APPENDIX

# C

# MOTOROLA 68000 INSTRUCTION SET

This appendix contains a summary of the Motorola 68000 instruction set. Part II of Chapter 3 gives an introductory discussion of the main characteristics of this processor, including a description of the register structure and the addressing modes, summarized in Figure 3.18 and Table 3.2, respectively. Note that Table 3.2 includes the assembler syntax for the addressing modes.

The general format for encoding the address field for an operand is shown in Table C.1. A 6-bit field specifies the addressing mode and the register involved. For the modes in which it is not necessary to specify a particular register, all 6 bits are used to specify the addressing mode.

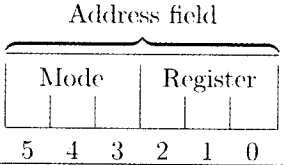
The names of addressing modes in Table C.1 are consistent with those used in this book. Some of these names differ from those used in Motorola literature. Because the reader will find it useful to consult the manufacturer's data sheets and user manuals, we summarize the differences in the terminology in Table C.2. The Motorola terminology is highly descriptive but somewhat awkward to use for discussion.

The 68000 instructions are presented in this appendix in the form of a table. To keep the table reasonably small, extensive notational abbreviations are used. Table C.3 gives the notational symbols and their meanings. Note that symbols that correspond to bit patterns in the OP-code field have one letter for each bit position involved.

Table C.4 provides a complete list of the available instructions. The addressing modes allowed for each instruction are indicated in a matrix format. For each source

*(Continued on page 766.)*

**Table C.1** Address field encoding for 68000




| Addressing mode | Mode field | Register field |
|---|---|---|
| Data register direct | 000 | Register number |
| Address register direct | 001 | Register number |
| Address register indirect | 010 | Register number |
| Autoincrement | 011 | Register number |
| Autodecrement | 100 | Register number |
| Indexed basic | 101 | Register number |
| Indexed full | 110 | Register number |
| Absolute short | 111 | 000 |
| Absolute long | 111 | 001 |
| Relative basic | 111 | 010 |
| Relative full | 111 | 011 |
| Immediate or status register | 111 | 100 |

**Table C.2** Differences from Motorola terminology

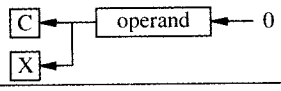
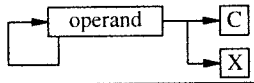
| Terminology used in this book | Motorola terminology |
|---|---|
| Autoincrement | Address register indirect with postincrement |
| Autodecrement | Address register indirect with predecrement |
| Indexed basic | Address register indirect with displacement |
| Indexed full | Address register indirect with index |
| Relative basic | Program counter with displacement |
| Relative full | Program counter with index |

**Table C.3** Notation for Table C.4

| Symbol | Meaning |
|---|---|
| s | Source operand |
| d | Destination operand |
| An | Address register $n$ |
| Dn | Data register $n$ |
| Xn | An address or data register, used as an index register |
| PC | Program counter |
| SP | Stack pointer |
| SR | Status register |
| CCR | Condition-code flags in SR |
| AAA | Address register number |
| DDD | Data register number |
| rrr | Source register number |
| RRR | Destination register number |
| eeeeee | Effective address of the source operand |
| EEEEEE | Effective address of the destination operand |
| MMM | Effective address mode of destination |
| CCCC | Specification for a condition code test |
| P...P | Displacement |
| Q...Q | Quick immediate data |
| SS | Size: 00 ≡ byte, 01 ≡ word, 10 ≡ long word (for most instructions); 01 ≡ byte, 11 ≡ word, 10 ≡ long word (for MOVE and MOVEA instructions) |
| VVVV | Trap vector number |
| u | Condition code flag state is undefined (meaningless) |
| d(An) | Indexed basic addressing mode |
| d(An,Xi) | Indexed full addressing mode |
| d(PC) | Relative basic addressing mode |
| d(PC,Xi) | Relative full addressing mode |

**Table C.4** 68000 instruction set

| Mnemonic (Name) | Size | Addressing mode | | Addressing mode: Dn | An | (An) | (An)+ | −(An) | d(An) | d(An,Xi) | Abs.W | Abs.L | d(PC) | d(PC,Xi) | Immed | SR or CCR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ABCD (Add BCD) | B | s = Dn | d = | x | | | | | | | | | | | | |
| | | s = −(An) | d = | | | | | x | | | | | | | | |
| ADD (Add) | B,W,L | s = Dn | d = | x | | x | x | x | x | x | x | x | | | | |
| | | d = Dn | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| ADDA (Add address) | W | d = An | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| | L | d = An | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| ADDI (Add immediate) | B,W,L | s = Immed | d = | x | | x | x | x | x | x | x | x | | | | |
| ADDQ (Add quick) | B,W,L | s = Immed3 | d = | x | x | x | x | x | x | x | x | x | | | | |
| ADDX (Add extended) | B,W,L | s = Dn | d = | x | | | | | | | | | | | | |
| | | s = −(An) | d = | | | | | x | | | | | | | | |
| AND (Logical AND) | B,W,L | s = Dn | d = | x | | x | x | x | x | x | x | x | | | | |
| | | d = Dn | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| ANDI (AND immediate) | B,W,L | s = Immed | d = | x | | x | x | x | x | x | x | x | | | | x |
| ASL (Arithmetic shift left) | B,W,L | count = [Dn] | d = | x | | | | | | | | | | | | |
| | | count = QQQ | d = | x | | | | | | | | | | | | |
| | | count = 1 | d = | | | x | x | x | x | x | x | x | | | | |
| ASR (Arithmetic shift right) | B,W,L | count = [Dn] | d = | x | | | | | | | | | | | | |
| | | count = QQQ | d = | x | | | | | | | | | | | | |
| | | count = 1 | d = | | | x | x | x | x | x | x | x | | | | |
| BCHG* (Test a bit and change it) | B | bit# = [Dn] | d = | | | x | x | x | x | x | x | x | | | | |
| | | bit# = Immed | d = | | | x | x | x | x | x | x | x | | | | |
| | L | bit# = [Dn] | d = | x | | | | | | | | | | | | |
| | | bit# = Immed | d = | x | | | | | | | | | | | | |
| BCLR* (Test a bit and clear it) | B | bit# = [Dn] | d = | | | x | x | x | x | x | x | x | | | | |
| | | bit# = Immed | d = | | | x | x | x | x | x | x | x | | | | |
| | L | bit# = [Dn] | d = | x | | | | | | | | | | | | |
| | | bit# = Immed | d = | x | | | | | | | | | | | | |

| OP code $b_{15} \ldots b_0$ | Operation performed | Condition flags X | N | Z | V | C |
|---|---|---|---|---|---|---|
| 1100 RRR1 0000 0rrr<br>1100 RRR1 0000 1rrr | d ← [s]+[d]+[X]<br>Binary-coded decimal addition | x | u | x | u | x |
| 1101 DDD1 SSEE EEEE<br>1101 DDD0 SSee eeee | d ← [Dn]+[d]<br>Dn ← [s]+[Dn] | x<br>x | x<br>x | x<br>x | x<br>x | x<br>x |
| 1101 AAA0 11ee eeee<br>1101 AAA1 11ee eeee | An ← [s]+[An] | | | | | |
| 0000 0110 SSEE EEEE | d ← s+[d] | x | x | x | x | x |
| 0101 QQQ0 SSEE EEEE | d ← QQQ+[d] | x | x | x | x | x |
| 1101 RRR1 SS00 0rrr<br>1101 RRR1 SS00 1rrr | d ← [s]+[d]+[X]<br>Multiprecision addition | x | x | x | x | x |
| 1100 DDD1 SSEE EEEE<br>1100 DDD0 SSee eeee | d ← [Dn] ∧ [d] | | x | x | 0 | 0 |
| 0000 0010 SSEE EEEE | d ← s ∧ [d] | | x | x | 0 | 0 |
| 1110 rrr1 SS10 0DDD<br>1110 QQQ1 SS00 0DDD<br>1110 0001 11EE EEEE |  | x | x | x | x | x |
| 1110 rrr0 SS10 0DDD<br>1110 QQQ0 SS00 0DDD<br>1110 0000 11EE EEEE |  | x | x | x | x | x |
| 0000 rrr1 01EE EEEE<br>0000 1000 01EE EEEE<br>0000 rrr1 01EE EEEE<br>0000 1000 01EE EEEE | Z ← $\overline{\text{(bit\# of d)}}$;<br>then complement the tested bit in d. | | | x | | |
| 0000 rrr1 10EE EEEE<br>0000 1000 10EE EEEE<br>0000 rrr1 10EE EEEE<br>0000 1000 10EE EEEE | Z ← $\overline{\text{(bit\# of d)}}$;<br>then clear the tested bit in d. | | | x | | |

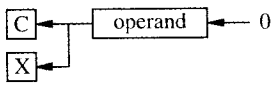
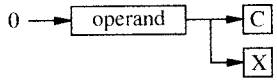
(Continued)

| Mnemonic (Name) | Size | Addressing mode | | Addressing mode | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Dn | An | (An) | (An)+ | -(An) | d(An) | d(An,Xi) | Abs.W | Abs.L | d(PC) | d(PC,Xi) | Immed | SR or CCR |
| BSET* (Test a bit and set it) | B | bit# = [Dn] | d = | | | x | x | x | x | x | x | x | | | | |
| | | bit# = Immed | d = | | | x | x | x | x | x | x | x | | | | |
| | L | bit# = [Dn] | d = | x | | | | | | | | | | | | |
| | | bit# = Immed | d = | x | | | | | | | | | | | | |
| BTST* (Test a bit) | B | bit# = [Dn] | d = | | | x | x | x | x | x | x | x | | | | |
| | | bit# = Immed | d = | | | x | x | x | x | x | x | x | | | | |
| | L | bit# = [Dn] | d = | x | | | | | | | | | | | | |
| | | bit# = Immed | d = | x | | | | | | | | | | | | |
| CHK (Check register) | W | d = Dn | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| CLR (Clear) | B,W,L | | d = | x | | x | x | x | x | x | x | x | | | | |
| CMP (Compare) | B,W,L | d = Dn | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| CMPA (Compare address) | W | d = An | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| | L | d = An | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| CMPI (Compare immed) | B,W,L | s = Immed | d = | x | | x | x | x | x | x | x | x | | | | |
| CMPM (Compare memory) | B,W,L | s = (An)+ | d = | | | | x | | | | | | | | | |
| DIVS (Divide signed) | W | d = Dn | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| DIVU (Divide unsigned) | W | d = Dn | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| EOR (Logical XOR) | B,W,L | s = Dn | d = | x | | x | x | x | x | x | x | x | | | | |
| EORI (XOR immediate) | B,W,L | s = Immed | d = | x | | x | x | x | x | x | x | x | | | | x |
| EXG (Exchange) | L | s = Dn | d = | x | x | | | | | | | | | | | |
| | | s = An | d = | x | x | | | | | | | | | | | |
| EXT (Sign extend) | W | | d = | x | | | | | | | | | | | | |
| | L | | d = | x | | | | | | | | | | | | |

| OP code $b_{15} \ldots b_0$ | Operation performed | Condition flags X | N | Z | V | C |
|---|---|---|---|---|---|---|
| 0000 rrr1 11EE EEEE<br>0000 1000 11EE EEEE<br>0000 rrr1 11EE EEEE<br>0000 1000 11EE EEEE | Z ← $\overline{\text{(bit\# of d)}}$:<br>then set to 1 the tested bit<br>in d. | | | x | | |
| 0000 rrr1 00EE EEEE<br>0000 1000 00EE EEEE<br>0000 rrr1 00EE EEEE<br>0000 1000 00EE EEEE | Z ← $\overline{\text{(bit\# of d)}}$: | | | x | | |
| 0100 DDD1 10ee eeee | If [Dn]< 0 or [Dn] > [s],<br>then raise an interrupt. | | x | u | u | u |
| 0100 0010 SSEE EEEE | d ← 0 | | 0 | 1 | 0 | 0 |
| 1011 DDD0 SSee eeee | [d] − [s] | | x | x | x | x |
| 1011 AAA0 11ee eeee<br>1011 AAA1 11ee eeee | [An] − [s] | | x | x | x | x |
| 0000 1100 SSEE EEEE | [d] − [s] | | x | x | x | x |
| 1011 RRR1 SS00 1rrr | [d] − [s] | | x | x | x | x |
| 1000 DDD1 11ee eeee | d ← [d] ÷ [s], using<br>32 bits of d and 16 bits of s. | | x | x | x | 0 |
| 1000 DDD0 11ee eeee | d ← [d] ÷ [s], using<br>32 bits of d and 16 bits of s. | | x | x | x | 0 |
| 1011 rrr1 SSEE EEEE | d ← [Dn] ⊕ [d] | | x | x | 0 | 0 |
| 0000 1010 SSEE EEEE | d ← s ⊕ [d] | | x | x | 0 | 0 |
| 1100 DDD1 0100 0DDD<br>1100 AAA1 0100 1AAA<br>1100 DDD1 1000 1AAA | [s] ↔ [d] | | | | | |
| 0100 1000 1000 0DDD<br>0100 1000 1100 0DDD | (bits 15–8 of d) ← (bit 7 of d)<br>(bits 31–16 of d) ← (bit 15 of d) | | x<br>x | x<br>x | 0<br>0 | 0<br>0 |

(*Continued*)

| Mnemonic (Name) | Size | Addressing mode | | Addressing mode | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Dn | An | (An) | (An)+ | −(An) | d(An) | d(An,Xi) | Abs.W | Abs.L | d(PC) | d(PC,Xi) | Immed | SR or CCR |
| JMP (Jump) | | | d= | | | x | | | x | x | x | x | x | x | | |
| JSR (Jump to subroutine) | | | d = | | | x | | | x | x | x | x | x | x | | |
| LEA (Load effective address) | L | d = An | s = | | | x | | | x | x | x | x | x | x | | |
| LINK (Link and allocate) | | disp = Immed | s = | | x | | | | | | | | | | | |
| LSL (Logical shift left) | B,W,L | count = [Dn] | d = | x | | | | | | | | | | | | |
| | | count = QQQ | d = | x | | | | | | | | | | | | |
| | W | count = I | d = | | | x | x | x | x | x | x | x | | | | |
| LSR (Logical shift right) | B,W,L | count = [Dn] | d = | x | | | | | | | | | | | | |
| | | count = QQQ | d = | x | | | | | | | | | | | | |
| | W | count = I | d = | | | x | x | x | x | x | x | x | | | | |
| MOVE (Move) | B,W,L | s =Dn | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = An | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = (An) | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = (An)+ | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = −(An) | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = d(An) | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = d(An,Xi) | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = Abs.W | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = Abs.L | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = d(PC) | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = d(PC,Xi) | d = | x | | x | x | x | x | x | x | x | | | | |
| | | s = Immed | d = | x | | x | x | x | x | x | x | x | | | | |
| | W | d = CCR | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| | | d = SR | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| | | s = SR | d = | x | | x | x | x | x | x | x | x | | | | |
| | L | s = SP | d = | | x | | | | | | | | | | | |
| | | d = SP | s = | | x | | | | | | | | | | | |
| MOVEA (Move address) | W,L | d = An | s = | x | x | x | x | x | x | x | x | x | x | x | x | |

| OP code $b_{15} \ldots b_0$ | Operation performed | Condition flags | | | | |
|---|---|---|---|---|---|---|
| | | X | N | Z | V | C |
| 0100 1110 11EE EEEE | PC ← effective address of d | | | | | |
| 0100 1110 10EE EEEE | SP ← [SP]−4;<br>[SP] ← [PC];<br>PC ← effective address of d | | | | | |
| 0100 AAA1 11ee eeee | An ← effective address of s | | | | | |
| 0100 1110 0101 0AAA | SP ← [SP]−4;<br>[SP] ← [An];<br>An ← [SP];<br>SP ← [SP] + disp | | | | | |
| 1110 rrr1 SS10 1DDD<br>1110 QQQ1 SS00 1DDD<br>1110 0011 11EE EEEE |  | x | x | x | 0 | x |
| 1110 rrr0 SS10 1DDD<br>1110 QQQ0 SS00 1DDD<br>1110 0010 11EE EEEE |  | x | x | x | 0 | x |
| 00SS RRRM MMee eeee | d ← [s] | | x | x | 0 | 0 |
| 0100 0100 11ee eeee | CCR ← [bits 7−0 of s] | x | x | x | x | x |
| 0100 0110 11ee eeee | SR ← [s] | x | x | x | x | x |
| 0100 0000 11EE EEEE | d ← [SR] | | | | | |
| 0100 1110 0110 1AAA | d ← [SP] | | | | | |
| 0100 1110 0110 0AAA | SP ← [d] | | | | | |
| 00SS AAA0 01ee eeee | An ← [s] | | | | | |

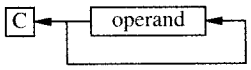
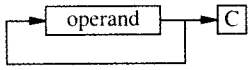
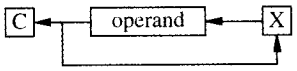
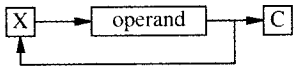
*(Continued)*

| Mnemonic (Name) | Size | Addressing mode | | Addressing mode | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Dn | An | (An) | (An)+ | –(An) | d(An) | d(An,Xi) | Abs.W | Abs.L | d(PC) | d(PC,Xi) | Immed | SR or CCR |
| MOVEM* (Move multiple registers) | W | s = Xn | d = | | | x | | x | x | x | x | x | | | | |
| | | d = Xn | s = | | | x | x | | x | x | x | x | x | x | | |
| | L | s = Xn | d = | | | x | | x | x | x | x | x | | | | |
| | | d = Xn | s = | | | x | x | | x | x | x | x | x | x | | |
| MOVEP* (Move peripheral data) | W | s = Dn | d = | | | | | | x | | | | | | | |
| | L | s = Dn | d = | | | | | | x | | | | | | | |
| | W | s = d(An) | d = | x | | | | | | | | | | | | |
| | L | s = d(An) | d = | x | | | | | | | | | | | | |
| MOVEQ (Move quick) | L | s = Immed8 | d = | x | | | | | | | | | | | | |
| MULS (Multiply signed) | W | d = Dn | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| MULU (Multiply unsigned) | W | d = Dn | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| NBCD (Negate BCD) | B | | d = | x | | x | x | x | x | x | x | x | | | | |
| NEG (Negate) | B.W.L | | d = | x | | x | x | x | x | x | x | x | | | | |
| NEGX (Negate extended) | B.W.L | | d = | x | | x | x | x | x | x | x | x | | | | |
| NOP (No operation) | | | | | | | | | | | | | | | | |
| NOT (Complement) | B.W.L | | d = | x | | x | x | x | x | x | x | x | | | | |
| OR (Logical OR) | B.W.L | s = Dn | d = | | | x | x | x | x | x | x | x | | | | |
| | | d = Dn | s = | x | | x | x | x | x | x | x | x | x | x | x | |
| ORI (OR immediate) | B.W.L | s = Immed | d = | x | | x | x | x | x | x | x | x | | | | x |
| PEA (Push effective address) | L | | s = | | | x | | | x | x | x | x | x | x | | |

| OP code $b_{15} \ldots b_0$ | Operation performed | Condition flags X | N | Z | V | C |
|---|---|---|---|---|---|---|
| 0100 1000 10EE EEEE<br>0100 1100 10ee eeee<br>0100 1000 11EE EEEE<br>0100 1100 11ee eeee | d ← [Xn]<br>Xn ← [s]<br>d ← [Xn]<br>Xn ← [s]<br>} A second word is used to specify the registers involved. | | | | | |
| 0000 DDD1 1000 1AAA<br>0000 DDD1 1100 1AAA<br>0000 DDD1 0000 1AAA<br>0000 DDD1 0100 1AAA | Alternate bytes of d ← [Dn]<br><br>Dn ← alternate bytes of d | | | | | |
| 0111 DDD0 QQQQ QQQQ | Dn ← QQQQQQQQ | | x | x | 0 | 0 |
| 1100 DDD1 11ee eeee | Dn ← [s] × [Dn] | | x | x | 0 | 0 |
| 1100 DDD0 11ee eeee | Dn ← [s] × [Dn] | | x | x | 0 | 0 |
| 0100 1000 00EE EEEE | d ← 0 − [d] − [X]<br>using BCD arithmetic | x | u | x | u | x |
| 0100 0100 SSEE EEEE | d ← 0 − [d] | x | x | x | x | x |
| 0100 0000 SSEE EEEE | d ← 0 − [d] − [X] | x | x | x | x | x |
| 0100 1110 0111 0001 | none | | | | | |
| 0100 0110 SSEE EEEE | d ← $\overline{[d]}$ | | x | x | 0 | 0 |
| 1000 DDD1 SSEE EEEE<br>1000 DDD0 SSee eeee | d ← [s] ∨ [d] | | x | x | 0 | 0 |
| 0000 0000 SSEE EEEE | d ← s ∨ [d] | | x | x | 0 | 0 |
| 0100 1000 01ee eeee | SP ← [SP] − 4:<br>[SP] ← effective address of s | | | | | |

*(Continued)*

| Mnemonic (Name) | Size | Addressing mode | | Addressing mode | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Dn | An | (An) | (An)+ | −(An) | d(An) | d(An,Xi) | Abs.W | Abs.L | d(PC) | d(PC,Xi) | Immed | SR or CCR |
| RESET | | | | | | | | | | | | | | | | |
| ROL (Rotate left without X) | B,W,L | count = [Dn] | d = | x | | | | | | | | | | | | |
| | | count = QQQ | d = | x | | | | | | | | | | | | |
| | W | count = 1 | d = | | | x | x | x | x | x | x | x | | | | |
| ROR (Rotate right without X) | B,W,L | count = [Dn] | d = | x | | | | | | | | | | | | |
| | | count = QQQ | d = | x | | | | | | | | | | | | |
| | W | count = 1 | | | | x | x | x | x | x | x | x | | | | |
| ROXL (Rotate left with X) | B,W,L | count = [Dn] | d = | x | | | | | | | | | | | | |
| | | count = QQQ | d = | x | | | | | | | | | | | | |
| | W | count = 1 | d = | | | x | x | x | x | x | x | x | | | | |
| ROXR (Rotate right with X) | B,W,L | count = [Dn] | d = | x | | | | | | | | | | | | |
| | | count = QQQ | d = | x | | | | | | | | | | | | |
| | W | count = 1 | d = | | | x | x | x | x | x | x | x | | | | |
| RTE (Return from exception) | | | | | | | | | | | | | | | | |
| RTR (Return and restore CCR) | | | | | | | | | | | | | | | | |
| RTS (Return from subroutine) | | | | | | | | | | | | | | | | |
| SBCD (Subtract BCD) | B | s = Dn | d = | x | | | | | | | | | | | | |
| | | s = − (An) | d = | | | | | x | | | | | | | | |
| Scc (Set on condition) | B | | d= | x | | x | x | x | x | x | x | x | | | | |
| STOP (Load SR and stop) | | | s = | | | | | | | | | | | | x | |

| OP code $b_{15} \ldots b_0$ | Operation performed | Condition flags X | N | Z | V | C |
|---|---|---|---|---|---|---|
| 0100 1110 0111 0000 | Assert RESET output line. | | | | | |
| 1110 rrr1 SS11 1DDD<br>1110 QQQ1 SS01 1DDD<br>1110 0111 11EE EEEE |  | | x | x | 0 | x |
| 1100 rrr1 SS11 1DDD<br>1110 QQQ0 SS01 1DDD<br>1110 0111 11EE EEEE |  | | x | x | 0 | x |
| 1110 rrr1 SS11 0DDD<br>1110 QQQ1 SS01 0DDD<br>1110 0101 11EE EEEE |  | x | x | x | 0 | x |
| 1110 rrr0 SS11 0DDD<br>1110 QQQ0 SS01 0DDD<br>1110 0100 11EE EEEE |  | x | x | x | 0 | x |
| 0100 1110 0111 0011 | SR ← [[SP]];<br>SP ← [SP] + 2;<br>PC ← [[SP]];<br>SP ← [SP] + 4; | x | x | x | x | x |
| 0100 1110 0111 0111 | CCR ← [[SP]];<br>SP ← [SP] + 2;<br>PC ← [[SP]];<br>SP ← [SP] + 4; | x | x | x | x | x |
| 0100 1110 0111 0101 | PC ← [[SP]];<br>SP ← [SP] + 4 | | | | | |
| 1000 RRR1 0000 0rrr<br>1000 RRR1 0000 1rrr | d ← [d] − [s] − [X]<br>Binary-coded decimal subtraction | x | u | x | u | x |
| 0101 CCCC 11EE EEEE | Set all 8 bits of d to 1 if cc is true, otherwise clear them to 0. | | | | | |
| 0100 1110 0111 0010 | SR ← s;<br>wait for interrupt. | x | x | x | x | x |

*(Continued)*

| Mnemonic (Name) | Size | Addressing mode | | Addressing mode | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Dn | An | (An) | (An)+ | −(An) | d(An) | d(An,Xi) | Abs.W | Abs.L | d(PC) | d(PC,Xi) | Immed | SR or CCR |
| SUB (Subtract) | B,W,L | s = Dn | d = | x | | x | x | x | x | x | x | x | | | | |
| | | d = Dn | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| SUBA (Subtract address) | W | d = An | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| | L | d = An | s = | x | x | x | x | x | x | x | x | x | x | x | x | |
| SUBI (Subtract immed) | B,W,L | s = Immed | d = | x | | x | x | x | x | x | x | x | | | | |
| SUBQ (Subtract quick) | B,W,L | s = Immed3 | d = | x | x | x | x | x | x | x | x | x | | | | |
| SUBX (Subtract extended) | B,W,L | s = Dn | d = | x | | | | | | | | | | | | |
| | | s = −(An) | d = | | | | | x | | | | | | | | |
| SWAP (Swap register halves) | W | | d = | x | | | | | | | | | | | | |
| TAS (Test and set) | B | | d = | x | | x | x | x | x | x | x | x | | | | |
| TRAP (Trap) | | s = Immed4 | | | | | | | | | | | | | | |
| TRAPV (Trap on overflow) | | | | | | | | | | | | | | | | |
| TST (Test) | B,W,L | | d = | x | | x | x | x | x | x | x | x | | | | |
| UNLK (Unlink) | | | | | x | | | | | | | | | | | |

| OP code $b_{15} \ldots b_0$ | Operation performed | Condition flags | | | | |
|---|---|---|---|---|---|---|
| | | X | N | Z | V | C |
| 1001 DDD1 SSEE EEEE<br>1001 DDD0 SSee eeee | d ← [d] − [s] | x | x | x | x | x |
| 1001 AAA0 11ee eeee<br>1001 AAA1 11ee eeee | An ← [An] − [s] | | | | | |
| 0000 0100 SSEE EEEE | d ← [d] − s | x | x | x | x | x |
| 0101 QQQ1 SSEE EEEE | d ← [d] − QQQ | x | x | x | x | x |
| 1001 RRR1 SS00 0rrr<br>1001 RRR1 SS00 1rrr | d ← [d] − [s] − [X] | x | x | x | x | x |
| 0100 1000 0100 0DDD | $[Dn]_{31-16} \leftrightarrow [Dn]_{15-0}$ | | x | x | 0 | 0 |
| 0100 1010 11EE EEEE | Test d and set N and Z flags:<br>set bit 7 of d to 1. | | x | x | 0 | 0 |
| 0100 1110 0100 VVVV | SP ← [SP] − 4:<br>[SP] ← [PC]:<br>SP ← [SP] − 2:<br>[SP] ← [SR]:<br>PC ← vector | | | | | |
| 0100 1110 0111 0110 | If V = 1, then<br>SP ← [SP] − 4:<br>[SP] ← [PC]:<br>SP ← [SP] − 2:<br>[SP] ← [SR]:<br>PC ← TRAPV vector | | | | | |
| 0100 1010 SSEE EEEE | Test d and set N and Z flags. | | x | x | 0 | 0 |
| 0100 1110 0101 1AAA | SP ← [An]:<br>An ← [[SP]]:<br>SP ← [SP] + 4 | | | | | |

*(Concluded)*

(destination) addressing mode provided, all destination (source) addressing modes permitted are denoted with an x. For example, for the AND instruction, if the source is a data register, the destination mode may be (An), (An)+, −(An), d(An), d(An,Xi), Abs.W, or Abs.L. Moreover, if the destination is a data register, the source can be specified in any of the 11 modes shown in the table.

The OP-code column shows the actual bit pattern of the first 16-bit word of an instruction. Instructions that have immediate source data use a second word for 8- and 16-bit operands, and a second and third word for 32-bit operands. For the indexed and relative addressing modes, the required index value (that is, the displacement) is given in the word that follows the OP code.

Shift and Rotate instructions can specify a count of the number of bit positions by which the operand is to be shifted or rotated. The count can be given as the contents of a data register or as an immediate 3-bit value within the OP code. If a memory operand is involved, the count is always equal to 1.

Branch instructions are listed in Table C.5. The branch offset (the displacement) is a signed 2's-complement number that specifies the relative distance in bytes. For conditional branch instructions, as well as for Scc (Set on condition) instructions, the condition code suffix possibilities (cc) are shown in Table C.6. This table also indicates the condition that is tested to determine if a branch is to be taken.

The operation performed for a given instruction is indicated in Tables C.4 and C.5. For most instructions, the action taken is obvious. However, for a few instructions, additional comments are in order. The instructions labeled with an asterisk in the mnemonic column are discussed further in the following paragraphs.

### BCHG, BCLR, BSET, and BTST

All of these instructions test a specified bit of the destination operand. The number of the bit position to be tested (bit#) is indicated either as the contents of a data register or as an immediate value within the instruction. The test is made by loading the complement of the tested bit into the condition flag Z.

### MOVEM

This instruction moves the contents of one or more registers to or from consecutive memory locations. The registers involved in the transfer are specified in the second word of the instruction. Bits 0 through 7 correspond to D0 through D7, and bits 8 through 15 correspond to A0 through A7. This arrangement is valid for all addressing modes except the autodecrement mode, in which case the order of registers is reversed.

### MOVEP

This instruction is useful for data transfers between the 68000 and 8-bit peripheral devices. The data are transferred in bytes, with the memory address incremented by 2 after each byte. Thus, if the starting address is even, all bytes are transferred to or from even-numbered address locations by means of the high-order eight lines of the data bus. Similarly, if the starting address is odd, then all transfers are done via the low-order eight lines of the data bus. The high-order byte of a data register is transferred first and the low-order byte is transferred second.

**Table C.5** 68000 branch instructions

| Mnemonic (Name) | Displacement size | OP code | Operation performed |
|---|---|---|---|
| BRA (Branch always) | 8<br>16 | 0100 0000 PPPP PPPP<br>0110 0000 0000 0000<br>PPPP PPPP PPPP PPPP | PC ← [PC] + disp |
| Bcc (Branch conditionally) | 8<br>16 | 0110 CCCC PPPP PPPP<br>0110 CCCC 0000 0000<br>PPPP PPPP PPPP PPPP | If cc is true, then<br>PC ← [PC] + disp |
| BSR (Branch to subroutine) | 8<br>16 | 0110 0001 PPPP PPPP<br>0110 0001 0000 0000<br>PPPP PPPP PPPP PPPP | SP ← [SP] − 4;<br>[SP] ← [PC];<br>PC ← [PC] + disp |
| DBcc (Decrement and branch conditionally) | 16 | 0101 CCCC 1100 1DDD<br>PPPP PPPP PPPP PPPP | If cc is false, then<br>Dn ← [Dn] − 1;<br>If [Dn] ≠ − 1, then<br>PC ← [PC] + disp |
| DBRA (Decrement and branch) | The assembler interprets this instruction as DBF (see the DBcc entry). | | |

The 68000 has two basic modes of operation. In the supervisor mode, all instructions can be used. In the user mode, some instructions cannot be executed. Instructions that can only be used in the supervisor mode are called privileged instructions. These are

- ANDI, EORI, ORI, and MOVE instructions when the destination is the status register SR
- MOVE instruction, which moves the contents of the user stack pointer to or from an address register
- RESET, RTE, and STOP instructions

The information presented in this appendix should enable the reader to write and debug assembly-language programs for the 68000. The size and structure of assembled

**Table C.6** Condition codes for Bcc, DBcc, and Scc instructions

| Machine code CCCC | Condition suffix cc | Name | Test condition |
|---|---|---|---|
| 0000 | T | True | Always true |
| 0001 | F | False | Always false |
| 0010 | HI | High | $C \vee Z = 0$ |
| 0011 | LS | Low or same | $C \vee Z = 1$ |
| 0100 | CC | Carry clear | $C = 0$ |
| 0101 | CS | Carry set | $C = 1$ |
| 0110 | NE | Not equal | $Z = 0$ |
| 0111 | EQ | Equal | $Z = 1$ |
| 1000 | VC | Overflow clear | $V = 0$ |
| 1001 | VS | Overflow set | $V = 1$ |
| 1010 | PL | Plus | $N = 0$ |
| 1011 | MI | Minus | $N = 1$ |
| 1100 | GE | Greater or equal | $N \oplus V = 0$ |
| 1101 | LT | Less than | $N \oplus V = 1$ |
| 1110 | GT | Greater than | $Z \vee (N \oplus V) = 0$ |
| 1111 | LE | Less or equal | $Z \vee (N \oplus V) = 1$ |

1 T and F suffixes cannot be used in the Bcc instruction

instructions can be determined on the basis of the OP codes given and the addressing modes employed. Lack of space has prevented the inclusion of timing information, such as the number of machine cycles needed to execute a given instruction. This information, as well as further details about the instruction set, can be found in the manufacturer's literature.