# 5

# *M68000 Addressing Modes and Assembler Instructions*

This chapter contains information specific to the Motorola M68000 processor architecture, including the MC68040, MC68030 processors, and the MC68882 floating point coprocessor. The first section, ªM68000 Registers and Addressing Modes,º lists the registers available and describes the addressing modes used by assembler instructions. The second section, ªM68000 Assembler Instructions,º lists each assembler instruction with NeXT and Motorola syntax.

## M68000 Registers and Addressing Modes

This section describes the conventions used to specify instruction mnemonics and addressing modes. The instructions themselves are detailed in the next section, ªM68000 Assembler Instructions.º

# Instruction Mnemonics

The instruction mnemonics that the assembler uses are based on the mnemonics described in the relevant Motorola processor manuals. However, the NeXT assembler uses the MIT assembler syntax, which deviates from the standard Motorola syntax in several areas. Points to be aware of include the following:

- Instead of using a qualifier of **.b**, **.w**, or **.l** to indicate byte, word, or long as in the Motorola assembler, **as** uses a suffix on the normal instruction mnemonic, thereby creating a separate mnemonic to indicate which length operand was intended. For example, there are three mnemonics for the Or instruction: **orb**, **orw**, and **orl**.
- Instruction mnemonics for instructions with unusual opcodes may have additional suffixes. Thus in addition to the normal Add variations, there also exist **addqb**, **addqw**, and **addql**, for the Add Quick instruction.
- Branch instructions are always long for non-local labels on the NeXT M68000 machines. This allows the link editor to do procedure ordering (see the description of the **-sectorder** option in the **ld**(1) man page, and the ªLink Optimizationº paper in the directory **/NextLibrary/Documentation/NextDev/Concepts/Performance**).

# Registers

Certain instructions accept a variety of special registers. The available registers are listed in this section.

**Note:** The assembler determines that an identifier is the name of a register based solely on its name, and not the context in which the identifier is used. Therefore, *all* register names are reserved, including those less commonly known register names for the control registers. In order to avoid problems, be careful not to pick a label name that's also the name of a register.

## M68000 CPU Registers

| | |
|---|---|
| **a0-a7** | address registers (**a7** is reserved for the stack pointer) |
| **sp** | stack pointer (**a7**) |

| | |
|---|---|
| **d0-d7** | data registers |
| **pc** | program counter |
| **zpc** | hack for program space, but 0 addressing (GAS) |
| **cc** or **ccr** | condition code register |
| **sr** | status register |
| **fp0-fp7** | floating point registers |
| **fpi** or **fpir** | floating point instruction register |
| **fpc** or **fpcr** | floating point control register |
| **fps** or **fpsr** | floating point status register |

## M68000 Control Registers

| | |
|---|---|
| **sfc** | source function control code register |
| **dfc** | destination function code register |
| **usp** | user stack pointer |
| **vbr** | vector base register |
| **cacr** | cache control register |
| **msp** | master stack pointer |
| **isp** | interrupt stack pointer |

## 68040-Specific Control Registers

| | |
|---|---|
| **tc** | MMU translation control register |
| **itt0** | instruction transparent translation register 0 |
| **itt1** | instruction transparent translation register 1 |
| **dtt0** | data transparent translation register 0 |
| **dtt1** | data transparent translation register 1 |
| **mmusr** or **psr** | MMU status register |
| **urp** | user root pointer |
| **srp** | supervisor root pointer |
| **ic** | instruction cache |
| **dc** | data cache |
| **bc** | both instruction and data cache |

## 68030-Specific Control Registers

| | |
|---|---|
| **srp** | supervisor root pointer |
| **tc** | translation control register |
| **crp** | CPU root pointer |
| **caar** | cache address register |
| **mmusr** or **psr** | MMU status register |

**tt0** transparent translation register 0

**tt1** transparent translation register 1

# Operands and Addressing Modes

Table€5-1 in this section describes the addressing modes that the assembler recognizes. The bulleted information below is intended to explain the notation used in the figure and give you the information you need in order to understand the contents of the figure.

| Mode | Notation | Example |
|---|---|---|
| Register | a*n*, d*n*, sp, pc, cc, sr, or usp | movw a3,d2 |
| Register Indirect | a*n*@ | movw a3@,d2 |
| Register List | r*i*/r*j* (that is., r*i* and r*j*)<br>r*i*-r*j* (r*i* through r*j*) | movem a0/a1/a2/a3,a6@-<br>movem d0-d7,a6@- |
| Floating-Point Register | fp*i* | fmoves fp1,a3@(24) |
| Postincrement | a*n*@+ | movw a3@+,d2 |
| Predecrement | a*n*@- | movw a3@-,d2 |
| Displacement | a*n*@(*d*) | movw a3@(24),d2 |
| Word Index | a*n*@(*d*,r*i*:w) | movw a3@(16, d2:w),d3 |
| Long Index | a*n*@(*d*,r*i*:l) | movw a3@(16, d2:l),d3 |
| Absolute Short | *xxx*:w | movw 14:w,d2 |
| Absolute Long | *xxx*:l | movw 14:l,d2 |
| PC Displacement | pc@(*d*) | movw pc@(20),d3 |
| PC Word Index | pc@(*d*,r*i*:w) | movw pc@(14, d2:w),d3 |
| PC Long Index | pc@(*d*,r*i*:l) | movw pc@(14, d2:l),d3 |
| PC-Memory Indirect Pre-Indexed | pc@(*d'*:*L*,r*i*:*L*:*s*)@(*d*:*L*) | movl pc@(2:w,d4:w:4)@(14:l),d3 |
| PC-Memory Indirect Post-Indexed | pc@(*d*:*L*)@(*d'*:*L*,r*i*:*L*:*s*) | movl pc@(d:l)@(3:w,d2:l:4),d3 |

| | | |
|---|---|---|
| Memory Indirect Pre-Indexed | a*n*@(*d'*:*L*,r*i*:*L*:*s*)@(*d*:*L*) | movl a1@(d:L,d2:1:4)@(14:w) |
| Memory Indirect Post-Indexed | a*n*@(*d*:*L*)@(*d'*:*L*,r*i*:*L*:*s*) | movl a2@(2:w)@(14:w,d4:w:2) |
| Normal | *identifier* | movw var,d3 |
| Immediate | *#xxx* | movw #27+3,d3 |

**Table 5-1: Addressing Modes**

## Reading the Table

The following notation is used in the table:

- **a***n* An address register
- **d***n* A data register
- **r***i* An address register or a data register
- **f***i* A floating-point register
- *d* A displacement, which is a constant expression. A length specifier may be appended to the displacement. Any forward or external references require the length specifier to be **:l**. All other references permit either **:l** or **:w** or nulls.
- *L* The index register's length. This may be either **l** (long), **w** (word), or null. If the only value permitted by a particular addressing mode or category is **l** or **w**, then *L* will be replaced by the appropriate value in the table notation.
- *s* A scale factor that may be used to multiply the index register's length. The scale factor may have a value of 1, 2, 4, or 8.

Colons precede items that may be optional. For example:

```
ri:L:s
```

In this example, you may not specify :*s* unless you have specified :*L*, which you may not specify unless you have specified **r***i*. The items in the list must appear in the order shown.

*xxx* refers to a constant expression.

In each addressing mode, up to four user-specified values are used to generate the final operand address:

- base register
- base displacement
- index register
- outer displacement

All four user-specified values are optional. Both base and outer displacements may be null, word, or long. When a displacement is null or when an element is suppressed, its value is taken as zero in the effective address calculation.

In the table where both *d* and *d'* are specified, *d* corresponds to an outer displacement and *d'* corresponds to a base displacement.

To suppress displacements, write them as ªº with no length qualifier. For example, the first expression shown here has a null base displacement and a null outer displacement, while the second has long displacements.:

```
a1@(0,d2)@(0)
a1@(0x0:l,d2)@(0x0:l)
```

To suppress the base register for modes that use address registers, omit the address register:

```
@(base_disp,d2)@(outer_disp)
```

For modes that use the program counter, use the pseudo register **zpc**:

```
zpc@(base_disp,d2)@(outer_disp)
```

**Note:** The assembler will never generate the addressing modes ªA Register Indirect with Indexº or ªProgram Counter Indirect with Indexº when the index register is suppressed and the base displacement is word-sized. Instead, it will use ªAddress Register with Displacementº or ªProgram Counter with Displacement.º

In the case of ªMemory Indirectº addressing, an address register (**a***n*) is used as a base register, and its value can be adjusted by an optional base displacement (*d'*). An index register (**r***i*) specifies an index operand (**r***i***:L:s**) and finally, an outer displacement (*d*) can be added to the address operand, yielding the effective address.

ªProgram Counter Memory-Indirectº mode is exactly the same. The only difference is that the program counter is used as the base register.

Normal mode assembles as absolute long.

The Motorola manuals present different mnemonics (and in fact different forms of the actual machine instructions) for instructions that use the literal effective address as data instead of using the contents of the effective address. For instance, they use the mnemonic **adda** for Add Address. The NeXT assembler doesn't make these distinctions because it can determine the type of opcode required, from the form of the operand. Thus, an instruction of the form:

```
avenue: .word 0
    ...
    addl  #avenue,a0
```

assembles to the Add Address instruction because the assembler can determine that **a0** is an address register. Similarly,

```
var: = 40000
    ...
    addl  #var,d0
```

assembles to the Add Immediate instruction because the assembler can determine that **var** is a constant. Because of this determination of operand forms, some of the mnemonics listed in the Motorola manuals are missing from the set of mnemonics that the assembler recognizes.

Certain classes of instructions accept only subsets of the addressing modes above. For example, the Add instruction doesn't accept a PC-relative address as a destination, and register lists may only be used with the **movem** and **fmovem** instructions. The assembler tries to check all these restrictions and generates the Illegal Operand error code for instructions that don't satisfy the address mode restrictions.

Currently the assembler doesn't accept immediate operands for floating-point instructions using the packed decimal format (this is because the assembler can't determine how to convert decimal floating-point format into packed decimal format).

The following tables show the differences between the NeXT and Motorola addressing mode syntax, and compares these to the equivalent format that would be used in the C programming language.

| C Equivalent | NeXT Syntax | Motorola Syntax | Mode | Register |
|---|---|---|---|---|
| Dn | Dn | Dn | 000 | reg. number:Dn |
| An | An | An | 001 | reg. number:An |
| *(An) | An@ | (An) | 010 | reg. number:An |
| *(An++) | An@+ | (An)+ | 011 | reg. number:An |
| *(-€-An) | An@- | -(An) | 100 | reg. number:An |
| *(An+d16) | An@(d16) | (d16,An) | 101 | reg. number:An |
| *(An+d8+Xn) | An@(d8,Xn) | (d8,An,Xn) | 110 | reg. number:An |
| *(An+bd+Xn) | An@(bd,Xn) | (bd,An,Xn) | 110 | reg. number:An |
| *(*(An+bd)+od+Xn) | An@(bd)@(od,Xn) | ([bd,An],Xn,od) | 110 | reg. number:An |
| *(*(An+bd+Xn)+od) | An@(bd,Xn)@(od) | ([bd,An,Xn],od) | 110 | reg. number:An |
| *(xxx) | xxx:w | (xxx).W | 111 | 000 |
| *(xxx) | xxx:l | (xxx).L | 111 | 001 |
| data | #data | #<data> | 111 | 100 |
| Đ | Đ | Đ | Đ | Đ |
| Đ | Đ | Đ | Đ | Đ |
| *(pc+d16) | pc@(d16) | (d16,pc) | 111 | 010 |
| *(pc+d8+Xn) | pc@(d8,Xn) | (d8,pc,Xn) | 111 | 011 |
| *(pc+bd+Xn) | pc@(bd,Xn) | (bd,pc,Xn) | 111 | 011 |
| *(*(pc+bd)+od+Xn) | pc@(bd)@(od,Xn) | ([bd,pc],Xn,od) | 111 | 011 |
| *(*(pc+bd+Xn)+od) | pc@(bd,Xn)@(od) | ([bd,pc,Xn],od) | 111 | 011 |

**Table 5-2: Addressing Mode Syntactic Comparison**

# Addressing Categories

The 68030 and 68040 processors group the effective address modes into categories, derived from the manner in which they are used to address operands (note the distinction between address *modes* and address *categories*).

There are 18 addressing modes in the 68030 and 68040, and each addressing mode belongs to one or more of

four addressing categories. The addressing categories are described here, followed by a table summarizing the grouping of the addressing modes into categories. Note that register lists can be used only by the **movem** and **fmovem** instructions.

| **Category** | **Meaning** |
|---|---|
| Data | means that the effective address mode is used to refer to data operands such as a **d** register or immediate data |
| Memory | means that the effective address mode can refer to memory operands. Examples include all the a-register indirect addressing modes and all the absolute addressing modes. |
| Alterable | means that the effective address mode refers to operands that are writeable (alterable). This category takes in every addressing mode except the PC-relative addressing modes and the immediate address mode. |
| Control | means that the effective address mode refers to memory operands with no explicit size specification. |

Some addressing categories can be intersected to make more restrictive ones. For example, the Motorola manual mentions the ªData Alterable Addressing Modeº to mean that the particular instruction can only use those modes which provide data addressing and are alterable as well.

| **Addressing Mode** | **Assembler Syntax** | **Data** | **Memory** | **Control** | **Alterable** |
|---|---|---|---|---|---|
| Register Direct | a*n*, d*n*, sp, pc, cc, sr, usp | √ | | | √ |
| A-Register Indirect | an@ | √ | √ | √ | √ |
| A-Register Indirect with Displacement | an@(*d*:*L*) | √ | √ | √ | √ |
| A-Register Indirect with Word Index | an@(*d*:*L*,r*i*:w:*s*) | √ | √ | √ | √ |
| A-Register Indirect with Long Index | an@(*d*:*L*,r*i*:l:*s*) | √ | √ | √ | √ |
| A-Register Indirect with Post Increment | an@+ | √ | √ | | √ |

| | | | | | |
|---|---|---|---|---|---|
| A-Register Indirect with Pre Decrement | an@- | √ | √ | | √ |
| A-Register Indirect with Displacement | an@(*d*) | √ | √ | √ | √ |
| A-Register Indirect with Word Index | an@(*d*,r*i*:w) | √ | √ | √ | √ |
| A-Register Indirect with Long Index | an@(*d*,*r*i:l) | √ | √ | √ | √ |
| Memory-Indirect Post-Indexed | an@(*d*:*L*)@(*d'*:*L*,r*i*:*L*:*s*) | √ | √ | √ | √ |
| Memory-Indirect Pre-Indexed | an@(*d'*:*L*,ri:*L*:*s*)@(*d*:*L*) | √ | √ | √ | √ |
| Absolute Short | *xxx*:w | √ | √ | √ | √ |
| Absolute Long | *xxx*:l√ | √ | √ | √ | |
| PC-Relative | pc@(*d*) | √ | √ | √ | |
| PC-Indirect with Displacement | pc@(*d*:*L*) | √ | √ | √ | |
| PC-Relative with Word Index | pc@(*d*,r*i*:w) | √ | √ | √ | |
| PC-Indirect with Word Index | pc@(*d*:*L*,r*i*:w:*s*) | √ | √ | √ | |
| PC-Relative with Long Index | pc@(*d*,r*i*:l) | √ | √ | √ | |
| PC-Indirect with Long Index | pc@(*d*:*L*,r*i*:l:*s*) | √ | √ | √ | |
| PC-Memory Indirect Post-Indexed | pc@(*d*:*L*)@(*d'*:*L*,r*i*:*L*:*s*) | √ | √ | √ | √ |
| PC-Memory Indirect Pre-Indexed | pc@(*d'*:*L*,r*i*:*L*:*s*)@(*d*:*L*) | √ | √ | √ | √ |
| Immediate Data | #*nnn* | √ | √ | | |

**Table 5-3: Addressing Categories**

# M68000 Assembler Instructions

Note the following points about the information contained in this section:

- **Name** is the name that appears in the upper left and right corner of a page in the Motorola manuals.
- **Operation Name** is the name that appears at the upper center of a page in the Motorola manuals.
- An asterisk (*) following the **Operand** description indicates an instruction whose syntax on NeXT computers is different from the standard Motorola syntax.

## Integer Instructions

| Name | Operator | Operand | Operation Name |
|---|---|---|---|
| abcd | abcd | Dn,Dn | Add Decimal with Extend |
| | abcd | An@-,An@- | |
| add | add{b,w,l} | <ea>,Dn | Add |
| | add{b,w,l} | Dn,<ea> | |
| adda | adda{w,l} | <ea>,An | Add Address |
| | add{w,l} | <ea>,An | |
| addi | addi{b,w,l} | #data,<ea> | Add Immediate |
| | add{b,w,l} | #data,<ea> | |

| | | | |
|---|---|---|---|
| addq | addq{b,w,l} | #data,<ea> | Add Quick |
| addx | addx{b,w,l} | Dn,Dn | Add Extended |
| | addx{b,w,l} | An@-,An@- | |
| and | and{b,w,l} | <ea>,Dn | AND Logical |
| | and{b,w,l} | Dn,<ea> | |
| andi | andi{b,w,l} | #data,<ea> | AND Immediate |
| | and{b,w,l} | #data,<ea> | |
| andi to ccr | andi{b} | #data,ccr | AND Immediate |
| | and{b} | #data,ccr | to Condition Codes |
| andi to sr | andi{w} | #data,sr | AND Immediate |
| | and{w} | #data,sr | to Status Register |
| asl | asl{b,w,l} | #data,Dn | Arithmetic Shift Left |
| | asl{b,w,l} | Dn,Dn | |
| | asl{w} | <ea> | |
| asr | asr{b,w,l} | #data,Dn | Arithmetic Shift Right |
| | asr{b,w,l} | Dn,Dn | |
| | asr{w} | <ea> | |
| b*cc* (and b*ccs*) | | | Branch Conditionally |
| | bcc | <ea> | carry clear |
| | bcs | <ea> | carry set |
| | beq | <ea> | equal |
| | bge | <ea> | greater or equal |

| | | | |
|---|---|---|---|
| | bgt | <ea> | greater than |
| | bhi | <ea> | high |
| | ble | <ea> | less or equal |
| | bls | <ea> | low or same |
| | blt | <ea> | less than |
| | bmi | <ea> | minus |
| | bne | <ea> | not equal |
| | bpl | <ea> | plus |
| | bvc | <ea> | overflow clear |
| | bvs | <ea> | overflow set |
| bchg | bchg | Dn,<ea> | Test a Bit and Change |
| | bchg | #data,<ea> | |
| bclr | bclr | Dn,<ea> | Test a Bit and Clear |
| | bclr | #data,<ea> | |
| bfchg | bfchg | <ea>{#data:#data} | Test Bit Field and Change |
| | bfchg | <ea>{#data:Dn} | |
| | bfchg | <ea>{Dn:#data} | |
| | bfchg | <ea>{Dn:Dn} | |
| bfclr | bfclr | <ea>{#data:#data} | Test Bit Field and Clear |
| | bfclr | <ea>{#data:Dn} | |
| | bfclr | <ea>{Dn:#data} | |
| | bfclr | <ea>{Dn:Dn} | |
| bfexts | bfexts | <ea>{#data:#data}Dn | Extract Bit Field Signed |
| | bfexts | <ea>{#data:Dn}Dn | |
| | bfexts | <ea>{Dn:#data}Dn | |

| | | | |
|---|---|---|---|
| | bfexts | <ea>{Dn:Dn}Dn | |
| bfextu | bfextu | <ea>{#data:#data}Dn | Extract Bit Field Unsigned |
| | bfextu | <ea>{#data:Dn}Dn | |
| | bfextu | <ea>{Dn:#data}Dn | |
| | bfextu | <ea>{Dn:Dn}Dn | |
| bfffo | bfffo | <ea>{#data:#data}Dn | Find First One in Bit Field |
| | bfffo | <ea>{#data:Dn}Dn | |
| | bfffo | <ea>{Dn:#data}Dn | |
| | bfffo | <ea>{Dn:Dn}Dn | |
| bfins | bfins | Dn,<ea>{#data:#data} | Insert Bit Field |
| | bfins | Dn,<ea>{#data:Dn} | |
| | bfins | Dn,<ea>{Dn:#data} | |
| | bfins | Dn,<ea>{Dn:Dn} | |
| bfset | bfset | <ea>{#data:#data} | Test Bit Field and Set |
| | bfset | <ea>{#data:Dn}Dn | |
| | bfset | <ea>{Dn:#data}Dn | |
| | bfset | <ea>{Dn:Dn}Dn | |
| bftst | bftst | <ea>{#data:#data} | Test Bit Field |
| | bftst | <ea>{#data:Dn}Dn | |
| | bftst | <ea>{Dn:#data}Dn | |
| | bftst | <ea>{Dn:Dn}Dn | |
| bkpt | bkpt | #data | Breakpoint |
| bra | bra | <ea> | Branch Always |

| | | | |
|---|---|---|---|
| | bras | <ea> | *(short form)* |
| bset | bset | Dn,<ea> | Test a Bit and Set |
| | bset | #data,<ea> | |
| bsr | bsr | <ea> | Branch to Subroutine |
| | bsrs | <ea> | *(short form)* |
| btst | btst | Dn,<ea> | Test a Bit |
| | btst | #data,<ea> | |
| callm | callm | #data,<ea> | CALL Module |
| cas | cas{b,w,l} | Dn,Dn,<ea> | Compare and Swap with |
| | cas2{w,l} | Dn,Dn,Dn,Dn,Rn,Rn * | Operand |
| chk | chk{w,l} | <ea>,Dn | Check Register Against Bounds |
| | chk2{b,w,l} | <ea>,Rn | |
| clr | clr{b,w,l} | <ea> | Clear an Operand |
| cmp | cmp{b,w,l} | <ea>,Dn | Compare |
| cmpa | cmpa{w,l} | <ea>,An | Compare Address |
| | cmp{w,l} | <ea>,An | |
| cmpi | cmpi{b,w,l} | #data,<ea> | Compare Immediate |
| | cmp{b,w,l} | #data,<ea> | |
| cmpm | cmpm{b,w,l} | An@+,An@+ | Compare Memory |

| | | | |
|---|---|---|---|
| cmp2 | cmp2{b,w,l} | <ea>,Rn | Compare Register Against Bounds |
| db*cc* | | | Test Cond., Decrement, Branch |
| | dbcc | Dn,<ea> | carry clear |
| | dbcs | Dn,<ea> | carry set |
| | dbeq | Dn,<ea> | equal |
| | dbf | Dn,<ea> | never equal |
| | dbge | Dn,<ea> | greater or equal |
| | dbgt | Dn,<ea> | greater than |
| | dbhi | Dn,<ea> | high |
| | dble | Dn,<ea> | less or equal |
| | dbls | Dn,<ea> | low or same |
| | dblt | Dn,<ea> | less than |
| | dbmi | Dn,<ea> | minus |
| | dbne | Dn,<ea> | not equal |
| | dbpl | Dn,<ea> | plus |
| | dbra | Dn,<ea> | always true |
| | dbt | Dn,<ea> | always true |
| | dbvc | Dn,<ea> | overflow clear |
| | dbvs | Dn,<ea> | overflow set |
| divs | divs{w,l} | <ea>,Dn | Signed Divide |
| | divs | <ea>,Dn | *(same as divsw)* |
| | divsl{l} | <ea>,Dn,Dn * | |
| divu | divu{w,l} | <ea>,Dn | Unsigned Divide |
| | divu | <ea>,Dn | *(same as divsw)* |
| | divul{l} | <ea>,Dn,Dn * | |

| | | | |
|---|---|---|---|
| eor | eor{b,w,l} | Dn,<ea> | Exclusive-OR Logical |
| eori | eori{b,w,l} | #data,<ea> | Exclusive-OR Immediate |
| | eor{b,w,l} | #data,<ea> | |
| eori to ccr | eori{b} | #data,ccr | Exclusive-OR Immediate |
| | eor{b} | #data,ccr | to Condition Code |
| eori to sr | eori{w} | #data,sr | Exclusive-OR Immediate |
| | eor{w} | #data,sr | to Status Register |
| exg | exg | Dn,Dn | Exchange Registers |
| | exg | An,An | |
| | exg | Dn,An | |
| | exg | An,Dn | |
| ext | ext{w,l} | Dn | Sign Extend |
| | extb{l} | Dn | |
| | extb.l | Dn | *(same as extbl)* |
| illegal | illegal | | Take Illegal Instruction Trap |
| j*cc* (and j*cc*s) | | | Jump or Branch Conditionally |
| | jhi | <ea> | high |
| | jls | <ea> | low or same |
| | jcc | <ea> | carry clear |
| | jcs | <ea> | carry set |
| | jne | <ea> | not equal |
| | jeq | <ea> | equal |

| | | | |
|---|---|---|---|
| | jvc | <ea> | overflow clear |
| | jvs | <ea> | overflow set |
| | jpl | <ea> | plus |
| | jmi | <ea> | minus |
| | jge | <ea> | greater or equal |
| | jlt | <ea> | less than |
| | jgt | <ea> | greater than |
| | jle | <ea> | less or equal |
| jmp | jmp | <ea> | Jump |
| | jra | <ea> | Jump or Branch |
| | jras | <ea> | Jump or Branch *(short form)* |
| jsr | jsr | <ea> | Jump to Subroutine |
| | jbsr | <ea> | Jump or Branch to Subroutine |
| | jbsrs | <ea> | *(short form)* |
| lea | lea | <ea>,An | Load Effective Address |
| link | link{w,l} | An,#data | Link and Allocate |
| | link | An,#data | *(same as linkw)* |
| lsl | lsl{b,w,l} | #data,Dn | Logical Shift Left |
| | lsl{b,w,l} | Dn,Dn | |
| | lsl{w} | <ea> | |
| lsr | lsr{b,w,l} | #data,Dn | Logical Shift Right |
| | lsr{b,w,l} | Dn,Dn | |
| | lsr{w} | <ea> | |

| | | | |
|---|---|---|---|
| move | move{b,w,l} | <ea>,<ea> | Move Data from Source |
| | mov{b,w,l} | <ea>,<ea> | to Destination |
| move | move{w} | sr,<ea> | Move from the |
| from sr | mov{w} | sr,<ea> | Status Register |
| move | move{w} | ccr,<ea> | Move from the |
| from ccr | mov{w} | ccr,<ea> | Condition Code Register |
| move | move{w} | <ea>,ccr | Move to the |
| to ccr | mov{w} | <ea>,ccr | Condition Code Register |
| move | move{w} | <ea>,sr | Move to the |
| to sr | mov{w} | <ea>,sr | Status Register |
| move usp | move{l} | An,usp | Move User Stack Pointer |
| | move{l} | usp,An | |
| movea | movea{w,l} | <ea>,An | Move Address |
| | move{w,l} | <ea>,An | *(alternate form)* |
| | mova{w,l} | <ea>,An | *( " " )* |
| | mov{w,l} | <ea>,An | *( " " )* |
| movec | movec | Rn,Rc | Move Control Register |
| | movec | Rc,Rn | |
| | movec | Rn,#data | |
| | movec | #data,Rn | |
| | movc | Rn,Rc | *(alternate form)* |
| | movc | Rc,Rn | *( " " )* |
| | movc | Rn,#data | *( " " )* |

| | | | |
|---|---|---|---|
| | movc | #data,Rn | *(" ")* |
| movem | movem{w,l} | reglist,<ea> | Move Multiple Registers |
| | movem{w,l} | <ea>,reglist | |
| | movm{w,l} | reglist,<ea> | *(alternate form)* |
| | movm{w,l} | <ea>,reglist | *(" ")* |
| | movem{w,l} | #data,<ea> | *(reglist can be immediate* |
| | movem{w,l} | <ea>,#data | *data, as shown here)* |
| movep | movep{w,l} | (d,An),Dn | Move Peripheral Data |
| | movep{w,l} | Dn,(d,An) | |
| | movp{w,l} | (d,An),Dn | *(alternate form)* |
| | movp{w,l} | Dn,(d,An) | *(" ")* |
| moveq | moveq | #data,Dn | Move Quick |
| | movq | #data,Dn | *(alternate form)* |
| | movel | #data,Dn | *(" ")* |
| | movl | #data,Dn | *(" ")* |
| | moveql | #data,Dn | *(" ")* |
| | movql | #data,Dn | *(" ")* |
| moves | moves{b,w,l} | <ea>,Rn | Move Address Space |
| | moves{b,w,l} | Rn,<ea> | |
| | movs{b,w,l} | <ea>,Rn | *(alternate form)* |
| | movs{b,w,l} | Rn,<ea> | *(" ")* |
| muls | muls{w,l} | <ea>,Dn | Signed Multiply |
| | muls{l} | <ea>,Dn,Dn * | |
| | muls | <ea>,Dn | *(same as mulsw)* |

| | | | |
|---|---|---|---|
| mulu | mulu{w,l} | <ea>,Dn | Unsigned Multiply |
| | mulu{l} | <ea>,Dn,Dn * | |
| | mulu | <ea>,Dn | *(same as muluw)* |
| nbcd | nbcd | <ea> | Negate Decimal with Extend |
| neg | neg{b,w,l} | <ea> | Negate |
| negx | negx{b,w,l} | <ea> | Negate with Extend |
| nop | nop | | No Operation |
| not | not{b,w,l} | <ea> | Logical Complement |
| or | or{b,w,l} | <ea>,Dn | Inclusive-OR Logical |
| | or{b,w,l} | Dn,<ea> | |
| ori | ori{b,w,l} | #data,<ea> | Inclusive-OR Immediate |
| | or{b,w,l} | #data,<ea> | |
| ori to ccr | ori{b} | #data,ccr | Inclusive-OR Immediate |
| | or{b} | #data,ccr | to Condition Codes |
| ori to ssr | ori{w} | #data,sr | Inclusive-OR Immediate |
| | or{w} | #data,sr | to the Status Register |
| pack | pack | Dn,Dn,#data | Pack |
| | pack | An@-,An@-,#data | |
| pea | pea | <ea> | Push Effective Address |

| | | | |
|---|---|---|---|
| reset | reset | | Reset External Devices |
| rol | rol{b,w,l}<br>rol{b,w,l}<br>rol{w} | #data,Dn<br>Dn,Dn<br><ea> | Rotate Left without Extend |
| ror | ror{b,w,l}<br>ror{b,w,l}<br>ror{w} | #data,Dn<br>Dn,Dn<br><ea> | Rotate Right without Extend |
| roxl | roxl{b,w,l}<br>roxl{b,w,l}<br>roxl{w} | #data,Dn<br>Dn,Dn<br><ea> | Rotate Left with Extend |
| roxr | roxr{b,w,l}<br>roxr{b,w,l}<br>roxr{w} | #data,Dn<br>Dn,Dn<br><ea> | Rotate Right with Extend |
| rtd | rtd | #data | Return and Deallocate |
| rte | rte | | Return from Exception |
| rtm | rtm | Rn | Return from Module |
| rtr | rtr | | Return and Restore Condition Codes |
| rts | rts | | Return from Subroutine |

| | | | |
|---|---|---|---|
| sbcd | sbcd | Dn,Dn | Subtract Decimal with Extend |
| | sbcd | An@-,An@- | |
| s*cc* | | | Set According to Condition |
| | scc | <ea> | carry clear |
| | scs | <ea> | carry set |
| | seq | <ea> | equal |
| | sf | <ea> | never equal |
| | sge | <ea> | greater or equal |
| | sgt | <ea> | greater than |
| | shi | <ea> | high |
| | sle | <ea> | less or equal |
| | sls | <ea> | low or same |
| | slt | <ea> | less than |
| | smi | <ea> | minus |
| | sne | <ea> | not equal |
| | spl | <ea> | plus |
| | st | <ea> | always true |
| | svc | <ea> | overflow clear |
| | svs | <ea> | overflow set |
| stop | stop | #data | Load Status Register and Stop |
| sub | sub{b,w,l} | <ea>,Dn | Subtract |
| | sub{b,w,l} | Dn,<ea> | |
| suba | suba{w,l} | <ea>,An | Subtract Address |
| | sub{w,l} | <ea>,An | |
| subi | subi{b,w,l} | #data,<ea> | Subtract Immediate |

| | | | |
|---|---|---|---|
| | sub{b,w,l} | #data,<ea> | |
| subq | subq{b,w,l} | #data,<ea> | Subtract Quick |
| | sub{b,w,l} | #data,<ea> | |
| subx | subx{b,w,l} | Dn,Dn | Subtract with Extend |
| | subx{b,w,l} | An@-,An@- | |
| swap | swap | Dn | Swap Register Halves |
| tas | tas | <ea> | Test and Set an Operand |
| trap | trap | #data | Trap |
| trap*cc* | | | Trap on Condition (Unsized) |
| | trapcc | | carry clear |
| | trapcs | | carry set |
| | trapeq | | equal |
| | trapf | | never equal |
| | trapge | | greater or equal |
| | trapgt | | greater than |
| | traphi | | high |
| | traple | | less or equal |
| | trapls | | low or same |
| | traplt | | less than |
| | trapmi | | minus |
| | trapne | | not equal |
| | trappl | | plus |
| | trapt | | always true |
| | trapvc | | overflow clear |

| | | | |
|---|---|---|---|
| | trapvs | | overflow set |
| trap*cc*. | | | Trap on Condition (Sized) |
| | trapcc.{w,l} | #data | carry clear |
| | trapcs.{w,l} | #data | carry set |
| | trapeq.{w,l} | #data | equal |
| | trapf.{w,l} | #data | never equal |
| | trapge.{w,l} | #data | greater or equal |
| | trapgt.{w,l} | #data | greater than |
| | traphi.{w,l} | #data | high |
| | traple.{w,l} | #data | less or equal |
| | trapls.{w,l} | #data | low or same |
| | traplt.{w,l} | #data | less than |
| | trapmi.{w,l} | #data | minus |
| | trapne.{w,l} | #data | not equal |
| | trappl.{w,l} | #data | plus |
| | trapt.{w,l} | #data | always true |
| | trapvc.{w,l} | #data | overflow clear |
| | trapvs.{w,l} | #data | overflow set |
| trapv | trapv | | Trap on Overflow |
| tst | tst{b,w,l} | <ea> | Test an Operand |
| unlk | unlk | An | Unlink |
| unpk | unpk | Dn,Dn,#data | Unpack BCD |
| | unpk | An@-,An@-,#data | |

# Floating-Point Instructions

In the following list of instructions, *condition_code* can be replaced by any of the following:

| Code | Meaning |
|---|---|
| eq | Equal |
| ne | Not Equal |
| ueq | Unordered or Equal |
| ge | Greater Than or Equal |
| nge | Not (Greater Than or Equal) |
| oge | Ordered Greater Than or Equal |
| uge | Unordered or Greater or Equal |
| gl | Greater or Less Than |
| ngl | Not (Greater or Less Than) |
| ogl | Ordered Greater or Less Than |
| gle | Greater, Less or Equal |
| ngle | Not (Greater, Less or Equal) |
| gt | Greater Than |
| ngt | Not Greater Than |
| ogt | Ordered Greater Than |
| ugt | Unordered or Greater Than |
| le | Less Than or Equal |
| nle | Not (Less Than or Equal) |
| ole | Ordered Less Than or Equal |
| ule | Unordered Less Than or Equal |
| lt | Less Than |
| nlt | Not Less Than |
| olt | Ordered Less Than |
| ult | Unordered or Less Than |
| t | True |

| | |
|---|---|
| f | False |
| st | Signaling True |
| sf | Signaling False |
| seq | Signaling Equal |
| sne | Signaling Not Equal |
| or | Ordered |
| un | Unordered |

| Name | Operator | Operand | Operation Name |
|---|---|---|---|
| fabs | fabs{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Absolute Value |
| | fabs{x} | FPn,FPn | |
| | fabs{x} | FPn | |
| facos | facos{b,w,l,s,d,x,p} | <ea>,FPn | Arc Cosine |
| | facos{x} | FPn,FPn | |
| | facos{x} | FPn | |
| fadd | fadd{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Add |
| | fadd{x} | FPn,FPn | |
| | fadd{x} | FPn | |
| fasin | fasin{b,w,l,s,d,x,p} | <ea>,FPn | Arc Sine |
| | fasin{x} | FPn,FPn | |
| | fasin{x} | FPn | |
| fatan | fatan{b,w,l,s,d,x,p} | <ea>,FPn | Arc Tangent |
| | fatan{x} | FPn,FPn | |
| | fatan{x} | FPn | |
| fatanh | fatanh{b,w,l,s,d,x,p} | <ea>,FPn | Hyperbolic Arc Tangent |

| | | | |
|---|---|---|---|
| | fatanh{x} | FPn,FPn | |
| | fatanh{x} | FPn | |
| fb*cc* | | | Floating-Point |
| | fb{*condition_code*} | <ea> | Branch Conditionally |
| fj*cc* | | | Floating-Point |
| | fj{*condition_code*} | <ea> | Branch or Jump Conditionally |
| fcmp | fcmp{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Compare |
| | fcmp{x} | FPn,FPn | |
| | fcmp{x} | FPn | |
| fcos | fcos{b,w,l,s,d,x,p} | <ea>,FPn | Cosine |
| | fcos{x} | FPn,FPn | |
| | fcos{x} | FPn | |
| fcosh | fcosh{b,w,l,s,d,x,p} | <ea>,FPn | Hyperbolic Cosine |
| | fcosh{x} | FPn,FPn | |
| | fcosh{x} | FPn | |
| fdb*cc* | | | Floating-Point Test Condition, |
| | fdb{*condition_code*} | Dn,<ea> | Decrement, and Branch |
| fdiv | fdiv{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Divide |
| | fdiv{x} | FPn,FPn | |
| | fdiv{x} | FPn | |
| fetox | fetox{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point $e^x$ |

| | | | |
|---|---|---|---|
| | fetox{x} | FPn,FPn | |
| | fetox{x} | FPn | |
| fetoxm1 | fetoxm1{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point $ex_{-1}$ |
| | fetoxm1{x} | FPn,FPn | |
| | fetoxm1{x} | FPn | |
| fgetexp | fgetexp{b,w,l,s,d,x,p} | <ea>,FPn | Get Exponent |
| | fgetexp{x} | FPn,FPn | |
| | fgetexp{x} | FPn | |
| fgetman | fgetman{b,w,l,s,d,x,p} | <ea>,FPn | Get Mantissa |
| | fgetman{x} | FPn,FPn | |
| | fgetman{x} | FPn | |
| fint | fint{b,w,l,s,d,x,p} | <ea>,FPn | Integer Part |
| | fint{x} | FPn,FPn | |
| | fint{x} | FPn | |
| fintrz | fintrz{b,w,l,s,d,x,p} | <ea>,FPn | Integer Part, Round-to-Zero |
| | fintrz{x} | FPn,FPn | |
| | fintrz{x} | FPn | |
| flog10 | flog10{b,w,l,s,d,x,p} | <ea>,FPn | $Log_{10}$ |
| | flog10{x} | FPn,FPn | |
| | flog10{x} | FPn | |
| flog2 | flog2{b,w,l,s,d,x,p} | <ea>,FPn | Log2 |

| | | | |
|---|---|---|---|
| | flog2{x} | FPn,FPn | |
| | flog2{x} | FPn | |
| flogn | flogn{b,w,l,s,d,x,p} | <ea>,FPn | Loge |
| | flogn{x} | FPn,FPn | |
| | flogn{x} | FPn | |
| flognp1 | flognp1{b,w,l,s,d,x,p} | <ea>,FPn | $Loge^{(x+1)}$ |
| | flognp1{x} | FPn,FPn | |
| | flognp1{x} | FPn | |
| fmod | fmod{b,w,l,s,d,x,p} | <ea>,FPn | Modulo Remainder |
| | fmod{x} | FPn,FPn | |
| | fmod{x} | FPn | |
| fmove | fmove{b,w,l,s,d,x,p} | <ea>,FPn | Move Floating-Point |
| | fmove{b,w,l,s,d,x} | FPn,<ea> | Data Register |
| | fmove{p} | FPn,<ea>{Dn} | |
| | fmove{p} | FPn,<ea>,{#k} | |
| | fmove{x} | FPn,FPn | |
| | fmove{x} | FPn | |
| fmove fcr | fmove{l} | FPcr,<ea> | Move Floating-Point |
| | fmove{l} | <ea>,FPcr | System Control Register |
| fmovecr | fmovecr{x} | #data,FPn | Move Constant ROM |
| | fmovecr | #data,FPn | *(same as above)* |

| | | | |
|---|---|---|---|
| fmovem | fmovem{l,x} | reglist,<ea> | Move Multiple Floating-Point Data Registers *(reglist can be immediate data, as shown here)* |
| | fmovem{l,x} | <ea>,reglist | |
| | fmovem{l,x} | #data,<ea> | |
| | fmovem{l,x} | <ea>,#data | |
| fpmovem ccr | fmovem{l} | FPcrlist,<ea> | Move Multiple Floating-Point Control Registers |
| | fmovem{l} | <ea>,FPcrlist | |
| fmul | fmul{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Multiply |
| | fmul{x} | FPn,FPn | |
| | fmul{x} | FPn | |
| fneg | fneg{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Negate |
| | fneg{x} | FPn,FPn | |
| | fneg{x} | FPn | |
| fnop | fnop | | No Operation |
| frem | frem{b,w,l,s,d,x,p} | <ea>,FPn | IEEE Remainder |
| | frem{x} | FPn,FPn | |
| | frem{x} | FPn | |
| frestore | frestore | <ea> | Restore Internal Floating-Point State |
| fsave | fsave | <ea> | Save Internal Floating-Point State |
| fscale | fscale{b,w,l,s,d,x,p} | <ea>,FPn | Scale Exponent |
| | fscale{x} | FPn,FPn | |

| | | | |
|---|---|---|---|
| | fscale{x} | FPn | |
| fs*cc* | fs{*condition_code*} | <ea> | Set According to Floating-Point Condition |
| fsgldiv | fsgldiv{b,w,l,s,d,x,p}<br>fsgldiv{x}<br>fsgldiv{x} | <ea>,FPn<br>FPn,FPn<br>FPn | Single Precision Divide |
| fsglmul | fsglmul{b,w,l,s,d,x,p}<br>fsglmul{x}<br>fsglmul{x} | <ea>,FPn<br>FPn,FPn<br>FPn | Single Precision Multiply |
| fsin | fsin{b,w,l,s,d,x,p}<br>fsin{x}<br>fsin{x} | <ea>,FPn<br>FPn,FPn<br>FPn | Sine |
| fsincos | fsincos{b,w,l,s,d,x,p}<br>fsincos{x} | <ea>,FPn,FPn *<br>FPn,FPn,FPn * | Simultaneous Sine and Cosine |
| fsinh | fsinh{b,w,l,s,d,x,p}<br>fsinh{x}<br>fsinh{x} | <ea>,FPn<br>FPn,FPn<br>FPn | Hyperbolic Sine |
| fsqrt | fsqrt{b,w,l,s,d,x,p}<br>fsqrt{x}<br>fsqrt{x} | <ea>,FPn<br>FPn,FPn<br>FPn | Floating-Point Square Root |
| fsub | fsub{b,w,l,s,d,x,p}<br>fsub{x} | <ea>,FPn<br>FPn,FPn | Floating-Point Subtract |

| | | | |
|---|---|---|---|
| | fsub{x} | FPn | |
| ftan | ftan{b,w,l,s,d,x,p}<br>ftan{x}<br>ftan{x} | <ea>,FPn<br>FPn,FPn<br>FPn | Tangent |
| ftanh | ftanh{b,w,l,s,d,x,p}<br>ftanh{x}<br>ftanh{x} | <ea>,FPn<br>FPn,FPn<br>FPn | Hyperbolic Tangent |
| ftentox | ftentox{b,w,l,s,d,x,p}<br>ftentox{x}<br>ftentox{x} | <ea>,FPn<br>FPn,FPn<br>FPn | $10^x$ |
| ftrap*cc* | | | Trap on Floating-Point |
| | ftrap{*condition_code*} | | Condition |
| | ftrap{*condition_code*}w | #data | |
| | ftrap{*condition_code*}l | #data | |
| ftst | ftst{b,w,l,s,d,x,p}<br>ftst{x} | <ea><br>FPn | Test Floating-Point Operand |
| ftwotox | ftwotox{b,w,l,s,d,x,p}<br>ftwotox{x}<br>ftwotox{x} | <ea>,FPn<br>FPn,FPn<br>FPn | 2x |

# 68030-Specific Instructions

This section lists several miscellaneous instructions specific to the 68030 processor.

| Name | Operator | Operand | Operation Name |
|---|---|---|---|
| pflush | pflusha030 | | Flush Entry in the ATC |
| | pflush | #data,#data | |
| | pflush | Dn,#data | |
| | pflush | sfc,#data | |
| | pflush | dfc,#data | |
| | pflush | #data,#data,<ea> | |
| | pflush | Dn,#data,<ea> | |
| | pflush | sfc,#data,<ea> | |
| | pflush | dfc,#data,<ea> | |
| pload | ploadr | #data,<ea> | Load an Entry into the ATC |
| | ploadr | Dn,<ea> | |
| | ploadr | sfc,<ea> | |
| | ploadr | dfc,<ea> | |
| | ploadw | #data,<ea> | |
| | ploadw | Dn,<ea> | |
| | ploadw | sfc,<ea> | |

| | | | |
|---|---|---|---|
| | ploadw | dfc,<ea> | |
| pmove | pmove | MRn,<ea> | Move to/from MMU registers |
| | pmove | <ea>,MRn | |
| | pmovefd | <ea>,MRn | |
| ptest | ptestr | #data,<ea>,#data | Test a Logical Address |
| | ptestr | Dn,<ea>,#data | |
| | ptestr | sfc,<ea>,#data | |
| | ptestr | dfc,<ea>,#data | |
| | ptestr | #data,<ea>,#data,An | |
| | ptestr | Dn,<ea>,#data,An | |
| | ptestr | sfc,<ea>,#data,An | |
| | ptestr | dfc,<ea>,#data,An | |
| | ptestw | #data,<ea>,#data | |
| | ptestw | Dn,<ea>,#data | |
| | ptestw | sfc,<ea>,#data | |
| | ptestw | dfc,<ea>,#data | |
| | ptestw | #data,<ea>,#data,An | |
| | ptestw | Dn,<ea>,#data,An | |
| | ptestw | sfc,<ea>,#data,An | |

| | | | |
|---|---|---|---|
| | ptestw | dfc,<ea>,#data,An | |

# 68040-Specific Instructions

This section lists several miscellaneous instructions specific to the 68040 processor.

| Name | Operator | Operand | Operation Name |
|---|---|---|---|
| cinv | cinvl | {ic,dc,bc},An@ | Invalidate Cache Lines |
| | cinvp | {ic,dc,bc},An@ | |
| | cinva | {ic,dc,bc} | |
| cpush | cpushl | {ic,dc,bc} | Push and Invalidate Cache Lines |
| | cpushp | {ic,dc,bc} | |
| | cpusha | {ic,dc,bc} | |
| fabs | fsabs{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Absolute Value |
| | fsabs{x} | FPn,FPn | |
| | fsabs{x} | FPn | |
| | fdabs{b,w,l,s,d,x,p} | <ea>,FPn | |
| | fdabs{x} | FPn,FPn | |
| | fdabs{x} | FPn | |
| fadd | fsadd{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Add |

| | | | |
|---|---|---|---|
| | fsadd{x} | FPn,FPn | |
| | fsadd{x} | FPn | |
| | fdadd{b,w,l,s,d,x,p} | <ea>,FPn | |
| | fdadd{x} | FPn,FPn | |
| | fdadd{x} | FPn | |
| fdiv | fsdiv{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Divide |
| | fsdiv{x} | FPn,FPn | |
| | fsdiv{x} | FPn | |
| | fddiv{b,w,l,s,d,x,p} | <ea>,FPn | |
| | fddiv{x} | FPn,FPn | |
| | fddiv{x} | FPn | |
| fmove | fsmove{b,w,l,s,d,x,p} | <ea>,FPn | Move Floating-Point |
| | fsmove{x} | FPn,FPn | Data Register |
| | fsmove{x} | FPn | |
| | fdmove{b,w,l,s,d,x,p} | <ea>,FPn | |
| | fdmove{x} | FPn,FPn | |
| | fdmove{x} | FPn | |
| fmul | fsmul{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Multiply |
| | fsmul{x} | FPn,FPn | |
| | fsmul{x} | FPn | |

| | | | |
|---|---|---|---|
| | fdmul{b,w,l,s,d,x,p} | <ea>,FPn | |
| | fdmul{x} | FPn,FPn | |
| | fdmul{x} | FPn | |
| fneg | fsneg{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Negate |
| | fsneg{x} | FPn,FPn | |
| | fsneg{x} | FPn | |
| | fdneg{b,w,l,s,d,x,p} | <ea>,FPn | |
| | fdneg{x} | FPn,FPn | |
| | fdneg{x} | FPn | |
| fsqrt | fssqrt{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Square Root |
| | fssqrt | FPn,FPn | |
| | fssqrt | FPn | |
| | fdsqrt{b,w,l,s,d,x,p} | <ea>,FPn | |
| | fdsqrt | FPn,FPn | |
| | fdsqrt | FPn | |
| fsub | fssub{b,w,l,s,d,x,p} | <ea>,FPn | Floating-Point Subtract |
| | fssub{x} | FPn,FPn | |
| | fssub{x} | FPn | |
| | fdsub{b,w,l,s,d,x,p} | <ea>,FPn | |
| | fdsub{x} | FPn,FPn | |
| | fdsub{x} | FPn | |

| | | | |
|---|---|---|---|
| move16 | move16 | An@+,An@+ | Move 16 Bytes Block |
| | move16 | #data,An@ | |
| | move16 | #data,An@+ | |
| | move16 | An@,#data | |
| | move16 | An@+,#data | |
| pflush | pflush | An@ | Flush ATC Entries |
| | pflushn | An@ | |
| | pflusha040 | | |
| | pflushan | | |
| ptest | ptestr | An@ | Test a Logical Address |
| | ptestw | An@ | |