This chapter describes how you can perform comparison and arithmetic numeric operations using PowerPC assembly language. This chapter describes the following types of instructions:

- comparison
- arithmetic
- multiply-add
- move

It shows the format of these instructions and gives examples of use. For complete details on any of these instructions, see the Motorola *PowerPC 601 RISC Microprocessor User's Manual*. For operations that manipulate the floating-point environment, see Chapter 12, "Assembly-Language Environmental Controls." For operations that perform conversions, see Chapter 13, "Assembly-Language Numeric Conversions."

Comparison Operations

The assembler provides two floating-point comparison instructions:

`fcmpo` Ordered comparison

`fcmpu` Unordered comparison

The only difference is that the ordered comparison instruction generates an invalid exception if one of the input registers contains a NaN.

The comparison instructions have three operands. They are of the form

instr *DST*, *SRC1*, *SRC2*

DST A field in the Condition Register (0 through 7) into which the result of the comparison is placed.

SRC1, SRC2 Two floating-point registers.

Comparison instructions are interpreted as

DST ← *SRC1* `compare` *SRC2*

The comparison instructions compare the contents of two floating-point registers and place the results of the comparison in a Condition Register field as well as in bits 16 through 19 (field 4) of the FPSCR. The results in the Condition Register and FPSCR are interpreted as follows:

Result	Meaning
0001	Unordered
0010	*SRC1* = *SRC2*
0100	*SRC1* > *SRC2*
1000	*SRC1* < *SRC2*

Use a conditional branch instruction after the comparison instruction to use the results of the comparison, as shown in the following example:

```
fcmpo    2,f0,f11 # compare f0 to f11 and put result in CR2
blt      2,addr1  # go to addr1 if bit 0 (<) of CR2 is 1
bgt      2,addr2  # go to addr2 if bit 1 (>) of CR2 is 1
beq      2,addr3  # go to addr3 if bit 2 (=) of CR2 is 1
bun      2,addr4  # go to addr4 if bit 3 (unordered) of CR2 is 1
```

Arithmetic Operations

PowerPC assembly language supports five of the seven IEEE arithmetic operations:

- add
- subtract
- multiply
- divide
- round-to-integer

Except for the round-to-integer operation, these operations may be performed by a variety of instructions. The instructions that perform arithmetic operations are divided into three categories: arithmetic instructions, multiply-add instructions, and move instructions. (`fctiw`, described in Chapter 13, "Assembly-Language Numeric Conversions," performs the round-to-integer operation.)

Arithmetic Instructions

There are four arithmetic instructions:

`fadd` Adds two floating-point values.

`fsub` Subtracts two floating-point values.

`fmul` Multiplies two floating-point values.

`fdiv` Divides two floating-point values.

Note
These instructions might raise floating-point exceptions. See the Motorola *PowerPC 601 RISC Microprocessor User's Manual* for more information. ◆

Floating-point arithmetic instructions have three operands, all of which are floating-point registers. They are of the form

instr DST, *SRC1*, *SRC2*

Arithmetic instructions are interpreted as

DST ← *SRC1 op SRC2*

where *SRC1, SRC2,* and *DST* are floating-point registers and *op* is some operation.

Each of these instructions works on both single and double floating-point numbers. There are four versions of each instruction:

instr	Perform operation specified by *instr*. Interpret data in floating-point registers as double format.
instr`.`	Perform operation specified by *instr*. Interpret data in floating-point registers as double format. Record any exceptions raised in the Condition Register.
instr`s`	Perform operation specified by *instr*. Interpret data in floating-point registers as single format.
instr`s.`	Perform operation specified by *instr*. Interpret data in floating-point registers as single format. Record any exceptions raised in the Condition Register.

Note that all exceptions are always recorded in the FPSCR and are sometimes recorded in the Condition Register as well.

The following example adds two double floating-point numbers and stores the results:

```
lfd      f1,d(r1)    # load double number into register f1
lfd      f2,d(r2)    # load double number into register f2
fadd     f0,f1,f2    # f0 contains result
stfd     f0,d(r3)    # store result in double format
```

And the next example adds two single floating-point numbers and stores the results:

```
lfs      f1,d(r4)    # load single number into register f1
frsp     f1,f1       # stay single
lfs      f2,d(r5)    # load single number into register f2
frsp     f2,f2       # stay single
fadds.   f0,f1,f2    # result placed in f0 in single format
                     # CR1 reflects any exceptions
stfs     f0,d(r6)    # store result in single format
```

Multiply-Add Instructions

There are four multiply-add instructions:

`fmadd`	Perform multiply, add.
`fmsub`	Perform multiply, subtract.
`fnmadd`	Perform multiply, add, and negate.
`fnmsub`	Perform multiply, subtract, and negate.

Note
These instructions might raise floating-point exceptions. See the Motorola *PowerPC 601 RISC Microprocessor User's Manual* for more information. ◆

PowerPC assembly language provides the **multiply-add instructions** to perform more complex operations with at most a single roundoff error rather than the two potential roundoff errors that would result from performing the operations separately.

The multiply-add instructions take four operands, all of which are floating-point registers:

instr *DST*, *SRC1*, *SRC2*, *SRC3*

Multiply-add instructions are interpreted as

$DST \leftarrow (SRC1 \times SRC2) \pm SRC3$

where *SRC1*, *SRC2*, *SRC3*, and *DST* are floating-point registers.

Multiply-add instructions can take one of four forms:

instr	Perform operation specified by *instr*. Interpret data in floating-point registers as double format.
instr`.`	Perform operation specified by *instr*. Interpret data in floating-point registers as double format. Record any exceptions raised in the Condition Register.
instr`s`	Perform operation specified by *instr*. Interpret data in floating-point registers as single format.
instr`s.`	Perform operation specified by *instr*. Interpret data in floating-point registers as single format. Record any exceptions raised in the Condition Register.

Note that all exceptions are always recorded in the FPSCR and are sometimes recorded in the Condition Register as well.

The following example multiplies two double-format numbers, adds a third, and stores the result:

```
lfd      f1,d(r1)    # load double number into register f1
lfd      f2,d(r2)    # load double number into register f2
lfd      f3,d(r3)    # load double number into register f3
fmadd    f0,f1,f2,f3 # f0 = f1 × f2 + f3
stfd     f0,d(r4)    # store result as double format
```

The following example performs the same operations on single-format numbers:

```
lfs      f1,d(r5)    # load single number into register f1
frsp     f1,f1       # stay single
lfs      f2,d(r6)    # load single number into register f2
frsp     f2,f2       # stay single
lfs      f3,d(r7)    # load single number into register f3
frsp     f3,f3       # stay single
fmadds.  f0,f1,f2    # f0 = f1 × f2 + f3
                     # f0 contains single format number
                     # CR1 reflects any exceptions
stfs     f0,d(r8)    # store result in single format
```

Move Instructions

There are four move instructions:

`fabs`	Move absolute value of register.
`fmr`	Move register value.
`fneg`	Move negative value of register.
`fnabs`	Move negative absolute value of register.

Move instructions perform sign manipulations while copying a value from one floating-point register to another. Because they manipulate only the sign bit, they generate no floating-point exceptions. They take two operands, both of which are floating-point registers. They are of the form

instr *DST*, *SRC*

Floating-point move instructions are interpreted as

DST ← *op SRC*

where *SRC* and *DST* are floating-point registers and *op* is some operation that is performed on the contents of *SRC*.

Note that you may copy a value from a register into the same register. For example:

```
fneg  f1,f1    # f1 has just been negated
```

Transcendental and Auxiliary Functions

PowerPC assembly language does not directly support any of the IEEE auxiliary functions or the transcendental functions listed in this book. If you are writing a numerics application in assembly language, you can access the routines in the C library MathLib to perform these operations, provided you set up the stack frame properly. For information on how to set up the stack frame, see the book *Assembler for Macintosh With PowerPC*.