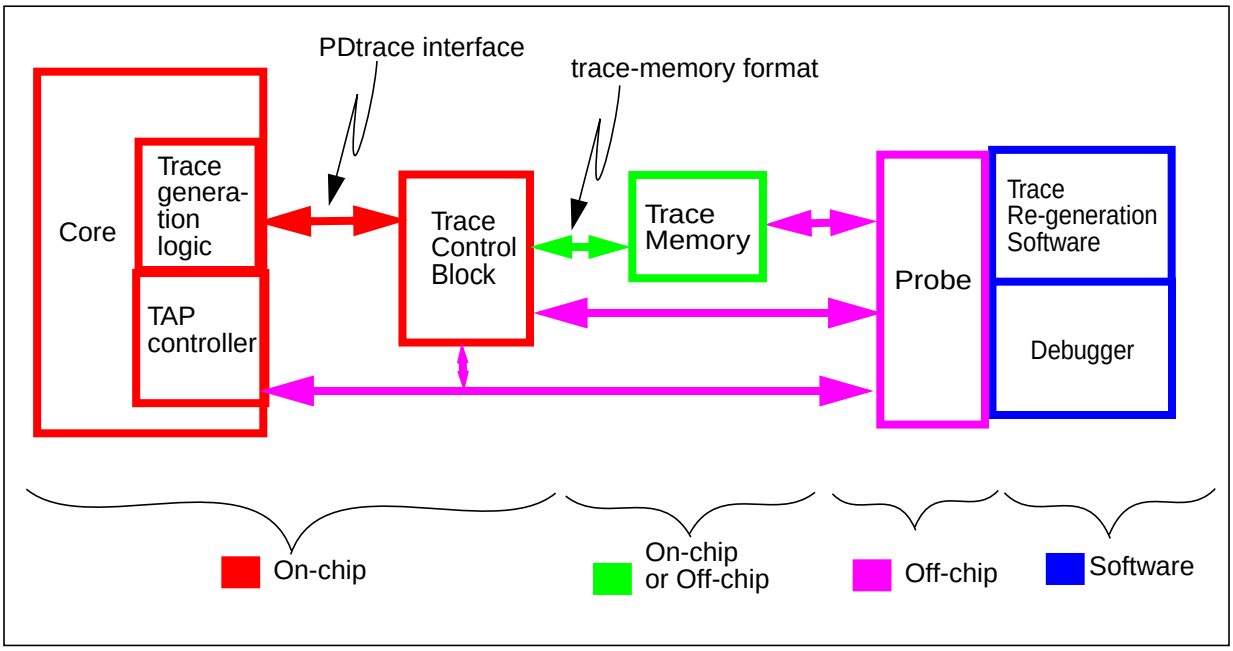
or via E-mail to architecture@mips.com.

Overview

This document contains a specification of the interface from the core used to capture PC and data trace (PDtrace™) information from each pipeline within the processor. A trace block external to the core (but on-chip), captures this PDtrace information and writes it to trace memory. The trace memory may be either on-chip or off-chip based on user requirements. The trace information written to memory is compressed and assumes that post-processing software has access to the static program image to reconstruct the dynamic program flow. Compression reduces the number of signals (hence pins) required to gather this information and also reduces the trace size.

Figure 2-1 illustrates one possible configuration for trace capture and post-analysis using software. The figure shows a core with trace generation logic and a TAP controller. This core is connected to a trace control block (TCB) via the PDtrace interface and via the TAP controller (since the TCB implements and uses TAP registers). Both these units are on-chip. The trace memory associated with the trace control block can either be located on-chip, or off-chip. An on-chip trace buffer will be smaller and will be writable by the TCB at higher speeds, while an off-chip trace memory can be much larger and is written via the potentially slower pin interface out of the core. Probe hardware and software connects to the TCB and the TAP controller via the chip's pin interface and allows debugger software to start, stop, and examine program execution traces.

Figure 2-1 Illustration of a PC and Data Trace Flow



The rest of this document describes the PDtrace interface in detail. This document together with the Trace Control Block Specification document serve three functions: (1) they provide a specification of the trace interface for the core designer, (2) they provide sufficient detail for a third-party vendor to build the trace control block, and (3) they provide sufficient details to design and code a post-processing software module for trace re-construction.

Implementation of PDtrace is optional for a given MIPS-compatible processor. Whether a core or processor implements PDtrace is indicated by a bit in the Coprocessor 0 Config3 register as shown in Figure 2-2 and Table 2-1.

Note that if a core or processor does not implement EJTAG, then the PDtrace tracing logic can still be implemented.

Figure 2-2 *Config3* Register Format

Table 2-1 *Config3* Register Field Descriptions

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
	31:1	As per the MIPS32 and MIPS64 Architecture specifications			
TL	0	This bit is used to indicate the presence of tracing logic in the processor. 0 : No tracing logic implemented 1 : Tracing logic implemented	R	Preset	Required

2.1 Processor Modes

The PDtrace specification allows tracing to be enabled or disabled based on various processor modes. This section precisely describes these modes, and the terminology is then used later in the document.

```
DebugMode ← (Debug_DM = 1)
ExceptionMode ← (not DebugMode) and ((Status_EXL = 1) or (Status_ERL = 1))
KernelMode ← (not (DebugMode or ExceptionMode)) and (Status_KSU = 2#00)
SupervisorMode ← (not (DebugMode or ExceptionMode)) and (Status_KSU = 2#01)
UserMode ← (not (DebugMode or ExceptionMode)) and (Status_KSU = 2#10)
```

2.2 Subsetting

The PDtrace specification allows four levels of subsetting. Within each level, all features required to support the level must be implemented. The allowable subsets are:

- No PDtrace implemented
- PDtrace with PC tracing only
- PDtrace with PC and load and store address tracing only
- PDtrace with PC, load and store address, and load and store data tracing

The specific subset implemented by a processor or core can be determined by reading the TL bit (0) of the ***Config3*** register (see Table 2-1) and the ImpSubset bits (6:5) in the ***TraceControl2*** register (see Table 4-9 on page 37).

In addition, Trace Trigger from EJTAG Hardware breakpoints (see Section 4.4, "Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints" on page 32) is optional.

The PDtrace Interface Signals

All signals are assumed to be asserted high unless otherwise noted. The signal direction “Out” refers to a signal that is output from the processor core, and “In” signals are those that are input to the processor core. The “PDO_” prefix to the signal names is used to uniquely identify the signals as belonging to the PDtrace Output interface. And the “PDI_” prefix is used to identify the PDtrace Input signals. Signals that have been repeated in the “Signal Name” column with a “_n” prefix are PDO_ signals that are to be duplicated for multi-issue processors.

3.1 PDtrace Interface Signal List

Table 3-1 PDtrace Interface Signals

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>Pclk</td><td></td><td>Processor clock, used by the core and the trace control block.</td></tr>
<tr><td>PDO_IamTracing</td><td>Out</td><td>The core uses this signal to validate all the other Out signals. The external trace control block cannot always predict if the trace data from the core is valid or not valid since tracing depends on core execution status such as the processor mode and also since tracing can be controlled by software running on the core.

This signal is used for all the _n signals, and is not duplicated.</td></tr>
<tr><td>PDO_InsComp[2:0]

PDO_InsComp_n[2:0]</td><td>Out</td><td>Instruction completion status signal. The values are interpreted as follows:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>000</td><td>No instruction completed this cycle (NI)</td></tr>
<tr><td>001</td><td>Instruction completed this cycle (I)</td></tr>
<tr><td>010</td><td>Instruction completed this cycle was a load (IL)</td></tr>
<tr><td>011</td><td>Instruction completed this cycle was a store (IS)</td></tr>
<tr><td>100</td><td>Instruction completed this cycle was a PC sync (IPC)</td></tr>
<tr><td>101</td><td>Instruction branched this cycle (IB)</td></tr>
<tr><td>110</td><td>Instruction branched this cycle was a load (ILB)</td></tr>
<tr><td>111</td><td>Instruction branched this cycle was a store (ISB)</td></tr>
</table>

A "No Instruction" (NI) can happen due to a pipeline stall or when the instruction was killed (due to an exception).

The three encoding (101, 110, 111) for branched instruction indicates a discontinuity in the PC value for the associated instruction. Note that it is only when the new PC can not be predicted from the static program flow that it is traced.

The IPC value is used for the periodic output of the full PC value for synchronization. The tracing hardware should ensure that this is not done on an unpredictable branch, load, or store instruction.</td></tr>
</table>

Table 3-1 PDtrace Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDO_MIPS16
PDO_MIPS16_n</td><td>Out</td><td>When asserted, this signal indicates that the current instruction specified in PDO_InsComp is a MIPS16e instruction. When de-asserted, the processor is not executing a MIPS16e instruction.

This signal (along with the PDO_MIPS16Ins signal) is used by the TCB to compute the current PC value. Hence this is irrelevant externally and not traced to memory. Note that since external software has access to the program image, it can always know whether an instruction is a MIPS16e instruction or not.

This is an optional signal for PDtrace specification revisions less than 03.00. This signal is only relevant if the processor also implements the MIPS16e ASE, and is not required otherwise. If a processor provides this signal, it is optional whether a TCB accepts this signal and uses it.</td></tr>
<tr><td>PDO_MIPS16Ins[1:0]
PDO_MIPS16Ins_n[1:0]</td><td>Out</td><td>This signal accompanies the PDO_MIPS16 signal and is used to indicate the type of MIPS16e instruction. Like PDO_MIPS16 this is optional, but must be implemented if PDO_MIPS16 is implemented.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>00</td><td>Is executing a MIPS16e instruction that is not a MACRO instruction and is not extended.</td></tr><tr><td>01</td><td>Is executing a MIPS16e instruction that is not a MACRO instruction and is extended.</td></tr><tr><td>10</td><td>Is executing a MIPS16e MACRO instruction.</td></tr><tr><td>11</td><td>Reserved</td></tr></table></td></tr>
<tr><td>PDO_AD[15:0] or
PDO_AD[31:0]
PDO_AD_n[15:0] or
PDO_AD_n[31:0]</td><td>Out</td><td>The address or data value is transmitted on this bus. The actual values must be correlated using the PDO_TType signal described below. It is recommended that a 64-bit processor core implement at least 32 bits for improved tracing capability.

A multi-cycle transaction sends the least-significant bits first, followed by the more-significant bits.

When the transmitted data width is less than the width of the bus, the data is transmitted on the least-significant bits of the bus. There is no necessity to indicate the validity since the post-analyzing software knows the width of the data. (For example, a LB implies one byte of data). The upper bits of the bus must be sign extended to allow the TCB to truncate the upper bits and hence avoid tracing unneeded bits to memory.</td></tr>
</table>

Table 3-1 PDtrace Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDO_TType[2:0]
PDO_TType_n[2:0]</td><td>Out</td><td>Specifies the transmission type for the transaction on the PDO_AD lines. The valid types are:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>000</td><td>No transmission this cycle (NT)</td></tr>
<tr><td>001</td><td>Transmitting the PC (TPC)</td></tr>
<tr><td>010</td><td>Transmitting the load address (TLA)</td></tr>
<tr><td>011</td><td>Transmitting the store address (TSA)</td></tr>
<tr><td>100</td><td>Transmitting the load/store data value (TD)</td></tr>
<tr><td>101</td><td>Transmitting the processor mode, the 8-bit ASID, and the SYNC bit. This is triggered by either a change in the processor mode, by a software write to the EntryHi register, or a trace synchronization operation. (TMOAS). If the processor does not implement the standard TLB-based MMU, it is UNPREDICTABLE whether a write to the EntryHi register triggers a TMOAS operation. (See Figure 4-3 on page 23).</td></tr>
<tr><td>110</td><td>Transmitting user-defined trace record - type 1 (TU1)</td></tr>
<tr><td>111</td><td>Transmitting user-defined trace record - type 2 (TU2)</td></tr>
</table></td></tr>
<tr><td>PDO_TEnd
PDO_TEnd_n</td><td>Out</td><td>Indicates the last cycle of the current transaction on the PDO_AD bus. This signal can be asserted in the same cycle that a transaction is started, implying that the particular transaction only took one cycle to complete.

In a multi-issue core, the PDO_TEnd signals are synchronized for all the PDO_AD_n transmissions associated with instructions that graduate together. See Section 4.3.3, "Coordinating the Instruction Completion Trace with the Address/Data Trace" on page 30 for details.

In PDtrace revision 3.00 and higher, the processor is allowed to assert this signal early if the tracing logic determines that the upper bits of the address or data being sent on the PDO_AD bus are redundant. For example, redundant upper sign bits may be omitted and software could easily reconstruct these bits. Note that the TCB must therefore be capable of accepting an early PDO_TEnd signal for any transmission type. This early assertion of PDO_TEnd is allowed, for all the values of PDO_TMode.</td></tr>
<tr><td>PDO_TMode
PDO_TMode_n</td><td>Out</td><td>Indicates the transmission mode for the bits transmitted on PDO_AD. The mode depends on the transmission type.
<table>
<tr><th>PDO_TType</th><th>PDO_TMode</th></tr>
<tr><td>000(NT)
101(TMOAS)</td><td>Reserved</td></tr>
<tr><td>001(TPC)</td><td>0 -> delta from last PC value
1 -> compression algorithm A (full address)</td></tr>
<tr><td>010(TLA)
011(TSA)</td><td>0 -> delta from last data address of that type
1 -> compression algorithm B (full address)</td></tr>
<tr><td>100(TD)
110(TU1)
111(TU2)</td><td>0 -> Reserved
1 -> compression algorithm C (full data)</td></tr>
</table></td></tr>
</table>

Table 3-1 PDtrace Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDO_DataOrder[3:0]
PDO_DataOrder_n[3:0]</td><td>Out</td><td>This signal is used to indicate the degree of out-of-order-ness of load and store data. Using this order value allows load and store data to be traced out as it becomes available, thus avoiding the need to internally buffer data. Note that only sixteen outstanding data values are allowed because of the limitation imposed by the signal width of 4 bits. This signal takes on the following values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0000</td><td>data from oldest load/store instruction (is in-order)</td></tr>
<tr><td>0001</td><td>data from second-oldest load/store instruction</td></tr>
<tr><td>0010</td><td>data from third-oldest load/store instruction</td></tr>
<tr><td>0011</td><td>data from fourth-oldest load/store instruction</td></tr>
<tr><td>0100</td><td>data from fifth-oldest load/store instruction</td></tr>
<tr><td>0101</td><td>data from sixth-oldest load/store instruction</td></tr>
<tr><td>0110</td><td>data from seventh-oldest load/store instruction</td></tr>
<tr><td>0111</td><td>data from eighth-oldest load/store instruction</td></tr>
<tr><td>1000</td><td>data from ninth-oldest load/store instruction</td></tr>
<tr><td>1001</td><td>data from tenth-oldest load/store instruction</td></tr>
<tr><td>1010</td><td>data from eleventh-oldest load/store instruction</td></tr>
<tr><td>1011</td><td>data from twelfth-oldest load/store instruction</td></tr>
<tr><td>1100</td><td>data from thirteenth-oldest load/store instruction</td></tr>
<tr><td>1101</td><td>data from fourteenth-oldest load/store instruction</td></tr>
<tr><td>1110</td><td>data from fifteenth-oldest load/store instruction</td></tr>
<tr><td>1111</td><td>data from sixteenth-oldest load/store instruction</td></tr>
</table></td></tr>
</table>

Table 3-1 PDtrace Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDO_DataForIns[7:0]
PDO_DataForIns_n[7:0]</td><td>Out</td><td>The value indicates which pieces of data will be transmitted for the corresponding instruction in PDO_InsComp.
<table>
<tr><th>Bit #</th><th>Description</th></tr>
<tr><td>0</td><td>A value of zero indicates that the TPC sent (indicated in bit position 1 of this signal) is associated with a statically predictable address change, i.e., the external software does not need to be sent the value in TPC for accurate tracing. A value of one indicates that the TPC sent is associated with a statically unpredictable address change.

Note that TPC values for statically predictable address changes are sent only when the PDI_TraceAllBranch option is set. Since the TCB does not know whether the TPC is for a statically predictable or unpredictable address change, it uses the information in this bit to determine whether to transmit through the TPC value to trace memory.

Note that any data in internal buffers when the PDI_TraceAllBranch signal is de-asserted will have valid values for this bit position, but the TCB does not explicitly know how much data was in the buffers. Hence there will be some small period of time, a few cycles probably, when more address information may be traced than explicitly required by external software.</td></tr>
<tr><td>1</td><td>A value of 1 indicates that TPC will be sent.</td></tr>
<tr><td>2</td><td>A value of 1 indicates that TLA will be sent.</td></tr>
<tr><td>3</td><td>A value of 1 indicates that TSA will be sent.</td></tr>
<tr><td>4</td><td>A value of 1 indicates that TD will be sent.</td></tr>
<tr><td>5</td><td>A value of 1 indicates that TMOAS will be sent.</td></tr>
<tr><td>6</td><td>A value of 1 indicates that TU1 will be sent.</td></tr>
<tr><td>7</td><td>A value of 1 indicates that TU2 will be sent.</td></tr>
</table></td></tr>
<tr><td>PDO_TrigI[N:0]</td><td>Out</td><td>This one-hot vector indicates which of the N+1 implemented EJTAG hardware instruction breakpoints caused a trigger. The instruction causing the trigger is indicated on the corresponding PDO_InsComp bus, if tracing has been turned on. Note that EJTAG restricts the maximum number of implementable hardware instruction breakpoints to 15.</td></tr>
<tr><td>PDO_TrigD[N:0]</td><td>Out</td><td>This one-hot vector indicates which of the N+1 implemented EJTAG hardware data breakpoints caused a trigger. The instruction causing the trigger is not necessarily the one on the PDO_InsComp bus since data triggers may be imprecise. Note that EJTAG restricts the maximum number of implementable hardware data breakpoints to 15.</td></tr>
<tr><td>PDO_TrigOn</td><td>Out</td><td>This bit is asserted if at least one trigger in PDO_TrigI[N:0] or PDO_TrigD[N:0] turns trace on. (See Section 4.4, "Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints" on page 32).</td></tr>
<tr><td>PDO_TrigOff</td><td>Out</td><td>This is asserted if no trigger turns trace on (i.e., PDO_TrigOn is not asserted), and at least one of the indicated triggers in PDO_TrigI[N:0] or PDO_TrigD[N:0] turns trace off. (See Section 4.4, "Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints" on page 32).</td></tr>
</table>

Table 3-1 PDtrace Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDO_Overflow</td><td>Out</td><td>This signals an internal FIFO overflow error in the core and implies the following:
• the current transmission is to be abandoned in the current cycle
• the FIFO is emptied so that previously collected trace information in the FIFO is lost
• a new transmission begins in the next cycle with a TMOAS and a full PC address</td></tr>
<tr><td>PDO_ValidModes[1:0]</td><td>Out</td><td>This signal specifies the subset of tracing that is supported by the processor (see Section 2.2, "Subsetting" on page 6).
<table><tr><th>Encoding</th><th>Meaning</th></tr><tr><td>00</td><td>PC tracing only</td></tr><tr><td>01</td><td>PC and load and store address tracing only</td></tr><tr><td>10</td><td>PC, load and store address, and load and store data</td></tr><tr><td>11</td><td>Reserved</td></tr></table></td></tr>
<tr><td>PDO_IssueTag_n[5:0]</td><td>Out</td><td>This signal is used in multi-issue processors and it is signaled with PDO_InsComp_n. In multi-issue processors, instructions that issue together are assigned a matching tag value, specified by this signal value.
A six bit internal counter increments each cycle, and the instructions that issue in that cycle are assigned the counter value When the maximum counter value is reached, it simply restarts at zero.
This feature facilitates the performance debugging of code schedulers for high-end processors. These tag values are available every cycle, but it is anticipated that the TCB will trace this to memory only when specially requested by the user.</td></tr>
<tr><td>PDI_TCBPresent</td><td>In</td><td>When asserted this indicates that the TCB hardware is present and connected to the core’s tracing logic. Hence the core can consider the other PDI_ signals to be valid.</td></tr>
<tr><td>PDI_TraceOn</td><td>In</td><td>This is the signal asserted by the external trace block into the core that states whether tracing is globally turned on or off. It is expected that this signal be continuously asserted to turn on tracing.
0 : tracing off
1 : tracing is turned on</td></tr>
</table>

Table 3-1 PDtrace Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDI_TraceMode[2:0]</td><td>In</td><td>When tracing is turned on, this signal specifies what information is to be traced by the core:
<table>
<tr><th>Mode</th><th>Trace Mode</th></tr>
<tr><td>000</td><td>Trace PC</td></tr>
<tr><td>001</td><td>Trace PC and load address</td></tr>
<tr><td>010</td><td>Trace PC and store address</td></tr>
<tr><td>011</td><td>Trace PC and both load/store addresses</td></tr>
<tr><td>100</td><td>Trace PC and load data (optional for all PDtrace specification revisions less than 03.00)</td></tr>
<tr><td>101</td><td>Trace PC and load address and data</td></tr>
<tr><td>110</td><td>Trace PC and store address and data</td></tr>
<tr><td>111</td><td>Trace PC and both load/store address and data</td></tr>
</table>
The PDO_ValidModes signal determines which of these encoding are supported by the processor. The operation of the processor is UNPREDICTABLE if PDI_TraceMode is set to a value which is not supported by the processor.</td></tr>
<tr><td>PDI_G</td><td>In</td><td>The global bit, which if asserted to 1, implies that all processes are to be traced. If 0, then trace data is sent only for a process that matches PDI_ASID[7:0]. If the processor does not implement the standard TLB-based MMU, this signal is ignored by the processor and is treated as if it were asserted.</td></tr>
<tr><td>PDI_ASID[7:0]</td><td>In</td><td>When the global bit is 0, only the process whose ASID matches this ASID value will be traced. If the processor does not implement the standard TLB-based MMU, this signal is ignored by the processor.</td></tr>
<tr><td>PDI_U</td><td>In</td><td>Enables tracing in User Mode (see Section 2.1, "Processor Modes" on page 6). This enables tracing only if the PDI_TraceOn is also asserted or the hardware breakpoint trace triggers on, and either the PDI_G bit is set or the PDI_ASID matches the current process ASID.</td></tr>
<tr><td>PDI_S</td><td>In</td><td>Enables tracing in Supervisor Mode (for those processors that implement Supervisor Mode), otherwise, this signal is not required (see Section 2.1, "Processor Modes" on page 6). This enables tracing only if the PDI_TraceOn is also asserted or the hardware breakpoint trace triggers on, and either the PDI_G bit is set or the PDI_ASID matches the current process ASID.</td></tr>
<tr><td>PDI_K</td><td>In</td><td>Enables tracing in Kernel Mode (see Section 2.1, "Processor Modes" on page 6). This enables tracing only if the PDI_TraceOn is also asserted or the hardware breakpoint trace triggers on, and either the PDI_G bit is set or the PDI_ASID matches the current process ASID.</td></tr>
<tr><td>PDI_E</td><td>In</td><td>Enables tracing when in Exception Mode (see Section 2.1, "Processor Modes" on page 6). This enables tracing only if the PDI_TraceOn is also asserted or the hardware breakpoint trace triggers on, and either the PDI_G bit is set or the PDI_ASID matches the current process ASID.</td></tr>
<tr><td>PDI_DM</td><td>In</td><td>Enables tracing in Debug Mode (see Section 2.1, "Processor Modes" on page 6). This feature is useful to debug the debug handler code via the EJTAG and TAP controller port.</td></tr>
<tr><td>PDI_InhibitOverflow</td><td>In</td><td>This signal is used by the external trace block to indicate to the core that the core pipeline should be back-pressured (and stalled) instead of allowing the trace FIFO to overflow and hence lose trace information.</td></tr>
</table>

Table 3-1 PDtrace Interface Signals (Continued)

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>PDI_StallSending</td><td>In</td><td>When asserted, this signal is used by the external trace block to indicate to the core that it must stop transmitting trace information in the next cycle. This request may be essential when the trace control block is in imminent danger of over-running its internal trace buffer.

In the cycle when the signal is asserted, the value on all the PDO_ signals are valid and must be captured by the TCB.

In the cycle after the one where the core sees an assertion of this signal the core must not transmit any valid trace information on any of the PDO_ output signal bits (including PDO_InsComp).

In the cycle after the TCB de-asserts this signal again, PDtrace PDO_ signals are valid and must be captured by the TCB. (Note that some processors cannot arbitrarily stall their pipeline on any given cycle. In this situation, the implementation on the processor side must provide sufficient buffering to hold trace information until the pipeline can be stalled).</td></tr>
<tr><td>PDI_SyncOffEn</td><td>In</td><td>This signal is an enable signal for the PDI_SyncPeriod, PDI_TBImpl, and PDI_OffChipTB signals. When asserted, the core latches these values. This signal, and the signals which it controls must be asserted before tracing can begin.</td></tr>
<tr><td>PDI_SyncPeriod[2:0]</td><td>In</td><td>This signal is used to set the synchronization period bits in the TraceControl2 register. The value specifies the period (in cycles) for sending synchronization information.
<table>
<tr><th>SyncPeriod</th><th>Period (in cycles) for On-chip memory</th><th>Period (in cycles) for Off-chip memory</th></tr>
<tr><td>000</td><td>2^2</td><td>2^7</td></tr>
<tr><td>001</td><td>2^3</td><td>2^8</td></tr>
<tr><td>010</td><td>2^4</td><td>2^9</td></tr>
<tr><td>011</td><td>2^5</td><td>2^{10}</td></tr>
<tr><td>100</td><td>2^6</td><td>2^{11}</td></tr>
<tr><td>101</td><td>2^7</td><td>2^{12}</td></tr>
<tr><td>110</td><td>2^8</td><td>2^{13}</td></tr>
<tr><td>111</td><td>2^9</td><td>2^{14}</td></tr>
</table>
The “On-chip” column value is used when the trace data is being written to an on-chip trace buffer. Conversely, the “Off-chip” column is used when the trace data is being written to an off-chip trace buffer. This selection is made by the value of PDI_OffChipTB signal, which is subsequently loaded into the TBU field in the TraceControl2 register.</td></tr>
<tr><td>PDI_TBImpl</td><td>In</td><td>When this signal is a 1, the TCB has implemented both an on-chip and an off-chip trace buffer, and the PDI_OffChipTB signal indicates to which the trace is currently being written. When this signal is a 0, the PDI_OffChipTB signal indicates which buffer is implemented. This value is written into the TraceControl2 CP0 register (as the TBI bit). It is optional for the TCB to provide this signal to the core logic for all TCB implementations compatible to PDtrace specifications less than 03.00.</td></tr>
<tr><td>PDI_OffChipTB</td><td>In</td><td>When one, this signal indicates that the trace data is being sent off-chip to an external trace memory. When zero, this indicates an on-chip trace buffer. The value of this signal to the core changes how the core interprets the trace synchronization period bits. This signal value is written into the TraceControl2 CP0 register (as the TBU bit).</td></tr>
</table>

Table 3-1 PDtrace Interface Signals (Continued)

Signal Name	Direction	Description
PDI_TraceAllBranch	In	When asserted, the core's tracing logic will emit PC values for all taken branches encountered in the execution stream, including all conditional and unconditional, predictable and unpredictable branches. When de-asserted, the core reverts to normal tracing mode.

PDtrace Interface Description

A program executes sequentially through instructions within a basic block followed by a jump (or branch) to the head (first instruction) of the next basic block. To reconstruct the dynamic execution path of the program, it is sufficient to provide the post-analyzer with the PC address of the head of each basic block. Even this is not always necessary, since it may be possible in some instances to statically predict the value of the branch target, provided there is a separate indication for the taken branch. Hence, PC addresses need to be traced only when it is not possible to statically predict the branch target PC. For the MIPS32 and MIPS64 instruction sets, the statically unpredictable instructions are JR and JALR (for branch target address), and BEQ, BNE, BGEZ, etc. (for branch condition). Other statically unpredictable PC changes happen with taken exceptions and return from exceptions (ERET and DERET). To enable the post-analyzer to re-synchronize itself with the program execution, the PC value is also output at predictable intervals and synchronization periods.

The tracing mechanism can be controlled either by hardware via the input signals from the external trace block or controlled by software. Software control is possible via bits in a Coprocessor 0 register. In addition, there is one extra bit in the register used to select control between the hardware and software mechanisms. The reset value of this bit selects hardware tracing control. If software wants to take over tracing, it can set all the tracing control bits in the Coprocessor 0 register to the desired value and then set the select bit to transfer tracing control to the bits in the register.

This chapter describes the details of the general tracing mechanism, including hardware and software trace control.

4.1 Trace Output Signals

4.1.1 The Instruction Completion Status Signal

The PDO_InsComp[2:0] signal from the core's tracing logic is used as an indicator of completed instructions and their type in the processor's pipeline. Once tracing is initiated, a valid PDO_InsComp value must be transmitted every cycle (except when the TCB has asserted the PDI_StallSending signal).

NI (No Instruction complete) is used when the internal pipe is stalled for some reason or the other, and no instruction completes in that cycle. It is also used when tracing has been turned off, but the internal FIFO is still emptying trace data out to the TCB on the PDO_AD bus.

Instructions within a basic block are indicated with a I, IL, or IS value. The I is used to indicate a simple instruction that is neither a load nor a store. The IL is used to indicate a load instruction and the IS is used to indicate a store instruction.

Unpredictable (and predictable) changes in the PC value is indicated as a branch-type instruction, i.e., IB, ILB, or ISB. Note that the first instruction in the basic block is always indicated as a branch instruction. When this first instruction is

a load or a store, then the PDO_InsComp[2:0] takes values ILB or ISB respectively, to indicate the combined condition of the branch and load or store.

Figure 4-1 A Sample Pipeline And The PDO_InsComp Trace Point

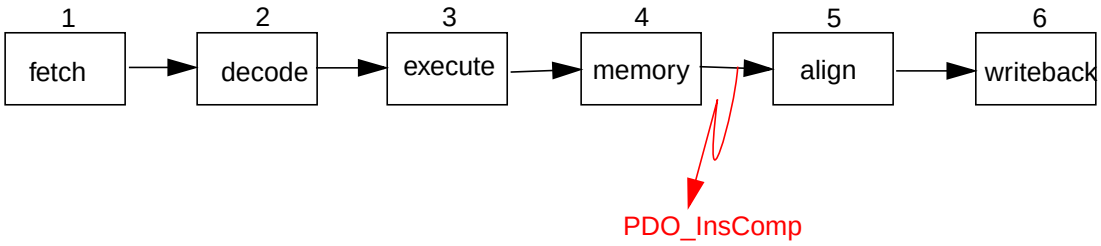


Figure 4-1 shows an example of when the PDO_InsComp signal might be output by the core tracing logic, with respect to the processor pipeline implementation. This example pipeline has six stages. They are: “fetch”, “decode”, “execute”, “memory”, “align”, and “write back”. The PDO_InsComp signal is output after the memory stage. That is, the instruction goes through the pipeline and is output after the last stage when the instruction **must** complete and can no longer be killed. In the example shown, this is after stage 4. This will differ, of course, with each pipeline implementation.

Some instructions might have to transmit more information for a complete picture of the program execution. For instance, a branch indicator might have to transmit the PC value if the unpredictability lies in the branch target address. If the unpredictability was in the branch condition (i.e., determining if the branch is taken or not), then the branch target PC value need not be transmitted; it suffices to indicate that it was a “taken” branch using the appropriate PDO_InsComp value.

The list below summarizes the three possible branching options, and the corresponding PDO_InsComp and PC tracing action:

- When the branch is unconditional and the branch target is predictable, IB, ILB, or ISB is used for the PDO_InsComp value, and the PC is not traced out.
- When the branch is conditional, and the branch target is predictable, IB, ILB, or ISB is used only when the branch is taken. The PC is not traced out.
- When the branch is conditional or unconditional, and the branch target is unpredictable, IB, ILB, or ISB is used and the PC is traced (using TPC for the PDO_TType signal).

There are four possible circumstances that cause the PC to be transmitted in the PDtrace MIPS architecture. They are:

1. after a JR or JALR instruction.
2. after a control transfer to an exception handler.
3. after a return from exception (ERET or DERET instruction).
4. the PC is traced out periodically for software synchronization of trace with the static program image.

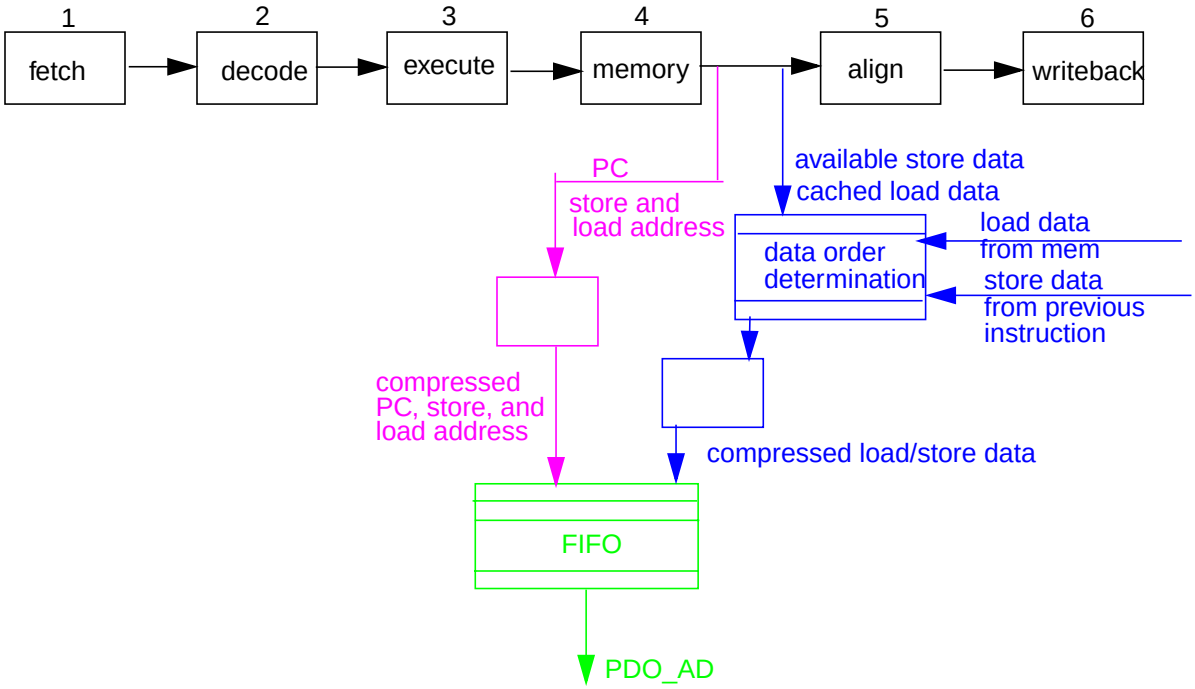
When the PDO_InsComp value indicates a store in the completing instruction with an IS, then the store address and data might have to be transmitted if the user requires these to be traced. With a ISB the PC value might also need to be traced out. In this situation, the PC value is sent first, followed by the store address, and finally the store data if it is immediately available.

An ILB is similar, and might require the tracing of the PC value as well as the load address and the load data. The PC value is sent first. If the load hits in the cache, then this works like the store described above, i.e., the PC value is sent, followed immediately by the load address and data.

The load or store data may not be immediately available. This can happen if the load misses in the cache and must be fetched from memory, or the store data is pending the completion of a previous (long latency) instruction that is computing the data value. In this situation, the load or store instruction is still indicated with the appropriate PDO_InsComp value of either IL, ILB, IS, or ISB. If the PC value needs to be sent, then it is sent first, followed by the load or store address, but the sending of the corresponding data is deferred until it is available. While the processor is waiting for this data, other instructions may complete in the pipeline and are indicated by the appropriate PDO_InsComp values. When the data is available, it is traced out as soon as possible by the processor using the appropriate PDO_DataOrder value to indicate the out-of-orderness of the data (see Section 4.1.8, "Data Order Signal" on page 24).

Figure 4-2 shows, for the hypothetical pipeline, the points at which the different pieces of information are tapped out to be traced. The PC value and the store address and load address are tapped out after stage 4. If the load hits in the primary cache, or the store data is available, then this information may be completely traced out at that point. If not, only the data's address is sent and the data value is traced out when it becomes available.

Figure 4-2 Illustration of a Pipeline and Trace Tap Points



4.1.2 Start of Tracing

When tracing is first started, or when it is re-started after a break, some basic information is first output that allows external software to identify the trace start point in the static program image, and make some reasonable conclusions about the processor mode at the start of tracing. The first record that is traced is a TMOAS (see PDO_TType in Table 3-1 on page 7 and Figure 4-3 on page 23). This trace record type shows the processor mode and the ASID value of the currently executing processor. This record is followed by a transmission of the full PC value for the first instruction traced. This first traced instruction must use a IB, ILB, or ISB PDO_InsComp value so that the external software can correlate the PC transmission with the PDO_InsComp value. In addition, if load/store address tracing is turned on, then the first encountered load or store instruction will send the full address instead of a delta value. Note that the synchronization counter is reset to the value in $\text{TraceControl2}_{\text{SyP}}$ when tracing is started.

4.1.3 Trace Synchronization

Once the full PC value, or the full address for the load/store instruction has been sent during the start of tracing, subsequent traced addresses may all be delta values. Hence, it is possible that occasionally the external software will lose track of the current execution point in the static program image. To fix this potential problem, the tracing logic will send periodic synchronization information.

The synchronization tracing function is triggered when the internal synchronization counter overflows based on the synchronization period bits as set in the *TraceControl2* CP0 register. Similar to the start of tracing, when the synchronization period is reached, a TMOAS record is first inserted in the PDO_AD tracing sequence, followed by a full PC value accompanied by an IPC for PDO_InsComp. To simplify this IPC transaction type, the hardware must ensure that the instruction used to synchronize the PC value is neither a branch, a load or a store instruction. Hence, the synchronization period is an approximate point, where the transmission of the IPC can be delayed by a few instructions until an instruction is found that is neither a branch, load, or a store instruction. Note that the TMOAS associated with synchronization is sent only when the IPC instruction has been identified, to prevent other PDO_TType records between the TMOAS and the full PC transmission for the synchronization. At this juncture, if load/store addresses are not being traced, then this completes all the transmissions needed for synchronization. If load/store addresses are being traced, then the first load or store instruction encountered after the IPC transmission sends a full address value, rather than a delta. This completes the synchronization process. Note that the synchronization counter is reset to the value in $\text{TraceControl2}_{\text{SyP}}$ once the IPC has been sent.

Note that the TMOAS record that is sent for synchronization uses a value of 1 for the SYNC bit field (see Figure 4-3). This is an aid used by external software to synchronize the PDO_InsComp stream and the data stream on the PDO_AD bus. To use this bit to synchronize, external software will look in the trace buffer for the first IPC entry, when it finds one, it starts looking in the trace buffer from the current cycle onwards for the first TMOAS record with the SYNC bit set to one. The first PC value following this TMOAS record will be a full PC transmission that corresponds to the IPC entry.

4.1.4 Trace Bus

When a PC, load/store address, or load/store value is to be traced, they are sent on the PDO_AD[15:0] (or PDO_AD[31:0]) signal bus, accompanied by an appropriate PDO_TType signal. Since the width of this bus may not be adequate to transmit the entire address or data in one cycle, each transaction can take multiple cycles to transmit. The PDO_TEnd signal is used to indicate the last cycle of the current transaction.

A FIFO at the core is used to hold pending transactions and values. The draining of the FIFO happens independently of the PDO_InsComp signal. Hence, there is some cycle delay between the tracing of the PDO_InsComp signal and the corresponding address or data on the PDO_AD bus. To provide external software with some means of synchronizing the two streams, the TMOAS record that is sent during a synchronization trace has a special SYNC bit that is set to one, as discussed above.

4.1.5 Trace Overflow and Restart

As noted earlier, a FIFO is used to hold address and data values waiting to be transmitted over the PDO_AD bus. It is possible to have a program sequence that overflows this FIFO. In the situation that the FIFO overflows, the core trace logic will assert the PDO_Overflow signal to indicate that the current tracing is being abandoned due to a FIFO overflow. In this situation, the internal core logic abandons tracing in the current cycle, discards all entries in the FIFO, and restarts tracing from the next completed instruction in the following cycle. Note that in this situation, the first instruction to be signalled after the assertion of the PDO_Overflow signal must have its full PC value sent, so this should be treated as a IB, ILB, or ISB. An example of this can be seen in the timing diagram in Figure 4-5. Similar to a trace start or re-start situation, a TMOAS record is first sent after the overflow, and before the full PC value is transmitted.

It is possible for the entire program trace to be captured under all circumstances, and no trace records lost. This is done using the PDI_InhibitOverflow signal. When asserted, this implies that the processor core must back-pressure the pipeline and stall it without overflowing the FIFO. (Hence, if PDI_InhibitOverflow is asserted, the core must ensure that PDO_Overflow never gets asserted.) The pipeline is restarted as soon as the FIFO starts emptying again.

Note that the size of the FIFO is implementation-dependent and its size should be based on the following factors:

- If all the traced data is to be saved at all times, and the FIFO is too small for the amount of trace data, then the processor pipeline will need to be stalled more frequently. Hence, a larger FIFO would be desired. Sometimes, it may not matter whether the processor is stalled when debugging, hence a smaller FIFO might be an acceptable solution.
- The width of the processor, the width of the PDtrace PDO_AD bus and how quickly the FIFO can be drained with respect to the processor speed.
- The frequency of the processor and the frequency of the external pin interface is also a factor in how quickly the FIFO can be drained.
- The amount of data that is expected to be traced in a typical usage. That is, only PC tracing, or load/store addresses, or load/store data is also traced. Each scenario will result in more data being traced, and consequently the FIFO will fill faster and overflow sooner, and will need to be larger if the processor is not to be stalled often in order to preserve all the traced data.

4.1.6 Trace Type and an Example Code Fragment

The PDO_TType[2:0] signal is used to indicate the type of information being transmitted on the PDO_AD trace bus. A PDO_InsComp[2:0] value of IB, ILB, or ISB is output when a branch instruction is taken, and the PDO_TType[2:0] can begin with a PC transmission TPC at this same cycle or later. We will use Table 4-1 to illustrate these transmissions. This table shows an example of a MIPS assembly fragment and the values of PDO_InsComp, PDO_TType, and PDO_TEnd that will be transmitted upon completion of each instruction of the code fragment in the pipeline. Assume that tracing was begun earlier, hence the start of tracing is not shown in this code fragment. The example also assumes a 32-bit processor and a 16-bit PDO_AD bus.

As described earlier, a taken branch is always indicated with an IB transmission. But when the branch target address can be deduced from the static program image, then there is no accompanying TPC transmission. An example of this can be seen in cycle 7, where the transmission of the IB indicates the taken branch from the JAL instruction in cycle 5.

An example of an IB transmission for the PDO_InsComp value accompanied by a corresponding TPC (to transmit the statically unpredictable PC value), can be seen in cycle 10. This is triggered by the JR instruction in cycle 8. Cycle 10 is the branch target, also the first instruction of the new basic block. (Cycle 9 is the execution of the instruction in the branch delay slot). Note that the TPC transmission could be directly started on cycle 10 since the FIFO was empty.

The PDO_TEnd signal is used to indicate the end of any previously-started transmission. If the PC change value can be transmitted in a single cycle, then the PDO_TEnd signal may be transmitted in the same cycle as the PDO_TType value TPC. An example of this is seen in cycle 10. Otherwise, it may follow the required number of cycles later, for example in cycle 4, where it took 2 cycles to transmit the store address value. After a transaction is begun, until PDO_TEnd is asserted, the value of PDO_TType must stay asserted at the original value.

Note that at the processor's discretion, the PDO_TEnd signal may be used to cut off redundant sign bits from an address or data transmission. That is the transmission is terminated early and hence not all the upper bits of an address or data needs to be stored in trace memory. The reconstruction software must recognize this situation and sign-extend the address or data appropriately before use.

When a load instruction is executed, PDO_InsComp indicates this using IL and ILB, and a store is indicated using IS and ISB. The user might have requested that load and store addresses (and data) be traced. In this situation, the PDO_TType value will transmit the load address and the store address using TLA or TSA respectively.

Table 4-1 Example Code Fragment With Some PDtrace Signal Value

Cycle No.	PC	Instruction	PDO_InsComp [2:0]	PDO_TType [2:0]	PDO_TEnd
1	0x00400188	SW $6, 0xe170($1)	IS	TSA	1
2	0x0040018c	SW $4, 0xb134($28)	IS	TSA	1
3	0x00400190	SW $5, 0xb130($28)	IS	TSA	1
4	0x00400194	SW $0, 0x1c($29)	IS	TSA	0
5	0x00400198	JAL 0x418d9c	I	TSA	1
6	0x0040019c	OR $30, $0, $0	I	NT	x
7	0x00418d9c	NOP	IB	NT	x
8	0x00418da0	JR $31	I	NT	x
9	0x00418da4	NOP	I	NT	x
10	0x004001a0	JAL 0x411c40	IB	TPC	1
11	0x004001a4	NOP	I	NT	x
12	0x00411c40	JR $31	IB	NT	x
13	0x00411c44	NOP	I	NT	x
14	0x00414adc	LW $4, 0xb134($28)	ILB	TPC	0
15	0x00414ae0	BEQ $14, $0, 0x414af8	I	TPC	1
16	0x00414ae4	ADDIU $29, $29, 0xffe0	I	TLA	1
17	0x00414af8	OR $7, $0, $0	IB	TD	0
18	0x00414afc	NOP	IPC	TD	1
19	0x00414b00	ADDU $6, $6, $2	I	TMOAS	1
20	0x00414b04	OR $7, $2, $0	I	TPC	0
21	0x00414b08	SLTU $1, $2, $1	I	TPC	1

An example of store address tracing is seen in Table 4-1 at cycles 1, 2, 3, and 4. The store instruction in cycles 1, 2, and 3 take only 1 cycle to send the store address. While the store address associated with the store in cycle 4 takes 2 cycles. (Perhaps it was not possible to compress the store address to less than 16 bits in this case). Note that in this case store data is not sent, only the store address is sent, as per the user request. If store data is also being traced, then the store data is sent immediately following the store address using a TD value on the PDO_TType signal lines. If the store data is not immediately available, it is sent later with the appropriate PDO_DataOrder value.

Assume that sometime between cycle 4 and cycle 14, the user changes the desired trace output, and wants load and store data to also be sent. Hence, the load instruction LW in cycle 14 will transmit not only the address, but also the associated data. Note that sometimes the load data is not immediately available since the load might miss in the first-level cache. In this situation, the load address is transmitted immediately and the load data is sent when it becomes available. The association of the load data with the corresponding load address is done using the PDO_DataOrder signal (not shown in the table).

The ILB in cycle 14 sends the PC value in two cycles, and then sends the load address using TLA in cycle 16. The load data is then sent using TD during cycles 17 and 18. The load must have hit the cache in this example, for otherwise, the

associated load could have been separated from the instruction by an arbitrary number of cycles (required to satisfy the load miss from secondary memory).

An example of the periodic PC trace IPC for synchronization is shown in cycle 18. The required transmissions for a synchronization includes sending a record of the process ASID and processor mode. This uses the PDO_TType[2:0] value of TMOAS, as seen in cycle 19 (sent as soon as the previous TD transmission completes). This is followed by a tracing of the full PC value, which takes 2 cycles (sent during cycles 20 and 21). As discussed in Section 4.1.3, "Trace Synchronization" on page 20, since load/store address tracing is turned on, the synchronization operation is not completed until a load and store full address transmission has also been sent (not shown in Table 4-1). A load and store address transmission is always tied to a load and store instruction, respectively. The full load and store address is thus not sent until the next respective occurrence of a load and store instruction after the IPC transmission.

Figure 4-3 A TMOAS Trace Record

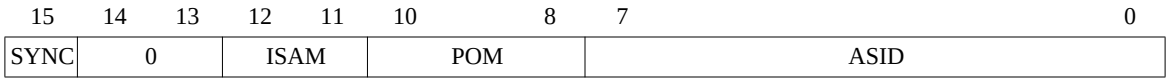


Table 4-2 A TMOAS Trace Record Field Descriptions

Fields		Description
Name	**Bits**	
SYNC	15	When 0, this record was sent when the ASID, POM, or ISAM changed. When 1, this record was sent for a synchronization event.
0	14:13	Reserved for future use
ISAM	12:11	**Value** / **Description**: 00 MIPS32; 01 MIPS64; 10 MIPS16 from MIPS32 mode; 01 MIPS16 from MIPS64 mode
POM	10:8	**Value** / **Description**: 000 Kernel Mode (EXL = 0, ERL = 0); 001 Exception Mode (EXL = 1, ERL = 0); 010 Exception Mode (EXL = don't care, ERL = 1); 011 Debug Mode; 100 Supervisor Mode; 101 User Mode; 110 Reserved; 111 Reserved
ASID	7:0	The ASID of the current process. If the processor does not implement the standard TLB-based MMU, this field is always traced as a zero because the *EntryHi* register, and hence the ASID, is not defined.

The TMOAS transaction is used to essentially track any modifications to the ASID and the processor mode. This tracking is enabled whenever tracing is on before the mode change takes place. If tracing is off when an ASID or mode change occurs, no mode transaction occurs. Figure 4-3 illustrates the bits that are traced in the right-most position on the PDO_AD bus for a TMOAS record.

In addition to the PDO_TType values discussed above, there are two, TU1 and TU2 which are used for user-triggered tracing. Whenever the user writes to a special register, the register values are traced out using one of the above PDO_TType values (depending on a bit in a control register). Details on this mechanism are described in Section 4.5.1, "Coprocessor 0 Trace Registers" on page 34.

4.1.7 Trace Mode

The PDO_TMode signal is used to indicate the compression method used to transmit the address or data value on the PDO_AD bus. This is used by the external software to regenerate the program flow. The compression technique depends on the particular type of value being transmitted. A more detailed description is provided in Chapter 5, “Trace Compression,” on page 43.

4.1.8 Data Order Signal

The data order signal PDO_DataOrder is used to indicate the out-of-order-ness of load and store data that is traced out. The main purpose of this signal is to allow load and store data to be traced out as and when it becomes available, and not maintain local storage that sequences it. This signal works by indicating the position of the traced load/store data in the list of current outstanding loads/stores starting at the oldest. For example, assume that the program issues 5 loads A, B, C, D, E, respectively.

Table 4-3 Load Order Example

Load	Cycle #	Cache Op	Load Data Available	Data Traced Out	PDO_DataOrder
A	1	Miss	-	-	-
B	2	Hit	B	B	1 (second oldest)
C	3	Hit	C	C	1 (second oldest)
D	4	Miss	-	-	-
E	5	Hit	E	E	2 (third oldest)
-	k	-	A	A	0 (oldest)
-	k+p	-	D	D	0 (oldest)

Table 4-3 shows an example of how these five loads may be traced. Load data that hits in the first-level cache is usually available at some fixed delay from instruction issue. So without loss of generality, we assume in the table that load data is available the same cycle as the issued instruction.

If the number of outstanding data supported by four bits is exceeded, then the processor simply issues the overflow signal, clears its internal buffers and restarts tracing. If the PDI_InhibitOverflow signal is asserted, then the processor must stall until at least some of the outstanding loads/stores are satisfied before continuing. Note that if data values are being traced, limits are being reached on other resources like the internal FIFO, and thus it is unlikely that the number-of-outstanding-data limit will be so easily reached.

Some processors will graduate a store instruction while still waiting for the store data to become available. Thus, a load can bypass a store and thus load data will be available before a preceding store’s store data is available. An example is illustrated in Figure 4-4.

Figure 4-4 An Example of Load Data Bypassing an Earlier Store

(1)

Cycle	Program	PDO_InsComp	PDO_AD	Comments
m+0	DIV r3, r2	I	NT	multi-cycle instr
m+1	MFHI r1	I	NT	
m+2	SW r1, 0(r3)	ISa	TSAa	data not available
m+3	SW r4, 0(r7)	ISb	TSAb	data not available
m+4	LW r4, 0(r6)	ILc	TDb	store data
m+5	LW r5, 4(r6)	ILd	TLAc	cache hit
m+6			TDc	load data
m+7			TLAd	cache hit
m+8			TDd	load data
m+9+k			TDa	store data

(2)

Required Data Order	PDO_DataOrder
TDa	1
TDb	1
TDc	1
TDd	0

Block **(1)** in Figure 4-4 shows a small program fragment and the sequence of the PDO_InsComp and PDO_AD bus transmissions. This processor will graduate and trace all instructions including the first store ISa. This store then waits for the data in r1 before it actually completes its execution. Some processors will order store data. Hence the second store ISb will wait for ISa before it can complete. But the following loads, ILc and ILd would complete without any delay. In this situation, the PDO_AD column of block **(1)** shows the sequence of data availability. But if the processor must trace data sequentially, then it is required to trace out data in-order as shown in the left column of block **(2)**. This sequentiality requirement can be avoided by using the PDO_DataOrder signal that orders both the loads and stores. The PDO_DataOrder values for the data is shown in the right column of block **(2)**.

Another example that illustrates the combined load/store ordering is shown in Table 4-4. This table shows in column one, a sequence of only the loads and stores from a program fragment. The second column shows the sequence in which the data associated with the loads and stores become available, and the third column shows the PDO_DataOrder signal that is needed to trace out the sequence as available.

Table 4-4 Data (Load/Store) Order Example

Load/Store	Data Trace Order	PDO_DataOrder
Load-A	-	-
Load-B	-	-
Store-C	-	-
Load-D	-	-
Store-E	-	-
Store-F	-	-
Store-G	-	-
Store-H	-	-
Load-I	I	8 (ninth oldest)
-	A	0 (oldest)
-	C	1 (second oldest)
-	E	2 (third oldest)
-	F	2 (third oldest)
-	G	2 (third oldest)
-	H	2 (third oldest)
-	B	0 (oldest)
-	D	0 (oldest)

4.1.9 Instruction-Data Map Signal

This signal provides some redundant per instruction information on the PDtrace interface. An 8-bit PDO_DataForIns[7:0] bus, per PDO_InsComp, defines which pieces of data will be transmitted on the PDO_AD bus for that instruction. See Table 3-1 for details on how the 8 signal bits are defined. This allows the logic block reading the interface to prepare up-front for the information that will be sent on the PDO_AD bus for a given instruction.

4.1.10 Trace Timing Example

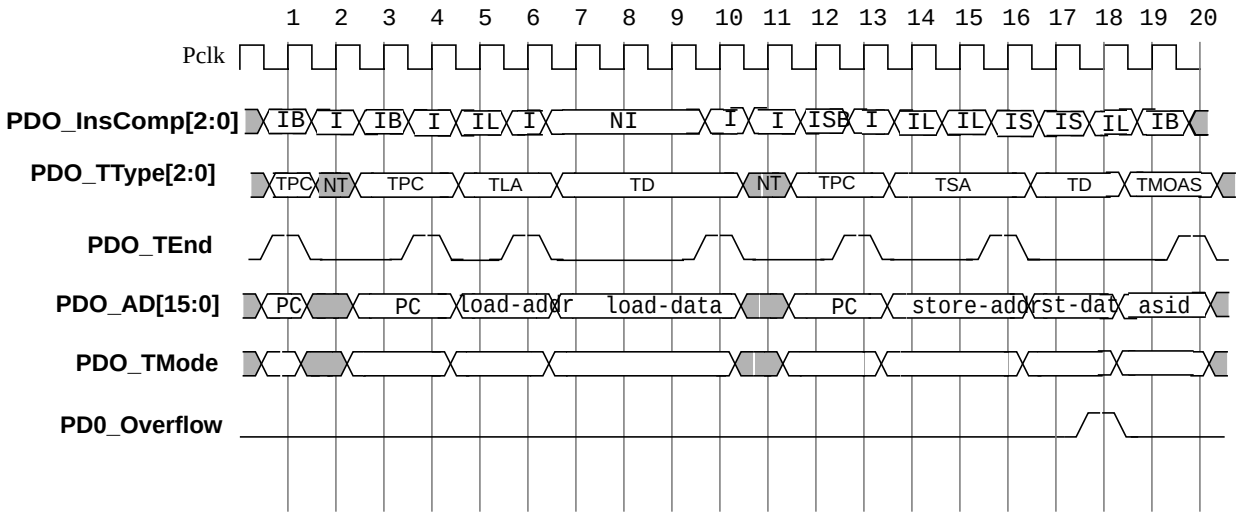
The timing diagram shown in Figure 4-5 illustrates an example of the timing and usage of the signals described in the previous sections.

The figure shows a single cycle PC transmission in cycle 1. Cycles 3 and 4 illustrate an example where it takes two cycles to transmit the new PC value.

The PDO_TType[2:0] signal is also used to indicate the begin of transmission of a load or store data address (TLA/TSA) and the actual data itself (TD). The PDO_TEnd signal is used to indicate the completion of the current transmission type. In Figure 4-5 for example, cycle 5 shows the beginning of a load data address transmission. This transaction takes two cycles, hence, the end of this transaction is indicated in cycle 6 by the asserted PDO_TEnd signal. The transmission of the data that corresponds to this load address then begins in cycle 7 and ends in cycle 10.

Cycle 12 indicates the beginning of an instruction where the PC changes value, and needs to be transmitted, but it is also a store data instruction, thus the need to transmit the store address and the data. In the meantime, cycles 14, 15, 16, 17, 18 all need to transmit data addresses and values, and the FIFO overflows at cycle 18. The PDO_Overflow signal is asserted at cycle 18. Now, all the load and store values from cycles 14-18 are discarded. Cycle 19 indicates the completion of an instruction that needs no tracing, but since this is after a PDO_Overflow, the full PC value needs to be transmitted, which is done during cycles 19 and 20, hence an IB is used for PDO_InsComp (rather than I).

Figure 4-5 PDtrace interface timing example



4.2 Trace Input Signals

The majority of the trace input signals are used to specify the conditions under which tracing is to be enabled. The list below briefly explains the various types of trace input signals to the core:

- The PDI_TCBPresent signal is really a validity signal for all the other input signals. This tells the core that TCB hardware is present and connected to the core. The core then regards the other input signals to have valid values.
- An overall trace control signal PDI_TraceOn controls whether tracing can be triggered on or not. If this signal is asserted, then the input signals that control the per instruction decision of whether the core should trace or not, include input trace signals such as PDI_G, PDI_ASID, PDI_U, PDI_S, PDI_K, PDI_E, and PDI_DM. Refer to Table 3-1 for an explanation of when each of these signals enable tracing.
- When tracing is turned on, the TCB needs to specify what kind of information is to be traced, i.e., just the PC, or also the load/store addresses and data. This is done using the PDO_TMode signal. See Table 3-1 for details. In addition to this, another signal, PDI_TraceAllBranch asks that the PC of all taken branches be traced, not just the ones that are statically unpredictable. When asserted, this signal will generate a lot of trace data, since in a RISC architecture like MIPS, typically every 3 or 4 instructions is a branch instruction. The main purpose of this all-branches tracing is to enable the TCB to track the execution addresses on the core without referring to the static program image. This knowledge can be used by the TCB to provide additional filtering on the trace data.
- Two signals, PDI_InhibitOverflow and PDI_StallSending are used to ensure that trace data is never lost because of internal FIFO or buffer overflow. (This condition would result when a large number of bits are traced each cycle on the average while the bandwidth out of the core or TCB is far less. The PDI_InhibitOverflow is used to ensure that the FIFO on the core's tracing logic does not overflow. If this signal is asserted and the FIFO is in imminent danger of overflowing, then the core must stall its pipe while the FIFO is emptied.

  If the TCB's internal buffer is in danger of imminent overflow, the PDI_StallSending signal is used by the TCB to

signal to the core to stop sending trace data on the PDtrace interface. The core stalls until the PDI_StallSending signal is de-asserted by the TCB. Note that the cycle after the PDI_StallSending signal is asserted, the TCB can ignore all PDO_ signals from the core, including the PDO_InsComp as well the PDO_AD bus signals. In the cycle after the TCB de-asserts PDI_StallSending, all PDO_ signals from the core are considered valid and must be read by the TCB.

- The signal PDI_SyncOffEn is an enable for the signals PDI_SyncPeriod and PDI_OffChipTB. These two signals are used to set the synchronization interval. As shown in Table 3-1, the synchronization interval is specified in cycles and is interpreted based on the value of PDI_OffChipTB. That is, whether the trace is being stored on-chip within a trace buffer in the TCB, or being sent off-chip to some larger trace memory.

4.3 Tracing Multi-Issue and High-Performance Processors

4.3.1 Background on High Performance Processors

This section addresses the tracing needs of multi-issue pipeline processors and describes a mechanism that allows a workable and efficient tracing of program execution on such processors. The features of high performance processors are not in general, very suitable for effectively tracing the sequential execution of a program. Such processor features include, but are not limited to:

- Superscalarity or multi-issue
- Aggressive, out-of-order dynamic scheduling with big fetch and issue windows
- Deep pipelines
- Multi-latency pipelines
- Multiple outstanding load misses

A processor that is designed to issue multiple instructions, and moreover out of order from the original program sequence, will also implement what is typically known as the re-order buffer. This re-order buffer and its control logic is responsible for putting the issued instructions back in-order (of the original program sequence). There is a stage in the pipeline when instructions are graduated from the re-order buffer, i.e., the point where it is certain that the instruction will not stop due to an exception (or any other reason), and can proceed to completion. This graduation of instructions from the re-order buffer is done in program sequence.

There are several things to note here, one, the graduated instructions have not completed their execution and will proceed to do so further in the pipeline, for example, the register write-back of the computed result of an arithmetic instruction will happen later in the pipeline. The second thing to note, is that, typically, the number of graduating instructions will not exceed the number of issue slots of the processor. But the number can vary from a minimum of zero up to the number of issue slots at the front of the pipe plus the number load miss completions from the bus and cache units, etc.

4.3.2 The Basic Tracing Methodology

The trace methodology in this document proposes that instructions be traced at the point of graduation. It is recommended that a number of instructions be simultaneously traced, the recommended number is the number of issue slots of the processor, let us call this the “number of instruction trace slots”. It is possible that in some cycles the number of graduating instructions is greater than the number of instruction trace slots. In these cases, the processor’s trace control logic must buffer the instruction(s) that could not be traced earlier, and trace them at the beginning of the next cycle, still maintaining the program sequence order. Note that the size of such a buffer need not be very large, since over time, the number of issued instructions will equal the number of graduated instructions. The size of this buffer can be calculated based on the maximum number of instructions that can graduate from the re-order buffer on any given cycle, and this number is based on the processor’s pipeline depths and other pipeline-related information.

All the signals marked “Out” in Table 3-1 are signals output from the processor core and represent the activity of a single instruction within the core. Most of these signals need to be duplicated as many times as the number of instruction trace slots within the core. Signals that must be duplicated are shown in Table 3-1 also with signal names appended with a “_n”, where n is used to designate the instruction trace slot number. For example, a two-issue core can trace two instructions and use the signals PDO_InsComp_0 and PDO_InsComp_1 to represent the completion status values of two simultaneously graduating instructions. If only one instruction graduates on any given cycle, then PDO_InsComp_1 sends a value of 000. When no instruction graduates on a given cycle, then both PDO_InsComp_0 and PDO_InsComp_1 send 000 values.

The same example code fragment from before is shown in Table 4-5 and this table shows the graduation cycle of each instruction and the number of the instruction trace slot that actually traces that instruction. This example assumes a simple two-issue processor that allows up to one load/store instruction per issue and one branch instruction per cycle.

Table 4-5 Example Code Fragment Showing the Graduation Cycle and Trace Bus Number

Instr No.	PC	Instruction	Graduation Cycle	Trace Bus Number
1	0x00400188	SW $6, 0xe170($1)	n+0	0
2	0x0040018c	SW $4, 0xb134($28)	n+1	0
3	0x00400190	SW $5, 0xb130($28)	n+2	0
4	0x00400194	SW $0, 0x1c($29)	n+3	0
5	0x00400198	JAL 0x418d9c	n+4	0
6	0x0040019c	OR $30, $0, $0	n+4	1
7	0x00418d9c	NOP	n+5	0
8	0x00418da0	JR $31	n+5	1
9	0x00418da4	NOP	n+6	0
10	0x004001a0	JAL 0x411c40	n+7	0
11	0x004001a4	NOP	n+7	1
12	0x00411c40	JR $31	n+8	0
13	0x00411c44	NOP	n+8	1
14	0x00414adc	LW $4, 0xb134($28)	n+9	0
15	0x00414ae0	BEQ $14, $0, 0x414af8	n+9	1
16	0x00414ae4	ADDIU $29, $29, 0xffe0	n+10	0
17	0x00414af8	OR $7, $0, $0	n+10	1
18	0x00414afc	NOP	n+11	0
19	0x00414b00	ADDU $6, $6, $2	n+11	1
20	0x00414b04	OR $7, $2, $0	n+12	0
21	0x00414b08	SLTU $1, $2, $1	n+12	1

4.3.3 Coordinating the Instruction Completion Trace with the Address/Data Trace

When an instruction is traced on a particular instruction trace slot, say using PDO_InsComp_k, then all other information for that instruction is sent on the signals of the "k"th instruction trace slot. For example, the address and data, if any, associated with that instruction is sent on the PDO_AD_k bus. Thus, once an instruction begins its trace life on a particular instruction trace slot, it must complete its life on the same slot. The exception to this occurs when the data is not immediately available. In this situation, the data can be sent on any of the PDO_AD_n bus that is temporarily free and hence chosen by the processor to send that data. See Section 4.3.4, "Out-of-Order Loads and Stores in the Multi-Pipe Core".

The process of identifying the data associated with particular instructions has been simplified by making it a requirement that all the data associated with instructions traced on the same cycle be in lock-step. Specifically, all the data associated with instructions that are traced together on the different PDO_InsComp_n are such that their end points (i.e., the last data cycle) are synchronized to be traced together. This requirement makes it easier for an external block to sequence all the data operations in the various PDO_AD_n buses into the program sequence. An example that illustrates this behavior is shown in Figure 4-6.

Figure 4-6 shows four blocks of information. The first one **(1)** shows the instruction complete (PDO_InsComp) values in the program sequence. The second block **(2)** shows these values as they would be transmitted on the two instruction

Figure 4-6 An Example Showing the Coordination of Instructions and their Data

(1)

Program Sequence
ILBa
ILb
ISc
ILd

(2)

PDO_InsComp_0	PDO_InsComp_1	cycle
ILBa	ILb	n
ISc	ILd	n+1

(3)

Cycle	PDO_AD_0	PDO_AD_1	PDO_TEnd_0	PDO_TEnd_1	Comments
m+0	TPCa1	NT	0	x	
m+1	TPCa2	NT	1	x	
m+2	TLAa1	NT	0	x	
m+3	TLAa2	TLAb1	1	1	
m+4	TDa1	TDb1	0	0	
m+5	TDa2	TDb2	1	1	completion of all _AD transfers for instructions traced in cycle n
m+6	TSAc1	NT	0	x	
m+7	TSAc2	TLAd1	1	1	
m+8	TDc1	TDd1	0	0	
m+9	TDc2	TDd2	1	1	completion of all _AD transfers for instructions traced in cycle n+1

(4)
Data in Program Sequence
TPCa1, TPCa2,
TLAa1, TLAa2,
TLAb1,
TDa1, TDa2
TDb1, TDb2
TSAc1, TSAc2
TLAd1
TDc1, TDc2
TDd1, TDd2

trace slots, i.e., PDO_InsComp_0 and PDO_InsComp_1. The third block **(3)** shows the PDO_AD and PDO_TEnd values for the two trace slots. Note that the data trace information for the instructions that were simultaneously traced on PDO_InsComp_0 and PDO_InsComp_1 are traced such that their PDO_TEnd is coordinated. For the PDO_InsComp values traced in cycle n (in block **(2)**), the data transmission ends in cycle m+5 (in block **(3)**). And for the PDO_InsComp values traced in cycle n+1 (in block **(2)**), the data transmission ends in cycle m+9 (in block **(3)**).

The external block reading the signals on the interface can then take the data values from the two PDO_AD buses, and knowing the program sequence order (in block **(1)**), can put the data trace in order, as shown in block **(4)**.

4.3.4 Out-of-Order Loads and Stores in the Multi-Pipe Core

When a multi-pipe core needs to send out-of-order data, it uses the PDO_DataOrder signal just like the single-pipe core. When an out-of-order data is returned, it can be traced on any free PDO_AD_n bus, not necessarily the one that traced the corresponding instruction. This is because, instruction tracing is sequentialized by the PDO_InsComp_n order and therefore the data can be associated with the correct instruction once the PDO_DataOrder value is known. Note that since the PDO_AD_n busses are implicitly ordered, for data transmissions that end on the same cycle, the data on PDO_AD_k is before the data on PDO_AD_k+1.

4.3.5 Tagging Instructions that Issue Together

With the method of tracing graduating instructions in sequence, it is not possible to know which instructions issue together without additional information. This information might be invaluable to tune a code optimizer for high performance processors. In order to trace this information, the processor tags all the instructions that issue together, using the signal PDO_IssueTag_n. This tag value is also traced out with each PDO_InsComp_n value. A tag value of 6 bits is being initially proposed, assuming an issue window of about 64 instructions. Note that this tag information can be traced out of the TCB only if the user requires it, hence it will not incur bandwidth on the external pins unless there is a real need for this information. Thus, it is recommended that the TCB allow the external tracing of this information under user discretion.

4.3.6 Miscellaneous

The input signals to a multi-pipeline core are not duplicated. A single set of signals control tracing options on all the pipelines.

When tracing is first started (or re-started after a break), PDO_InsComp_0 is the first traced instruction in the static program image and this will output the TMOAS record and the full PC.

When there is a need for synchronization, the core can choose any PDO_InsComp_n to send the TMOAS record and the full PC value, as long as these two are both done on the same instruction in the trace slot. Note that if load/store addresses are also being traced, then a full load/store address value is part of the synchronization tracing. This may not always be possible on the instruction chosen by the core. But these should be sent on the next sequential load/store instruction. This is a situation that the external software has to take into account when recognizing synchronization transmissions in the multi-pipeline core or processor.

4.4 Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints

The MIPS EJTAG Specification describes the hardware data and instruction breakpoint feature. In brief, a core or processor can optionally implement up to 15 instruction and up to 15 data EJTAG hardware breakpoints. These breakpoints, when encountered during program execution, cause the processor to take a debug exception. Important to this discussion is that a bit (TE bit 2) in the breakpoint control register, when set, allows a trigger signal to be generated (instead of, or in addition to, causing a debug exception). The PC/Data tracing interface uses this trigger signal to trigger trace on/off.

In addition, when a trigger is generated, all information relating to this trigger is sent on the PDtrace interface to the TCB. The TCB passes this information on to the trace memory so the trace software can have knowledge of when trace triggers were generated. The signals that comprise this information is described in Table 3-1 as PDO_TrigOn, PDO_TrigOff, PDO_TrigI[N-1:0] and PDO_TrigD[N-1:0].

4.4.1 The *TraceBPC* Register (CP0 Register 23, Select 4)

Whether a particular hardware breakpoint triggers trace on or off, is determined by 30 separate bits in the *TraceBPC* register (Trace Break Point Control). (15 bits for hardware instruction breakpoint plus 15 bits for hardware data breakpoints). The type of tracing that is triggered is determined by the tracing mode signal PDO_TMode, or if in software control, by the Mode[3:1] bits in the *TraceControl* register (described in Section 4.5.1.1, "The TraceControl Register (CP0 Register 23, Select 1)" on page 34 of this document).

The EJTAG control logic, upon encountering a hardware breakpoint, will signal the triggered breakpoint to the trace logic. If more than one breakpoint triggers every cycle, the tracing logic will trigger trace on even if one of them triggers off. The trace is turned off only if all of them triggers off.

Note that it is possible for the tracing mechanism to globally disable the hardware breakpoint-enabled triggering of tracing using two bits in the *TraceBPC* register. One bit is used to disable instruction breakpoints, and the other is used to disable data breakpoints, as shown in Figure 4-7 and Table 4-6.

It is possible that PDtrace tracing logic is implemented with no EJTAG implementation. Thus, it is the responsibility of (external or internal) software to read the Coprocessor 0 Config1 register to determine if EJTAG is implemented before assuming the presence of the *TraceBPC* register. Moreover, the EJTAG hardware breakpoints are optional for a core implementing EJTAG. The Debug Control Register (at offset 0x0000 in drseg) has bits DataBrk and InstBrk that specify whether any EJTAG data or instruction hardware breakpoints are implemented. If both these bits are set to 0, then no hardware breakpoints are implemented in EJTAG on that core, and the *TraceBPC* register specified in this section is also not implemented, i.e., the tracing logic does not implement the feature of trace triggering from EJTAG. Thus one must first ensure that EJTAG is implemented, then examine the values of DataBrk and InstBrk in the Debug Control register, and ensure that at least one of them is not zero, before assuming the presence of the *TraceBPC* register.

Figure 4-7 *TraceBPC* Register Format

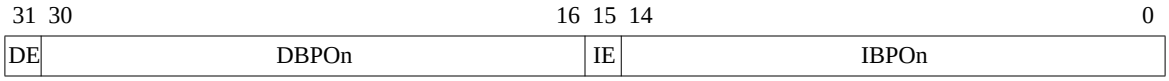


Table 4-6 *TraceBPC* Register Field Descriptions

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
DE	31	Used to specify whether the trigger signal from EJTAG data breakpoint should trigger tracing functions or not: 0 : disables trigger signals from data breakpoints 1 : enables trigger signals from data breakpoints	R/W	0	Required
DBPOn	30:16	Each of the 15 bits corresponds to the 15 possible EJTAG hardware data breakpoints that may be implemented. For example, bit 16 corresponds to the first data breakpoint. If only 4 data breakpoints are present in the EJTAG implementation, then only bits 16,17,18, and 19 are used. The rest are always ignored by the tracing logic since they will never be triggered. A value of one for each bit implies that a trigger from the corresponding data breakpoint should start tracing. And a value of zero implies that tracing should be turned off with the trigger signal.	R/W	0	Required
IE	15	Used to specify whether the trigger signal from EJTAG instruction breakpoint should trigger tracing functions or not: 0 : disables trigger signals from instruction breakpoints 1 : enables trigger signals from instruction breakpoints	R/W	0	Required
IBPOn	14:0	Each of the 15 bits corresponds to the 15 possible EJTAG hardware instruction breakpoints that may be implemented. Bit 0 corresponds to the first instruction breakpoint, and so on. If only 2 instruction breakpoints are present in the EJTAG implementation, then only bits 0 and 1 are used. The rest are always ignored by the tracing logic since they will never be triggered. A value of one for each bit implies that a trigger from the corresponding instruction breakpoint should start tracing. And a value of zero implies that tracing should be turned off with the trigger signal.	R/W	0	Required

4.5 Software Trace Control

Just as the TCB hardware can control tracing functionality using the input PDI_ signals, the PDtrace architecture allows software to control tracing with similar enables and with the same flexibility. This is done by setting bits in the Coprocessor 0 *TraceControl* register to appropriate values. To ensure that only one of hardware or software can control tracing at any given point in time, a trace select bit is used in the trace control register (*TraceControl*). A processor reset sets the trace select bit to default trace input select from the TCB hardware.

4.5.1 Coprocessor 0 Trace Registers

This section describes all the Coprocessor 0 trace registers needed for implementing PDtrace tracing logic in the core (with the exception of *TraceBPC*, which was described in Section 4.4, "Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints").

Table 4-7 shows a list of all the Coprocessor 0 tracing-related registers. The compliance level is specified assuming that tracing is implemented, i.e., the TL bit in Coprocessor 0 Config3 is 1 (Table 2-1).

Table 4-7 A List of Coprocessor 0 Trace Registers

Register Number	Sel	Register Name	Reference	Compliance
23	1	TraceControl	Section 4.5.1.1, "The TraceControl Register (CP0 Register 23, Select 1)" on page 34	Required
23	2	TraceControl2	Section 4.5.1.2, "The TraceControl2 Register (CP0 Register 23, Select 2)" on page 37	Required
23	3	UserTraceData	Section 4.5.1.3, "The UserTraceData Register (CP0 Register 23, Select 3)" on page 38	Required
23	4	TraceBPC	Section 4.4.1, "The TraceBPC Register (CP0 Register 23, Select 4)" on page 32	Required (only if EJTAG hardware data or instruction breakpoint has been implemented, otherwise not required).

4.5.1.1 The *TraceControl* Register (CP0 Register 23, Select 1)

The *TraceControl* register configuration is shown in Figure 4-8 and Table 4-8. Note the special behavior of the ASID_M, ASID, and G fields if the processor does not implement the standard TLB-based MMU.

Figure 4-8 *TraceControl* Register Format

Table 4-8 *TraceControl* Register Field Descriptions

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
TS	31	The trace select bit is used to select between the hardware and the software trace control bits. A value of zero selects the external hardware trace block signals, and a value of one selects the trace control bits in the *TraceControl* register.	R/W	0	Required

Table 4-8 *TraceControl* Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
UT	30	This bit is used to indicate the type of user-triggered trace record. A value of zero implies a user type 1 and a value of one implies a user type 2. The actual triggering of a user trace record happens on a write to the *UserTraceData* register. This is a 32-bit register for 32-bit processors and a 64-bit register for 64-bit processors.	R/W	Undefined	Required
0	29:28	Reserved for future use; Must be written as zero; returns zero on read.	0	0	Reserved
TB	27	Trace All Branch. When set to 1, this tells the processor to trace the PC value for all taken branches, not just the ones whose branch target address is statically unpredictable.	R/W	Undefined	Required
IO	26	Inhibit Overflow. This signal is used to indicate to the core trace logic that slow but complete tracing is desired. Hence, the core tracing logic must not allow a FIFO overflow and discard trace data. This is achieved by stalling the pipeline when the FIFO is nearly full, so that no trace records are ever lost.	R/W	Undefined	Required
D	25	When set to one, this enables tracing in Debug Mode (see Section 2.1, "Processor Modes" on page 6). For trace to be enabled in Debug mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in Debug Mode, irrespective of other bits.	R/W	Undefined	Required
E	24	When set to one, this enables tracing in Exception Mode (see Section 2.1, "Processor Modes" on page 6). For trace to be enabled in Exception mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in Exception Mode, irrespective of other bits.	R/W	Undefined	Required
K	23	When set to one, this enables tracing in Kernel Mode (see Section 2.1, "Processor Modes" on page 6). For trace to be enabled in Kernel mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in Kernel Mode, irrespective of other bits.	R/W	Undefined	Required
S	22	When set to one, this enables tracing in Supervisor Mode (see Section 2.1, "Processor Modes" on page 6). For trace to be enabled in Supervisor mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register. When set to zero, trace is disabled in Supervisor Mode, irrespective of other bits. If the processor does not implement Supervisor Mode, this bit is ignored on write and returns zero on read.	R/W	Undefined	Required (if Supervisor Mode is implemented, is Reserved otherwise)

Table 4-8 *TraceControl* Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read/ Write</th><th rowspan="2">Reset State</th><th rowspan="2">Compliance</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>U</td><td>21</td><td>When set to one, this enables tracing in User Mode (see Section 2.1, "Processor Modes" on page 6). For trace to be enabled in User mode, the On bit must be one, and either the G bit must be one, or the current process ASID must match the ASID field in this register.

When set to zero, trace is disabled in User Mode, irrespective of other bits.</td><td>R/W</td><td>Undefined</td><td>Required</td></tr>
<tr><td>ASID_M</td><td>20:13</td><td>This is a mask value applied to the ASID comparison (done when the G bit is zero). A “1” in any bit in this field inhibits the corresponding ASID bit from participating in the match. As such, a value of zero in this field compares all bits of ASID. Note that the ability to mask the ASID value is not available in the hardware signal bit; it is only available via the software control register.

If the processor does not implement the standard TLB-based MMU, this field is ignored on write and returns zero on read.</td><td>R/W</td><td>Undefined</td><td>Required</td></tr>
<tr><td>ASID</td><td>12:5</td><td>The ASID field to match when the G bit is zero. When the G bit is one, this field is ignored.

If the processor does not implement the standard TLB-based MMU, this field is ignored on write and returns zero on read.</td><td>R/W</td><td>Undefined</td><td>Required</td></tr>
<tr><td>G</td><td>4</td><td>When set, this implies that tracing is to be enabled for all processes, provided that other enabling functions (like U, S, etc.,) are also true.

If the processor does not implement the standard TLB-based MMU, this field is ignored on write and returns 1 on read. This causes all match equations to work correctly in the absence of an ASID.</td><td>R/W</td><td>Undefined</td><td>Required</td></tr>
<tr><td>Mode</td><td>3:1</td><td>These three bits provide the same trace mode functions as the PDI_TraceMode[2:0] signal, and is described here again.

<table><tr><th>Mode</th><th>Trace Mode</th></tr><tr><td>000</td><td>Trace PC</td></tr><tr><td>001</td><td>Trace PC and load address</td></tr><tr><td>010</td><td>Trace PC and store address</td></tr><tr><td>011</td><td>Trace PC and both load/store addresses</td></tr><tr><td>100</td><td>Trace PC and load data (optional for all PDtrace specification revisions less than 03.00)</td></tr><tr><td>101</td><td>Trace PC and load address and data</td></tr><tr><td>110</td><td>Trace PC and store address and data</td></tr><tr><td>111</td><td>Trace PC and both load/store address and data</td></tr></table>
The TraceControl2$_{ValidModes}$ field determines which of these encoding are supported by the processor. The operation of the processor is UNPREDICTABLE if this field is set to a value which is not supported by the processor.</td><td>R/W</td><td>Undefined</td><td>Required</td></tr>
</table>

Table 4-8 *TraceControl* Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
On	0	This is the master trace enable switch in software control. When zero, tracing is always disabled. When set to one, tracing is enabled whenever the other enabling functions are also true.	R/W	0	Required

4.5.1.2 The *TraceControl2* Register (CP0 Register 23, Select 2)

The TraceControl2 register provides additional control and status information. It is described here in Figure 4-9 and Table 4-9. Note that some fields in the *TraceControl2* register are read-only, but have a reset state of “Undefined”. This is because these values are loaded from various PDtrace Interface Signals when the PDI_SyncOffEn signal is asserted. As such, these fields in the *TraceControl2* register will not have valid values until the TCB asserts the PDI_SyncOffEn signal.

Figure 4-9 *TraceControl2* Register Format

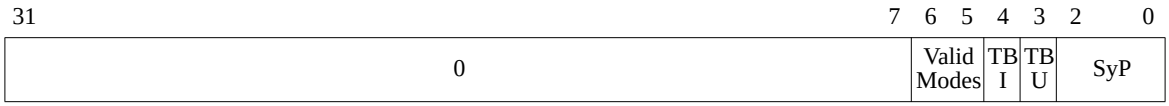


Table 4-9 *TraceControl2* Register Field Descriptions

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
0	31:7	Reserved for future use; Must be written as zero; returns zero on read.	0	0	Reserved
ValidModes	6:5	This field specifies the subset of tracing that is supported by the processor (see Section 2.2, "Subsetting" on page 6). **Encoding** / **Meaning** 00 — PC tracing only 01 — PC and load and store address tracing only 10 — PC, load and store address, and load and store data 11 — Reserved	R	Preset	Required
TBI	4	This bit indicates how many trace buffers are implemented by the TCB, as follows: **Encoding** / **Meaning** 0 — Only one trace buffer is implemented, and the TBU bit of this register indicates which trace buffer is implemented 1 — Both on-chip and off-chip trace buffers are implemented by the TCB and the TBU bit of this register indicates to which trace buffer the traces is currently written. This bit is loaded from the PDI_TBImpl signal when the PDI_SyncOffEn signal is asserted.	R	Undefined	Required

Table 4-9 *TraceControl2* Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State	Compliance
Name	**Bits**				
TBU	3	This bit denotes to which trace buffer the trace is currently being written and is used to select the appropriate interpretation of the $TraceControl2_{SyP}$ field. **Encoding** / **Meaning** 0 — Trace data is being sent to an on-chip trace buffer 1 — Trace Data is being sent to an off-chip trace buffer This bit is loaded from the PDI_OffChipTB signal when the PDI_SyncOffEn signal is asserted.	R	Undefined	Required
SyP	2:0	The period (in cycles) to which the internal synchronization counter is reset when tracing is started, or when the synchronization counter has overflowed. **SyP** / **On-chip** / **Off-chip** 000 / 2^2 / 2^7 001 / 2^3 / 2^8 010 / 2^4 / 2^9 011 / 2^5 / 2^{10} 100 / 2^6 / 2^{11} 101 / 2^7 / 2^{12} 110 / 2^8 / 2^{13} 111 / 2^9 / 2^{14} The “On-chip” column value is used when the trace data is being written to an on-chip trace buffer (e.g, $TraceControl2_{TBU} = 0$). Conversely, the “Off-chip” column is used when the trace data is being written to an off-chip trace buffer (e.g, $TraceControl2_{TBU} = 1$). This field is loaded from the PDI_SyncPeriod signal when the PDI_SyncOffEn signal is asserted.	R	Undefined	Required

4.5.1.3 The *UserTraceData* Register (CP0 Register 23, Select 3)

A software write to any bits in the *UserTraceData* register will trigger a trace record to be written indicating a type 1 or type 2 user format. The type is based on the UT bit in the *TraceControl* register. This register cannot be written to in consecutive cycles. The trace output data is UNPREDICTABLE if this register is written in consecutive cycles.

Figure 4-10 *UserTraceData* Register Format

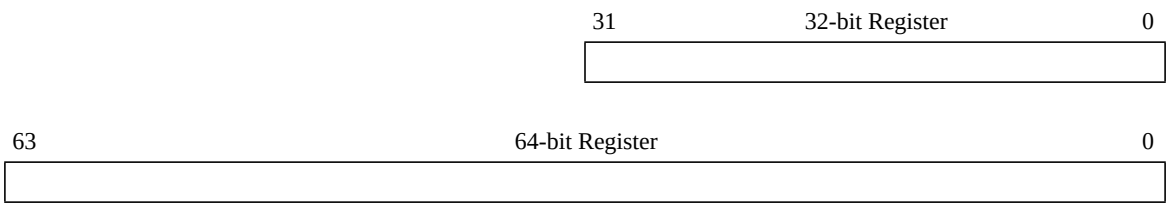


Table 4-10 *UserTraceData* Register Field Descriptions

Fields		Description	Read/ Write	Reset State	Compliance
Name	Bits				
Data	31:0 or 63:0	Software readable/writable data. When written, this triggers a user format trace record out of the PDtrace interface that transmits the Data field to trace memory.	R/W	0	Required

4.6 Trace Enabling/Disabling Condition

As seen in Section 4.2, "Trace Input Signals" on page 27 there are several input signals into the core that enable tracing. In addition, trace can also be triggered on and off by the EJTAG hardware instruction and data breakpoint settings, as seen in Section 4.4, "Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints" on page 32. The equations specified here clarify the conditions under which different input factors will enable or disable tracing.

Trace enabling and disabling from software is similar to the hardware method, with the exception that the bits in the control register are used instead of the input enable signals from the TCB. The $TraceControl_{TS}$ bit controls whether hardware (via the TCB), or software (via the *TraceControl* register) controls tracing functionality.

In any given cycle n, an instruction is traced if the following equation evaluates true:

TraceOn & (TriggerOn(n) | MatchEnable | TriggerEnable) (EQ 1)

And every cycle, the following state variable is set and then used in the next cycle:

TriggerOn(n+1) <- TraceOn & (TriggerEnable | (TriggerOn(n) & (~TriggerDisable)) (EQ 2)

The various expressions used in (EQ 1) and (EQ 2) are defined here.

```
TraceOn  ← ((TraceControl_TS & TraceControl_On)  |
            ((~TraceControl_TS) & PDI_TraceOn))

MatchEnable ←
(TraceControl_TS                                                                  &
 (TraceControl_G | (((TraceControl_ASID ^ EntryHi_ASID) & (~TraceControl_ASID_M))=0)) &
 ((TraceControl_U & UserMode)       |
  (TraceControl_K & KernelMode)     |
  (TraceControl_S & SupervisorMode) |
  (TraceControl_E & ExceptionMode)  |
  (TraceControl_D & DebugMode)))            |
((not TraceControl_TS)                    &
 (PDI_G or (PDI_ASID = EntryHi_ASID))      &
 ((PDI_U & UserMode)       |
  (PDI_K & KernelMode)     |
  (PDI_S & SupervisorMode) |
  (PDI_E & ExceptionMode)  |
  (PDI_DM & DebugMode)))

TriggerEnable ←
((EJTAG_data_trigger[i]) & TraceBPC_DE & (TraceBPC_DBPOn[i] = 1))  |
((EJTAG_inst_trigger[i]) & TraceBPC_IE & (TraceBPC_IBPOn[i] = 1))

TriggerDisable ←
((EJTAG_data_trigger[i]) & TraceBPC_DE & (TraceBPC_DBPOn[i] = 0))  |
((EJTAG_inst_trigger[i]) & TraceBPC_IE & (TraceBPC_IBPOn[i] = 0))
```

As seen in the (EQ 1), trace can be turned on only if the master switch On or PDI_TraceOn is first asserted (TraceOn). Once asserted, there are three ways in which instruction tracing can occur:

1. A trigger had occurred in the past that turned on tracing, but no trace disabling trigger had occurred since then (TriggerOn(n)).
2. The input enable signals from the TCB or the trace control register indicate a tracing condition (MatchEnable). This type of tracing is done over general program areas. For example, all of user-level code for a particular process (ASID specified), or some such conditions.
3. The third method to turn on tracing is from the processor side using the EJTAG hardware breakpoint triggers (TriggerEnable). If EJTAG is implemented, and hardware breakpoints can be set, then using this method, fine grain tracing control is possible. It is possible to send a trigger signal that turns on tracing at a particular instruction. For example, it would be possible to trace a single procedure in a program by triggering on trace at the first instruction, and triggering off trace at the last instruction.

Trace is turned off when (EQ 1) evaluates false. Note that tracing can be unilaterally turned off by de-asserting the On bit or the PDI_TraceOn signal.

4.7 Tracing During Processor Mode Changes

Note that during normal execution, the processor will change its operation mode frequently. For example, when executing user-level code, an interrupt may cause the processor to jump to kernel mode to service the interrupt. When the interrupt has been serviced, the processor will switch back to user mode. A mode change is indicated in the tracing logic by tracing out a TMOAS for PDO_TType.

In the situation that the mode change affects tracing, for example, the tracing system has been set up to trace only in user mode and not in kernel mode, then the interrupt service routine should not be traced. Upon jumping to kernel mode, the core tracing logic will add a TMOAS as the last record in the FIFO (or if the FIFO is empty, will output directly). While the entries in the FIFO until the TMOAS entry are being traced out, the core will use a PDO_InsComp value of NI (No Instruction). Once the TMOAS record has been output, the core tracing logic will de-assert its PDO_IamTracing signal until the interrupt service routine is done and execution jumps back into user mode. By knowing the static instruction stream in the user program, and using the TMOAS record, the external trace reconstruction software can figure out that tracing was suspended when the processor jumped to kernel mode.

When jumping from a non-tracing mode to a tracing mode, the first record output is TMOAS to indicate the mode change. This is followed by a full PC value of the first instruction in the tracing mode. This will enable the external trace reconstruction software to re-synchronize itself and track program execution in the desired mode.

4.8 Tracing Store Conditionals

A store conditional instruction part of an LL/SC operation may or may not perform the actual store operation. This section describes the tracing rules for such an instruction.

- A store conditional that performs the store operation is traced out as an IS or ISB for PDO_InsComp. If store address or data is being traced, then this associated information is traced as well.
- A store conditional that does not perform its store is traced as an I or IB for PDO_InsComp (or even an IPC). And no associated address or data will therefore be traced for this instruction.

It is the responsibility of software to determine from context of the tracing and the program source whether the store conditional was successful or not.

4.9 Tracing MIPS16e Macro Instructions

In the MIPS16e™ ASE, several single MIPS16e instructions expand to a fixed sequence of multiple 32-bit instructions. These include the SAVE and RESTORE and ASMACRO instructions. (See the "MIPS32™ Architecture for Programmers Volume IV-a: The MIPS16e™ Application-Specific Extension to the MIPS32™ Architecture", document number MD00076).

When executing a Macro instruction, note that the PC address does not change for the instructions that comprise the macro instruction, hence the core does not output a PC value until it executes the first instruction outside the Macro. In fact, the core indicates the completion of the Macro instruction by outputting a full PC value for the first instruction executed after the macro instruction. This instruction could either have been reached sequentially by falling out of the macro sequence, or by executing a branch instruction from within the macro sequence. This full PC value is output using a branch indication e.g., IB for the PDO_InsComp value, even though this instruction is most likely not a branch target. The external re-construction software will note the preceding Macro instruction, and hence be able to handle this situation correctly.

Within the macro sequence, normal tracing is carried out. Note that the macro sequence can include, in addition to arithmetic and logical instructions, load and store instructions, which will be traced in a manner similar to loads/stores that are not in a macro instruction sequence. (Note that any branch instruction inside the Macro sequence can only branch out of the Macro sequence and not to any location within the sequence since all instructions within the sequence have the same PC value).

4.10 Tracing MIPS16e Extend Instructions

A MIPS16e™ extend instruction is considered a single instruction, and therefore the PC of the extend part is traced. Note that a branch to a MIPS16e extend instruction is to the extend part of the instruction. (For details, refer to the "MIPS32™ Architecture for Programmers Volume IV-a: The MIPS16e™ Application-Specific Extension to the MIPS32™ Architecture", document number MD00076).

Trace Compression

This section is a discussion of compression techniques that may be used when tracing different values. The methods used are quite different for each “type" of value. For example, the PC may be sent as a delta from the previous PC address. Sometimes the full PC value needs to be sent when the trace process starts either at the beginning of tracing, after a buffer overflow, or for synchronization. In this case, the PC can be sent un-compressed, or some method such as bit-block compression can be used. The sections below discuss these various techniques as they correspond to the PDO_TMode signal value. Note that the single-bit PDO_TMode signal allows two ways in which to send the information being currently traced.

5.1 PC tracing

When PDO_TMode is zero, this implies that the delta of the PC value is transmitted. This delta is computed from the PC value of the instruction executed just before the branch target instruction (e.g., the instruction executed in the branch delay slot after a branch instruction). The computed delta is then right-shifted by one bit, since this bit is never used. Note that the value can be negative or positive, hence is a signed 16-bit value, and the upper bits need to be sign-extended before transmission.

PC_delta = (new_PC - last_PC) >> 1 (EQ 3)

If the width of the computed delta value is bigger than the width of the PDO_AD bus, then the lower bits are sent first, followed by the upper bits.

When the PDO_TMode value is one, this implies that the full PC value is transmitted. Depending on the width of the bus, this may take multiple cycles. Again, the first cycle transmits the least significant bits, and so on.

5.2 Load or Store Address Tracing

With a PDO_TMode zero value, the load address transmitted is a delta from the last transmitted load address. Stores are similar, where the computed delta is from the last transmitted store address. Note that the last load instruction can be a load instruction of any type, i.e., LB, LW, etc. The same is true for stores.

load_address_delta = current_load_address - last_load_address (EQ 4)

store_address_delta = current_store_address - last_store_address (EQ 5)

Note that the delta transmission is quite effective when the load or store addresses are increasing or decreasing sequentially.

With a PDO_TMode value of one, the value transmitted is the full address of either the load or the store. Depending on the width of the trace bus and the processor data width, this could take multiple cycles to transmit.

5.3 Load or Store Data Tracing

The data values are less prone to good compression techniques. But delta values and bit-block compression techniques might be useful in achieving some compression ratio. This version of the PDtrace specification does not dictate any

compression for data values. The PDO_TMode value of zero is reserved for a future compression scheme. And the PDO_TMode value of one is used to transmit the full data value.

5.4 Using Early PDO_TEnd Assertion

This technique was discussed in the Table on page 9. When the processor is transmitting data on the PDO_AD bus, it can optionally make a decision to cut off the trailing sign bits of the data and assert PDO_TEnd early, before all the bits of the address or data has been sent. For example, redundant sign bits need not be transmitted for accurate reconstruction of the data. Note that this data compression technique can be applied to any transmission on the PDO_AD bus, be it PC address, load/store address, or load/store data. Also note that this technique is optional for the processor tracing logic, but the TCB and software must be capable of handling this situation for implementations with PDtrace Specification 03.00 and higher.

Appendix A

Revision History

A.1 Revision History

Table A-1 Revision History

Revision	Date	Description
1.6	August 29, 2000	Changes in this revision: Add the requirement that the data address be also periodically gathered for synchronization purposes, per FS2. Modify Figure 3 to show that the load data is picked up after alignment, per lhh. typo fixes
1.7	September12, 2000	Changes in this revision: Add a separate input signal that says whether to trace in Debug mode or not (i.e., DM = 1 in the *Debug* register), per Scott who wants to be able to debug the debug handler code. Put back Figure 3 to tap load/store data pre-alignment, per Franz. Add a section (3.17) to show when tracing is enabled. Allow the ASID to be masked under software control, per Scott. Amend Figure 1 to show the EJTAG/TAP controller and its connection to the debugger. Add to Table 2, to show the use of the PDO_InsComp signal value IPC (100). Add a chapter (6) on the trace capture block and its interaction with the external debugger software. Add TOC Fix typos, grammar, sentence construction.
1.8	October 27, 2000	Changes in this revision: Change the way loads are tracked and traced out. Add the tracing out of ASID and processor mode as part of the periodic synchronization. Add details to the multi-issue tracing section. The above changes require a modification to the output format section. Add a chapter to discuss the trace capture block (TCB), that includes: a definition of the control registers within the TCB, and the mechanism to write these registers from the external probe (or debugger). Define tracing with an on-chip trace buffer versus off-chip trace buffer. Add another Out signal from the core, PDO_IamTracing, that the core uses to signal to the TCB that it is actually sending valid trace data.

Table A-1 Revision History

Revision	Date	Description
1.9	November 20, 2000	Changes in this revision: Add tracing of processor ISA mode, and whether processor is in Debug mode or not. Get rid of the TCBTraceMask register, is not really needed. Allocate some bits in the TraceControl register as implementation dependent. Specify that full addresses are used for on-chip trace memory. Change the encoding of bits from the EJTAG logic to the tracing logic, send all 30 bits of breakpoint trigger. Fix the logical expression in 3.1.8.
2.0	December 19, 2000	Changes in this revision: Add a signal from the TCB to the core tracing logic, PDI_StallSending, that inhibits the core from sending trace data. Note that the core does not stop tracing, only stops sending trace information to the TCB. Used by the TCB when its internal buffer is in imminent danger of overflowing. (The core will stall if its internal FIFO will overflow). Make the synchronization period programmable, by using some bits in a register to hold this value. These bits can be updated by either software or by the TCB (based on the trace buffer size). Add a signal from the TCB to the core tracing logic that signals whether the TCB is using an on-chip or off-chip trace buffer. This changes the way in which the core interprets the synchronization period bits in the register. The chapter on trace control block (TCB) has been cut off into another document, since it is not directly relevant to the PDtrace architecture.
2.01	January 25, 2001	Changes in this revision: Add a signal PDI_TCBPresent to indicate that the TCB hardware is present. Clearer explanation of how the PDI_StallSending signal works. Change in how the PDI_EXL and the corresponding X bit in the *TraceControl* register works. Coding change in the PDI_TraceMode[2:0] signal.
2.02	February 12, 2001	Changes in this revision: Change in how the PDI_EXL and the corresponding X bit in the *TraceControl* register works. Tracing triggers on when either EXL or the ERL bit is a 1, this enables tracing after a cold reset.

Table A-1 Revision History

Revision	Date	Description
2.03	March 22, 2001	Changes in this revision: • Add a register description table for *UserTraceData*. • Add a PDI_TraceAllBranch signal to indicate that all branches (conditional, unconditional, predictable, and unpredictable) are to be traced. • Change the PDO_InsComp definition for unconditional predictable branches (jumps), so that these trace out as IB, ILB, and ISB (rather than I, IL, and IS). • Document how tracing is handled within MACRO instructions and the SAVE/RESTORE instruction. • Document what happens when a mode change happens within the processor and this changes the tracing mode, i.e., either turns it off or on. • Fix typos.
2.04	June 20, 2001	Changes in this revision: • Converted document to new template • PDO_TMode's reserved bit field of 100 is now used for tracing PC values and load data (this is optional for all PDtrace specifications less than 03.00 and conforming TCB implementations. • Three PDO_ signal bits have been added, PDO_MIPS16 and PDO_MIPS16Ins that are used only by processors implementing the MIPS16 ASE, and are optional. • The sense of EQ1, EQ2, and EQ3 used to compute the delta address values have been reversed. • Add the PDI_TraceAllBranch to the Trace Control Register. • Note that the select position of the COP0 registers implemented for tracing have all been changed, so that the control registers are together and the optional register *TraceBPC* is the last one. • Note that the end of a MIPS16 Macro instruction was indicated by the transmission of a full PC value. This was more fully specified so that this full PC value is accompanied by an PDO_InsComp value that indicates a branch, e.g., IB, ILB, etc. • The PDI_EXL has been changed to PDI_E, and similarly in the *TraceControl* register, X has been changed to E. • Bits 22 and 23 in the *TraceControl* register (K and S), have switched places. • The *TraceControl2* register has been re-arranged, and instead of the bit OfC, two new bits TBU and TBI have been added. • The TMOAS record has been augmented with an extra bit for the POM field and with a new bit called the SYNC bit. • Add an Input signal PDI_TBImpl from the TCB to the core tracing logic to say whether on-chip, off-chip, or both buffers are implemented by the TCB. This signal is optional for all TCB implementations that are compatible to PDtrace specifications less than 03.00.

Table A-1 Revision History

Revision	Date	Description
2.05	June 28, 2001	Changes in this revision: • Convert the stand-alone document to a book format and add LOF and LOT pages. • Add trademark symbol to PDtrace • Fix minor typos.
2.06	August 8, 2001	Changes in this revision: • Define the behavior if the processor implements a fixed mapping MMU, rather than the standard TLB-based MMU. • Define the polarity of the $\text{TraceControl}_{\text{ASID\_M}}$ field. • Precisely define the processor modes which for which tracing may be enabled. See Section 2.1, "Processor Modes" on page 6 for these definitions. • Make the equations for turning on and off trace more precise and convert to standard notation. • Add the standard “About This Book” chapter to define syntax and conventions. • Eliminate the R/W fields in TraceControl2. • More fully describe the synchronization counter, including when it must be restarted. • Make it explicit that ASID and processor mode changes are not traced if tracing is off when the change occurs. That is, ASID and processor mode changes are not traced if tracing is currently off. • Add subsetting rules for PDtrace (see Section 2.2, "Subsetting" on page 6) • Add the PDO_ValidModes signal and the ValidModes field in the *TraceControl2* register to specify which tracing modes the processor supports.
2.07	March 21, 2002	Changes in this revision: (RT) • Change the name of the TraceControl2 register field ValidModes to ImpSubset since this field indicated the implemented subset of tracing. • Get ready for commercial release, breakup the single file into individual chapter files, fix typos, cross-references, etc.

Table A-1 Revision History

Revision	Date	Description
3.00	November 26, 2002	Changes in this revision: (RT) • Change the way multi-issue tracing is done (see Section 4.3.1, "Background on High Performance Processors" on page 28). • Change the use of PDO_LoadOrder signal to PDO_DataOrder (see Section 4.3.4, "Out-of-Order Loads and Stores in the Multi-Pipe Core" on page 31). • Increase the width of PDO_DataOrder signal to 4 bits (see Table 3-1 on page 7). • Add a new signal called PDO_DataPerIns[7:0] (see Table 3-1 on page 7). • Allow PDO_TEnd to be asserted early to cut off redundant upper bits of an address or data (see Table 3-1 on page 7). • Add a section to clarify how tracing is handled for store conditional instructions (see Section 4.8, "Tracing Store Conditionals" on page 40). • Make the PDO_TMode bit 0 value for PDO_TType values of TD, TU1, and TU2 to be Reserved. • Add PDO_Trig signals on the PDtrace interface that transmit trace trigger information to the TCB. See Section 4.4, "Trace Trigger from EJTAG Hardware Instruction/Data Breakpoints" on page 32. • Add MIPS16 in MIPS64 option to ISAM in TMOAS. See Table 4-2 on page 23. • Rewrite the trace enable equation to fix errors in the first version. See Section 4.6, "Trace Enabling/Disabling Condition" on page 39. • Fix grammatical errors and typos.
3.01	May 14, 2003	Removed the trace slot-specific signals PDO_TrigI_n, PDO_TrigD_n, PDO_TrigOn, and PDI_TrigOff, since these are superfluous. Fix minor typos.