ALLEGREX™ FPU User's Manual

Table of Contents

1. ALLEGREX™ FPU Overview

1.1. Main Features

The ALLEGREX™ FPU is an arithmetic calculation unit that performs single-precision floating-point operations in conformance with the IEEE 754 specification. The ALLEGREX™ FPU is tightly coupled to the ALLEGREX™ CPU as a coprocessor, enabling it to execute floating-point instructions in the MIPS ISA.

Figure 1-1 shows a block diagram of the ALLEGREX™ FPU.

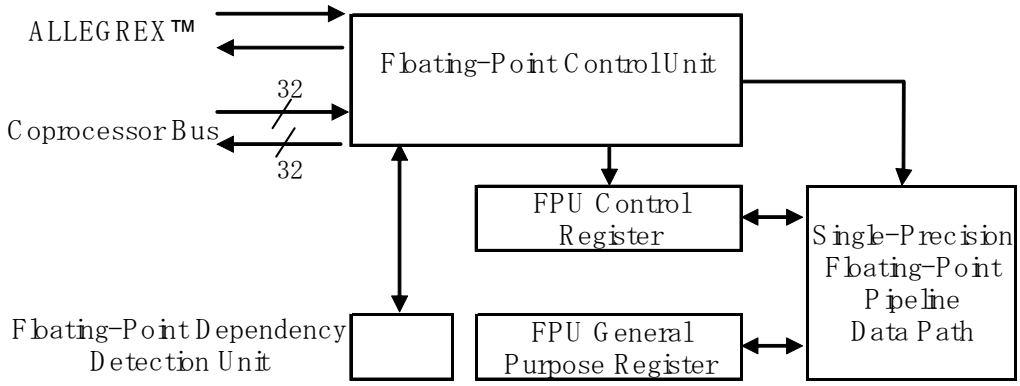


Figure 1-1: ALLEGREX™ FPU Block Diagram

Coprocessor configuration

As a tightly-coupled coprocessor (CP1), the ALLEGREX™ FPU extends the instruction set of the ALLEGREX™ CPU with the ability to perform floating-point arithmetic operations. One of the system control coprocessor (CP0) registers in the ALLEGREX™ CPU is the Status Register, which has bits (CU fields) for controlling the use of each coprocessor. The execution of floating-point instructions can be enabled by setting the CU1 bit.

Independent eight-stage pipeline

The floating-point pipeline of the ALLEGREX™ FPU is independent of the integer arithmetic pipeline of the ALLEGREX™ CPU. This enables floating-point instructions to be processed in parallel with integer instructions. The ALLEGREX™ FPU pipeline is eight stages long and will only stall when an instruction is being executed that depends on the result of a previous operation or when a multi-cycle floating-point instruction is being executed.

IEEE 754 support

The ALLEGREX™ FPU conforms to the ANSI/IEEE 754-1985 specification for single precision, namely, the IEEE standard for binary floating-point arithmetic. The FPU also supports recommendations included in that standard and many detailed exceptions of the MIPS architecture.

1.2. FPU Registers

The ALLEGREX™ FPU has 32 FPU general-purpose registers (FGRs) and two FPU control registers.

1.2.1. FPU General-purpose Registers (FGRs)

The ALLEGREX™ FPU has thirty-two 32-bit FPU general-purpose registers (FGRs). These registers can be used for storing single-precision floating-point numbers. The FGRs can be accessed from the CPU by executing the move instructions (MFC1, MTC1), the load instruction (LWC1), or the store instruction (SWC1).
Figure 1-2 shows the assignment of the FGRs.

Register No.	FGR General Register (FGR)
fs	31 0
00000	FGR0
00001	FGR1
00010	FGR2
00011	FGR3
---	---
11100	FGR28
11101	FGR29
11110	FGR30
11111	FGR31

Figure 1-2: FPU General-purpose Registers (FGRs)

1.2.2. FPU Control Registers (FCRs)

The ALLEGREX™ FPU has two FPU control registers (FCRs). One is the Implementation/Revision register (FCR0) and the other is the Control/Status register (FCR31).

FCR1 ~ FCR30 are reserved and unimplemented.

The FCRs can only be accessed from the CPU by using the dedicated move instructions (CFC1, CTC1).

Table 1-1 shows the assignment of the FCRs.

Table 1-1: FPU Control Registers (FCRs)

FCR number	Usage
FCR0	Revision information
FCR1 ~ FCR30	Reserved
FCR31	Rounding mode, exception causes, trap enables, flags

Implementation/Revision register (FCR0)

The Implementation/Revision register (FCR0) is a read-only register that contains design information and the revision number of the ALLEGREX™ FPU. Applications must not be written that depend on the revision number.

Figure 1-3 shows the fields of the Implementation/Revision register and Table 1-2 gives their descriptions.

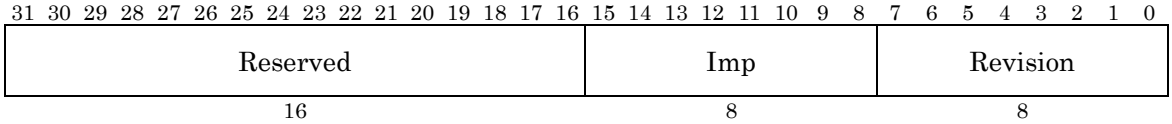


Figure 1-3: Implementation/Revision Register

Table 1-2: FCR0 Fields

Field	Description
Reserved	Reserved (read as “0”)
Imp	Code showing design information
Revision	Code showing revision number

Control/Status register (FCR31)

The Control/Status register (FCR31) is a read/write register that holds control and status information. This register has bits that control the rounding mode for arithmetic operations and trap-enable bits for floating-point operations. The register also has bits for recording the status of untrapped exceptions generated during floating-point operations.

The Control/Status register can be accessed using the CFC1 and CTC1 instructions regardless of the operating mode. Although the CTC1 instruction can be used to set or clear bits in the Control/Status register, FCR31 must be updated when you perform non-decimal point processing in order to guarantee the contents of registers.

Figure 1-4 shows the fields of the Control/Status register and Table 1-3 gives their descriptions.

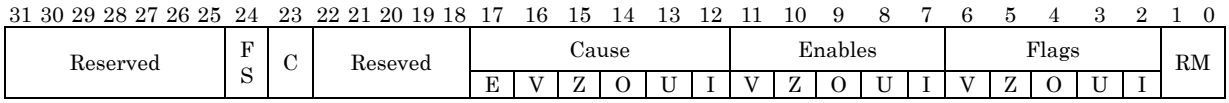
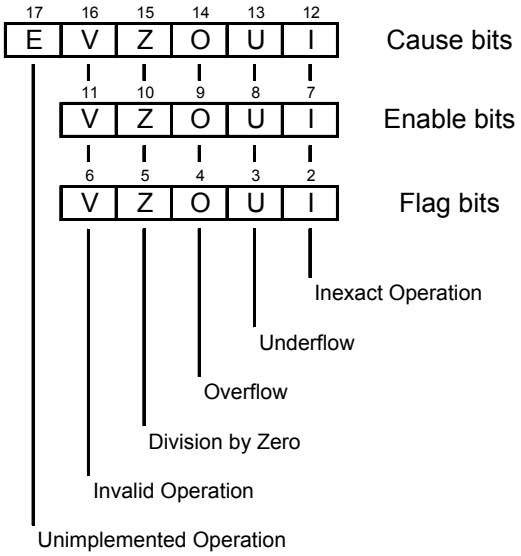


Figure 1-4: Control/Status Register

Table 1-3: FCR31 Fields

Field	Description
FS	Processing mode for denormalized numbers. When this bit is 1, a denormalized result will be flashed to 0.
C	Result of floating-point compare. 1, if the result of a floating-point comparison is true, otherwise 0.
Cause	Status of exceptions generated by floating-point operations. When an IEEE 754 exception occurs, the corresponding bit is set to 1.
Enables	Exception enable flags (software trap control). When these bits are set to 1, the corresponding IEEE 754 exceptions are reported to the CPU as FPU exceptions.
Flags	Exception generation flags. These bits are set when the corresponding IEEE 754 exception occurs but the exception is not reported to the CPU because the corresponding Enable bit is clear.
RM	Rounding mode
Reserved	Reserved (read as 0)



1. FS bit

Bit 24 of the Control/Status register (FCR31) is the FS bit (flash bit).

When this bit is set to 1, the result of a floating-point operation will be flashed to 0 if the

result cannot be normalized (a denormalized number).

2. C bit

Bit 23 of the Control/Status register (FCR31) is the C bit (condition bit).
The value of the C bit is set to 1 when the result of a floating-point compare instruction is true and set to 0 when the result is false. The C bit is only affected by a floating-point compare instruction and the CTC1 instruction.

3. Cause bits

Bits 17 to 12 of the Control/Status register (FCR31) are the Cause bits.
The Cause bits are logical extensions of the CP0 Cause register and show the cause of IEEE 754 exceptions generated by the last floating-point operation. If an instruction generates more than one IEEE 754 exception, each corresponding bit is set to 1.
The Cause bits are updated by the conversion instructions, arithmetic instructions (except for LOAD, STORE and MOV.S), CTC1 instruction, reserved instructions and unimplemented instructions.
When any of the Cause bits are set to 1 and the corresponding Enable bits are also set to 1, an FPU exception will be generated for the CPU. If one or more exceptions occur during the execution of an instruction, each of the relevant Cause bits will be set. If software emulation is needed, the Unimplemented Operation(E) bit will be set to 1. In all other cases, the Unimplemented Operation(E) bit will be set to 0. The setting of the other bits indicates whether or not an IEEE 754 exception was generated. When a floating point exception occurs, it is reflected only in the corresponding Cause bit. The actual results from processing are not stored. For details on FPU exceptions refer to “4 Floating-point Exceptions.”
If an Enable bit is 0 then an FPU exception will not be raised for the CPU even if the corresponding Cause bit is set to 1. From this state, if a register operation is performed by the CTC1 instruction such that the Cause bit becomes 0 and the Enable bit becomes 1 at the same time, then an unexpected FPU exception will be raised. In order to prevent this, an operation which first zeroes out the CTC1 so that the Cause and Enable bits do not change at the same time should be done first.

4. Enable bits

Bits 11 to 7 of the Control/Status register (FCR31) are the Enable bits.
The Enable bits control whether or not an FPU exception will be generated for the CPU when an IEEE 754 exception occurs. When any of the Enable bits is set to 1 and the corresponding Cause bits are also set to 1, an FPU exception will be generated. Similarly, if the CTC1 instruction sets both a Cause bit and its corresponding Enable bit, an exception

will be generated. However, there is no Enable bit for the Unimplemented Operation(E) exception. This exception will only occur due to a floating-point exception. To prevent repeated exceptions from occurring, you must use the CTC1 instruction to clear the Cause bit before returning from floating-point exception processing. As a result, it is not possible for a user mode program to determine if the Cause bit was enabled. If a user mode handler needs this information, temporarily save the value of the Status register in a separate location from where it can be accessed. If only the Cause bit is set, IEEE 754 stipulates that only the default result should be stored without generating an exception. In this case, the exception which occurred immediately before can be determined by reading the Cause bit.
For details on FPU exceptions refer to "4 Floating-point Exceptions."
The Enable bits are only updated by the CTC1 instruction.
When setting an Enable bit to 1, it is sometimes necessary to zero out FCR31 first so that the Cause bit and Enable bit do not change at the same time.

5. Flag bits

Bits 6 to 2 of the Control/Status register (FCR31) are the Flag bits.
A Flag bit is set to 1 when its corresponding IEEE 754 exception occurs but an FPU exception is not generated for the CPU because the Enable bit for that exception is cleared. There are no other cases when the Flag bits are changed. In other words, when the Enable bits allow an FPU exception to be generated, the Flag bits will not change even though an IEEE 754 exception occurs.
The Flag bits are not cleared during floating-point operations. Because of this, the Flag bits will indicate any FPU exceptions that occur after a reset which are not reported to the CPU and have accumulated. However, the CTC1 instruction can be used to clear (or set) the bits in software.
For details on FPU exceptions refer to "4 Floating-point Exceptions."

6. RM bits

Bits 1 and 0 of the Control/Status register (FCR31) are the RM (rounding mode control) bits.
The settings of the RM bits determine the rules that will be used during all floating-point operations. They control how values will be approximated when the exact results of an operation cannot be represented. Table 1-4 lists the available rounding modes.

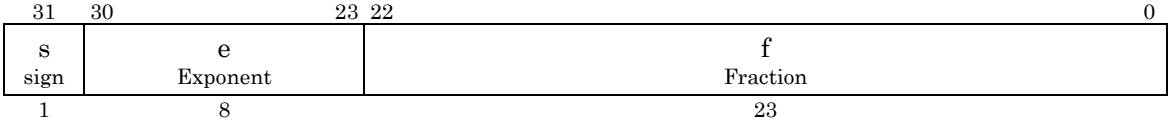
Table 1-4: RM Bits and Rounding Modes

RM bits [1:0]	Rounding mode	Description
00	RN	Rounds to a value that is closest to the exact value but can still be represented. When the exact value lies midway between two representable values, rounds to the value whose least significant bit is 0.
01	RZ	Rounds towards 0. Namely, rounds to a value that is closest to the exact value without exceeding its absolute value with unbounded precision.
10	RP	Rounds towards +∞. Namely, rounds to the closest value that is greater than or equal to the exact value with unbounded precision.
11	RM	Rounds towards -∞. Namely, rounds to the closest value that is less than or equal to the exact value with unbounded precision.

1.3. Number Representation Formats

1.3.1. Single-precision Floating-point Format (S)

The ALLEGREX™ FPU processes 32-bit (single-precision) IEEE 754 floating-point arithmetic. A single-precision floating-point number consists of a 24-bit signed fraction (s+f) and an 8-bit exponent (e). Figure 1-5 shows single-precision floating-point format.



Item	Specification
Total length (number of bits)	32
Fraction length (number of bits)	23
Integer bit	Implied
Exponent length (number of bits)	8
Exponent bias value	+127
Exponent range (E_{max} to E_{min})	+127 to -126

Figure 1-5: Single-precision Floating-point Format

The meaning of each field is as follows.

- s: Sign
- e: The part of the exponent that includes the bias. Produced by adding the bias (+127) to the actual exponent (E) (-126 to +127)
- f: Fraction, $f = .b_1b_2b_3b_4\ldots b_{23}$

Table 1-5 shows the meaning of different combinations of s, e and f. Every number has exactly one binary representation, except for 0.

Table 1-5: Floating-point Number Representations

Condition	Numeric value
$E = E_{max} + 1$ and $f \neq 0$	NaN（regardless of s）
$E = E_{max} + 1$ and $f = 0$	$(-1)^s \infty$
$E_{min} \leq E \leq E_{max}$	$(-1)^s 2^E (1.f)$
$E = E_{min} - 1$ and $f \neq 0$	$(-1)^s 2^{Emin} (0.f)$
$E = E_{min} - 1$ and $f = 0$	$(-1)^s 0$

Table 1-6 shows the range of values that can be represented in floating-point format.

Table 1-6: Range of Floating-point Numbers

Item	Value（decimal）
Smallest number that can be represented	1.40129846e−45
Smallest number that can be represented (normalized)	1.17549435e−38
Largest number that can be represented	3.40282347e+38

Representing zero

Figure 1-6 and Figure 1-7 show binary representations of +0 (positive zero) and -0 (negative zero), respectively. The comparison instructions of the ALLEGREX™ FPU treat these values as being identical.

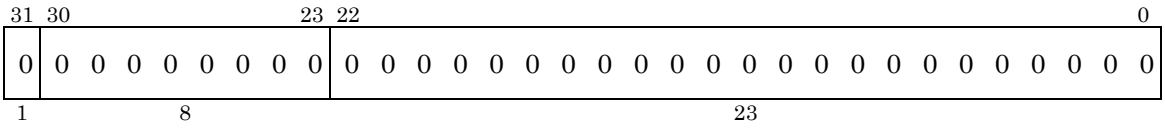


Figure 1-6: Binary Representation of +0 (positive zero)

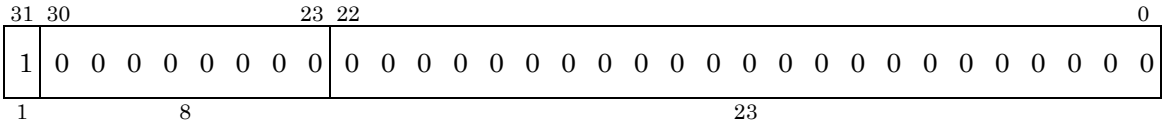


Figure 1-7: Binary Representation of -0 (negative zero)

Representing denormalized numbers

Numbers having an exponent E smaller than E_{min} cannot be represented as normalized numbers and are handled as denormalized numbers in the IEEE 754 specification.

Figure 1-8 shows the binary representation of a denormalized number.

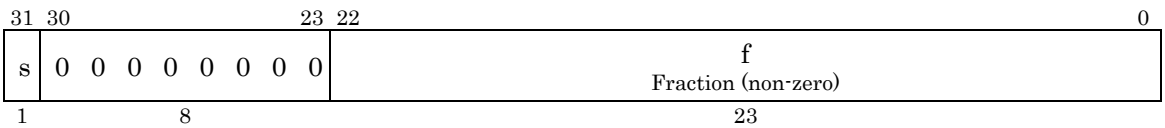


Figure 1-8: Binary Representation of a Denormalized Number

Representing infinity

Numbers having an exponent E larger than E_{max} cannot be represented as normalized numbers and are handled as infinity in the IEEE 754 specification.

Figure 1-9 and Figure 1-10 show binary representations of +∞ (positive infinity) and -∞ (negative infinity), respectively.

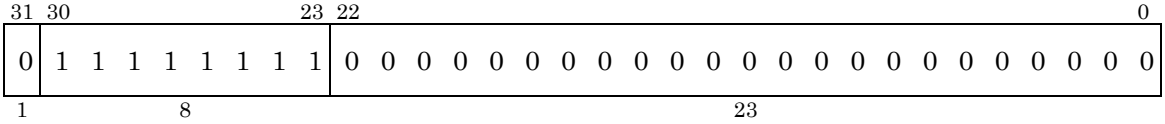


Figure 1-9: Binary Representation of +∞ (positive infinity)

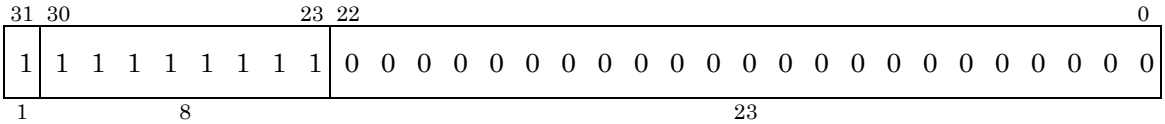


Figure 1-10: Binary Representation of -∞ (negative infinity)

Not-a-Number (NaN)

The IEEE 754 specification defines a floating-point value called a NaN (Not-a-Number). A NaN is not a numeric value so it does not have a sign, nor is its magnitude defined. Two types of NaNs are defined, a signaling NaN and a quiet NaN. They are distinguished by the most significant bit of the fraction f.

Figure 1-11 shows the binary representation of the signaling NaN and Figure 1-12 shows the binary representation of the quiet NaN. Note that other MIPS processors have different representations for these NaNs.

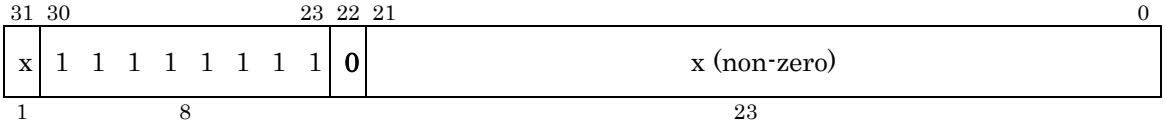


Figure 1-11: Binary Representation of Signaling NaN

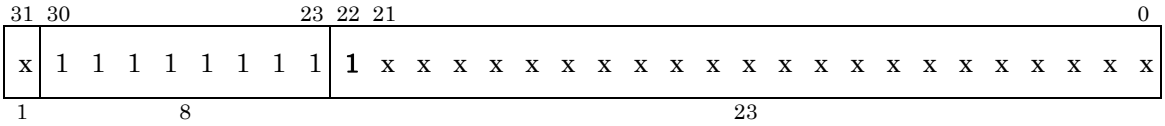


Figure 1-12: Binary Representation of Quiet NaN

1.3.2. 32-bit Fixed-point Format (W)

32-bit fixed-point format is used to represent signed 32-bit integers using two's complement notation. Figure 1-13 shows 32-bit fixed-point format.

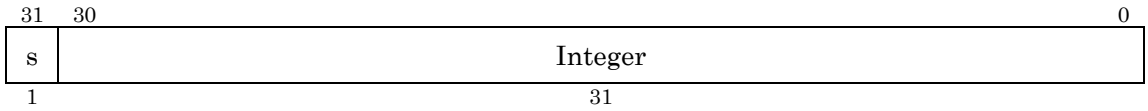


Figure 1-13: 32-bit Fixed-point Format

- s: sign
- Integer: Integer part

Note that unsigned fixed-point numbers cannot be handled directly by the ALLEGREX™ FPU floating-point instruction set.

2. FPU Instruction Set Overview

All FPU instructions are 32 bits long and aligned on a word boundary. FPU instructions are classified as follows.

- Load, Store, Move Instructions
- Conversion Instructions
- Arithmetic Instructions
- Comparison Instructions
- FPU Branch Instructions

2.1. Data Formats

In FPU instructions that handle numeric data, the data format of the operands is specified by a 5-bit fmt field appended to the instruction opcode. Table 2-1 shows how this fmt field is interpreted.
The ALLEGREX™ FPU supports only two types of data formats for the fmt field: S (single-precision floating point) and W (32-bit fixed-point). D (double-precision floating-point) and L (64-bit fixed-point) are not supported. If either of these is specified, an Unimplemented Instruction exception will occur.

Table 2-1: fmt Field of FPU Instructions

Code [25:21]	Mnemonic	Data size	Format	Notes
10000	S	Single precision (32 bit)	Binary floating-point	
10001	D	Double precision (64bit)	Binary floating-point	Cannot be used (*)
10010-10011			Undefined (reserved)	
10100	W	Single (32 bit)	Binary fixed-point	
10101	L	Long word (64bit)	Binary fixed-point	Cannot be used (*)
10110-11111			Undefined (reserved)	

(*) If specified, an Unimplemented Instruction exception will occur.

Table 2-2 shows combinations of valid FPU instructions and data formats that can be used with the ALLEGREX™ FPU. A “O” symbol indicates a valid combination. An “×” symbol indicates that an Unimplemented Instruction exception will occur. A “–” symbol indicates an invalid combination even though no exception will occur.

Table 2-2: Valid Combinations of FPU Instructions and Data Formats

Operation	fmt			
	S	D	W	L
ADD	O	×	–	×
SUB	O	×	–	×
MUL	O	×	–	×
DIV	O	×	–	×
SQRT	O	×	–	×
ABS	O	×	–	×
MOV	O	×	–	×
NEG	O	×	–	×
TRUNC.W	O	×	–	×
ROUND.W	O	×	–	×
CEIL.W	O	×	–	×
FLOOR.W	O	×	–	×
CVT.S	×	×	O	×
CVT.W	O	×	–	×
C	O	×	–	×

2.2. Load / Store / Move Instructions

1. Loading and Storing Data Between FPU and Memory

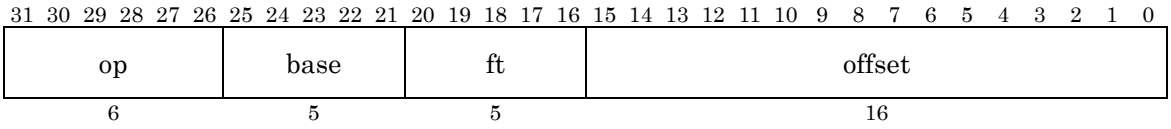
Loads and stores between the general-purpose registers of the FPU (FGRs) and memory are performed by the following two instructions.

- LWC1
- SWC1

These instructions handle bit patterns directly. Their operation is independent of the data format and they do not perform any format conversion. As a result, these instructions do not generate floating-point exceptions.

Table 2-3: FPU Load / Store Instructions

Format



Instructions

Instruction	Format	Description
Load Word to FPU	LWC1 ft,offset(base)	The offset is sign-extended and added to the contents of the base register to generate an address. The memory word at this address is loaded into FPU register ft.

Instruction	Format	Description
Store Word from FPU	SWC1 ft,offset(base)	The offset is sign-extended and added to the contents of the base register to generate an address. The contents of FPU register ft are stored in the memory word at this address.

2. Moving Data between FPU and CPU

Data is transferred between an FPU general-purpose register (FGR) and the CPU using the following two instructions.

- MTC1
- MFC1

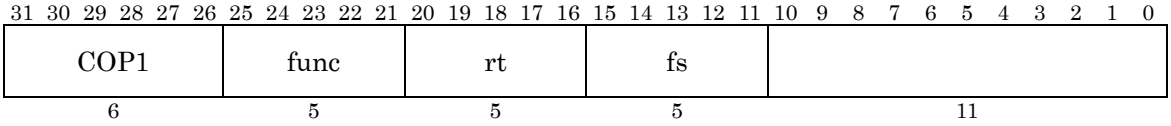
These instructions handle bit patterns directly. Their operation is independent of the data format and they do not perform any format conversion. As a result, these instructions do not generate floating-point exceptions.

Data is transferred between an FPU control register (FCR) and the CPU using the following two instructions.

- CTC1
- CFC1

Table 2-4: FPU Move Instructions

Format



Instructions

Instruction	Format	Description
Move Word To FPU	MTC1 rt,fs	Move contents of CPU register rt to FPU general-purpose register fs.
Move Word From FPU	MFC1 rt,fs	Move contents of FPU general-purpose register fs to CPU register rt.
Move Control Word To FPU	CTC1 rt,fs	Move contents of CPU register rt to FPU control register fs.
Move Control Word From FPU	CFC1 rt,fs	Move contents of FPU control register fs to CPU register rt.

3. Load Delay and Hardware Interlock

An FPU general-purpose register (FGR) that is the target of a Load to FPU (LWC1) or Move to FPU (MTC1) instruction can be used by other instructions immediately after the LWC1/MTC1 instructions complete. It is not necessary to insert NOPs because the hardware will automatically interlock the pipeline. However, since this adds extra cycles, it

is necessary to schedule the load delay slot properly in order to avoid extra delay.

The FPU branch instructions (BC1F, BC1FL, BC1T, BC1TL) that reference condition bits in the Control/Status register cannot be executed immediately after the Move to FPU control register instruction (CTC1). In this case, the smallest number of NOPs should be inserted. Note that the CTC1 instruction cannot be placed in the delay slot of the BC1* instructions.

4. Alignment of Data

Data accessed by all FPU Load / Store instructions is aligned on a 32-bit word boundary. Access is always by word, so the low-order 2 bits of the address must be 0.

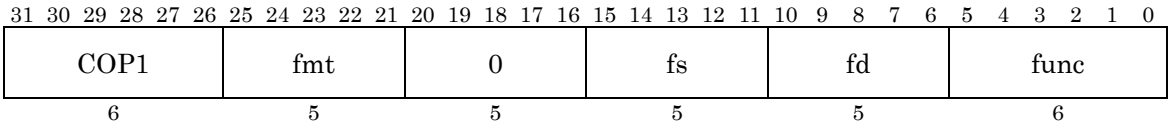
2.3. Conversion Instructions

Conversion instructions convert the formats of single-precision floating-point numbers and binary fixed-point numbers in FPU general-purpose registers.

In the ALLEGREX™ FPU, only two types of data formats can be specified in the fmt field for the source operand of a conversion instruction: S (single-precision floating point) and W (32-bit fixed-point). D (double-precision floating-point) and L (64-bit fixed-point) are not supported. If either of these is specified, an Unimplemented Instruction exception will occur.

Table 2-5: FPU Conversion Instructions

Format



Instructions

Instruction	Format	Description
Floating-point Convert To Single Floating-point	CVT.S.fmt fd,fs	Convert the contents of FPU register fs from the format specified by fmt to single-precision floating-point. Round the result and store it in FPU register fd.
Floating-point Convert To Word Fixed-point	CVT.W.fmt fd,fs	Convert the contents of FPU register fs from the format specified by fmt to 32-bit fixed point. Round the result and store it in FPU register fd.
Floating-point Round Convert To Word Fixed-point	ROUND.W.fmt fd,fs	Convert the contents of FPU register fs from the format specified by fmt, by rounding to the closest value that can be represented in 32-bit fixed point format then store the result in FPU register fd.

Instruction	Format	Description
Floating-point Truncate Convert To Word Fixed-point	TRUNC.W.fmt fd,fs	Convert the contents of FPU register fs from the format specified by fmt to 32-bit fixed point by rounding the result towards 0, then store it in FPU register fd.
Floating-point Ceiling Convert To Word Fixed-point	CEIL.W.fmt fd,fs	Convert the contents of FPU register fs from the format specified by fmt to 32-bit fixed point by rounding the result towards +∞, then store it in FPU register fd.
Floating-point Floor Convert To Word Fixed-point	FLOOR.W.fmt fd,fs	Convert the contents of FPU register fs from the format specified by fmt to 32-bit fixed point by rounding the result towards -∞, then store it in FPU register fd.

2.4. Arithmetic Instructions

The arithmetic instructions perform arithmetic operations on floating-point values in FPU general-purpose registers.

In the ALLEGREX™ FPU, only two types of data formats can be specified in the fmt field: S (single-precision floating point) and W (32-bit fixed-point). D (double-precision floating-point) and L (64-bit fixed-point) are not supported. If either of these is specified, an Unimplemented Instruction exception will occur.

Table 2-6: Arithmetic Instructions (3 operand)

Format

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
COP1	fmt	ft	fs	fd	func
6	5	5	5	5	6

Instructions

Instruction	Format	Description
Floating-point Add	ADD.fmt fd,fs,ft	Arithmetically add the contents of FPU register fs to the contents of FPU register ft in the format specified by fmt. Round the result and store it in FPU register fd.
Floating-point Subtract	SUB.fmt fd,fs,ft	Arithmetically subtract the contents of FPU register ft from the contents of FPU register fs in the format specified by fmt. Round the result and store it in FPU register fd.
Floating-point Multiply	MUL.fmt fd,fs,ft	Arithmetically multiply the contents of FPU register fs by the contents of FPU register ft in the format specified by fmt. Round the result and store it in FPU register fd.

Instruction	Format	Description
Floating-point Divide	DIV.fmt fd,fs,ft	Arithmetically divide the contents of FPU register fs by the contents of FPU register ft in the format specified by fmt. Round the result and store it in FPU register fd.

Table 2-7: FPU Arithmetic Instructions (2 operand)

Format

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
COP1	fmt	0	fs	fd	func
6	5	5	5	5	6

Instructions

Instruction	Format	Description
Floating-point Absolute Value	ABS.fmt fd,fs	Find the arithmetic absolute value of the contents of FPU register fs in the format specified by fmt and store the result in FPU register fd.
Floating-point Move	MOV.fmt fd,fs	Move the contents of FPU register fs in the format specified by fmt to FPU register fd.
Floating-point Negate	NEG.fmt fd,fs	Reverse the sign of the contents of FPU register fs in the format specified by fmt and store the result in FPU register fd.
Floating-point Square Root	SQRT.fmt fd,fs	Find the arithmetic positive square root of the contents of FPU register fs in the format specified by fmt. Round the result and store it in FPU register fd.

2.5. Comparison Instructions

The comparison instructions (C.cond.fmt) interpret the contents of two FPU general-purpose registers (fs, ft) as being in the data format specified by fmt, then compare them using the condition specified by cond.

In the ALLEGREX™ FPU, only two types of data formats can be specified in the fmt field: S (single-precision floating point) and W (32-bit fixed-point). D (double-precision floating-point) and L (64-bit fixed-point) are not supported. If either of these is specified, an Unimplemented Instruction exception will occur.

Table 2-8: FPU Comparison Instructions

Format

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
COP1	fmt	ft	fs	0	FC	cond
6	5	5	5	5	2	4

Instructions

Instruction	Format	Description
Floating-point Compare	C.cond.fmt fs,ft	Interpret the contents of FPU registers fs and ft as being in the format specified by fmt. Arithmetically compare the two registers and determine whether or not the comparison result matches the condition specified by cond. The comparison result can be referenced by an FPU branch instruction after a 1 instruction delay.

As can be seen in Table 2-9, an FPU branch instruction will need to test 32 conditions to find out the true/false condition of the coprocessor. However, as can be seen in Table 2-10, this can be done in as few as 16 comparisons depending on whether the logic of the condition is inverted.

Table 2-9: Comparison Conditions

cond [3:0]	Mnemonic	Condition	Mnemonic	Condition
0000	F	False	T	True
0001	UN	Unordered	OR	Ordered
0010	EQ	Equal	NEQ	Not Equal
0011	UEQ	Unordered or Equal	OLG	Ordered Less Than or Greater Than
0100	OLT	Ordered Less Than	UGE	Unordered or Greater Than or Equal
0101	ULT	Unordered or Less Than	OGE	Ordered Greater Than
0110	OLE	Ordered Less Than or Equal	UGT	Unordered or Greater Than
0111	ULE	Unordered or Less Than or Equal	OGT	Ordered Greater Than
1000	SF	Signaling False	ST	Signaling True
1001	NGLE	Not Greater Than or Less Than or Equal	GLE	Greater Than or Less Than or Equal
1010	SEQ	Signaling Equal	SNE	Signaling Not Equal
1011	NGL	Not Greater Than or Less Than	GL	Greater Than or Less Than
1100	LT	Less Than	NLT	Not Less Than
1101	NGE	Not Greater Than or Equal	GE	Greater Than or Equal
1110	LE	Less Than or Equal	NLE	Not Less Than or Equal
1111	NGT	Not Greater Than	GT	Greater Than

Table 2-10: Change in Meaning of Conditional Test According to Inversion of Logic

Condition			Relationship				Does invalid operation exception occur when unordered?
Mnemonic		Cond [3:0]	Greater than	Less than	Equal	Unordered	
True	False						
F	T	0000	F	F	F	F	No
UN	OR	0001	F	F	F	T	No
EQ	NEQ	0010	F	F	T	F	No
UEQ	OGL	0011	F	F	T	T	No
OLT	UGE	0100	F	T	F	F	No
ULT	OGE	0101	F	T	F	T	No
OLE	UGT	0110	F	T	T	F	No
ULE	OGT	0111	F	T	T	T	No
SF	ST	1000	F	F	F	F	Yes
NGLE	GLE	1001	F	F	F	T	Yes
SEQ	SNE	1010	F	F	T	F	Yes
NGL	GL	1011	F	F	T	T	Yes
LT	NLT	1100	F	T	F	F	Yes
NGE	GE	1101	F	T	F	T	Yes
LE	NLE	1110	F	T	T	F	Yes
NGT	GT	1111	F	T	T	T	Yes

F : False

T : True

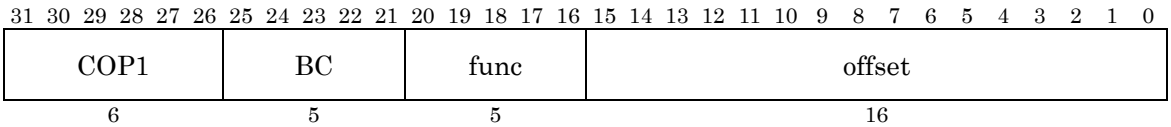
2.6. FPU Branch Instructions

FPU branch instructions branch according to the result of a previously executed comparison instruction (C.cond.fmt). One NOP must be inserted between a comparison instruction and the FPU branch instruction that references its result.

FPU branch instructions have a branch delay slot equal to one instruction. The instruction in the branch delay slot is executed before the branch operation. However, with a Branch Likely instruction, the instruction in the delay slot is nullified (discarded) when the branch is not taken.

Table 2-11: FPU Branch Instructions

Format



Instructions

Instruction	Format	Description
Branch on FPU True	BC1T offset	When the result of the previously executed comparison instruction is true(1), the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the offset after it has been left-shifted 2 bits and sign-extended to a 32-bit value.
Branch on FPU True Likely	BC1TL offset	When the result of the previously executed comparison instruction is true(1), the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the offset after it has been left-shifted 2 bits and sign-extended to a 32-bit value. If the branch is not taken, the instruction immediately after the branch instruction is nullified.
Branch on FPU False	BC1F offset	When the result of the previously executed comparison instruction is false(0), the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the offset after it has been left-shifted 2 bits and sign-extended to a 32-bit value.
Branch on FPU False Likely	BC1FL offset	When the result of the previously executed comparison instruction is false(0), the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the offset after it has been left-shifted 2 bits and sign-extended to a 32-bit value. If the branch is not taken, the instruction immediately after the branch instruction is nullified.

3. FPU Pipeline

3.1. Overview

The ALLEGREX™ FPU provides an instruction pipeline with eight stages as shown below.

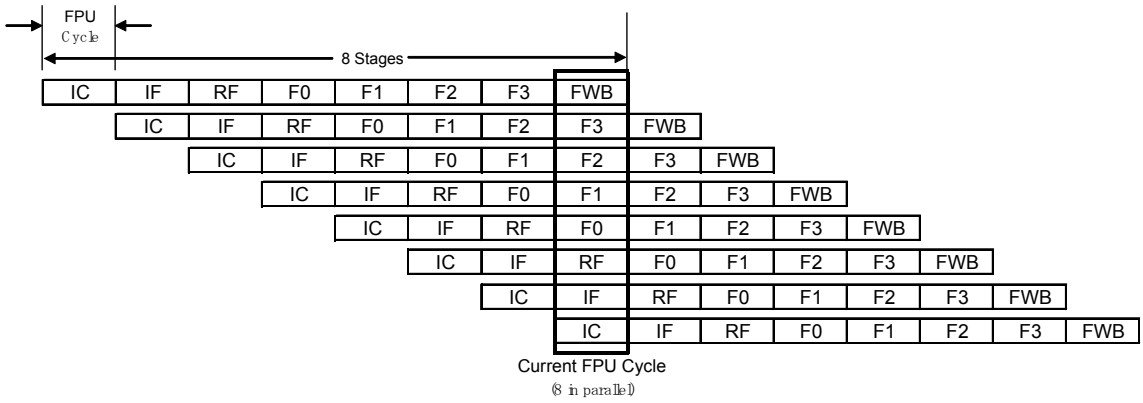


Figure 3-1: FPU Pipeline

The ALLEGREX™ FPU pipeline will not stall except when it is performing an operation that depends on the result of another operation, or when it is executing a multi-cycle instruction (division and square root). Table 3-1 shows the smallest number of pipeline cycles required for each FPU instruction.

Table 3-1: Minimum Number of Cycles to Execute FPU Instructions

Operation	No. of pipeline cycles	Operation	No. of pipeline cycles
ADD.fmt	1	BC1F	1
SUB.fmt	1	BC1TL	1
MUL.fmt	1	BC1FL	1
DIV.fmt	28	LWC1	1
SQRT.fmt	28	SWC1	1
ABS.fmt	1	CVT.S.fmt	1
MOV.fmt	1	CVT.W.fmt	1
NEG.fmt	1	MTC1	1
ROUND.fmt	1	MFC1	1
CEIL.W.fmt	1	CTC1	1
FLOOR.W.fmt	1	CFC1	1
TRUNC.W.fmt	1	C.cond.fmt	1
BC1T	1		

3.2. Synchronization with CPU Pipeline

The seven-stage pipeline of the ALLEGREX™ CPU and the eight-stage pipeline of the ALLEGREX™ FPU are synchronized as shown in the figures below.

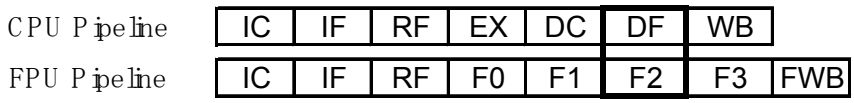


Figure 3-2: CPU and FPU Pipelines

Pipelines are synchronized at the DF/F2 stage when moving data between the FPU and CPU or memory. The instructions that perform the synchronization are the FPU Load/Store instructions (LWC1, SWC1), the FPU Move instructions (MFC1, MTC1) and the FPU Control Register Move instructions (CFC1, CTC1).

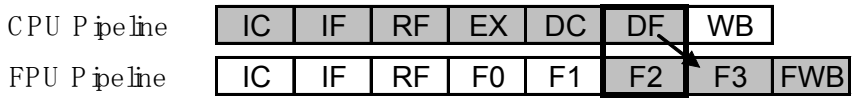


Figure 3-3: CPU → FPU Pipeline Synchronization (MTC1, CTC1)

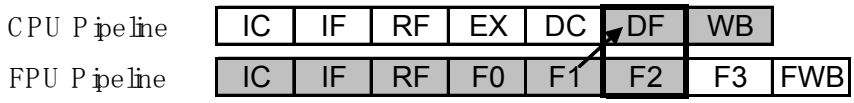


Figure 3-4: FPU → CPU, MEM Pipeline Synchronization (MFC1, CFC1, SWC1)

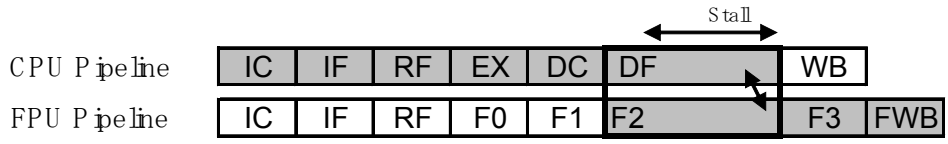


Figure 3-5: MEM → FPU Pipeline Synchronization (LWC1)

3.3. FPU Pipeline Interlocks

The figures below show timing and pipeline operation for the different types of pipeline interlocks in the ALLEGREX™ FPU.

FPU operation that uses the result of a previous FPU operation

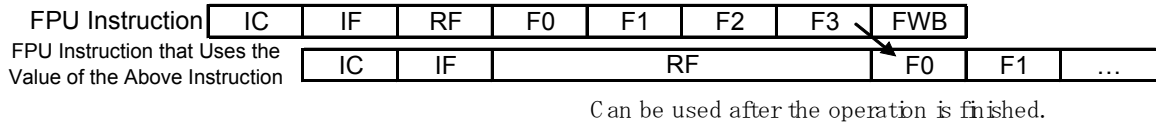


Figure 3-6: FPU Pipeline Operation When a Dependency With Another FPU Operation Occurs

FPU operation that moves data from the CPU and depends on that data

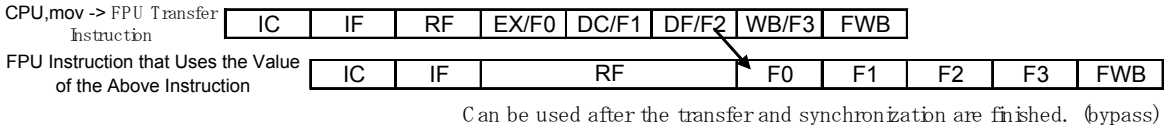


Figure 3-7: FPU Pipeline Operation When a Dependency With a CPU → FPU Operation Occurs

Writing to the same register

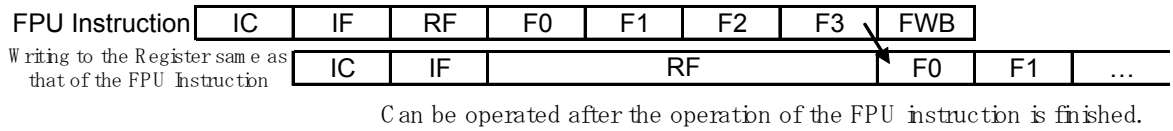


Figure 3-8: FPU Pipeline Operation When Successively Writing to the Same Register

Multi-cycle instructions (division, square root)

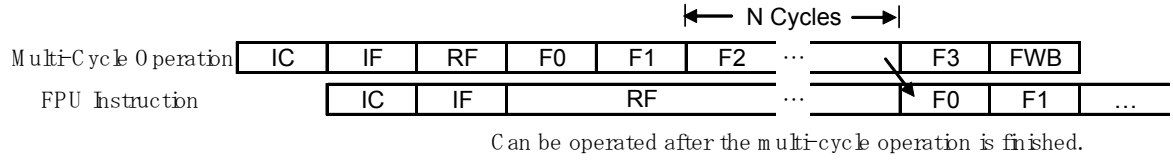


Figure 3-9: FPU Pipeline Operation When Executing Multi-cycle Instructions

Writing to a control register

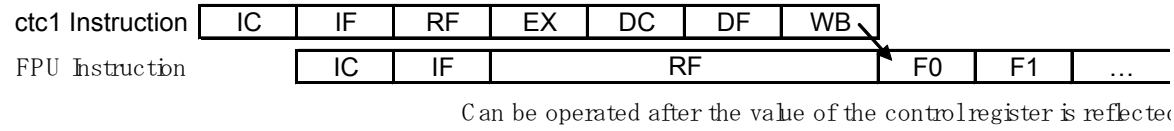


Figure 3-10: FPU Pipeline Operation During Control Register Write

Instruction cache miss

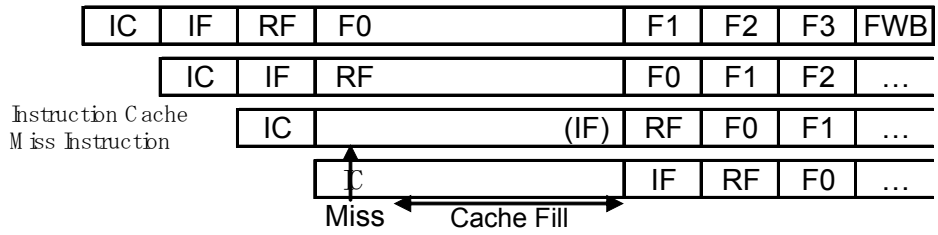


Figure 3-11: FPU Pipeline Operation When Instruction Cache Miss Occurs

Data cache miss

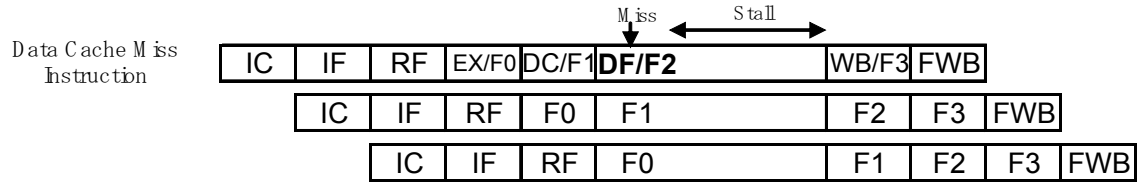


Figure 3-12: FPU Pipeline Operation When Data Cache Miss Occurs

4. Floating-point Exceptions

The ALLEGREX™ FPU is based on the IEEE 754 specification and will generate an IEEE 754 exception when either an operand or the result of an operation cannot be processed normally. There are two approaches to dealing with exceptions when they occur and these can be set up in advance by software. One approach is to have the FPU exception cause a software trap for the CPU. The other approach is to set a status flag and use a default value as the result.
In addition, because the ALLEGREX™ FPU is based on a single-precision specification, performing a 64-bit (double-precision) operation will cause a trap to occur, resulting in an Unimplemented Instruction exception.
Please refer to the discussion of exceptions in the “ALLEGREX™ User's Manual” for more information.

4.1. Generation of Floating-point Exceptions

The ALLEGREX™ FPU supports the following five types of IEEE 754 exceptions.

- I: Inexact Operation exception
- O: Overflow exception
- U: Underflow exception
- Z: Division by Zero exception
- V: Invalid Operation exception

Software traps can be individually enabled and disabled for these exceptions by using the corresponding Enable bits of the Control/Status register (FCR31).
The Unimplemented Instruction exception (E) is an additional exception that can be reported by the ALLEGREX™ FPU. However, this exception is always generated for the CPU and its software trap cannot be disabled. The bits of the Control/Status register are shown in Figure 4-1

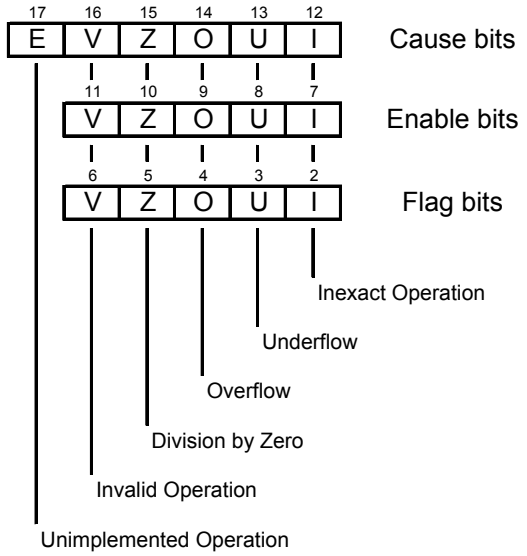


Figure 4-1: Cause, Flags, Enable Fields of the Control/Status Register

4.1.1. Generation Conditions for IEEE 754 Exceptions

When any one of the seven states shown in Table 4-1 is detected internally, the ALLEGREX™ FPU will generate an IEEE 754 exception.

Table 4-1: Generation Conditions for IEEE 754 Exceptions

State	IEEE 754	ALLEGREX™ FPU
Inexact operation (result is not exact)	I	I
Exponent overflow (normalized exponent > E_{max})	O, I (*)	O, I
Division by Zero (exponent of divisor = E_{min} −1, fraction = 0)	Z	Z
Overflow during conversion (source operand is out of integer range)	V	V
Signaling NaN (source operand is Signaling NaN)	V	V
Invalid operation (such as 0 / 0)	V	V
Exponent underflow (normalized exponent < E_{min})	U	U

(*) In IEEE 754, if Overflow exceptions are disabled, an Inexact Operation exception will occur in these cases.

4.1.2. Exception Generation Action

When an IEEE 754 exception occurs, the Cause bit of the Control/Status register (FCR31)

corresponding to the exception is set to 1. What happens next depends on the setting of the Enable bit.

When traps are enabled

If the Enable bit corresponding to the exception's Cause bit is set to 1, an FPU exception will be generated for the CPU. The software can use an FPU exception handler (software trap) to perform processing that corrects abnormal operations.

When traps are disabled

If the Enable bit corresponding to the exception's Cause bit is set to 0, an FPU exception will not be generated for the CPU. Instead, a default value defined in IEEE 754 will be stored as the operation result and the corresponding Flags bit will be set to 1.

Note that there is no Enable bit corresponding to an Unimplemented Instruction exception. When the Cause bit (E) is set to 1, an FPU exception will always be generated for the CPU.

An IEEE 754 exception will also be generated when the CTC1 instruction is executed and an Enable bit is set to 1 at the same time its corresponding Cause bit is set when the Control/Status register (FCR31) is written.
Normally, as long as a Cause bit and the corresponding Enable bit do not become 1 at the same time, an FPU exception will not be raised, but if a register operation is performed by the CTC1 instruction in which the Cause bit changes from 1 to 0 and the Enable bit changes from 0 to 1, then an unexpected FPU exception is raised. In order to prevent this, when changing the Enable bit from 0 to 1, the CTC1 instruction must be divided into two instances such that the FCR31 register is first zeroed out and then changed to the target value. In addition, be aware of the fact that the CTC1 instruction cannot be issued successively.

4.2. FPU Exception Handler Processing

In user mode, in place of the five standard exceptions, IEEE 754 strongly recommends implementing exception handlers for storing calculated results in the destination register.

4.2.1. Distinguishing FPU Exception Causes

When an FPU exception is generated, the cause of the exception can be determined from the system control coprocessor (CP0) Cause register and the FPU's Control/Status register.
When the ExcCode field of the system control coprocessor (CP0) Cause register contains the value FPE (FPU exception), it means that the FPU caused the exception. Each bit of the Cause

field in the FPU's Control/Status register indicates the type of exception that occurred.

4.2.2. Inexact FPU Exceptions

Because the number of pipeline stages in the ALLEGREX™ CPU and ALLEGREX™ FPU are different, when an IEEE 754 exception is generated for an operation result, please be careful because the EPC register may not be pointing to the address where the program will resume.

FPU exceptions whose location can be precisely determined

For the FPU exceptions shown below, the instruction that caused the exception can be precisely determined using the system control coprocessor (CP0) EPC register and the BD bit of the Cause register.

- Invalid Operation exception
- Division by Zero exception
- Unimplemented exception

FPU exceptions whose location cannot be precisely determined

The FPU exceptions shown below are detected after a floating-point operation is performed.

- Inexact Operation exception
- Overflow exception
- Underflow exception

For these exceptions, the floating-point operation will not have been completed when the CPU is in the DF stage where exceptions are generated, so there is a possibility that the CPU might already be executing the next instruction when the exception is actually generated. Consequently, the instruction that generated the FPU exception cannot be precisely determined even if the EPC register is referenced. The instruction that caused the exception must by inferred by a technique such as using the EPC register to find the location of the previous instruction, then checking the source register of that instruction.

4.2.3. Saving / Restoring FPU State

Normally, an FPU exception handler saves and restores the state of the FPU general-purpose registers in memory 32 words at a time by executing the SWC1 and LWC1 instructions. The information in the Control/Status register (FCR31) is also saved and restored in memory via a CPU register, using the CFC1 and CTC1 instructions. Normally, saving of the Control/Status register is done as the very first step, and restoring the register is done as the very last step. Note that when the Control/Status register is restored after exception processing, the Cause bit will indicate that an unprocessed IEEE 754 exception is still pending. If the Control/Status register is restored as is, the same FPU exception will occur again. Consequently, after processing is completed, the Cause bit should be cleared first before restoring the register.

4.3. Default Values when Traps are Disabled

When an Enable bit in the Control/Status register is set to 0 and the corresponding IEEE 754 exception occurs, the FPU will write a default value to the destination register as the result. The default value is determined by the type of exception and the rounding mode as shown in Table 4-2.

Table 4-2: Default Values When Traps are Disabled

IEEE 754 exception	Rounding mode	Default value
I (Inexact operation)	All	Rounded result
O (Overflow)	RN	∞ with sign of intermediate result
	RZ	Largest finite number with sign of intermediate result
	RP	Negative overflow: Largest finite normalized number Positive overflow: +∞
	RM	Positive overflow: Largest positive finite number Negative overflow: −∞
U (Underflow)	RN	0 with sign of intermediate result
	RZ	0 with sign of intermediate result
	RP	Positive underflow: Smallest positive finite number Negative underflow: 0
	RM	Negative underflow: Smallest negative finite number Positive underflow: 0
Z (Division by Zero)	All	∞ with exact sign
V (Invalid operation)	All	Quiet NaN（Not-a-Number）

When traps are disabled and an IEEE 754 exception occurs, the corresponding Flag bit of the Control/Status register is set to 1. The Flag bit is not cleared to 0 by a normal FPU instruction but can be set and reset by writing new values using the CTC1 instruction.

4.4. Detailed Description of Each FPU Exception

The following section gives the cause of each FPU exception and the resultant action when the exception occurs.

4.4.1. Inexact Operation Exception

The FPU generates an Inexact Operation exception in the following cases.

- When the rounded result is imprecise
- When the rounded result has overflowed
- When the rounded result has underflowed, Underflow and Inexact Operation

exceptions do not have their enable bits set, and the FS bit of the Control/Status register is set.

When an Inexact Operation exception is generated, the actions shown below take place depending on the setting of the Inexact Operation exception Enable bit (bit 7) in the Control/Status register (FCR31).

Bit 7 = 1: Traps enabled

If Inexact Operation exceptions are enabled, the result register will not be updated and the source register will also be saved.

Bit 7 = 0: Traps disabled

If a trap does not simultaneously occur due to another exception, the value that lost precision, underflowed, or overflowed when it was rounded is stored in the destination register as is.

Before executing floating-point processing, the FPU normally determines whether an exception will occur based on the exponents of the operands. If a trap would cause an exception, the instruction is executed using the coprocessor stall function.
It is not possible to predict whether an FPU operation will result in a loss in precision. For example, the instruction that converts a floating-point number to an integer will almost always generate an Inexact Operation exception due to a loss in precision. As a result, care is required when using the Inexact Operation exception Enable bit.

4.4.2. Invalid Operation Exception

When the value of one or both operands is invalid for a given operation, the FPU generates an Invalid Operation exception. When this kind of exception occurs without a trap, MIPS ISA defines it as a NaN. Specifically, this exception occurs in the following cases.

- When one or both operands of an arithmetic instruction is a signaling NaN (The Move instruction (MOV.S) is not considered an arithmetic instruction)
- In addition and subtraction, when both operands are infinity
  Example: $(+\infty) + (-\infty)$ or $(-\infty) - (-\infty)$
- In multiplication, when one operand is 0 and the other operand is infinity
  Example: $0 \times \infty$, regardless of sign
- In division, when both operands are 0 or both are infinity
  Example: 0 / 0 or ∞ / ∞, regardless of sign
- In square root, when the operand is less than 0
- In conversion to an integer, when the operand is out of integer range, infinity or a NaN

- In a comparison operation, when the operand is a signaling NaN
- In a comparison operation, when the operand is a NaN but the comparison condition does not include NaN

When an Invalid Operation exception is generated, the actions shown below take place depending on the setting of the Invalid Operation exception Enable bit (bit 11) in the Control/Status register (FCR31).

Bit 11 = 1: Traps enabled

There is no change to the original operands.

Bit 11 = 0: Traps disabled

If a trap does not simultaneously occur due to another exception, a quiet NaN is stored in the destination register.

4.4.3. Division by Zero Exception

When the divisor is 0 and the dividend is a finite number other than 0, a Division by Zero exception is generated. When this exception occurs, the actions shown below take place depending on the setting of the Division by Zero exception Enable bit (bit 10) in the Control/Status register (FCR31).

Bit 10 = 1: Traps enabled

The contents of the destination register are left unchanged, the contents of the source register are saved, and an FPU exception is generated.

Bit 10 = 0: Traps disabled

If a trap does not simultaneously occur due to another exception, an infinite value ($\pm \infty$) determined by the sign of the operand is stored in the destination register.

4.4.4. Overflow Exception

An Overflow exception occurs when the exponent range is unbounded and the value produced after rounding is larger than the largest finite number that can be represented in the destination format. An Inexact Operation exception is also generated at the same time. An Overflow exception will not be generated by the CVT.W.S instruction that converts floating-point numbers to fixed-point numbers.
When an Overflow exception is generated, the actions shown below take place depending on the setting of the Overflow exception Enable bit (bit 9) in the Control/Status register (FCR31).

Bit 9 = 1: Traps enabled

The contents of the destination register are left unchanged, the contents of the source register are saved, and an FPU exception is generated.

Bit 9 = 0: Traps disabled

If a trap does not simultaneously occur due to another exception, the result is determined by the rounding mode and the sign of the intermediate result.

4.4.5. Underflow Exception

An Underflow exception occurs in the following two cases.

- Underflow: When a small number can cause an exception after the result is within $\pm 2^{Emin}$ (excluding 0)
- Loss of precision: If a loss in precision occurs when the result of an operation on small numbers that are not normalized, is rounded

The IEEE 754 specification permits any method to be used for detecting underflow, but all processing must detect underflow using the same method. The IEEE 754 specification recognizes any of the following methods for detecting underflow.

- After rounding (underflow occurs if the calculation were performed as though the exponent range were unbounded and the result would be within the range $\pm 2^{Emin}$ excluding 0)
- Before rounding (underflow occurs if the calculation were performed as though the exponent range and precision were unbounded and the result would be within the range $\pm 2^{Emin}$ excluding 0)

The ALLEGREX™ FPU detects underflow after rounding, just like other MIPS processors.
In addition, the IEEE 754 specification recognizes any of the following methods for detecting a loss of precision.

- Denormalized loss (loss of precision occurs when there is a difference between the result calculated if the exponent range were unbounded, and the actual result)
- Inexact result (loss of precision occurs when there is a difference between the result calculated if the exponent range and the precision were both unbounded, and the actual result)

The ALLEGREX™ FPU detects loss of precision as an inexact result, just like other MIPS processors.
When an Underflow exception occurs, the actions shown below take place depending on the setting of the Underflow exception Enable bit (bit 8) in the Control/Status register (FCR31).

Bit 8 = 1: Traps enabled

An Underflow or Inexact Operation trap is permitted, but if the FS bit is not set, an Unimplemented Instruction exception will be generated. The contents of the destination register are saved.

Bit 8 = 0: Traps disabled

Underflow and Inexact Operation traps are not permitted, and if the FS bit is not set, a default value determined by the rounding mode is stored in the destination register.

4.4.6. Unimplemented Instruction Exception

An Unimplemented Instruction exception occurs when an attempt is made to execute an instruction with unsupported operation codes or format codes. Specifically, an Unimplemented exception occurs in the following cases.

- When an operand of an instruction, other than the Compare instruction, is denormalized
- When an operand of an instruction, other than the Compare instruction, is a NaN
- When a denormalized result is produced or an underflow occurs, if either an Underflow or Inexact Operation is permitted or the FS bit is not set
- When executing an instruction with a reserved opcode
- When executing an instruction that performs a double-precision floating-point operation
- When an unimplemented data format (D, L) is specified
- When an invalid format is specified in an instruction (example: CVT.S.S)

The Enable field of the Control/Status register does not have a bit that corresponds to the Unimplemented Instruction exception, so traps cannot be disabled. Consequently, an Unimplemented Instruction exception will always generate an FPU exception.