15. Floating-Point Unit

This section describes the operation of the FPU, including the register definitions.

The Floating-Point unit consists of the following functional units:

- add unit
- multiply unit
- divide unit
- square-root unit

The **add unit** performs floating-point add and subtract, compare, and conversion operations. Except for Convert Integer To Single-Precision (float), all operations have a 2-cycle latency and a 1-cycle repeat rate.

The **multiply unit** performs single-precision or double-precision multiplication with a 2-cycle latency and a 1-cycle repeat rate.

The **divide and square-root units** do single- or double-precision operations. They have long latencies and low repeat rates (20 to 40 cycles).

15.1 Floating Point Unit Operations

The floating-point add, multiply, divide, and square-root units read their operands and store their results in the floating-point register file. Values are loaded to or stored from the register file by the load/store and move units.

A logic diagram of floating-point operations is shown in Figure 15-1, in which data and instructions are read from the secondary cache into the primary caches, and then into the processor. There they are decoded and appended to the floating-point queue, passed into the FP register file where each is dynamically issued to the appropriate functional unit. After execution in the functional unit, results are stored, through the register file, in the primary data cache.

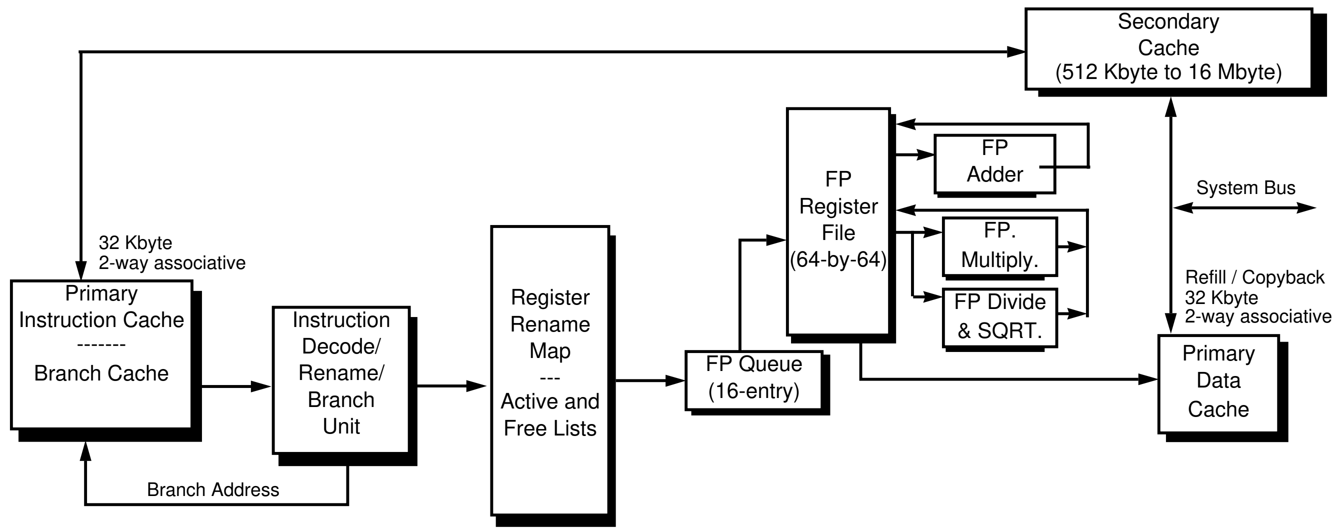

Figure 15-1 Logical Diagram of FP Operations

The floating-point queue can issue one instruction to the adder unit and one instruction to the multiplier unit. The adder and multiplier each have two dedicated read ports and a dedicated write port in the floating-point register file.

Because of their low repeat rates, the divide and square-root units do not have their own issue port. Instead, they decode instructions issued to the multiplier unit, using its operand registers and bypass logic. They appropriate a second cycle later for storing their result.

When an instruction is issued, up to two operands are read from dedicated read ports in the floating-point register file. After the operation has been completed, the result can be written back into the register file using a dedicated write port. For the add and multiply units, this write occurs four cycles after its operands were read.

15.2 Floating-Point Unit Control

The control of floating-point execution is shared by the following units:

- The floating-point queue determines operand dependencies and dynamically issues instructions to the execution units. It also controls the destination registers and register bypass.
- The execution units control the arithmetic operations and generate status.
- The graduate unit saves the status until the instructions graduate, and then it updates the *Floating-Point Status* register.

15.3 Floating-Point General Registers (FGRs)

The Floating-Point Unit is the hardware implementation of Coprocessor 1 in the MIPS IV Instruction Set Architecture. The MIPS IV ISA defines 32 logical floating-point general registers (FGRs), as shown in Figure 15-2. Each FGR is 64 bits wide and can hold either 32-bit single-precision or 64-bit double-precision values. The hardware actually contains 64 physical 64-bit registers in the Floating-Point Register File, from which the 32 logical registers are taken.

FP instructions use a 5-bit logical number to select an individual FGR. These logical numbers are mapped to physical registers by the rename unit (in pipeline stage 2), before the Floating-Point Unit executes them. Physical registers are selected using 6-bit addresses.

32- and 64-Bit Operations

The *FR* bit (26) in the *Status* register determines the number of logical floating-point registers available to the program, and it alters the operation of single-precision load/store instructions, as shown in Figure 15-2.

- *FR* is reset to 0 for compatibility with earlier MIPS I and MIPS II ISAs, and instructions use only the 16 physical even-numbered floating-point registers (32 logical registers). Each logical register is 32 bits wide.
- *FR* is set to 1 for normal MIPS III and MIPS IV operations, and all 32 of the 64-bit logical registers are available.

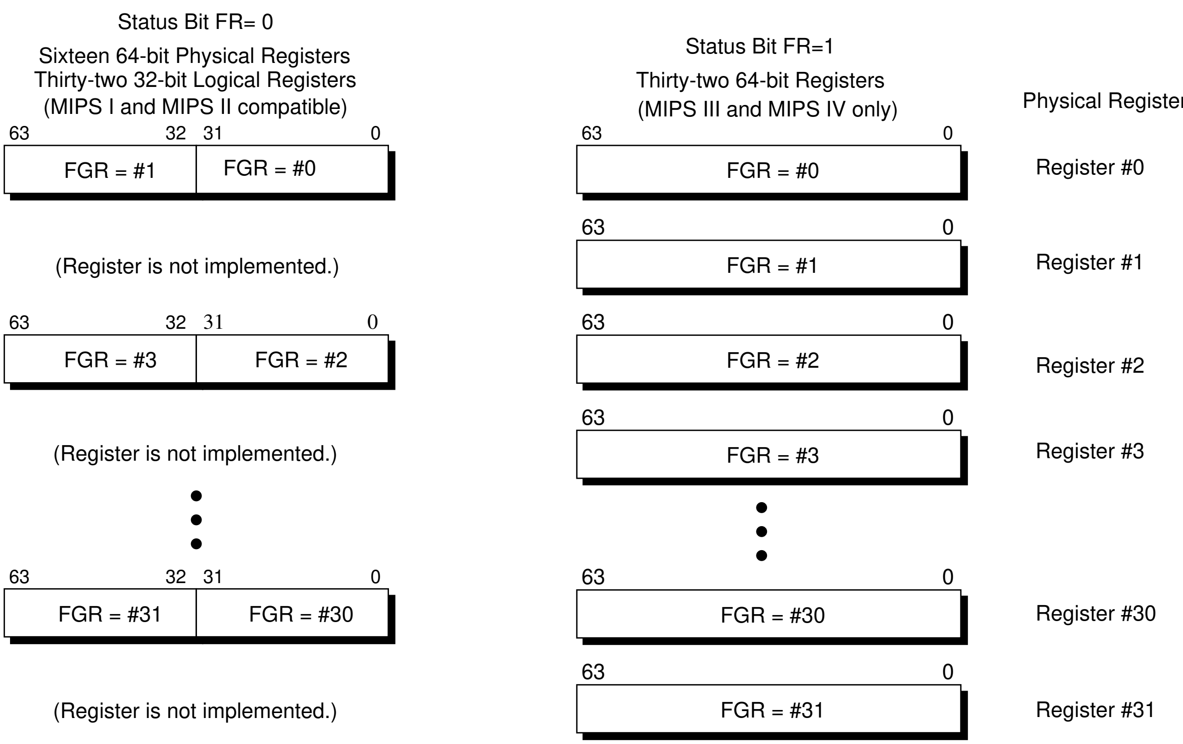

Figure 15-2 Floating-Point Registers

Load and Store Operations

When FR = 0, floating-point load and stores operate as follows:

- A doubleword load or store is handled the same as if the FR bit was set to 1, as long as the register selected is even (0, 2, 4, etc.).
- If the register selected is odd, the load/store is invalid.

These operations are shown in Figure 15-3. Singleword loads/stores to even and odd registers are also shown.

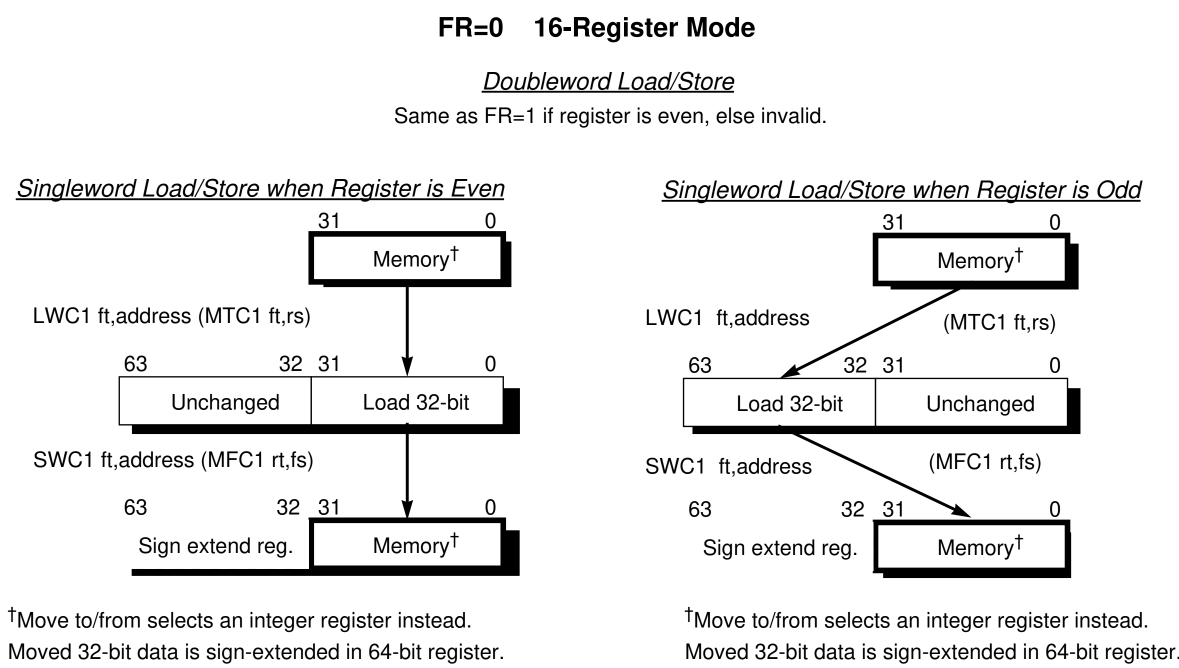

Figure 15-3 Loading and Storing Floating-Point Registers in 16-Register Mode

NOTE: Move (MOV) and conditional move (MOVC, MOVN, MOVZ are included in these arithmetic operations, although no arithmetic is actually performed.

When FR = 1, floating-point load and stores operate as follows:

- Single-precision operands are read from the low half of a register, leaving the upper half ignored. Single-precision results are written into the low half of the register. The high half of the result register is architecturally undefined; in the R10000 implementation, it is set to zero.
- Double-precision arithmetic operations use the entire 64-bit contents of each operand or result register.

Because of register renaming, every new result is written into a temporary register, and conditional move instructions select between a new operand and the previous old value. The high half of the destination register of a single-precision conditional move instruction is undefined (shown in Figure 15-5), even if no move occurs.

Singleword and doubleword loads and stores with the FPU in 32-register mode (FR=1) are shown in Figure 15-4.

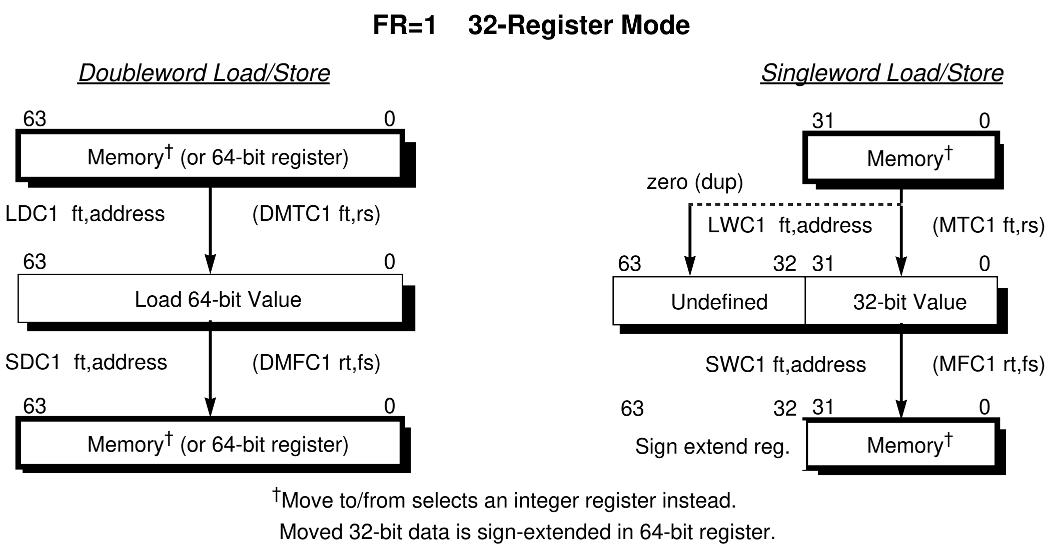

Figure 15-4 Loading and Storing Floating-Point Registers in 32-Register Mode

Doubleword load, store and move to/from instructions load or store an entire 64-bit floating-point register, as shown in Figure 15-5.

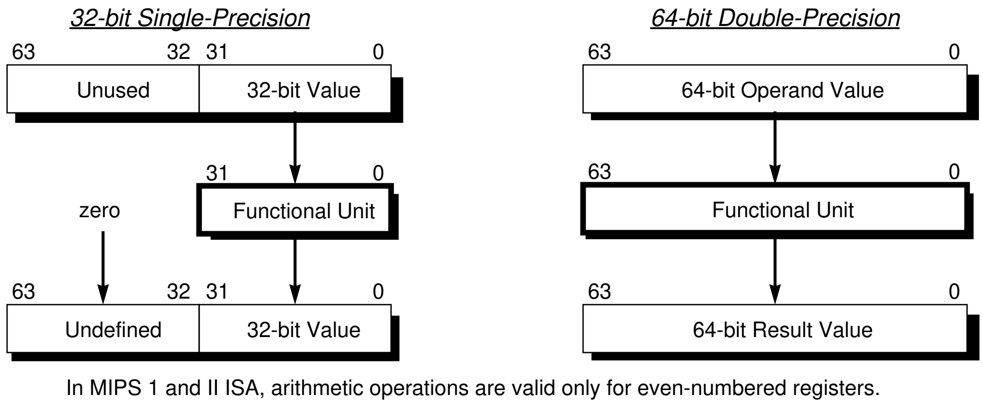

Figure 15-5 Operators on Floating-Point Registers

In MIPS I and MIPS II ISAs, all arithmetic instructions, whether single- or double-precision, are limited to using even register numbers. Load, store and move instructions transfer only a single word. Even and odd register numbers are used to access the low and high halves, respectively, of double-precision registers. When storing a floating-point register (SWC1 or MFC1), the processor reads the entire register but writes only the selected half to memory or to an integer register.

Because the register renaming scheme creates a new physical register for every destination, it is not sufficient just to enable writing half of the Floating-Point register file when loading (LWC1 or MTC1); the unchanged half must also be copied into the destination. This old value is read using the shared read port, it is then merged with the new word, and the merged doubleword value is written. (A write to the register file writes all 64 bits in parallel.)

When instructions are renamed in MIPS I or II, the low bit of any FGR field is forced to zero. Thus, each even/odd logical register number pair is treated as an even-numbered double-precision register. Odd numbered logical registers are not used in the mapping tables and dependency logic, but they remain mapped to their latest physical registers.

15.4 Floating-Point Control Registers

The MIPS IV ISA permits up to 32 control registers to be defined for each coprocessor, but the Floating-Point Unit uses only two:

- Control register 0, the FP *Implementation and Revision* register
- Control register 31, the *Floating-Point Status* register (*FSR*)

Floating-Point *Implementation and Revision* Register

The following fields are defined for control register 0 in Coprocessor 1, the FP *Implementation and Revision* register, as shown in Figure 15-6:

- The *Implementation* field holds an 8-bit number, 0x09, which identifies the R10000 implementation of the floating point coprocessor.
- The *Revision* field is an 8-bit number that defines a particular revision of the floating point coprocessor. Since it can be arbitrarily changed, it is not defined here.

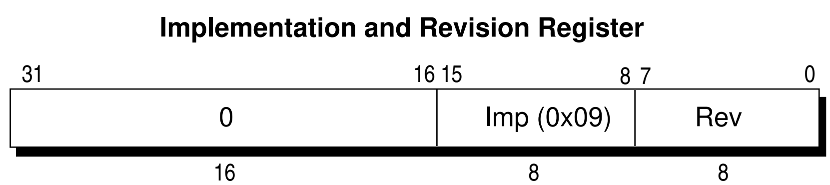

Figure 15-6 FP Implementation and Revision Register Format

Floating-Point Status Register (FSR)

Figure 15-7 shows the Floating-Point *Status* register (*FSR*), control register 31 in Coprocessor 1. It is implemented in the graduation unit rather than the Floating-Point Unit, because it is closely tied to the active list.

Bits 22:18 are unimplemented and must be set to zero. All other bits may be read or written using Control Move instructions from or to Coprocessor 1 (subfunctions CFC1 or CTC1). These move instructions are fully interlocked; they are delayed in the decode stage until all previous instructions have been graduated, and no subsequent instruction is decoded until they have been completed.

FP Status Register

31	30	29	28	27	26	25	24	23	22 21 20 19 18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1 0
7	6	5	4	3	2	1	FS	0	0	E	V	Z	O	U	I	V	Z	O	U	I	V	Z	O	U	I	RM
1	1	1	1	1	1	1	1	1	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
Condition Bits 7..0										Cause						Enables					Flags					

Condition bits are True/False values set by floating-point compare instructions.
Flush (FS) bit: 0: A denormalized result causes an Unimplemented Operation exception.
1: A denormalized result is replaced with zero. No exception is flagged.
Cause bits indicate the status of each floating-point arithmetic instruction. (Not by load, store, or move.)
Enable bits enable an exception if the corresponding *Cause* bit is set.
Flag bits are set whenever the corresponding *Cause* bit is a 1. These bits are cumulative. Once a bit is set, it remains set until the FSR is written by a CTC1 instruction.

E Unimplemented operation. This exception is always enabled.
IEEE 754 Exception bits: The following bits may be individually enabled:
V Invalid operation.
Z Division by zero. (Divide unit only.)
O Overflow.
U Underflow.
I Inexact operation. (Result can not be stored precisely.)

Round Mode (RM): (IEEE specification)
0: *RN*, Round to nearest representable value. If two values are equally near, set the lowest bit to zero.
1: *RZ*, Round toward Zero. Round to the closest value whose magnitude is not greater than the result.
2: *RP*, Round to Plus Infinity. Round to the closest value whose magnitude is not less than the result.
3: *RM*, Round to Minus Infinity. Round to the closest value whose magnitude is not greater.

Figure 15-7 Floating-Point Status Register (FSR)

Bit Descriptions of the FSR

Description of the bits in the FSR are as follows:

Condition Bits [31:25,23]: The *Condition* bits indicate the result of floating-point compare instructions. The active list keeps track of these bits.

Cause Bits [17:12]: Each functional unit can detect exceptional cases in their function codes, operands, or results. These cases are indicated by setting one of six specific *Cause* bits. The *Cause* bits indicate the status of the floating-point arithmetic instruction which graduated most recently or caused an exception to be taken. The *FSR* is not modified by load, store, or move instructions. All cause bits, except *E,* have corresponding *Enable* and *Flag* bits in the FSR.

E *Unimplemented operation*: the execution unit does not perform the specified operation. This exception is always enabled.

V *Invalid operation*: this operation is not valid for the given operands.

Z *Division by zero*: (divide unit only) the result of division by zero is not defined.

O *Overflow*: the result is too large in magnitude to be correctly represented in the result format.

U *Underflow*: the result is too small in magnitude to be correctly represented in the result format.

I *Inexact Result*: the result cannot be represented exactly.

NOTE: The *FSR* is modified only for instructions issued by the floating-point queue. Move From (MFC or DMFC) instructions never set the *Cause* field; status bits from the functional unit (multiplier) must be ignored. Move or Move Conditional instructions can set the Unimplemented Operation exception only in the *Cause* field. Load and store instructions are issued by the address queue.)

The functional units generate the *Cause* bits and send them to the graduation unit when the operation is completed.

Enable Bits [11:7]: The five *Enable* bits individually enable (when set to a 1) or disable (when set to a 0) exceptions when the corresponding *Cause* bit is set.

Flag Bits [6:2]: One of the five *Flag* bits is set when a floating-point arithmetic instruction graduates, if the corresponding *Cause* bit is set. The *Flag* bits are sticky and remain set until the *FSR* is written. Thus, the *Flag* bits indicate the status of all floating-point instructions graduated since the *FSR* was last written. The *Flag* bits are not modified for any instructions which cause an exception to be taken.

Round Mode [1:0]: RM bits select one of the four IEEE rounding modes. Most floating-point results cannot be precisely represented by the 32-bit or 64-bit register formats, and must be truncated and rounded to a representable value. The modes selected by the *RM* bit values are:

0: *RN*, round to nearest representable value. If two values are equally near, set the lowest bit to zero.

1: *RZ*, round toward zero. Round to the closest value whose magnitude is not greater than the result.

2: *RP*, round to plus infinity. Round to the closest value whose magnitude is not less than the result.

3: *RM*, round to minus infinity. Round to the closest value whose magnitude is not greater.

The *Round* and *Enable* bits only change when the *FSR* is written by a CTC1 (Move To Coprocessor 1 Control Register) instruction. Each CTC1 instruction is executed sequentially, after all previous floating-point instructions have been completed, so these *FSR* bits do not change while any floating-point instruction is active. These bits are broadcast from the graduation unit to all the floating-point functional units.

When a *Cause* bit is set and its corresponding *Enable* bit is also set, an exception is taken on the instruction. The result of the instruction is not stored, and the *Flag* bits are not changed. If no exception is taken, the corresponding *Flag* bits are set.

The *Cause* and *Flag* bits may be read or written. If a CTC1 instruction sets both a *Cause* bit and its *Enable* bit, an exception is taken immediately. The *FSR* is written, but the exception is reported on the move instruction.

Loading the FSR

The *FSR* may be loaded from an integer register by a CTC1 instruction which selects control register 31. This instruction is executed serially; that is, it is delayed during decode until the entire pipeline has emptied, and it is completed before the next instruction is decoded. This instruction writes all *FSR* bits.

If any *Cause* bit and its corresponding *Enable* bit are both set, an exception is taken after FSR has been modified. The CTC1 instruction is aborted; it does not graduate, even though it has changed the processor state.

15.5 FPU Instructions

This section describes the R10000 processor-specific implementations of the following FPU instructions:

- CVT.L.fmt
- moves and conditional moves
- CFC1/CTC1CVT.L.fmt

CVT.L.fmt

The CVT.L.fmt instruction has a slight change in the R10000 processor implementation. The R4400 processor allows conversion from a single or a double up to a 53-bit long integer. If the result is greater than 53 bits after the conversion, an Unimplemented Operation exception is taken. A back- conversion from a 53-bit long integer to single/double also takes an Unimplemented Operation exception.

In the R10000, the conversion allows only up to 52 bits; otherwise an Unimplemented Operation exception is taken. The back-conversion from a 52-bit long integer to single/double no longer takes an Unimplemented Operation exception.

Moves and Conditional Moves

The only legal formats for the move and conditional move instructions are single and double precision. The move instructions do not trap if their operands are either denormalized or NaNs, which is consistent with the R4400 implementation. Execution of floating-point move and conditional move instructions do not affect the *Cause* field of the floating-point *Status* register unless they take an Unimplemented Operation exception because an illegal formats was used.†

The upper 32 bits of the destination registers are undefined in architecture for all the floating-point arithmetic operations in single-precision or 32-bit fixed format (S or W). In the R10000 processor, the implementation clears the upper 32 bits, including MOV.S, whereas R4400 and R4200 processors preserve the upper 32 bits during the move.

For the floating-point conditional move instructions, MOVT.S, MOVF.S, MOVZ.S, and MOVN.S, the R10000 processor always clears the upper 32 bits of the destination register even though the condition is false.

In *32 floating-point register* mode (FR=1), the upper 32 bits of the destination register for the MTC1 and LWC1 instructions are architecturally undefined. The R10000 processor implementation clears the upper 32 bits.

CFC1/CTC1

There are only two valid Floating-Point Control registers: 0 and 31. Access to other registers is undefined.

† The *Cause* field is set to 100000 (*E* bit is 1).