

MOTOROLA

M68000 FAMILY

Programmer's Reference Manual

(Includes CPU32 Instructions)

TABLE OF CONTENTS

| Paragraph Number | Title | Page Number |
|---------------------|---|-------------|
| Section 1 | | |
| Introduction | | |
| 1.1 | Integer Unit User Programming Model | 1-2 |
| 1.1.1 | Data Registers (D7 – D0) | 1-2 |
| 1.1.2 | Address Registers (A7 – A0) | 1-2 |
| 1.1.3 | Program Counter | 1-3 |
| 1.1.4 | Condition Code Register | 1-3 |
| 1.2 | Floating-Point Unit User Programming Model | 1-4 |
| 1.2.1 | Floating-Point Data Registers (FP7 – FP0) | 1-4 |
| 1.2.2 | Floating-Point Control Register (FPCR) | 1-5 |
| 1.2.2.1 | Exception Enable Byte. | 1-5 |
| 1.2.2.2 | Mode Control Byte. | 1-5 |
| 1.2.3 | Floating-Point Status Register (FPSR) | 1-5 |
| 1.2.3.1 | Floating-Point Condition Code Byte. | 1-5 |
| 1.2.3.2 | Quotient Byte. | 1-6 |
| 1.2.3.3 | Exception Status Byte. | 1-6 |
| 1.2.3.4 | Accrued Exception Byte. | 1-7 |
| 1.2.4 | Floating-Point Instruction Address Register (FPIAR) | 1-8 |
| 1.3 | Supervisor Programming Model. | 1-8 |
| 1.3.1 | Address Register 7 (A7) | 1-10 |
| 1.3.2 | Status Register | 1-10 |
| 1.3.3 | Vector Base Register (VBR) | 1-11 |
| 1.3.4 | Alternate Function Code Registers (SFC and DFC) | 1-11 |
| 1.3.5 | Acu Status Register (MC68EC030 only) | 1-11 |
| 1.3.6 | Transparent Translation/access Control Registers | 1-12 |
| 1.3.6.1 | Transparent Translation/access Control Register Fields for the M68030. | 1-12 |
| 1.3.6.2 | Transparent Translation/access Control Register Fields for the M68040. | 1-13 |
| 1.4 | Integer Data Formats | 1-14 |
| 1.5 | Floating-Point Data Formats | 1-15 |
| 1.5.1 | Packed Decimal Real Format | 1-15 |
| 1.5.2 | Binary Floating-Point Formats | 1-16 |
| 1.6 | Floating-Point Data Types | 1-17 |
| 1.6.1 | Normalized Numbers. | 1-18 |
| 1.6.2 | Denormalized Numbers. | 1-18 |
| 1.6.3 | Zeros | 1-19 |
| 1.6.4 | Infinities | 1-19 |
| 1.6.5 | Not-A-Numbers | 1-19 |
| 1.6.6 | Data Format and Type Summary | 1-20 |
| 1.7 | Organization of Data in Registers | 1-25 |
| 1.7.1 | Organization of Integer Data Formats in Registers | 1-25 |

TABLE OF CONTENTS (Continued)

| Paragraph Number | Title | Page Number |
|------------------|--|-------------|
| 1.7.2 | Organization of Integer Data Formats in Memory | 1-27 |
| 1.7.3 | Organization of Fpu Data Formats in Registers and Memory | 1-30 |

Section 2 Addressing Capabilities

| | | |
|---------|--|------|
| 2.1 | Instruction Format | 2-1 |
| 2.2 | Effective Addressing Modes | 2-4 |
| 2.2.1 | Data Register Direct Mode | 2-5 |
| 2.2.2 | Address Register Direct Mode | 2-5 |
| 2.2.3 | Address Register Indirect Mode | 2-5 |
| 2.2.4 | Address Register Indirect with Postincrement Mode | 2-6 |
| 2.2.5 | Address Register Indirect with Predecrement Mode | 2-7 |
| 2.2.6 | Address Register Indirect with Displacement Mode | 2-8 |
| 2.2.7 | Address Register Indirect with Index (8-Bit Displacement) Mode | 2-9 |
| 2.2.8 | Address Register Indirect with Index (Base Displacement) Mode | 2-10 |
| 2.2.9 | Memory Indirect Postindexed Mode | 2-11 |
| 2.2.10 | Memory Indirect Preindexed Mode | 2-12 |
| 2.2.11 | Program Counter Indirect with Displacement Mode | 2-13 |
| 2.2.12 | Program Counter Indirect with Index (8-Bit Displacement) Mode | 2-14 |
| 2.2.13 | Program Counter Indirect with Index (Base Displacement) Mode | 2-15 |
| 2.2.14 | Program Counter Memory Indirect Postindexed Mode | 2-16 |
| 2.2.15 | Program Counter Memory Indirect Preindexed Mode | 2-17 |
| 2.2.16 | Absolute Short Addressing Mode | 2-18 |
| 2.2.17 | Absolute Long Addressing Mode | 2-18 |
| 2.2.18 | Immediate Data | 2-19 |
| 2.3 | Effective Addressing Mode Summary | 2-19 |
| 2.4 | Brief Extension Word Format Compatibility | 2-21 |
| 2.5 | Full Extension Addressing Modes | 2-22 |
| 2.5.1 | No Memory Indirect Action Mode | 2-24 |
| 2.5.2 | Memory Indirect Modes | 2-25 |
| 2.5.2.1 | Memory Indirect with Preindex | 2-25 |
| 2.5.2.2 | Memory Indirect with Postindex | 2-26 |
| 2.5.2.3 | Memory Indirect with Index Suppressed | 2-27 |
| 2.6 | Other Data Structures | 2-28 |
| 2.6.1 | System Stack | 2-28 |
| 2.6.2 | Queues | 2-29 |

Section 3 Instruction Set Summary

| | | |
|-------|---|-----|
| 3.1 | Instruction Summary | 3-1 |
| 3.1.1 | Data Movement Instructions | 3-5 |
| 3.1.2 | Integer Arithmetic Instructions | 3-6 |

TABLE OF CONTENTS (Continued)

| Paragraph Number | Title | Page Number |
|------------------|---|-------------|
| 3.1.3 | Logical Instructions | 3-8 |
| 3.1.4 | Shift and Rotate Instructions | 3-8 |
| 3.1.5 | Bit Manipulation Instructions | 3-10 |
| 3.1.6 | Bit Field Instructions | 3-10 |
| 3.1.7 | Binary-Coded Decimal Instructions | 3-11 |
| 3.1.8 | Program Control Instructions | 3-11 |
| 3.1.9 | System Control Instructions | 3-12 |
| 3.1.10 | Cache Control Instructions (MC68040) | 3-14 |
| 3.1.11 | Multiprocessor Instructions | 3-14 |
| 3.1.12 | Memory Management Unit (MMU) Instructions | 3-15 |
| 3.1.13 | Floating-Point Arithmetic Instructions | 3-15 |
| 3.2 | Integer Unit Condition Code Computation | 3-17 |
| 3.3 | Instruction Examples | 3-20 |
| 3.3.1 | Using the Cas and Cas2 Instructions | 3-20 |
| 3.3.2 | Using the Moves Instruction | 3-20 |
| 3.3.3 | Nested Subroutine Calls | 3-20 |
| 3.3.4 | Bit Field Instructions | 3-20 |
| 3.3.5 | Pipeline Synchronization with the Nop Instruction | 3-21 |
| 3.4 | Floating-Point Instruction Details | 3-21 |
| 3.5 | Floating-Point Computational Accuracy | 3-23 |
| 3.5.1 | Intermediate Result | 3-24 |
| 3.5.2 | Rounding the Result | 3-25 |
| 3.6 | Floating-Point Postprocessing | 3-27 |
| 3.6.1 | Underflow, Round, Overflow | 3-28 |
| 3.6.2 | Conditional Testing | 3-28 |
| 3.7 | Instruction Descriptions | 3-32 |

Section 4 Integer Instructions

Section 5 Floating Point Instructions

Section 6 Supervisor (Privileged) Instructions

Section 7 CPU32 Instructions

Section 8 Instruction Format Summary

| | | |
|-----|------------------------------|-----|
| 8.1 | Instruction Format | 8-1 |
|-----|------------------------------|-----|

TABLE OF CONTENTS (Continued)

| Paragraph Number | Title | Page Number |
|------------------|---|-------------|
| 8.1.1 | Coprocessor ID Field | 8-1 |
| 8.1.2 | Effective Address Field | 8-1 |
| 8.1.3 | Register/Memory Field | 8-1 |
| 8.1.4 | Source Specifier Field | 8-1 |
| 8.1.5 | Destination Register Field | 8-2 |
| 8.1.6 | Conditional Predicate Field | 8-2 |
| 8.1.7 | Shift and Rotate Instructions | 8-2 |
| 8.1.7.1 | Count Register Field. | 8-2 |
| 8.1.7.2 | Register Field. | 8-2 |
| 8.1.8 | Size Field. | 8-4 |
| 8.1.9 | Opmode Field | 8-4 |
| 8.1.10 | Address/Data Field | 8-4 |
| 8.2 | Operation Code Map | 8-4 |

Appendix A

Processor Instruction Summary

| | | |
|-------|--|------|
| A.1 | MC68000, MC68008, MC68010 Processors | A-12 |
| A.1.1 | M68000, MC68008, and MC68010 Instruction Set | A-12 |
| A.1.2 | MC68000, MC68008, and MC68010 Addressing Modes | A-16 |
| A.2 | MC68020 Processors. | A-17 |
| A.2.1 | MC68020 Instruction Set. | A-17 |
| A.2.2 | MC68020 Addressing Modes | A-20 |
| A.3 | MC68030 Processors. | A-21 |
| A.3.1 | MC68030 Instruction Set. | A-21 |
| A.3.2 | MC68030 Addressing Modes | A-24 |
| A.4 | MC68040 Processors. | A-25 |
| A.4.1 | MC68040 Instruction Set. | A-25 |
| A.4.2 | MC68040 Addressing Modes | A-29 |
| A.5 | MC68881/MC68882 Coprocessors | A-30 |
| A.5.1 | MC68881/MC68882 Instruction Set | A-30 |
| A.5.2 | MC68881/MC68882 Addressing Modes | A-31 |
| A.6 | MC68851 Coprocessors. | A-31 |
| A.6.1 | MC68851 Instruction Set. | A-31 |
| A.6.2 | MC68851 Addressing Modes | A-31 |

Appendix B

Exception Processing Reference

| | | |
|-----|--|------|
| B.1 | Exception Vector Assignments for the M68000 Family | B-1 |
| B.2 | Exception Stack Frames | B-3 |
| B.3 | Floating-Point Stack Frames | B-10 |

TABLE OF CONTENTS (Concluded)

| Paragraph Number | Title | Page Number |
|-----------------------------|-------------------------------|------------------------|
| | Appendix C | |
| | S-Record Output Format | |
| C.1 | S-Record Content. | C-1 |
| C.2 | S-Record Types | C-2 |
| C.3 | S-Record Creation | C-3 |

LIST OF FIGURES

| Figure Number | Title | Page Number |
|------------------|--|----------------|
| 1-1 | M68000 Family User Programming Model..... | 1-2 |
| 1-2 | M68000 Family Floating-Point Unit User Programming Model..... | 1-4 |
| 1-3 | Floating-Point Control Register..... | 1-5 |
| 1-4 | FPSR Condition Code Byte..... | 1-6 |
| 1-5 | FPSR Quotient Code Byte..... | 1-6 |
| 1-6 | FPSR Exception Status Byte..... | 1-6 |
| 1-7 | FPSR Accrued Exception Byte..... | 1-7 |
| 1-8 | Status Register..... | 1-11 |
| 1-9 | MC68030 Transparent Translation/MC68EC030 Access Control Register Format..... | 1-12 |
| 1-10 | MC68040 and MC68LC040 Transparent Translation/MC68EC040 Access Control Register Format..... | 1-13 |
| 1-11 | Packed Decimal Real Format..... | 1-16 |
| 1-12 | Binary Floating-Point Data Formats..... | 1-16 |
| 1-13 | Normalized Number Format..... | 1-18 |
| 1-14 | Denormalized Number Format..... | 1-18 |
| 1-15 | Zero Format..... | 1-19 |
| 1-16 | Infinity Format..... | 1-19 |
| 1-17 | Not-A-Number Format..... | 1-19 |
| 1-19 | Organization of Integer Data Formats in Address Registers..... | 1-26 |
| 1-18 | Organization of Integer Data Formats in Data Registers..... | 1-26 |
| 1-20 | Memory Operand Addressing..... | 1-27 |
| 1-21 | Memory Organization for Integer Operands..... | 1-29 |
| 1-22 | Organization of FPU Data Formats in Memory..... | 1-30 |
| | | |
| 2-1 | Instruction Word General Format..... | 2-1 |
| 2-2 | Instruction Word Specification Formats..... | 2-2 |
| 2-3 | M68000 Family Brief Extension Word Formats..... | 2-21 |
| 2-4 | Addressing Array Items..... | 2-23 |
| 2-5 | No Memory Indirect Action..... | 2-24 |
| 2-6 | Memory Indirect with Preindex..... | 2-26 |
| 2-7 | Memory Indirect with Postindex..... | 2-27 |
| 2-8 | Memory Indirect with Index Suppress..... | 2-27 |
| | | |
| 3-1 | Intermediate Result Format..... | 3-24 |
| 3-2 | Rounding Algorithm Flowchart..... | 3-26 |
| 3-3 | Instruction Description Format..... | 3-33 |
| | | |
| B-1 | MC68000 Group 1 and 2 Exception Stack Frame..... | B-3 |
| B-2 | MC68000 Bus or Address Error Exception Stack Frame..... | B-3 |
| B-3 | Four-Word Stack Frame, Format \$0..... | B-3 |
| B-4 | Throwaway Four-Word Stack Frame, Format \$1..... | B-3 |

LIST OF FIGURES (Concluded)

| Figure Number | Title | Page Number |
|------------------|---|----------------|
| B-5 | Six-Word Stack Frame, Format \$2..... | B-4 |
| B-6 | MC68040 Floating-Point Post-Instruction Stack Frame, Format \$3..... | B-4 |
| B-7 | MC68EC040 and MC68LC040 Floating-Point Unimplemented Stack Frame, Format \$4 | B-5 |
| B-8 | MC68040 Access Error Stack Frame, Format \$7 | B-5 |
| B-9 | MC68010 Bus and Address Error Stack Frame, Format \$8 | B-6 |
| B-10 | MC68020 Bus and MC68030 Coprocessor Mid-Instruction Stack Frame, Format \$9 | B-6 |
| B-11 | MC68020 and MC68030 Short Bus Cycle Stack Frame, Format \$A..... | B-7 |
| B-12 | MC68020 and MC68030 Long Bus Cycle Stack Frame, Format \$B..... | B-8 |
| B-13 | CPU32 Bus Error for Prefetches and Operands Stack Frame, Format \$C..... | B-8 |
| B-14 | CPU32 Bus Error on MOVEM Operand Stack Frame, Format \$C | B-9 |
| B-15 | CPU32 Four- and Six-Word Bus Error Stack Frame, Format \$C..... | B-9 |
| B-16 | MC68881/MC68882 and MC68040 Null Stack Frame..... | B-10 |
| B-17 | MC68881 Idle Stack Frame | B-10 |
| B-18 | MC68881 Busy Stack Frame | B-11 |
| B-19 | MC68882 Idle Stack Frame | B-11 |
| B-20 | MC68882 Busy Stack Frame | B-11 |
| B-21 | MC68040 Idle Busy Stack Frame | B-12 |
| B-22 | MC68040 Unimplemented Instruction Stack Frame..... | B-12 |
| B-23 | MC68040 Busy Stack Frame | B-13 |
| C-1 | Five Fields of an S-Record..... | C-1 |
| C-2 | Transmission of an S1 Record..... | C-4 |

LIST OF TABLES

| Table Number | Title | Page Number |
|-----------------|---|----------------|
| 1-1 | Supervisor Registers Not Related To Paged Memory Management | 1-9 |
| 1-2 | Supervisor Registers Related To Paged Memory Management..... | 1-10 |
| 1-3 | Integer Data Formats | 1-15 |
| 1-4 | Single-Precision Real Format Summary Data Format | 1-21 |
| 1-5 | Double-Precision Real Format Summary..... | 1-22 |
| 1-6 | Extended-Precision Real Format Summary..... | 1-23 |
| 1-6 | Extended-Precision Real Format Summary (Continued) | 1-24 |
| 1-7 | Packed Decimal Real Format Summary | 1-24 |
| 1-8 | MC68040 FPU Data Formats and Data Types | 1-30 |
| | | |
| 2-1 | Instruction Word Format Field Definitions | 2-3 |
| 2-2 | IS-I/IS Memory Indirect Action Encodings..... | 2-4 |
| 2-3 | Immediate Operand Location..... | 2-19 |
| 2-4 | Effective Addressing Modes and Categories | 2-20 |
| | | |
| 3-1 | Notational Conventions | 3-2 |
| 3-1 | Notational Conventions (Continued) | 3-3 |
| 3-1 | Notational Conventions (Concluded) | 3-4 |
| 3-2 | Data Movement Operation Format..... | 3-6 |
| 3-3 | Integer Arithmetic Operation Format..... | 3-7 |
| 3-4 | Logical Operation Format..... | 3-8 |
| 3-5 | Shift and Rotate Operation Format | 3-9 |
| 3-6 | Bit Manipulation Operation Format | 3-10 |
| 3-7 | Bit Field Operation Format | 3-10 |
| 3-8 | Binary-Coded Decimal Operation Format..... | 3-11 |
| 3-9 | Program Control Operation Format..... | 3-12 |
| 3-10 | System Control Operation Format | 3-13 |
| 3-11 | Cache Control Operation Format | 3-14 |
| 3-12 | Multiprocessor Operations | 3-14 |
| 3-13 | MMU Operation Format | 3-15 |
| 3-14 | Dyadic Floating-Point Operation Format..... | 3-16 |
| 3-15 | Dyadic Floating-Point Operations | 3-16 |
| 3-16 | Monadic Floating-Point Operation Format | 3-16 |
| 3-17 | Monadic Floating-Point Operations..... | 3-17 |
| 3-18 | Integer Unit Condition Code Computations..... | 3-18 |
| 3-19 | Conditional Tests | 3-19 |
| 3-20 | Operation Table Example (FADD Instruction)..... | 3-22 |
| 3-21 | FPCR Encodings..... | 3-25 |
| 3-22 | FPCC Encodings..... | 3-29 |
| 3-23 | Floating-Point Conditional Tests | 3-31 |
| 5-1 | Directly Supported Floating-Point Instructions..... | 5-2 |
| 5-2 | Indirectly Supported Floating-Point Instructions..... | 5-3 |

LIST OF TABLES (Continued)

| Table Number | Title | Page Number |
|-----------------|--|----------------|
| 7-1 | MC68020 Instructions Not Supported | 7-1 |
| 7-2 | M68000 Family Addressing Modes | 7-2 |
| 7-3 | CPU32 Instruction Set..... | 7-3 |
| 8-1 | Conditional Predicate Field Encoding | 8-3 |
| 8-2 | Operation Code Map..... | 8-4 |
| A-1 | M68000 Family Instruction Set And Processor Cross-Reference..... | A-1 |
| A-2 | M68000 Family Instruction Set..... | A-8 |
| A-3 | MC68000 and MC68008 Instruction Set..... | A-12 |
| A-4 | MC68010 Instruction Set..... | A-14 |
| A-5 | MC68000, MC68008, and MC68010 Data Addressing Modes | A-16 |
| A-6 | MC68020 Instruction Set Summary | A-17 |
| A-7 | MC68020 Data Addressing Modes | A-20 |
| A-8 | MC68030 Instruction Set Summary | A-21 |
| A-9 | MC68030 Data Addressing Modes | A-24 |
| A-10 | MC68040 Instruction Set..... | A-25 |
| A-11 | MC68040 Data Addressing Modes | A-29 |
| A-12 | MC68881/MC68882 Instruction Set..... | A-30 |
| A-13 | MC68851 Instruction Set..... | A-31 |
| B-1 | Exception Vector Assignments for the M68000 Family..... | B-2 |
| C-1 | Field Composition of an S-Record | C-1 |
| C-2 | ASCII Code | C-5 |

SECTION 1

INTRODUCTION

This manual contains detailed information about software instructions used by the microprocessors and coprocessors in the M68000 family, including:

| | | |
|-----------|---|--|
| MC68000 | — | 16-/32-Bit Microprocessor |
| MC68EC000 | — | 16-/32-Bit Embedded Controller |
| MC68HC000 | — | Low Power 16-/32-Bit Microprocessor |
| MC68008 | — | 16-Bit Microprocessor with 8-Bit Data Bus |
| MC68010 | — | 16-/32-Bit Virtual Memory Microprocessor |
| MC68020 | — | 32-Bit Virtual Memory Microprocessor |
| MC68EC020 | — | 32-Bit Embedded Controller |
| MC68030 | — | Second-Generation 32-Bit Enhanced Microprocessor |
| MC68EC030 | — | 32-Bit Embedded Controller |
| MC68040 | — | Third-Generation 32-Bit Microprocessor |
| MC68LC040 | — | Third-Generation 32-Bit Microprocessor |
| MC68EC040 | — | 32-Bit Embedded Controller |
| MC68330 | — | Integrated CPU32 Processor |
| MC68340 | — | Integrated Processor with DMA |
| MC68851 | — | Paged Memory Management Unit |
| MC68881 | — | Floating-Point Coprocessor |
| MC68882 | — | Enhanced Floating-Point Coprocessor |

NOTE

All references to the MC68000, MC68020, and MC68030 include the corresponding embedded controllers, MC68EC000, MC68EC020, and MC68EC030. All references to the MC68040 include the MC68LC040 and MC68EC040. This referencing method applies throughout the manual unless otherwise specified.

The M68000 family programming model consists of two register groups: user and supervisor. User programs executing in the user mode only use the registers in the user group. System software executing in the supervisor mode can access all registers and uses the control registers in the supervisor group to perform supervisor functions. The following paragraphs provide a brief description of the registers in the user and supervisor models as well as the data organization in the registers.

1.1 INTEGER UNIT USER PROGRAMMING MODEL

Figure 1-1 illustrates the integer portion of the user programming model. It consists of the following registers:

- 16 General-Purpose 32-Bit Registers (D7 – D0, A7 – A0)
- 32-Bit Program Counter (PC)
- 8-Bit Condition Code Register (CCR)

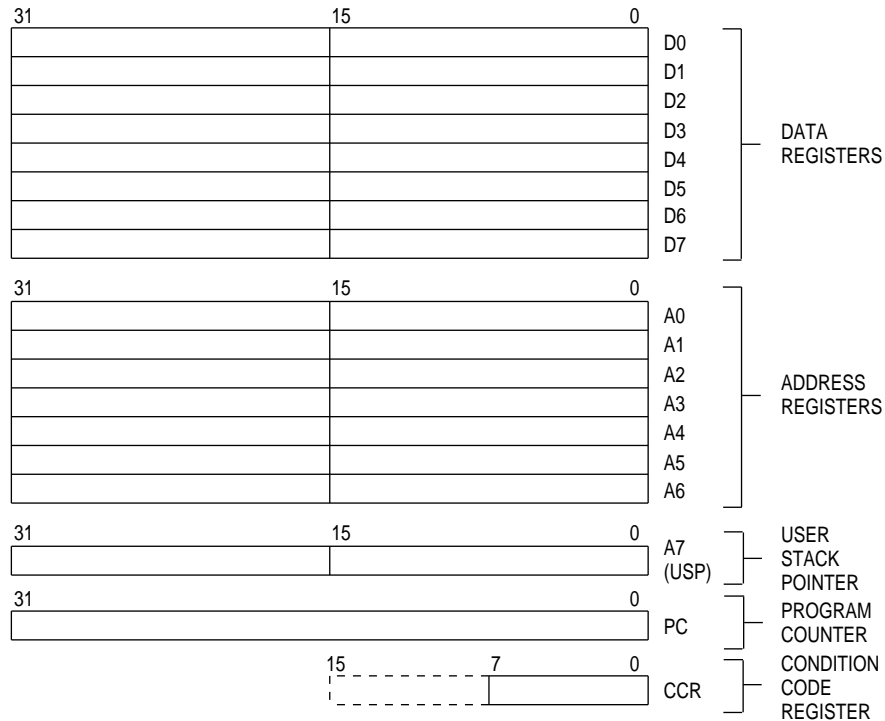


Figure 1-1. M68000 Family User Programming Model

1.1.1 Data Registers (D7 – D0)

These registers are for bit and bit field (1 – 32 bits), byte (8 bits), word (16 bits), long-word (32 bits), and quad-word (64 bits) operations. They also can be used as index registers.

1.1.2 Address Registers (A7 – A0)

These registers can be used as software stack pointers, index registers, or base address registers. The base address registers can be used for word and long-word operations. Register A7 is used as a hardware stack pointer during stacking for subroutine calls and exception handling. In the user programming model, A7 refers to the user stack pointer (USP).

1.1.3 Program Counter

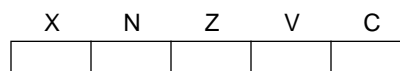
The PC contains the address of the instruction currently executing. During instruction execution and exception processing, the processor automatically increments the contents or places a new value in the PC. For some addressing modes, the PC can be used as a pointer for PC relative addressing.

1.1.4 Condition Code Register

Consisting of five bits, the CCR, the status register's lower byte, is the only portion of the status register (SR) available in the user mode. Many integer instructions affect the CCR, indicating the instruction's result. Program and system control instructions also use certain combinations of these bits to control program and system flow. The condition codes meet two criteria: consistency across instructions, uses, and instances and meaningful results with no change unless it provides useful information.

Consistency across instructions means that all instructions that are special cases of more general instructions affect the condition codes in the same way. Consistency across uses means that conditional instructions test the condition codes similarly and provide the same results whether a compare, test, or move instruction sets the condition codes. Consistency across instances means that all instances of an instruction affect the condition codes in the same way.

The first four bits represent a condition of the result generated by an operation. The fifth bit or the extend bit (X-bit) is an operand for multiprecision computations. The carry bit (C-bit) and the X-bit are separate in the M68000 family to simplify programming techniques that use them (refer to Table 3-18 as an example). In the instruction set definitions, the CCR is illustrated as follows:



X—Extend

Set to the value of the C-bit for arithmetic operations; otherwise not affected or set to a specified result.

N—Negative

Set if the most significant bit of the result is set; otherwise clear.

Z—Zero

Set if the result equals zero; otherwise clear.

V—Overflow

Set if an arithmetic overflow occurs implying that the result cannot be represented in the operand size; otherwise clear.

C—Carry

Set if a carry out of the most significant bit of the operand occurs for an addition, or if a borrow occurs in a subtraction; otherwise clear.

1.2 FLOATING-POINT UNIT USER PROGRAMMING MODEL

The following paragraphs describe the registers for the floating-point unit user programming model. Figure 1-2 illustrates the M68000 family user programming model's floating-point portion for the MC68040 and the MC68881/MC68882 floating-point coprocessors. It contains the following registers:

- 8 Floating-Point Data Registers (FP7 – FP0)
- 16-Bit Floating-Point Control Register (FPCR)
- 32-Bit Floating-Point Status Register (FPSR)
- 32-Bit Floating-Point Instruction Address Register (FPIAR)

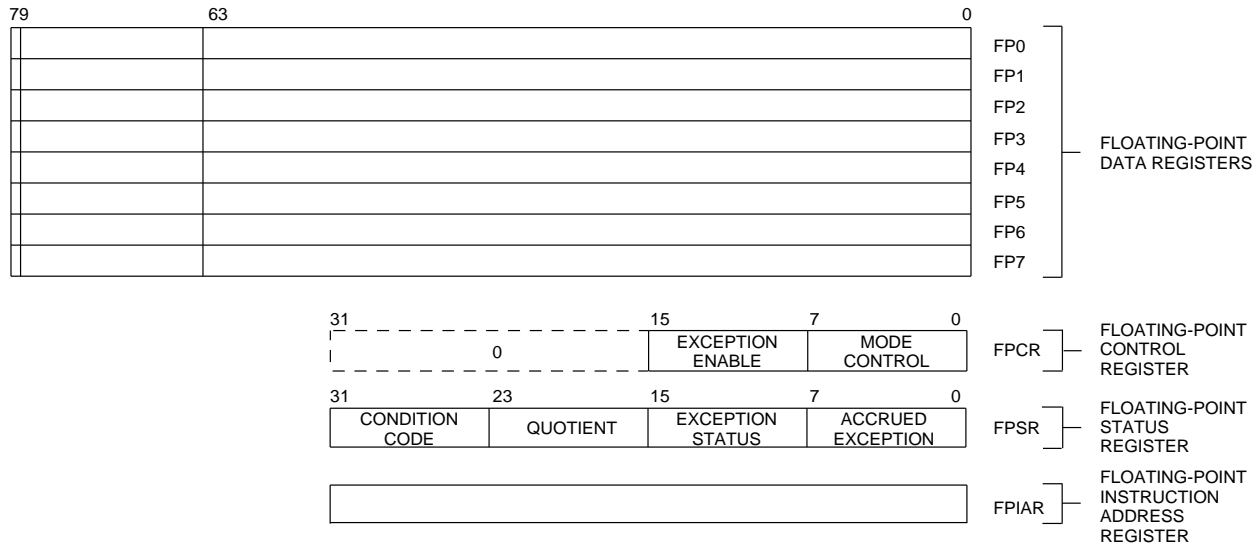


Figure 1-2. M68000 Family Floating-Point Unit User Programming Model

1.2.1 Floating-Point Data Registers (FP7 – FP0)

These floating-point data registers are analogous to the integer data registers for the M68000 family. They always contain extended-precision numbers. All external operands, despite the data format, are converted to extended-precision values before being used in any calculation or being stored in a floating-point data register. A reset or a null-restore operation sets FP7 – FP0 positive, nonsignaling not-a-numbers (NaNs).

1.2.2 Floating-Point Control Register (FPCR)

The FPCR (see Figure 1-3) contains an exception enable (ENABLE) byte and a mode control (MODE) byte. The user can read or write to the FPCR. Motorola reserves bits 31 – 16 for future definition; these bits are always read as zero and are ignored during write operations. The reset function or a restore operation of the null state clears the FPCR. When cleared, this register provides the IEEE 754 Standard for Binary Floating-Point Arithmetic defaults.

1.2.2.1 EXCEPTION ENABLE BYTE. Each bit of the ENABLE byte (see Figure 1-3) corresponds to a floating-point exception class. The user can separately enable traps for each class of floating-point exceptions.

1.2.2.2 MODE CONTROL BYTE. MODE (see Figure 1-3) controls the user-selectable rounding modes and precisions. Zeros in this byte select the IEEE 754 standard defaults. The rounding mode (RND) field specifies how inexact results are rounded, and the rounding precision (PREC) field selects the boundary for rounding the mantissa. Refer to Table 3-21 for encoding information. .

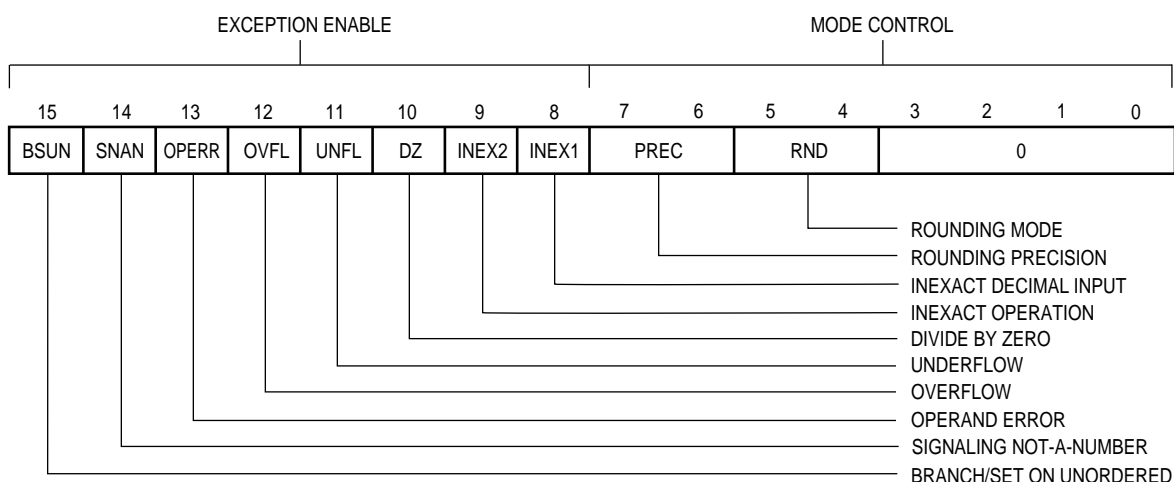


Figure 1-3. Floating-Point Control Register

1.2.3 Floating-Point Status Register (FPSR)

The FPSR (see Figure 1-2) contains a floating-point condition code (FPCC) byte, a floating-point exception status (EXC) byte, a quotient byte, and a floating-point accrued exception (AEXC) byte. The user can read or write to all the bits in the FPSR. Execution of most floating-point instructions modifies this register. The reset function or a restore operation of the null state clears the FPSR.

1.2.3.1 FLOATING-POINT CONDITION CODE BYTE. The FPCC byte, illustrated in Figure 1-4, contains four condition code bits that set after completion of all arithmetic instructions involving the floating-point data registers. The move floating-point data register

to effective address, move multiple floating-point data register, and move system control register instructions do not affect the FPCC. .

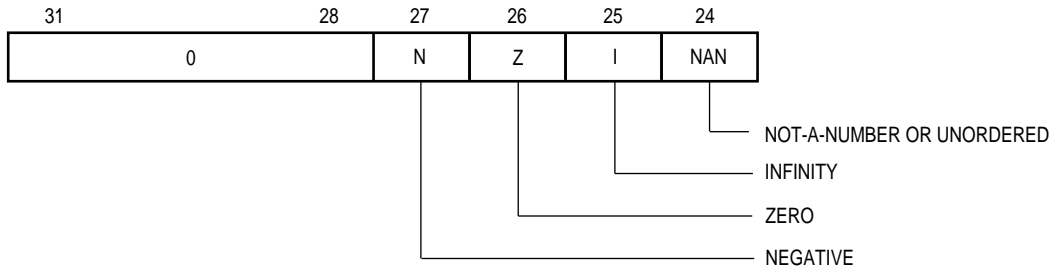


Figure 1-4. FPSR Condition Code Byte

1.2.3.2 QUOTIENT BYTE. The quotient byte contains the seven least significant bits of the unsigned quotient as well as the sign of the entire quotient (see Figure 1-5). The quotient bits can be used in argument reduction for transcendentals and other functions. For example, seven bits are more than enough to figure out the quadrant of a circle in which an operand resides. The quotient bits remain set until the user clears them. .

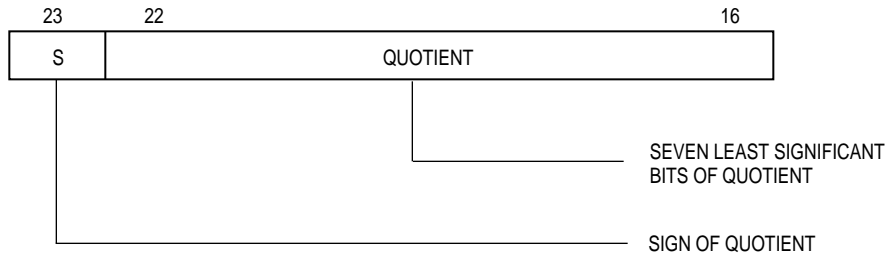


Figure 1-5. FPSR Quotient Code Byte

1.2.3.3 EXCEPTION STATUS BYTE. The EXC byte, illustrated in Figure 1- 6, contains a bit for each floating-point exception that might have occurred during the most recent arithmetic instruction or move operation. This byte is cleared at the start of all operations that generate floating-point exceptions. Operations that do not generate floating-point exceptions do not clear this byte. An exception handler can use this byte to determine which floating-point exception(s) caused a trap. .

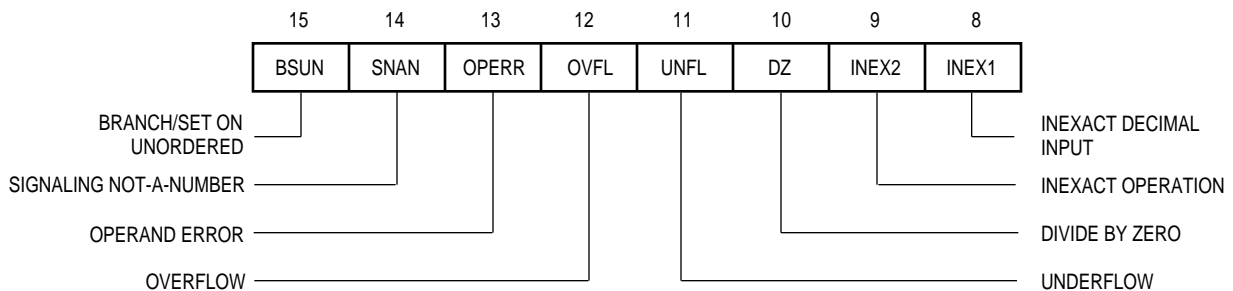


Figure 1-6. FPSR Exception Status Byte

1.2.3.4 ACCRUED EXCEPTION BYTE. The AEXC byte contains five exception bits (see Figure 1-7) required by the IEEE 754 standard for trap disabled operations. These exceptions are logical combinations of the bits in the EXC byte. The AEXC byte contains a history of all floating-point exceptions that have occurred since the user last cleared the AEXC byte. In normal operations, only the user clears this byte by writing to the FPSR; however, a reset or a restore operation of the null state can also clear the AEXC byte.

Many users elect to disable traps for all or part of the floating-point exception classes. The AEXC byte makes it unnecessary to poll the EXC byte after each floating-point instruction. At the end of most operations (FMOVE and FMOVE excluded), the bits in the EXC byte are logically combined to form an AEXC value that is logically ORed into the existing AEXC byte. This operation creates "sticky" floating-point exception bits in the AEXC byte that the user needs to poll only once—i.e., at the end of a series of floating-point operations.

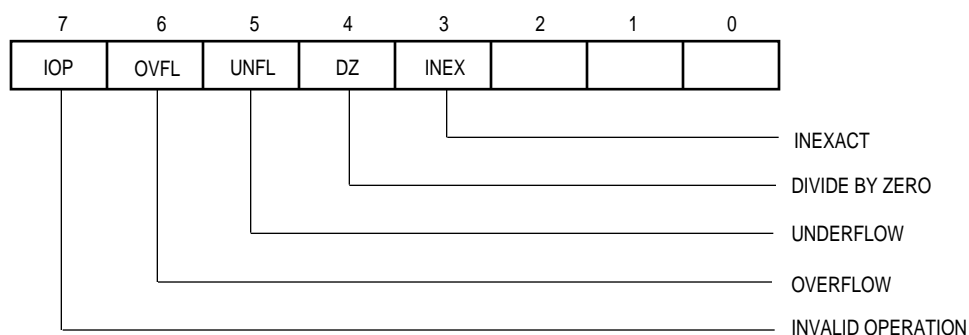


Figure 1-7. FPSR Accrued Exception Byte

Setting or clearing the AEXC bits neither causes nor prevents an exception. The following equations show the comparative relationship between the EXC byte and AEXC byte. Comparing the current value in the AEXC bit with a combination of bits in the EXC byte derives a new value in the corresponding AEXC bit. These equations apply to setting the AEXC bits at the end of each operation affecting the AEXC byte:

| New AEXC Bit | = Old AEXC Bit | V | EXC Bits |
|--------------|----------------|---|------------------------|
| IOP | = IOP | V | (SNAN V OPERR) |
| OVFL | = OVFL | V | (OVFL) |
| UNFL | = UNFL | V | (UNFL L INEX2) |
| DZ | = DZ | V | (DZ) |
| INEX | = INEX | V | (INEX1 V INEX2 V OVFL) |

1.2.4 Floating-Point Instruction Address Register (FPIAR)

The integer unit can be executing instructions while the FPU is simultaneously executing a floating-point instruction. Additionally, the FPU can concurrently execute two floating-point instructions. Because of this nonsequential instruction execution, the PC value stacked by the FPU, in response to a floating-point exception trap, may not point to the offending instruction.

For the subset of the FPU instructions that generate exception traps, the 32-bit FPIAR is loaded with the logical address of the instruction before the processor executes it. The floating-point exception handler can use this address to locate the floating-point instruction that caused an exception. Since the FPU FMOVE to/from the FPCR, FPSR, or FPIAR and FMOVEM instructions cannot generate floating-point exceptions, these instructions do not modify the FPIAR. A reset or a null-restore operation clears the FPIAR.

1.3 SUPERVISOR PROGRAMMING MODEL

System programmers use the supervisor programming model to implement sensitive operating system functions—e.g., I/O control and memory management unit (MMU) subsystems. The following paragraphs briefly describe the registers in the supervisor programming model. They can only be accessed via privileged instructions. Table 1-1 lists the supervisor registers and the processors not related to paged memory management. For information concerning page memory management programming, refer to the device-specific user's manual. Table 1-2 lists the supervisor registers and the processors related to paged memory management.

**Table 1-1. Supervisor Registers
Not Related To Paged Memory Management**

| Registers | Devices | | | | | | | | |
|-----------------|---|-------|------------------|-------|-------|---------|-------|---------|---------|
| | 68000 68008 68HC000 68HC001 68EC000 | 68010 | 68020 68EC020 | CPU32 | 68030 | 68EC030 | 68040 | 68EC040 | 68LC040 |
| AC1, AC0 | | | | | | x | | | |
| ACUSR | | | | | | x | | | |
| CAAR | | | x | | x | x | | | |
| CACR | | | x | | x | x | x | x | x |
| DACR1, DACR0 | | | | | | | | x | |
| DFC | | x | x | x | x | x | x | x | x |
| DTT1, DTT0 | | | | | | | x | | x |
| IACR1, IACR0 | | | | | | | | x | |
| ITT1, ITT0 | | | | | | | x | | x |
| MSP | | | x | | x | x | x | x | x |
| SFC | | x | x | x | x | x | x | x | x |
| SR | x | x | x | x | x | x | x | x | x |
| SSP/ISP | x | x | x | x | x | x | x | x | x |
| TT1, TT0 | | | | | x | | | | |
| VBR | | x | x | x | x | x | x | x | x |

| | |
|---|---|
| AC1, AC0 = Access Control Registers | ITT1, ITT0 = Instruction Transparent Translation Registers |
| ACUSR = Access Control Unit Status Register | MSP = Master Stack Pointer Register |
| CAAR = Cache Address Register | SFC = Source Function Code Register |
| CACR = Cache Control Register | SR = Status Register |
| DACR1, DACR0 = Data Access Control Registers | SSP/ISP = Supervisor and Interrupt Stack Pointer |
| DFC = Destination Function Code Register | TT1, TT0 = Transparent Translation Registers |
| DTT1, DTT0 = Data Transparent Translation Registers | VBR = Vector Base Register |
| IACR1, IACR0 = Instruction Access Control Registers | |

**Table 1-2. Supervisor Registers
Related To Paged Memory Management**

| Registers | Devices | | | |
|------------------|---------|-------|-------|---------|
| | 68851 | 68030 | 68040 | 68LC040 |
| AC | x | | | |
| CAL | x | | | |
| CRP | x | x | | |
| DRP | x | | | |
| PCSR | x | | | |
| PMMUSR, MMUSR | x | x | x | x |
| SCC | x | | | |
| SRP | x | x | x | x |
| TC | x | x | x | x |
| URP | | | x | x |
| VAL | x | | | |

| | | |
|--------|---|--|
| AC | = | Access Control Register |
| CAL | = | Current Access Level Register |
| CRP | = | CPU Root Pointer |
| DRP | = | DMA Root Pointer |
| PCSR | = | PMMU Control Register |
| PMMUSR | = | Paged Memory Management Unit Status Register |
| MMUSR | = | Memory Management Unit Status Register |
| SCC | = | Stack Change Control Register |
| SRP | = | Supervisor Root Pointer Register |
| TC | = | Translation Control Register |
| URP | = | User Root Pointer |
| VAL | = | Valid Access Level Register |

1.3.1 Address Register 7 (A7)

In the supervisor programming model register, A7 refers to the interrupt stack pointer, A7'(ISP) and the master stack pointer, A7" (MSP). The supervisor stack pointer is the active stack pointer (ISP or MSP). For processors that do not support ISP or MSP, the system stack is the system stack pointer (SSP). The ISP and MSP are general-purpose address registers for the supervisor mode. They can be used as software stack pointers, index registers, or base address registers. The ISP and MSP can be used for word and long-word operations.

1.3.2 Status Register

Figure 1-8 illustrates the SR, which stores the processor status and contains the condition codes that reflect the results of a previous operation. In the supervisor mode, software can access the full SR, including the interrupt priority mask and additional control bits. These bits indicate the following states for the processor: one of two trace modes (T1, T0), supervisor or user mode (S), and master or interrupt mode (M). For the MC68000, MC68EC000, MC68008, MC68010, MC68HC000, MC68HC001, and CPU32, only one trace mode

supported, where T0 is always zero, and only one system stack where the M-bit is always zero. I2, I1, and I0 define the interrupt mask level.

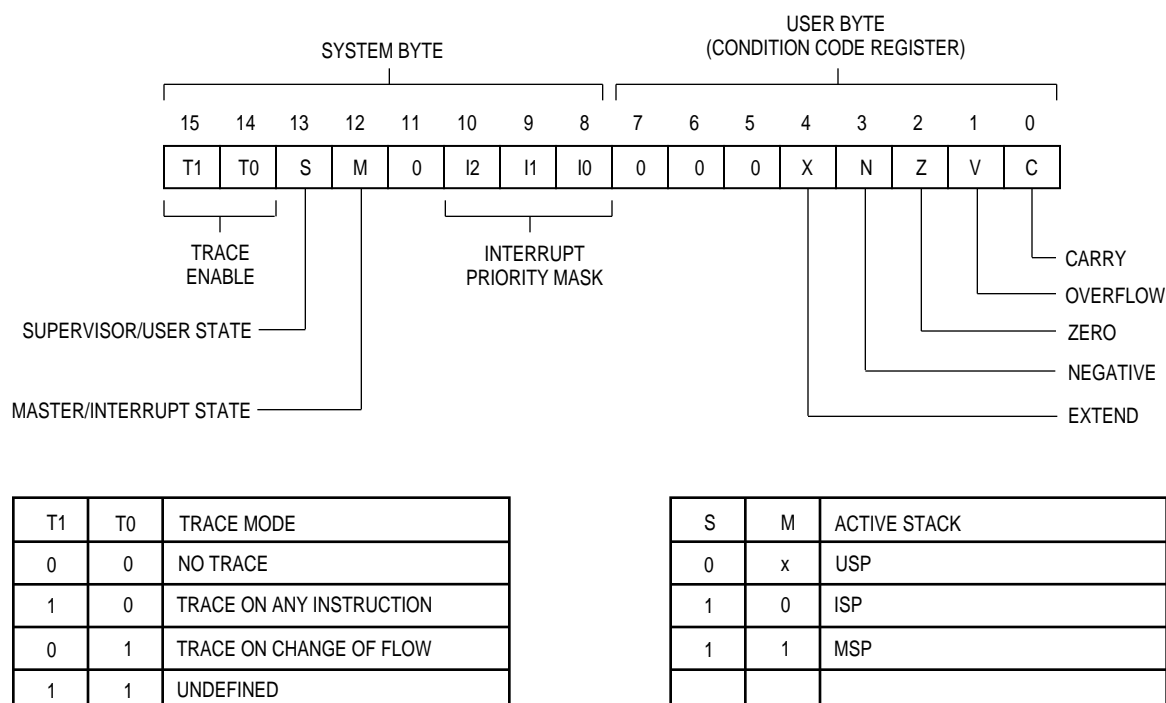


Figure 1-8. Status Register

1.3.3 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in memory. The displacement of an exception vector adds to the value in this register, which accesses the vector table.

1.3.4 Alternate Function Code Registers (SFC and DFC)

The alternate function code registers contain 3-bit function codes. Function codes can be considered extensions of the 32-bit logical address that optionally provides as many as eight 4-Gbyte address spaces. The processor automatically generates function codes to select address spaces for data and programs at the user and supervisor modes. Certain instructions use SFC and DFC to specify the function codes for operations.

1.3.5 Acu Status Register (MC68EC030 only)

The access control unit status register (ACUSR) is a 16-bit register containing the status information returned by execution of the PTEST instruction. The PTEST instruction searches the access control (AC) registers to determine a match for a specified address. A match in either or both of the AC registers sets bit 6 in the ACUSR. All other bits in the ACUSR are undefined and must not be used.

1.3.6 Transparent Translation/access Control Registers

Transparent translation is actually a misnomer since the whole address space transparently translates in an embedded control environment with no on-chip MMU present as well as in processors that have built-in MMUs. For processors that have built-in MMUs, such as the MC68030, MC68040, and MC68LC040, the transparent translation (TT) registers define blocks of logical addresses that are transparently translated to corresponding physical addresses. These registers are independent of the on-chip MMU. For embedded controllers, such as the MC68EC030 and MC68EC040, the access control registers (AC) are similar in function to the TT registers but just named differently. The AC registers, main function are to define blocks of address space that control address space properties such as cachability. The following paragraphs describe these registers.

NOTE

For the paged MMU related supervisor registers, please refer to the appropriate user's manual for specific programming detail.

1.3.6.1 TRANSPARENT TRANSLATION/ACCESS CONTROL REGISTER FIELDS FOR THE M68030. Figure 1-9 illustrates the MC68030 transparent translation/MC68EC030 access control register format.

| | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|-----|-----|--------------|---------|----|---|---------|---|--|
| 31 | | | | | | | | 24 | 23 | 16 | | | | |
| ADDRESS BASE | | | | | | | | ADDRESS MASK | | | | | | |
| E | 0 | 0 | 0 | 0 | CI | R/W | RWM | 0 | FC BASE | | 0 | FC MASK | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 4 | 3 | 2 | 0 | |

Figure 1-9. MC68030 Transparent Translation/MC68EC030 Access Control Register Format

Address Base

This 8-bit field is compared with address bits A31 – A24. Addresses that match in this comparison (and are otherwise eligible) are transparently translated/access controlled.

Address Mask

This 8-bit field contains a mask for the address base field. Setting a bit in this field causes the corresponding bit of the address base field to be ignored. Blocks of memory larger than 16 Mbytes can be transparently translated/accessed controlled by setting some logical address mask bits to ones. The low-order bits of this field normally are set to define contiguous blocks larger than 16 Mbytes, although this is not required.

E—Enable

- 0 = Transparent translation/access control disabled
- 1 = Transparent translation/access control enabled

CI—Cache Inhibit

- 0 = Caching allowed
- 1 = Caching inhibited

R/W—Read/Write

- 0 = Only write accesses permitted
- 1 = Only read accesses permitted

R/WM—Read/Write Mask

- 0 = R/W field used
- 1 = R/W field ignored

FC BASE—Function Code Base

This 3-bit field defines the base function code for accesses to be transparently translated with this register. Addresses with function codes that match the FC BASE field (and are otherwise eligible) are transparently translated.

FC MASK—Function Code Mask

This 3-bit field contains a mask for the FC BASE field. Setting a bit in this field causes the corresponding bit of the FC BASE field to be ignored.

1.3.6.2 TRANSPARENT TRANSLATION/ACCESS CONTROL REGISTER FIELDS FOR THE M68040. Figure 1-10 illustrates the MC68040 and MC68LC040 transparent translation/ MC68EC040 access control register format.

| | | | | | | | | | | | | | | | |
|--------------|---------|----|----|----|----|----|----|--------------|----|---|---|---|---|---|---|
| 31 | | | | | | | 24 | 23 | 16 | | | | | | |
| ADDRESS BASE | | | | | | | | ADDRESS MASK | | | | | | | |
| E | S FIELD | | 0 | 0 | 0 | U1 | U0 | 0 | CM | 0 | 0 | W | 0 | 0 | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Figure 1-10. MC68040 and MC68LC040 Transparent Translation/MC68EC040 Access Control Register Format

Address Base

This 8-bit field is compared with address bits A31 – A24. Addresses that match in this comparison (and are otherwise eligible) are transparently translated/access controlled.

Address Mask

This 8-bit field contains a mask for the address base field. Setting a bit in this field causes the corresponding bit in the address base field to be ignored. Blocks of memory larger than 16 Mbytes can be transparently translated/access controlled by setting some logical address mask bits to ones. The low-order bits of this field normally are set to define contiguous blocks larger than 16 Mbytes, although this not required.

E—Enable

This bit enables and disables transparent translation/access control of the block defined by this register.

0 = Transparent translation/access control disabled

1 = Transparent translation/access control enabled

S—Supervisor/User Mode

This field specifies the use of the FC2 in matching an address.

00 = Match only if FC2 is 0 (user mode access)

01 = Match only if FC2 is 1 (supervisor mode access)

1X = Ignore FC2 when matching

U1, U2—User Page Attributes

The MC68040, MC68E040, MC68LC040 do not interpret these user-defined bits. If an external bus transfer results from the access, U0 and U1 are echoed to the UPA0 and UPA1 signals, respectively.

CM—Cache Mode

This field selects the cache mode and access serialization for a page as follows:

00 = Cachable, Writethrough

01 = Cachable, Copyback

10 = Noncachable, Serialized

11 = Noncachable

W—Write Protect

This bit indicates if the block is write protected. If set, write and read-modify-write accesses are aborted as if the resident bit in a table descriptor were clear.

0 = Read and write accesses permitted

1 = Write accesses not permitted

1.4 INTEGER DATA FORMATS

The operand data formats supported by the integer unit, as listed in Table 1-3, include those supported by the MC68030 plus a new data format (16-byte block) for the MOVE16 instruction. Integer unit operands can reside in registers, memory, or instructions themselves. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation.

Table 1-3. Integer Data Formats

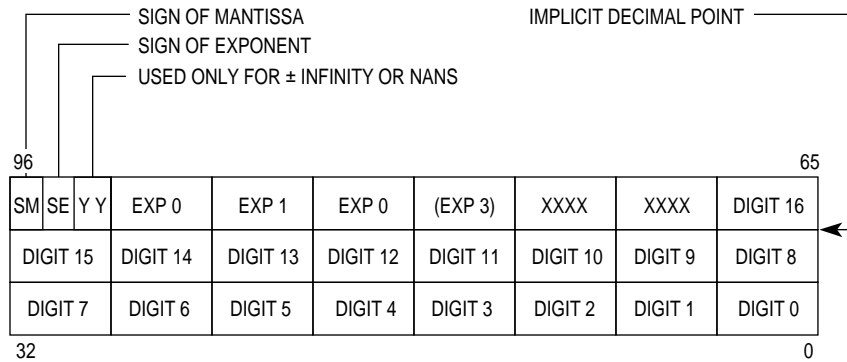
| Operand Data Format | Size | Notes |
|----------------------------|-------------|---|
| Bit | 1 Bit | — |
| Bit Field | 1 – 32 Bits | Field of Consecutive Bit |
| Binary-Coded Decimal (BCD) | 8 Bits | Packed: 2 Digits/Byte; Unpacked: 1 Digit/Byte |
| Byte Integer | 8 Bits | — |
| Word Integer | 16 Bits | — |
| Long-Word Integer | 32 Bits | — |
| Quad-Word Integer | 64 Bits | Any Two Data Registers |
| 16-Byte | 128 Bits | Memory Only, Aligned to 16- Byte Boundary |

1.5 FLOATING-POINT DATA FORMATS

The following paragraphs describe the FPU's operand data formats. The FPU supports seven data formats. There are three signed binary integer formats (byte, word, and long word) that are identical to those supported by the integer unit. The FPU supports the use of the packed decimal real format. The MC68881 and MC68882 support this format in hardware and the processors starting with the MC68040 support it in software. The FPU also supports three binary floating- point formats (single, double, and extended precision) that fully comply with the IEEE 754 standard. All references in this manual to extended-precision format imply the double-extended-precision format defined by the IEEE 754 standard.

1.5.1 Packed Decimal Real Format

Figure 1-11 illustrates the packed decimal real format which is three long words consisting of a 3-digit base 10 exponent and a 17-digit base 10 mantissa. The first two long words, digits 15 – 0, are 64 bits and map directly to bit positions 63 – 0 of the extended-precision real format. There are two separate sign bits, one for the exponent, the other for the mantissa. An extra exponent (EXP3) is defined for overflows that can occur when converting from the extended-precision real format to the packed decimal real format.



NOTE: XXXX indicates "don't care", which is zero when written and ignored when read.

Figure 1-11. Packed Decimal Real Format

1.5.2 Binary Floating-Point Formats

Figure 1-12 illustrates the three binary floating-point data formats. The exponent in the three binary floating-point formats is an unsigned binary integer with an implied bias added to it. When subtracting the bias from the exponent's value, the result represents a signed two's complement power of two. This yields the magnitude of a normalized floating-point number when multiplied by the mantissa. A program can execute a CMP instruction that compares floating-point numbers in memory using biased exponents, despite the absolute magnitude of the exponents.

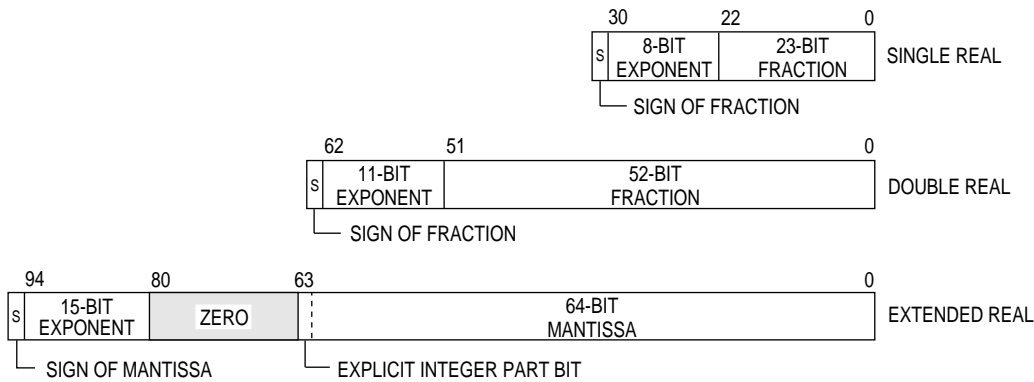


Figure 1-12. Binary Floating-Point Data Formats

Data formats for single- and double-precision numbers differ slightly from those for extended-precision numbers in the representation of the mantissa. For all three precisions, a normalized mantissa is always in the range (1.0...2.0). The extended-precision data format represents the entire mantissa, including the explicit integer part bit. Single- and double-precision data formats represent only a fractional portion of the mantissa (the fraction) and always imply the integer part as one.

The IEEE 754 standard has created the term significand to bridge the difference between mantissa and fraction and to avoid the historical implications of the term mantissa. The IEEE 754 standard defines a significand as the component of a binary floating-point number that includes an explicit or implicit leading bit to the left of the implied binary point. However, this manual uses the term mantissa for extended-precision formats and fraction for single- and double- precision formats instead of the IEEE term significand.

NOTE

This section specifies ranges using traditional set notation with the format "bound...bound" specifying the boundaries of the range. The bracket types enclosing the range define whether the endpoint is inclusive or exclusive. A square bracket indicates inclusive, and a parenthesis indicates exclusive. For example, the range specification "[1.0...2.0]" defines the range of numbers greater than or equal to 1.0 and less than or equal to 2.0. The range specification "(0.0... + inf)" defines the range of numbers greater than 0.0 and less than positive infinity, but not equal to.

1.6 FLOATING-POINT DATA TYPES

Each floating-point data format supports five, unique, floating-point data types: 1) normalized numbers, 2) denormalized numbers, 3) zeros, 4) infinities, and 5) NaNs. Exponent values in each format represent these special data types. The normalized data type never uses the maximum or minimum exponent value for a given format, except the extended-precision format. The packed decimal real data format does not support denormalized numbers.

There is a subtle difference between the definition of an extended- precision number with an exponent equal to zero and a single- or double-precision number with an exponent equal to zero. The zero exponent of a single- or double-precision number denormalizes the number's definition, and the implied integer bit is zero. An extended- precision number with an exponent of zero may have an explicit integer bit equal to one. This results in a normalized number, though the exponent is equal to the minimum value. For simplicity, the following discussion treats all three floating-point formats in the same manner, where an exponent value of zero identifies a denormalized number. However, remember the extended-precision format can deviate from this rule.

1.6.1 Normalized Numbers

Normalized numbers encompass all numbers with exponents laying between the maximum and minimum values. Normalized numbers can be positive or negative. For normalized numbers in single and double precision the implied integer bit is one. In extended precision, the mantissa's MSB, the explicit integer bit, can only be a one (see Figure 1-13); and the exponent can be zero.

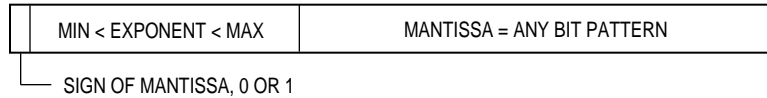


Figure 1-13. Normalized Number Format

1.6.2 Denormalized Numbers

Denormalized numbers represent real values near the underflow threshold. The detection of the underflow for a given data format and operation occurs when the result's exponent is less than or equal to the minimum exponent value. Denormalized numbers can be positive or negative. For denormalized numbers in single and double precision the implied integer bit is a zero. In extended precision, the mantissa's MSB, the explicit integer bit, can only be a zero (see Figure 1-14).

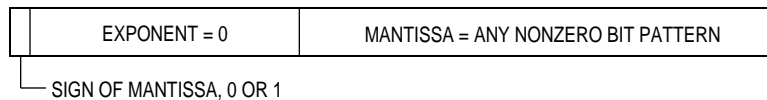


Figure 1-14. Denormalized Number Format

Traditionally, the detection of underflow causes floating-point number systems to perform a "flush-to-zero". This leaves a large gap in the number line between the smallest magnitude normalized number and zero. The IEEE 754 standard implements gradual underflows: the result mantissa is shifted right (denormalized) while the result exponent is incremented until reaching the minimum value. If all the mantissa bits of the result are shifted off to the right during this denormalization, the result becomes zero. Usually a gradual underflow limits the potential underflow damage to no more than a round-off error. This underflow and denormalization description ignores the effects of rounding and the user-selectable rounding modes. Thus, the large gap in the number line created by "flush-to-zero" number systems is filled with representable (denormalized) numbers in the IEEE "gradual underflow" floating-point number system.

Since the extended-precision data format has an explicit integer bit, a number can be formatted with a nonzero exponent, less than the maximum value, and a zero integer bit. The IEEE 754 standard does not define a zero integer bit. Such a number is an unnormalized number. Hardware does not directly support denormalized and unnormalized numbers, but implicitly supports them by trapping them as unimplemented data types, allowing efficient conversion in software.

1.6.3 Zeros

Zeros can be positive or negative and represent the real values + 0.0 and – 0.0 (see Figure 1-15).

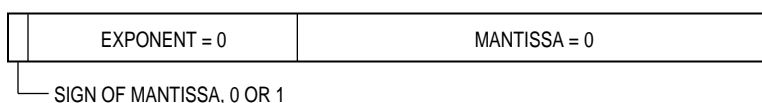


Figure 1-15. Zero Format

1.6.4 Infinities

Infinities can be positive or negative and represent real values that exceed the overflow threshold. A result's exponent greater than or equal to the maximum exponent value indicates the overflow for a given data format and operation. This overflow description ignores the effects of rounding and the user-selectable rounding models. For single- and double-precision infinities the fraction is a zero. For extended-precision infinities, the mantissa's MSB, the explicit integer bit, can be either one or zero (see Figure 1-16).

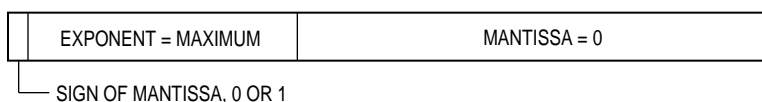


Figure 1-16. Infinity Format

1.6.5 Not-A-Numbers

When created by the FPU, NaNs represent the results of operations having no mathematical interpretation, such as infinity divided by infinity. All operations involving a NaN operand as an input return a NaN result. When created by the user, NaNs can protect against uninitialized variables and arrays or represent user-defined data types. For extended-precision NaNs, the mantissa's MSB, the explicit integer bit, can be either one or zero (see Figure 1-17).

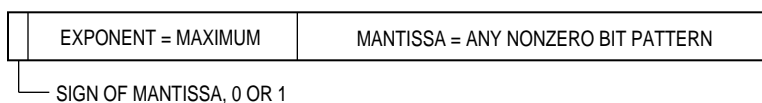


Figure 1-17. Not-A-Number Format

The FPU implements two different types of NaNs identified by the value of the MSB of the mantissa for single- and double-precision, and the MSB of the mantissa minus one for extended-precision. If the bit is set, it is a nonsignaling NaN, otherwise, it is an SNaN. An

SNAN can be used as an escape mechanism for a user-defined, non-IEEE data type. The FPU never creates an SNAN resulting from an operation.

The IEEE specification defines NAN processing used as an input to an operation. A nonsignaling NAN must be returned when using an SNAN as an input and there is a disabled SNAN trap. The FPU does this by using the source SNAN, setting the MSB of the mantissa, and storing the resulting nonsignaling NAN in the destination. Because of the IEEE formats for NANs, the result of setting an SNAN MSB is always a nonsignaling NAN.

When the FPU creates a NAN, the NAN always contains the same bit pattern in the mantissa. All bits of the mantissa are ones for any precision. When the user creates a NAN, any nonzero bit pattern can be stored in the mantissa.

1.6.6 Data Format and Type Summary

Tables 1-4 through 1-6 summarize the data type specifications for single-, double-, and extended-precision data formats. Packed decimal real formats support all data types except denormalized numbers. Table 1-7 summarizes the data types for the packed decimal real format.

Table 1-4. Single-Precision Real Format Summary Data Format

| Data Format | | | | |
|--|---|---|---|---|
| <div style="display: flex; justify-content: space-around; align-items: center;"> 31 30 23 22 0 </div> <table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td style="width: 20px; text-align: center;">s</td> <td style="width: 100px; text-align: center;">e</td> <td style="width: 100px; text-align: center;">f</td> </tr> </table> | | s | e | f |
| s | e | f | | |
| Field Size In Bits | | | | |
| Sign (s) | 1 | | | |
| Biased Exponent (e) | 8 | | | |
| Fraction (f) | 23 | | | |
| Total | 32 | | | |
| Interpretation of Sign | | | | |
| Positive Fraction | s = 0 | | | |
| Negative Fraction | s = 1 | | | |
| Normalized Numbers | | | | |
| Bias of Biased Exponent | +127 (\$7F) | | | |
| Range of Biased Exponent | 0 < e < 255 (\$FF) | | | |
| Range of Fraction | Zero or Nonzero | | | |
| Fraction | 1.f | | | |
| Relation to Representation of Real Numbers | $(-1)^s \times 2^{e-127} \times 1.f$ | | | |
| Denormalized Numbers | | | | |
| Biased Exponent Format Minimum | 0 (\$00) | | | |
| Bias of Biased Exponent | +126 (\$7E) | | | |
| Range of Fraction | Nonzero | | | |
| Fraction | 0.f | | | |
| Relation to Representation of Real Numbers | $(-1)^s \times 2^{-126} \times 0.f$ | | | |
| Signed Zeros | | | | |
| Biased Exponent Format Minimum | 0 (\$00) | | | |
| Fraction | 0.f = 0.0 | | | |
| Signed Infinities | | | | |
| Biased Exponent Format Maximum | 255 (\$FF) | | | |
| Fraction | 0.f = 0.0 | | | |
| NaNs | | | | |
| Sign | Don't Care | | | |
| Biased Exponent Format Maximum | 255 (\$FF) | | | |
| Fraction | Nonzero | | | |
| Representation of Fraction | Nonsignaling Signaling Nonzero Bit Pattern Created by User Fraction When Created by FPCP | | | |
| | 0.1xxxx...xxxx 0.0xxxx...xxxx xxxxx...xxxx 11111...1111 | | | |
| Approximate Ranges | | | | |
| Maximum Positive Normalized | 3.4×10^{38} | | | |
| Minimum Positive Normalized | 1.2×10^{-38} | | | |
| Minimum Positive Denormalized | 1.4×10^{-45} | | | |

Table 1-5. Double-Precision Real Format Summary

| Data Format | |
|---|--|
| <div style="display: flex; justify-content: space-around; align-items: center;"> 63 62 52 51 0 </div> <div style="display: flex; justify-content: space-around; align-items: center; border: 1px solid black; padding: 2px;"> s e f </div> | |
| Field Size (in Bits) | |
| Sign (s) | 1 |
| Biased Exponent (e) | 11 |
| Fraction (f) | 52 |
| Total | 64 |
| Interpretation of Sign | |
| Positive Fraction | s = 0 |
| Negative Fraction | s = 1 |
| Normalized Numbers | |
| Bias of Biased Exponent | +1023 (\$3FF) |
| Range of Biased Exponent | $0 < e < 2047$ (\$7FF) |
| Range of Fraction | Zero or Nonzero |
| Fraction | 1.f |
| Relation to Representation of Real Numbers | $(-1)^s \times 2^{e-1023} \times 1.f$ |
| Denormalized Numbers | |
| Biased Exponent Format Minimum | 0 (\$000) |
| Bias of Biased Exponent | +1022 (\$3FE) |
| Range of Fraction | Nonzero |
| Fraction | 0.f |
| Relation to Representation of Real Numbers | $(-1)^s \times 2^{-1022} \times 0.f$ |
| Signed Zeros | |
| Biased Exponent Format Minimum | 0 (\$00) |
| Fraction (Mantissa/Significand) | 0.f = 0.0 |
| Signed Infinities | |
| Biased Exponent Format Maximum | 2047 (\$7FF) |
| Fraction | 0.f = 0.0 |
| NaNs | |
| Sign | 0 or 1 |
| Biased Exponent Format Maximum | 255 (\$7FF) |
| Fraction | Nonzero |
| Representation of Fraction | 1xxxx...xxxx 0xxxx...xxxx xxxxx...xxxx 11111...1111 |
| Nonsignaling Signaling Nonzero Bit Pattern Created by User Fraction When Created by FPCP | |
| Approximate Ranges | |
| Maximum Positive Normalized | 18×10^{308} |
| Minimum Positive Normalized | 2.2×10^{-308} |
| Minimum Positive Denormalized | 4.9×10^{-324} |

Table 1-6. Extended-Precision Real Format Summary

| Data Format | |
|---|--|
| <div style="display: flex; justify-content: space-around; align-items: center;"> 95 94 80 79 64 63 62 0 </div> <div style="display: flex; justify-content: space-around; align-items: center; border: 1px solid black; width: fit-content; margin: 0 auto;"> s e z i f </div> | |
| Field Size (in Bits) | |
| Sign (s) | 1 |
| Biased Exponent (e) | 15 |
| Zero, Reserved (u) | 16 |
| Explicit Integer Bit (j) | 1 |
| Mantissa (f) | 63 |
| Total | 96 |
| Interpretation of Unused Bits | |
| Input | Don't Care |
| Output | All Zeros |
| Interpretation of Sign | |
| Positive Mantissa | $s = 0$ |
| Negative Mantissa | $s = 1$ |
| Normalized Numbers | |
| Bias of Biased Exponent | +16383 (\$3FFF) |
| Range of Biased Exponent | $0 < e < 32767$ (\$7FFF) |
| Explicit Integer Bit | 1 |
| Range of Mantissa | Zero or Nonzero |
| Mantissa (Explicit Integer Bit and Fraction) | 1.f |
| Relation to Representation of Real Numbers | $(-1)^s \times 2^{e-16383} \times 1.f$ |
| Denormalized Numbers | |
| Biased Exponent Format Minimum | 0 (\$0000) |
| Bias of Biased Exponent | +16383 (\$3FFF) |
| Explicit Integer Bit | 0 |
| Range of Mantissa | Nonzero |
| Mantissa (Explicit Integer Bit and Fraction) | 0.f |
| Relation to Representation of Real Numbers | $(-1)^s \times 2^{-16383} \times 0.f$ |
| Signed Zeros | |
| Biased Exponent Format Minimum | 0 (\$0000) |
| Mantissa (Explicit Integer Bit and Fraction) | 0.0 |
| Signed Infinities | |
| Biased Exponent Format Maximum | 32767 (\$7FFF) |
| Explicit Integer Bit | Don't Care |
| Mantissa (Explicit Integer Bit and Fraction) | x.000...0000 |

Table 1-6. Extended-Precision Real Format Summary (Continued)

| NANs | |
|-------------------------------------|-------------------------|
| Sign | Don't Care |
| Explicit Integer Bit | Don't Care |
| Biased Exponent Format Maximum | 32767 (\$7FFF) |
| Mantissa | Nonzero |
| Representation of Fraction | |
| Nonsignaling | x.1xxxx...xxxx |
| Signaling | x.0xxxx...xxxx |
| Nonzero Bit Pattern Created by User | x.xxxxx...xxxx |
| Fraction When Created by FPCP | 1.11111...1111 |
| Approximate Ranges | |
| Maximum Positive Normalized | 1.2×10^{4932} |
| Minimum Positive Normalized | 1.7×10^{-4932} |
| Minimum Positive Denormalized | 3.7×10^{4951} |

Table 1-7. Packed Decimal Real Format Summary

| Data Type | SM | SE | Y | Y | 3-Digit Exponent | 1-Digit Integer | 16-Digit Fraction |
|-----------|-----|-----|---|---|------------------|-----------------|---------------------|
| ±Infinity | 0/1 | 1 | 1 | 1 | \$FFF | \$XXXX | \$00...00 |
| ±NAN | 0/1 | 1 | 1 | 1 | \$FFF | \$XXXX | Nonzero |
| ±SNAN | 0/1 | 1 | 1 | 1 | \$FFF | \$XXXX | Nonzero |
| +Zero | 0 | 0/1 | X | X | \$000–\$999 | \$XXX0 | \$00...00 |
| –Zero | 1 | 0/1 | X | X | \$000–\$999 | \$XXX0 | \$00...00 |
| +In-Range | 0 | 0/1 | X | X | \$000–\$999 | \$XXX0–\$XXX9 | \$00...01–\$99...99 |
| –In-Range | 1 | 0/1 | X | X | \$000–\$999 | \$XXX0–\$XXX9 | \$00...01–\$99...99 |

A packed decimal real data format with the SE and both Y bits set, an exponent of \$FFF, and a nonzero 16-bit decimal fraction is a NAN. When the FPU uses this format, the fraction of the NAN is moved bit- by-bit into the extended-precision mantissa of a floating-point data register. The exponent of the register is set to signify a NAN, and no conversion occurs. The MSB of the most significant digit in the decimal fraction (the MSB of digit 15) is a don't care, as in extended-precision NANs, and the MSB of minus one of digit 15 is the SNAN bit. If the NAN bit is a zero, then it is an SNAN.

If a non-decimal digit (\$A – \$F) appears in the exponent of a zero, the number is a true zero. The FPU does not detect non-decimal digits in the exponent, integer, or fraction digits of an in-range packed decimal real data format. These non-decimal digits are converted to binary in the same manner as decimal digits; however, the result is probably useless although it is repeatable. Since an in-range number cannot overflow or underflow when converted to extended precision, conversion from the packed decimal real data format always produces normalized extended-precision numbers.

1.7 ORGANIZATION OF DATA IN REGISTERS

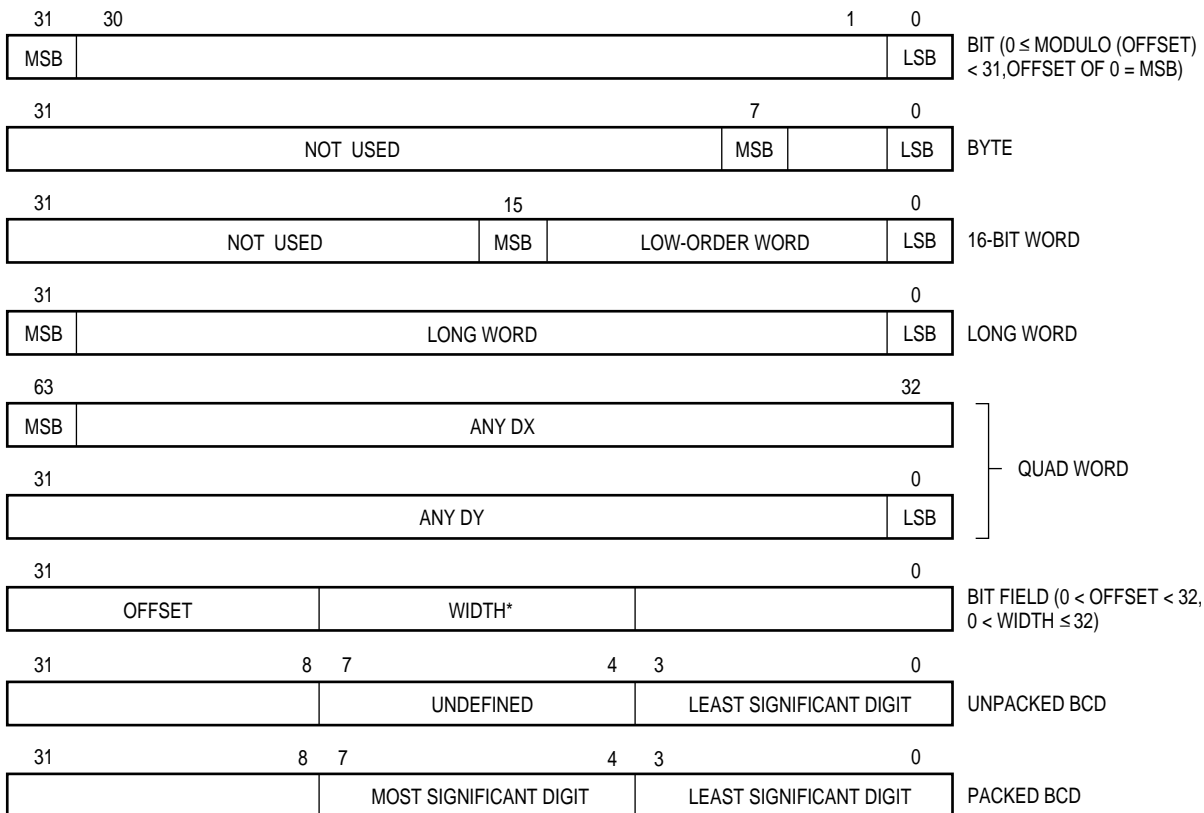
The following paragraphs describe data organization within the data, address, and control registers.

1.7.1 Organization of Integer Data Formats in Registers

Each integer data register is 32 bits wide. Byte and word operands occupy the lower 8- and 16-bit portions of integer data registers, respectively. Long- word operands occupy the entire 32 bits of integer data registers. A data register that is either a source or destination operand only uses or changes the appropriate lower 8 or 16 bits (in byte or word operations, respectively). The remaining high-order portion does not change and goes unused. The address of the least significant bit (LSB) of a long-word integer is zero, and the MSB is 31. For bit fields, the address of the MSB is zero, and the LSB is the width of the register minus one (the offset). If the width of the register plus the offset is greater than 32, the bit field wraps around within the register. Figure 1-18 illustrates the organization of various data formats in the data registers.

An example of a quad word is the product of a 32-bit multiply or the quotient of a 32-bit divide operation (signed and unsigned). Quad words may be organized in any two integer data registers without restrictions on order or pairing. There are no explicit instructions for the management of this data format, although the MOVEM instruction can be used to move a quad word into or out of registers.

Binary-coded decimal (BCD) data represents decimal numbers in binary form. Although there are many BCD codes, the BCD instructions of the M68000 family support two formats, packed and unpacked. In these formats, the LSBs consist of a binary number having the numeric value of the corresponding decimal number. In the unpacked BCD format, a byte defines one decimal number that has four LSBs containing the binary value and four undefined MSBs. Each byte of the packed BCD format contains two decimal numbers; the least significant four bits contain the least significant decimal number and the most significant four bits contain the most significant decimal number.



* IF WIDTH + OFFSET > 32, BIT FIELD WRAPS AROUND WITHIN THE REGISTER.

Figure 1-18. Organization of Integer Data Formats in Data Registers

Because address registers and stack pointers are 32 bits wide, address registers cannot be used for byte-size operands. When an address register is a source operand, either the low-order word or the entire long-word operand is used, depending upon the operation size. When an address register is the destination operand, the entire register becomes affected, despite the operation size. If the source operand is a word size, it is sign-extended to 32 bits and then used in the operation to an address register destination. Address registers are primarily for addresses and address computation support. The instruction set includes instructions that add to, compare, and move the contents of address registers. Figure 1-19 illustrates the organization of addresses in address registers.

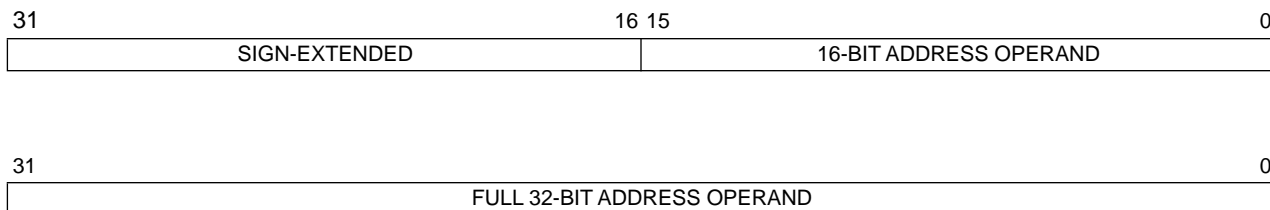


Figure 1-19. Organization of Integer Data Formats in Address Registers

Control registers vary in size according to function. Some control registers have undefined bits reserved for future definition by Motorola. Those particular bits read as zeros and must be written as zeros for future compatibility.

All operations to the SR and CCR are word-size operations. For all CCR operations, the upper byte is read as all zeros and is ignored when written, despite privilege mode. The alternate function code registers, supervisor function code (SFC) and data function code (DFC), are 32-bit registers with only bits 0P2 implemented. These bits contain the address space values for the read or write operands of MOVES, PFLUSH, and PTEST instructions. Values transfer to and from the SFC and DFC by using the MOVEC instruction. These are long-word transfers; the upper 29 bits are read as zeros and are ignored when written.

1.7.2 Organization of Integer Data Formats in Memory

The byte-addressable organization of memory allows lower addresses to correspond to higher order bytes. The address N of a long-word data item corresponds to the address of the highest order word's MSB. The lower order word is located at address N + 2, leaving the LSB at address N + 3 (see Figure 1-20). Organization of data formats in memory is consistent with the M68000 family data organization. The lowest address (nearest \$00000000) is the location of the MSB, with each successive LSB located at the next address (N + 1, N + 2, etc.). The highest address (nearest \$FFFFFFF) is the location of the LSB.

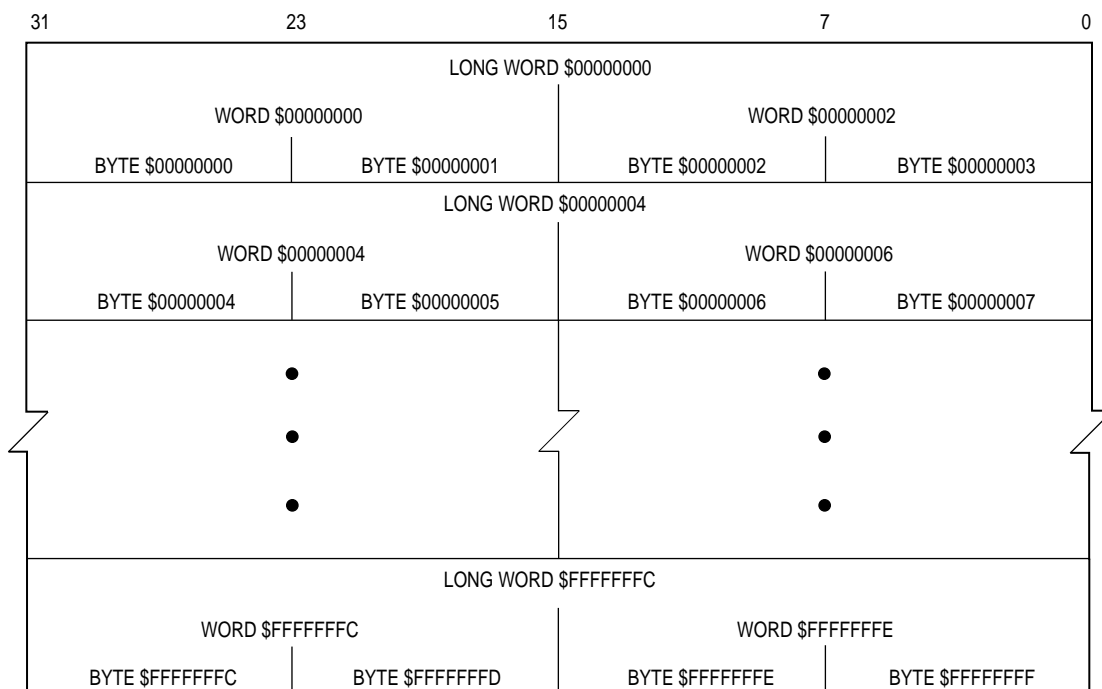


Figure 1-20. Memory Operand Addressing

Figure 1-21 illustrates the organization of IU data formats in memory. A base address that selects one byte in memory, the base byte, specifies a bit number that selects one bit, the bit operand, in the base byte. The MSB of the byte is seven.

The following conditions specify a bit field operand:

1. A base address that selects one byte in memory.
2. A bit field offset that shows the leftmost (base) bit of the bit field in relation to the MSB of the base byte.
3. A bit field width that determines how many bits to the right of the base bit are in the bit field.

The MSB of the base byte is bit field offset 0; the LSB of the base byte is bit field offset 7; and the LSB of the previous byte in memory is bit field offset – 1. Bit field offsets may have values between 2 – 31 to 231 – 1, and bit field widths may range from 1 to 32 bits.

A 16-byte block operand, supported by the MOVE16 instruction, has a block of 16 bytes, aligned to a 16-byte boundary. An address that can point to any byte in the block specifies this operand.

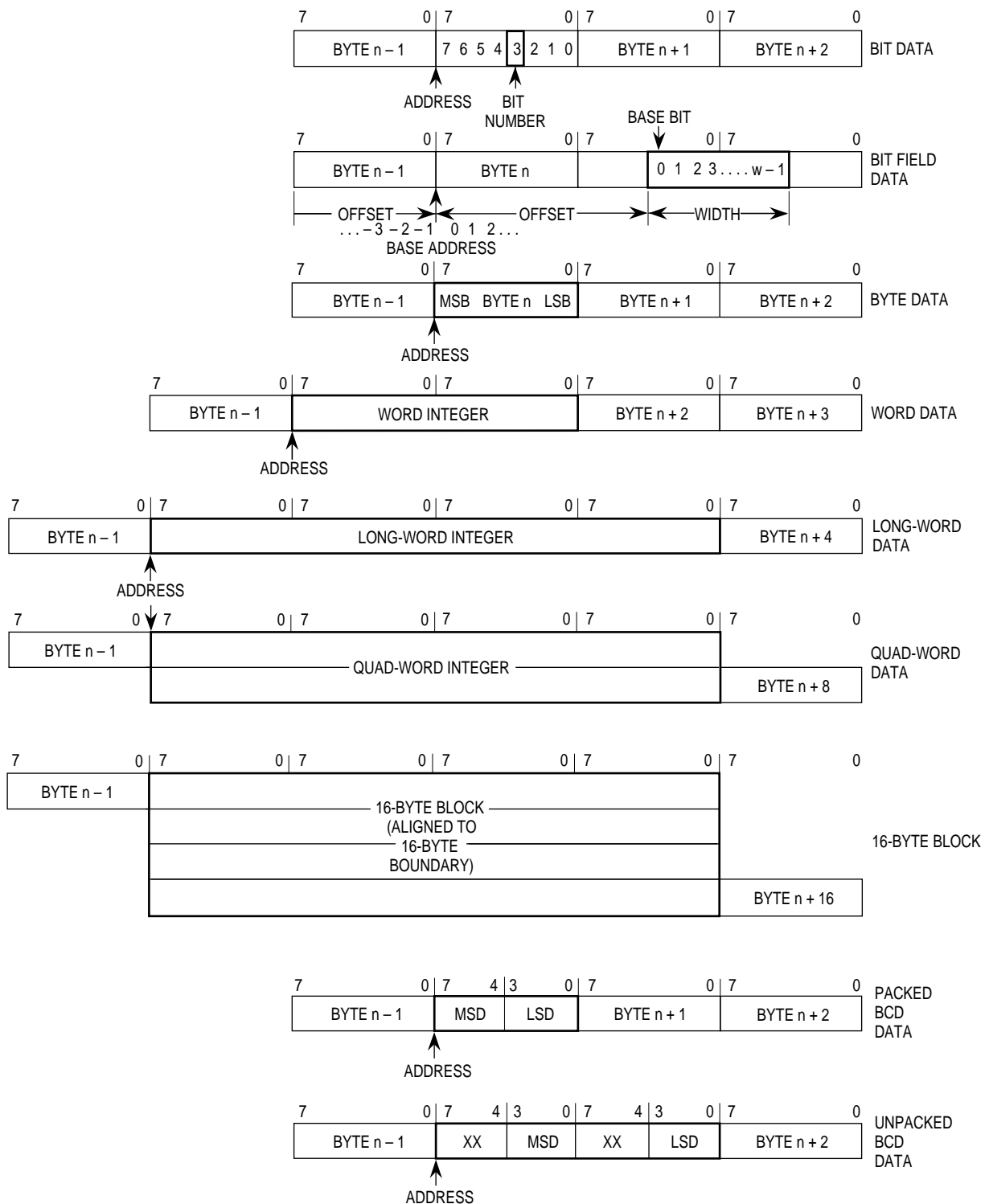


Figure 1-21. Memory Organization for Integer Operands

1.7.3 Organization of Fpu Data Formats in Registers and Memory

The eight, 80-bit floating-point data registers are analogous to the integer data registers and are completely general purpose (i.e., any instruction may use any register). The MC68040 supports only some data formats and types in hardware. Table 1-8 lists the data formats supported by the MC68040.

Table 1-8. MC68040 FPU Data Formats and Data Types

| Number Types | Data Formats | | | | | | |
|--------------|-----------------------|-----------------------|-------------------------|---------------------|--------------|--------------|-------------------|
| | Single-Precision Real | Double-Precision Real | Extended-Precision Real | Packed-Decimal Real | Byte Integer | Word Integer | Long-Word Integer |
| Normalized | * | * | * | † | * | * | * |
| Zero | * | * | * | † | * | * | * |
| Infinity | * | * | * | † | | | |
| NAN | * | * | * | † | | | |
| Denormalized | † | † | † | † | | | |
| Unnormalized | | | † | † | | | |

NOTES:

* = Data Format/Type Supported by On-Chip MC68040 FPU Hardware

† = Data Format/Type Supported by Software (MC68040FPSP)

Figure 1-22 illustrates the floating-point data format for the single-, double-, and extended-precision binary real data organization in memory.

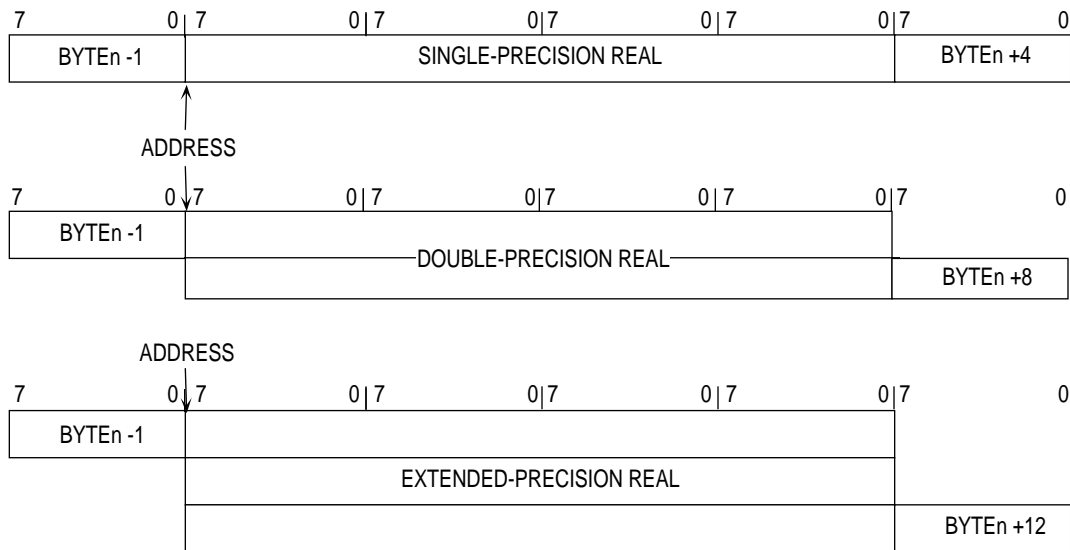


Figure 1-22. Organization of FPU Data Formats in Memory

SECTION 2

ADDRESSING CAPABILITIES

Most operations take a source operand and destination operand, compute them, and store the result in the destination location. Single-operand operations take a destination operand, compute it, and store the result in the destination location. External microprocessor references to memory are either program references that refer to program space or data references that refer to data space. They access either instruction words or operands (data items) for an instruction. Program space is the section of memory that contains the program instructions and any immediate data operands residing in the instruction stream. Data space is the section of memory that contains the program data. Data items in the instruction stream can be accessed with the program counter relative addressing modes; these accesses classify as program references.

2.1 INSTRUCTION FORMAT

M68000 family instructions consist of at least one word; some have as many as 11 words. Figure 2-1 illustrates the general composition of an instruction. The first word of the instruction, called the simple effective address operation word, specifies the length of the instruction, the effective addressing mode, and the operation to be performed. The remaining words, called brief and full extension words, further specify the instruction and operands. These words can be floating-point command words, conditional predicates, immediate operands, extensions to the effective addressing mode specified in the simple effective address operation word, branch displacements, bit number or bit field specifications, special register specifications, trap operands, pack/unpack constants, or argument counts.

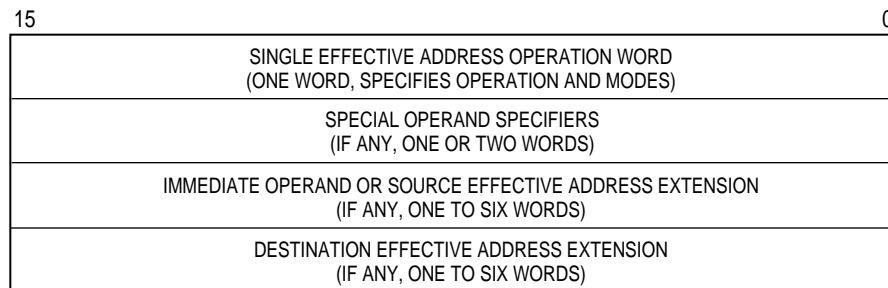


Figure 2-1. Instruction Word General Format

An instruction specifies the function to be performed with an operation code and defines the location of every operand. Instructions specify an operand location by register specification, the instruction's register field holds the register's number; by effective address, the instruction's effective address field contains addressing mode information; or by implicit reference, the definition of the instruction implies the use of specific registers.

The single effective address operation word format is the basic instruction word (see Figure 2-2). The encoding of the mode field selects the addressing mode. The register field contains the general register number or a value that selects the addressing mode when the mode field contains opcode 111. Some indexed or indirect addressing modes use a combination of the simple effective address operation word followed by a brief extension word. Other indexed or indirect addressing modes consist of the simple effective address operation word and a full extension word. The longest instruction is a MOVE instruction with a full extension word for both the source and destination effective addresses and eight other extension words. It also contains 32-bit base displacements and 32-bit outer displacements for both source and destination addresses. Figure 2-2 illustrates the three formats used in an instruction word; Table 2-1 lists the field definitions for these three formats.

SINGLE EFFECTIVE ADDRESS OPERATION WORD FORMAT

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| X | X | X | X | X | X | X | X | X | X | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

BRIEF EXTENSION WORD FORMAT

| | | | | | | | | | | | | | | | |
|-----|----------|----|----|-----|-------|---|--------------|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| D/A | REGISTER | | | W/L | SCALE | 0 | DISPLACEMENT | | | | | | | | |

FULL EXTENSION WORD FORMAT

| | | | | | | | | | | | | | | | |
|---------------------------------------|----------|----|----|-----|-------|---|----|----|---------|---|---|---|------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| D/A | REGISTER | | | W/L | SCALE | 1 | BS | IS | BD SIZE | | | 0 | I/IS | | |
| BASE DISPLACEMENT (0, 1, OR 2 WORDS) | | | | | | | | | | | | | | | |
| OUTER DISPLACEMENT (0, 1, OR 2 WORDS) | | | | | | | | | | | | | | | |

Figure 2-2. Instruction Word Specification Formats

Table 2-1. Instruction Word Format Field Definitions

| Field | Definition |
|--------------------|---|
| Instruction | |
| Mode | Addressing Mode |
| Register | General Register Number |
| Extensions | |
| D/A | Index Register Type 0 = Dn 1 = An |
| W/L | Word/Long-Word Index Size 0 = Sign-Extended Word 1 = Long Word |
| Scale | Scale Factor 00 = 1 01 = 2 10 = 4 11 = 8 |
| BS | Base Register Suppress 0 = Base Register Added 1 = Base Register Suppressed |
| IS | Index Suppress 0 = Evaluate and Add Index Operand 1 = Suppress Index Operand |
| BD SIZE | Base Displacement Size 00 = Reserved 01 = Null Displacement 10 = Word Displacement 11 = Long Displacement |
| I/IS | Index/Indirect Selection Indirect and Indexing Operand Determined in Conjunction with Bit 6, Index Suppress |

For effective addresses that use a full extension word format, the index suppress (IS) bit and the index/indirect selection (I/IS) field determine the type of indexing and indirect action. Table 2-2 lists the index and indirect operations corresponding to all combinations of IS and I/IS values.

Table 2-2. IS-I/IS Memory Indirect Action Encodings

| IS | Index/Indirect | Operation |
|----|----------------|---|
| 0 | 000 | No Memory Indirect Action |
| 0 | 001 | Indirect Preindexed with Null Outer Displacement |
| 0 | 010 | Indirect Preindexed with Word Outer Displacement |
| 0 | 011 | Indirect Preindexed with Long Outer Displacement |
| 0 | 100 | Reserved |
| 0 | 101 | Indirect Postindexed with Null Outer Displacement |
| 0 | 110 | Indirect Postindexed with Word Outer Displacement |
| 0 | 111 | Indirect Postindexed with Long Outer Displacement |
| 1 | 000 | No Memory Indirect Action |
| 1 | 001 | Memory Indirect with Null Outer Displacement |
| 1 | 010 | Memory Indirect with Word Outer Displacement |
| 1 | 011 | Memory Indirect with Long Outer Displacement |
| 1 | 100–111 | Reserved |

2.2 EFFECTIVE ADDRESSING MODES

Besides the operation code, which specifies the function to be performed, an instruction defines the location of every operand for the function. Instructions specify an operand location in one of three ways. A register field within an instruction can specify the register to be used; an instruction's effective address field can contain addressing mode information; or the instruction's definition can imply the use of a specific register. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used. **Section 1 Introduction** contains detailed register descriptions.

An instruction's addressing mode specifies the value of an operand, a register that contains the operand, or how to derive the effective address of an operand in memory. Each addressing mode has an assembler syntax. Some instructions imply the addressing mode for an operand. These instructions include the appropriate fields for operands that use only one addressing mode.

2.2.1 Data Register Direct Mode

In the data register direct mode, the effective address field specifies the data register containing the operand.

GENERATION: EA = Dn
 ASSEMBLER SYNTAX: Dn
 EA MODE FIELD: 000
 EA REGISTER FIELD: REG. NO.
 NUMBER OF EXTENSION WORDS: 0



2.2.2 Address Register Direct Mode

In the address register direct mode, the effective address field specifies the address register containing the operand.

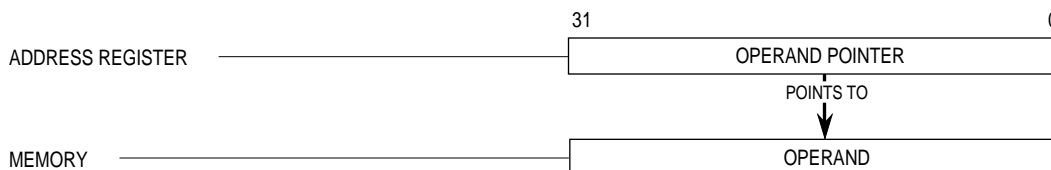
GENERATION: EA = An
 ASSEMBLER SYNTAX: An
 EA MODE FIELD: 001
 EA REGISTER FIELD: REG. NO.
 NUMBER OF EXTENSION WORDS: 0



2.2.3 Address Register Indirect Mode

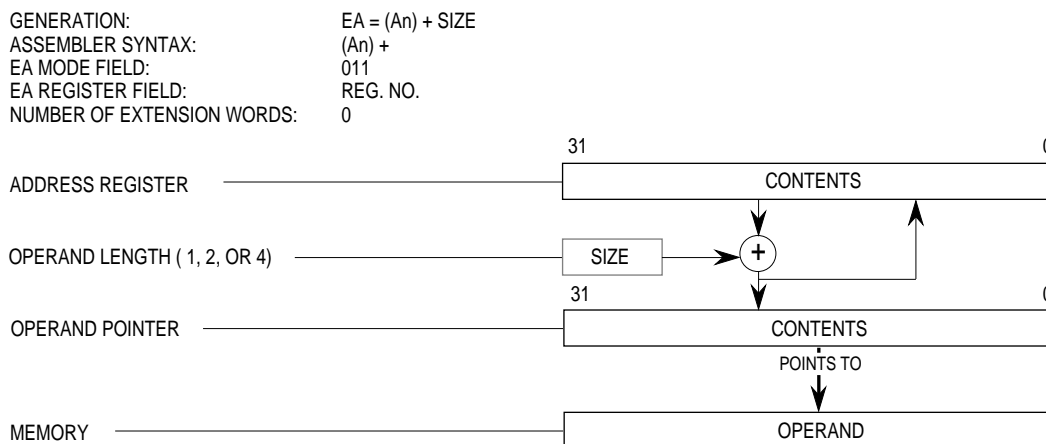
In the address register indirect mode, the operand is in memory. The effective address field specifies the address register containing the address of the operand in memory.

GENERATION: EA = (An)
 ASSEMBLER SYNTAX: (An)
 EA MODE FIELD: 010
 EA REGISTER FIELD: REG. NO.
 NUMBER OF EXTENSION WORDS: 0



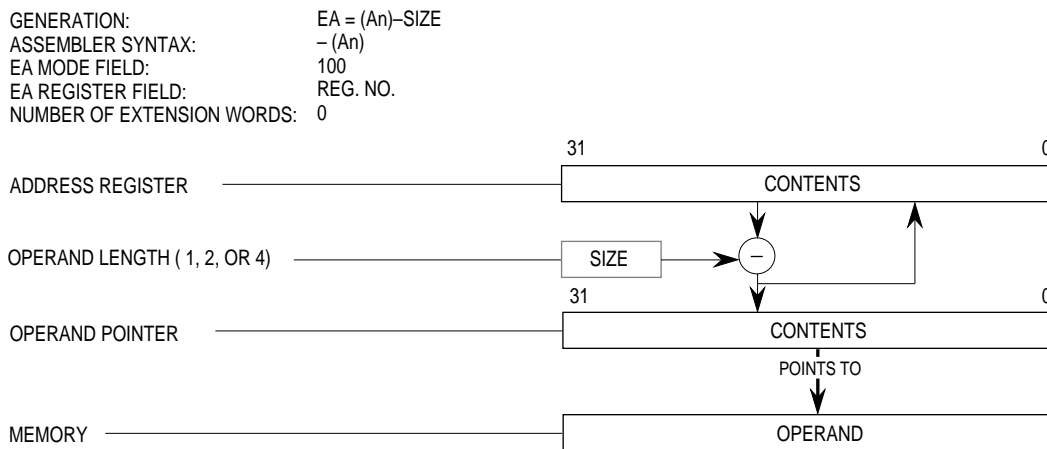
2.2.4 Address Register Indirect with Postincrement Mode

In the address register indirect with postincrement mode, the operand is in memory. The effective address field specifies the address register containing the address of the operand in memory. After the operand address is used, it is incremented by one, two, or four depending on the size of the operand: byte, word, or long word, respectively. Coprocessors may support incrementing for any operand size, up to 255 bytes. If the address register is the stack pointer and the operand size is byte, the address is incremented by two to keep the stack pointer aligned to a word boundary.



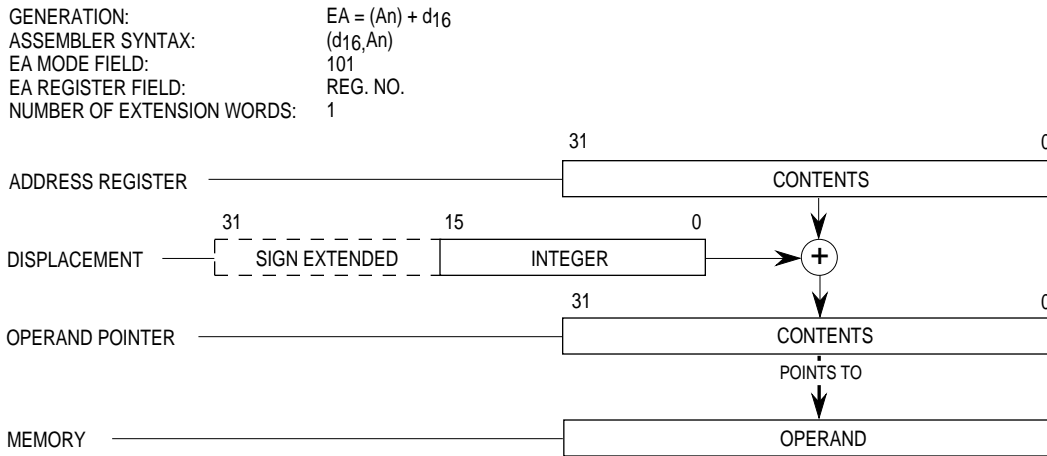
2.2.5 Address Register Indirect with Predecrement Mode

In the address register indirect with predecrement mode, the operand is in memory. The effective address field specifies the address register containing the address of the operand in memory. Before the operand address is used, it is decremented by one, two, or four depending on the operand size: byte, word, or long word, respectively. Coprocessors may support decrementing for any operand size up to 255 bytes. If the address register is the stack pointer and the operand size is byte, the address is decremented by two to keep the stack pointer aligned to a word boundary.



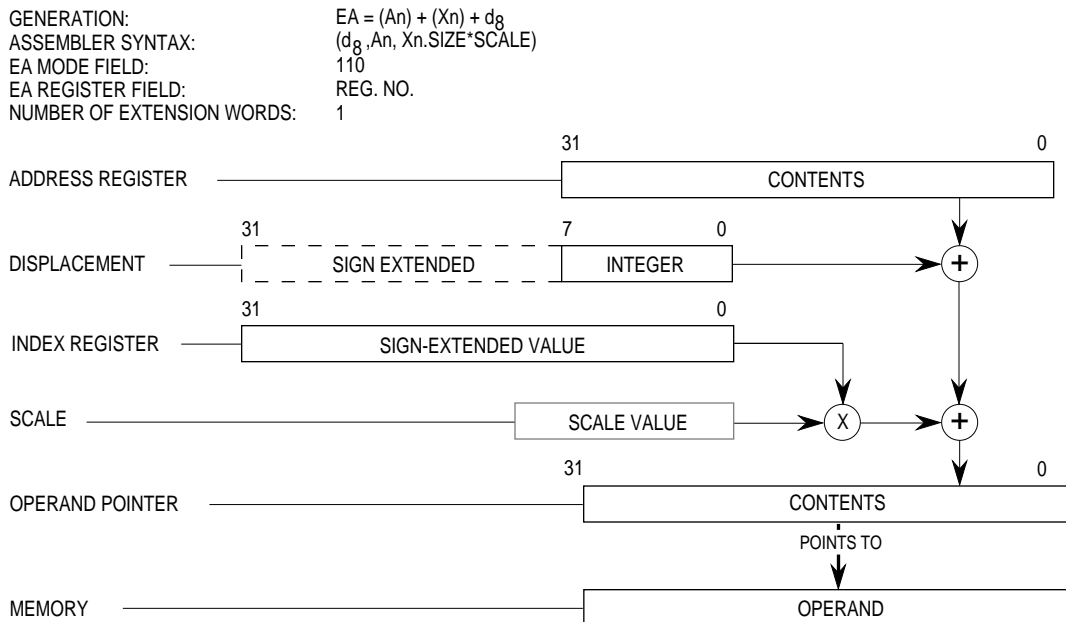
2.2.6 Address Register Indirect with Displacement Mode

In the address register indirect with displacement mode, the operand is in memory. The sum of the address in the address register, which the effective address specifies, plus the sign-extended 16-bit displacement integer in the extension word is the operand's address in memory. Displacements are always sign-extended to 32 bits prior to being used in effective address calculations.



2.2.7 Address Register Indirect with Index (8-Bit Displacement) Mode

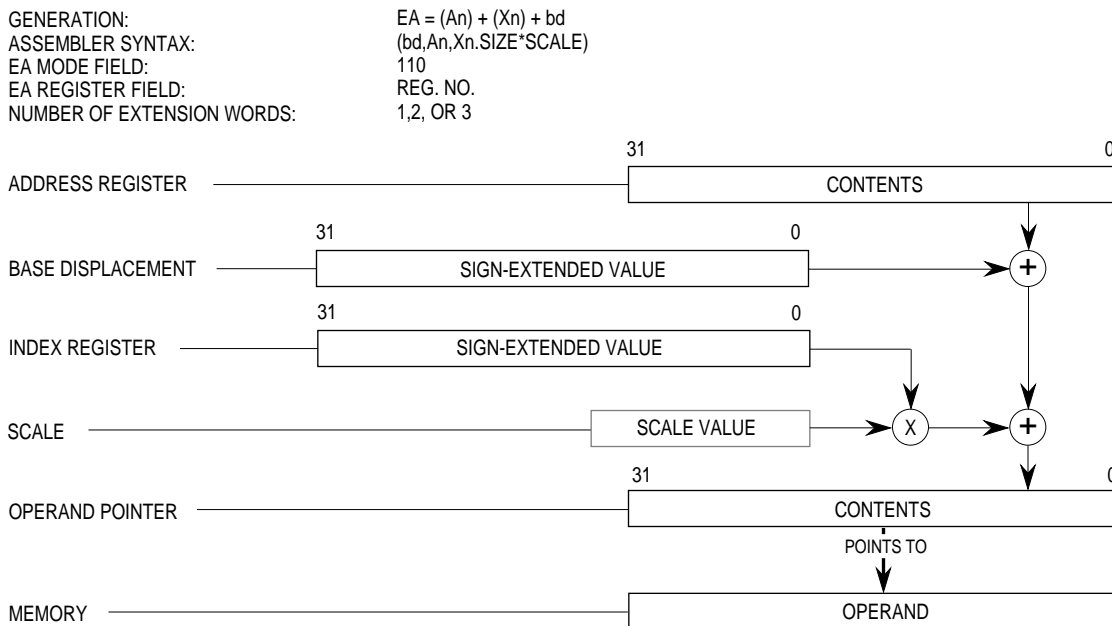
This addressing mode requires one extension word that contains an index register indicator and an 8-bit displacement. The index register indicator includes size and scale information. In this mode, the operand is in memory. The operand's address is the sum of the address register's contents; the sign-extended displacement value in the extension word's low-order eight bits; and the index register's sign-extended contents (possibly scaled). The user must specify the address register, the displacement, and the index register in this mode.



2.2.8 Address Register Indirect with Index (Base Displacement) Mode

This addressing mode requires an index register indicator and an optional 16- or 32-bit sign-extended base displacement. The index register indicator includes size and scaling information. The operand is in memory. The operand's address is the sum of the contents of the address register, the base displacement, and the scaled contents of the sign-extended index register.

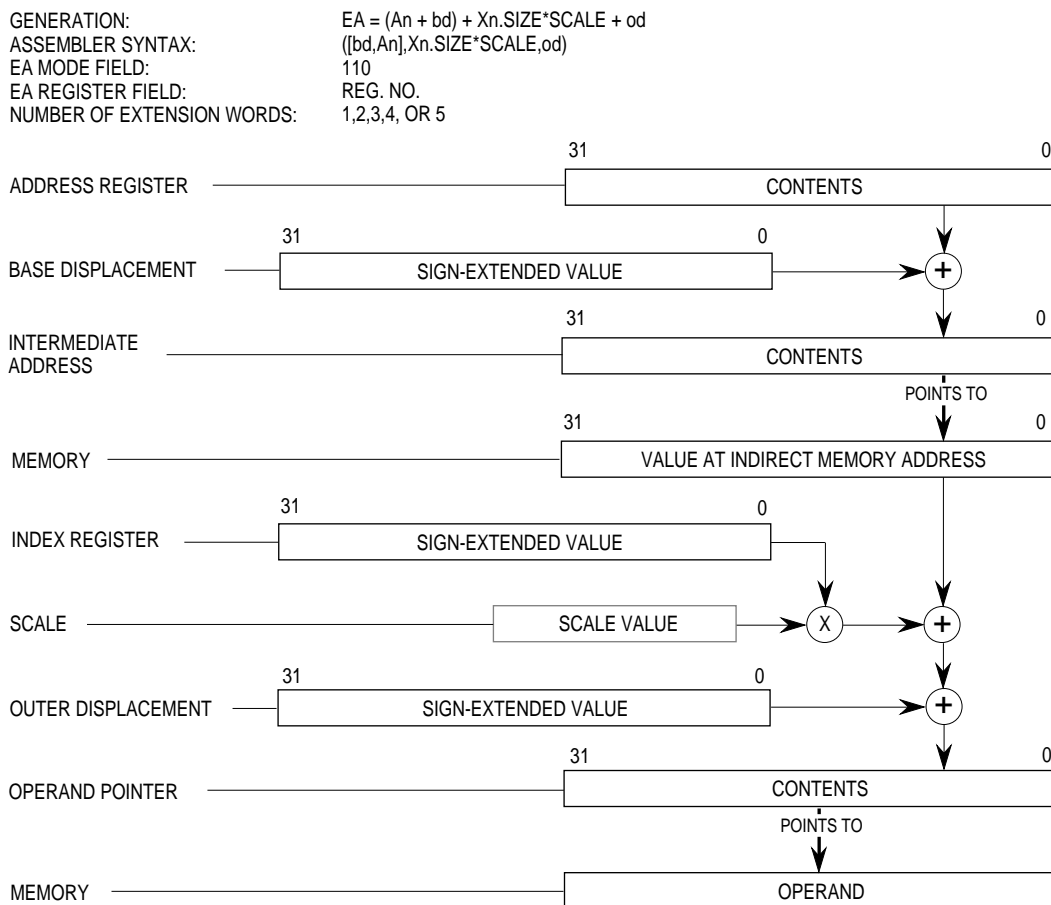
In this mode, the address register, the index register, and the displacement are all optional. The effective address is zero if there is no specification. This mode provides a data register indirect address when there is no specific address register and the index register is a data register.



2.2.9 Memory Indirect Postindexed Mode

In this mode, both the operand and its address are in memory. The processor calculates an intermediate indirect memory address using a base address register and base displacement. The processor accesses a long word at this address and adds the index operand ($Xn.SIZE * SCALE$) and the outer displacement to yield the effective address. Both displacements and the index register contents are sign-extended to 32 bits.

In the syntax for this mode, brackets enclose the values used to calculate the intermediate memory address. All four user-specified values are optional. Both the base and outer displacements may be null, word, or long word. When omitting a displacement or suppressing an element, its value is zero in the effective address calculation.

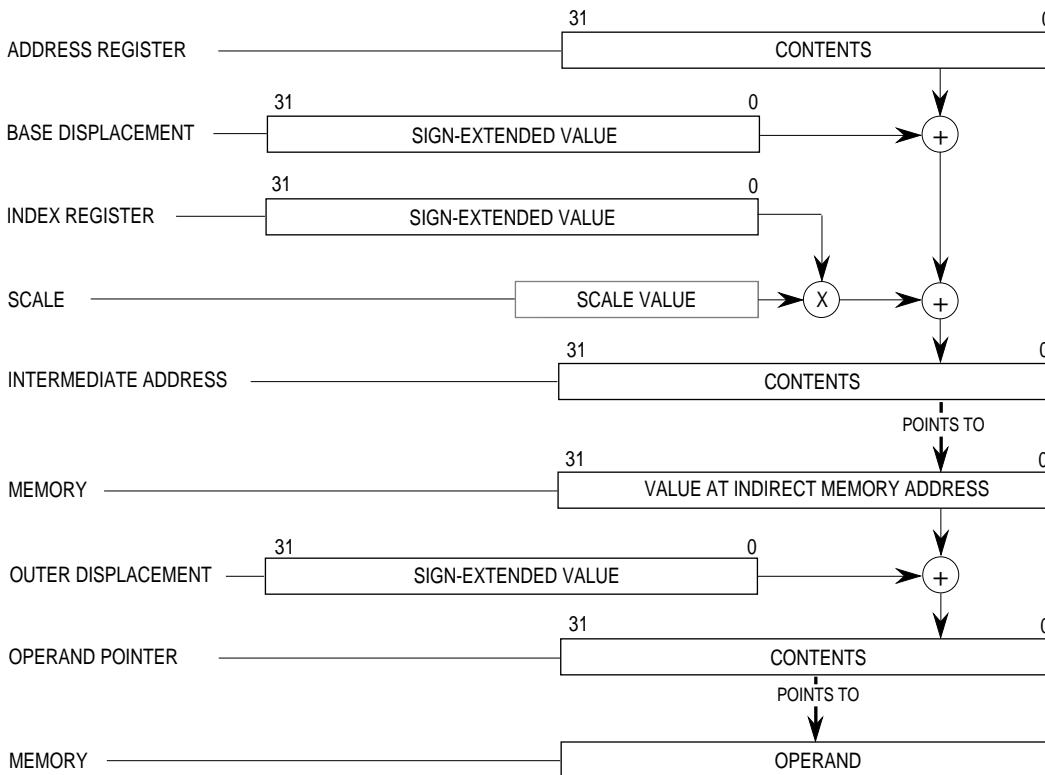


2.2.10 Memory Indirect Preindexed Mode

In this mode, both the operand and its address are in memory. The processor calculates an intermediate indirect memory address using a base address register, a base displacement, and the index operand ($Xn.SIZE*SCALE$). The processor accesses a long word at this address and adds the outer displacement to yield the effective address. Both displacements and the index register contents are sign-extended to 32 bits.

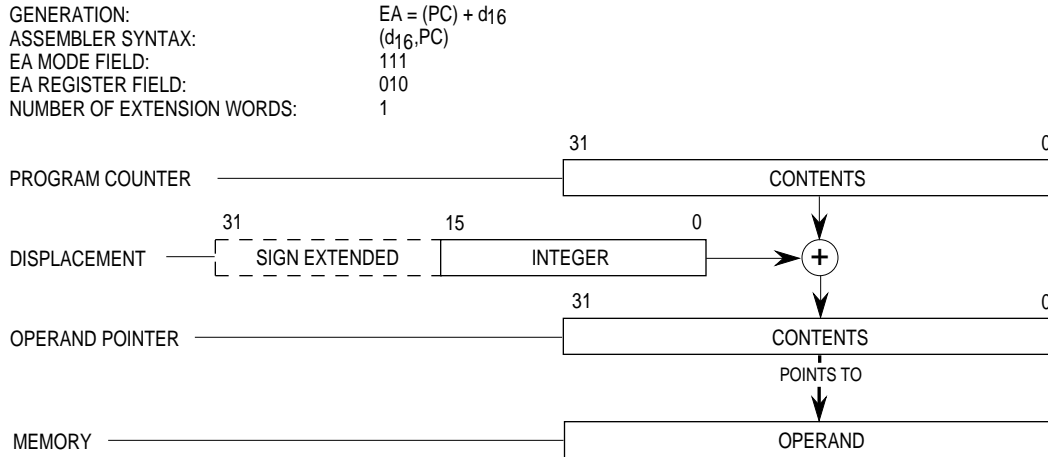
In the syntax for this mode, brackets enclose the values used to calculate the intermediate memory address. All four user-specified values are optional. Both the base and outer displacements may be null, word, or long word. When omitting a displacement or suppressing an element, its value is zero in the effective address calculation.

GENERATION: EA = (bd + An) + Xn.SIZE*SCALE + od
 ASSEMBLER SYNTAX: ([bd, An, Xn.SIZE*SCALE], od)
 EA MODE FIELD: 110
 EA REGISTER FIELD: REG. NO.
 NUMBER OF EXTENSION WORDS: 1,2,3,4, OR 5



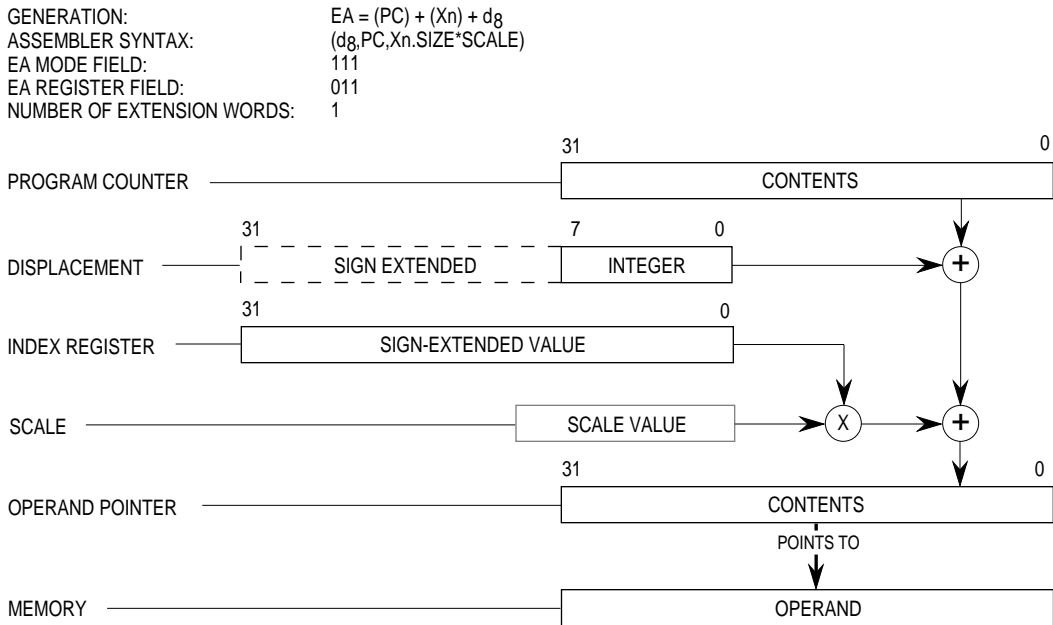
2.2.11 Program Counter Indirect with Displacement Mode

In this mode, the operand is in memory. The address of the operand is the sum of the address in the program counter (PC) and the sign-extended 16-bit displacement integer in the extension word. The value in the PC is the address of the extension word. This is a program reference allowed only for reads.



2.2.12 Program Counter Indirect with Index (8-Bit Displacement) Mode

This mode is similar to the mode described in **2.2.7 Address Register Indirect with Index (8-Bit Displacement) Mode**, except the PC is the base register. The operand is in memory. The operand's address is the sum of the address in the PC, the sign-extended displacement integer in the extension word's lower eight bits, and the sized, scaled, and sign-extended index operand. The value in the PC is the address of the extension word. This is a program reference allowed only for reads. The user must include the displacement, the PC, and the index register when specifying this addressing mode.

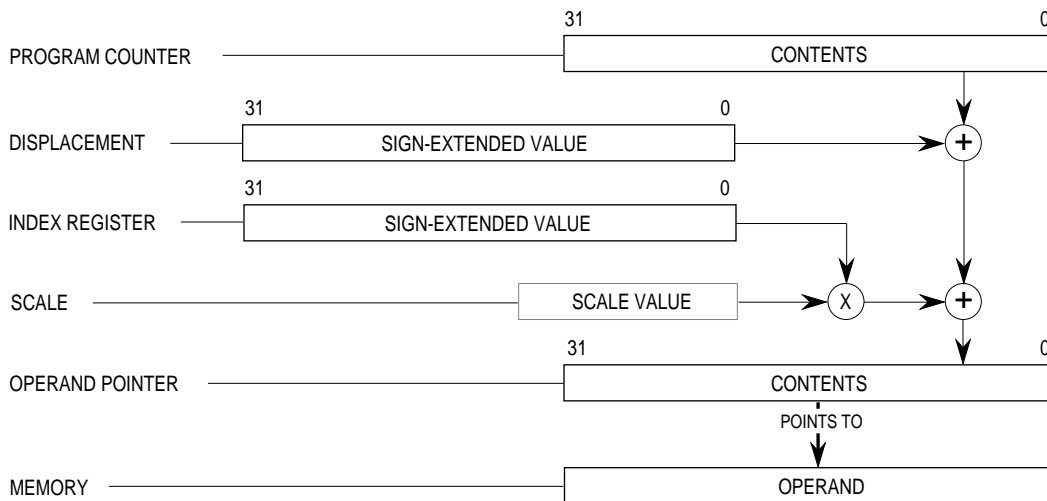


2.2.13 Program Counter Indirect with Index (Base Displacement) Mode

This mode is similar to the mode described in **2.2.8 Address Register Indirect with Index (Base Displacement) Mode**, except the PC is the base register. It requires an index register indicator and an optional 16- or 32-bit sign-extended base displacement. The operand is in memory. The operand's address is the sum of the contents of the PC, the base displacement, and the scaled contents of the sign-extended index register. The value of the PC is the address of the first extension word. This is a program reference allowed only for reads.

In this mode, the PC, the displacement, and the index register are optional. The user must supply the assembler notation ZPC (a zero value PC) to show that the PC is not used. This allows the user to access the program space without using the PC in calculating the effective address. The user can access the program space with a data register indirect access by placing ZPC in the instruction and specifying a data register as the index register.

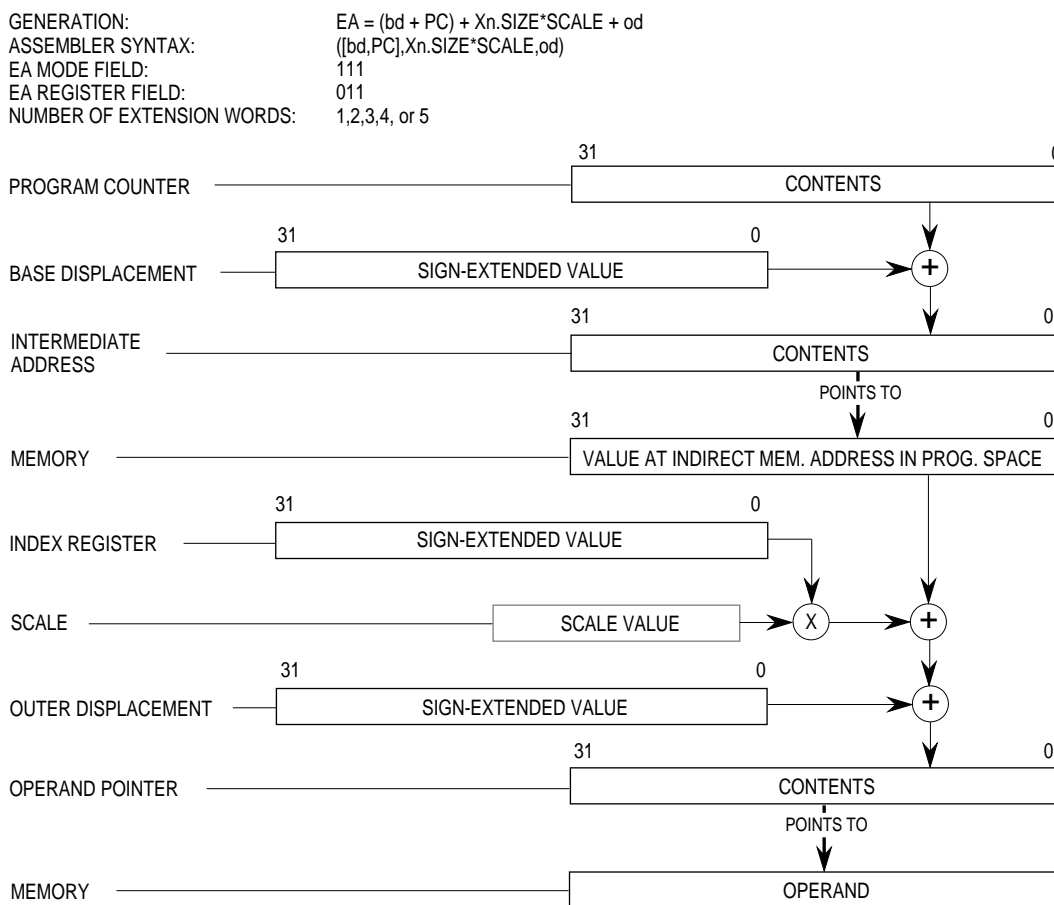
GENERATION: EA = (PC) + (Xn) + bd
 ASSEMBLER SYNTAX: (bd, PC, Xn, SIZE*SCALE)
 EA MODE FIELD: 111
 EA REGISTER FIELD: 011
 NUMBER OF EXTENSION WORDS: 1,2, OR 3



2.2.14 Program Counter Memory Indirect Postindexed Mode

This mode is similar to the mode described in **2.2.9 Memory Indirect Postindexed Mode**, but the PC is the base register. Both the operand and operand address are in memory. The processor calculates an intermediate indirect memory address by adding a base displacement to the PC contents. The processor accesses a long word at that address and adds the scaled contents of the index register and the optional outer displacement to yield the effective address. The value of the PC used in the calculation is the address of the first extension word. This is a program reference allowed only for reads.

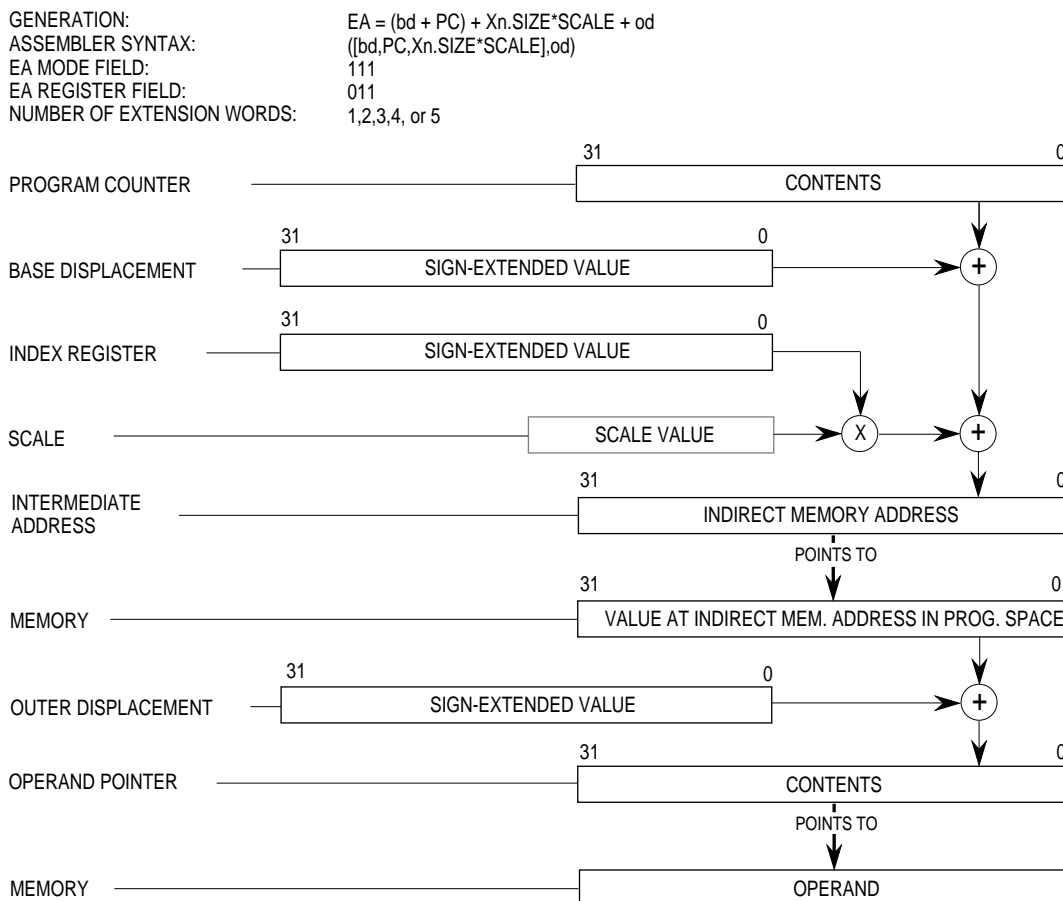
In the syntax for this mode, brackets enclose the values used to calculate the intermediate memory address. All four user-specified values are optional. The user must supply the assembler notation ZPC (a zero value PC) to show the PC is not used. This allows the user to access the program space without using the PC in calculating the effective address. Both the base and outer displacements may be null, word, or long word. When omitting a displacement or suppressing an element, its value is zero in the effective address calculation.



2.2.15 Program Counter Memory Indirect Preindexed Mode

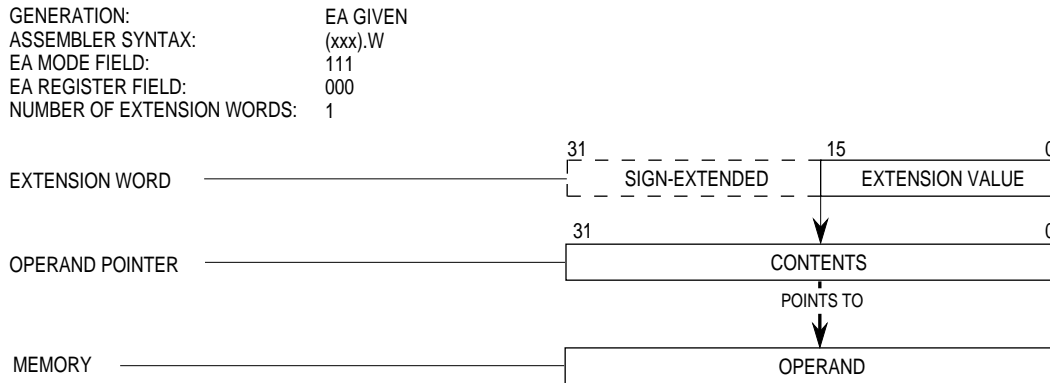
This mode is similar to the mode described in **2.2.10 Memory Indirect Preindexed Mode**, but the PC is the base register. Both the operand and operand address are in memory. The processor calculates an intermediate indirect memory address by adding the PC contents, a base displacement, and the scaled contents of an index register. The processor accesses a long word at immediate indirect memory address and adds the optional outer displacement to yield the effective address. The value of the PC is the address of the first extension word. This is a program reference allowed only for reads.

In the syntax for this mode, brackets enclose the values used to calculate the intermediate memory address. All four user-specified values are optional. The user must supply the assembler notation ZPC showing that the PC is not used. This allows the user to access the program space without using the PC in calculating the effective address. Both the base and outer displacements may be null, word, or long word. When omitting a displacement or suppressing an element, its value is zero in the effective address calculation.



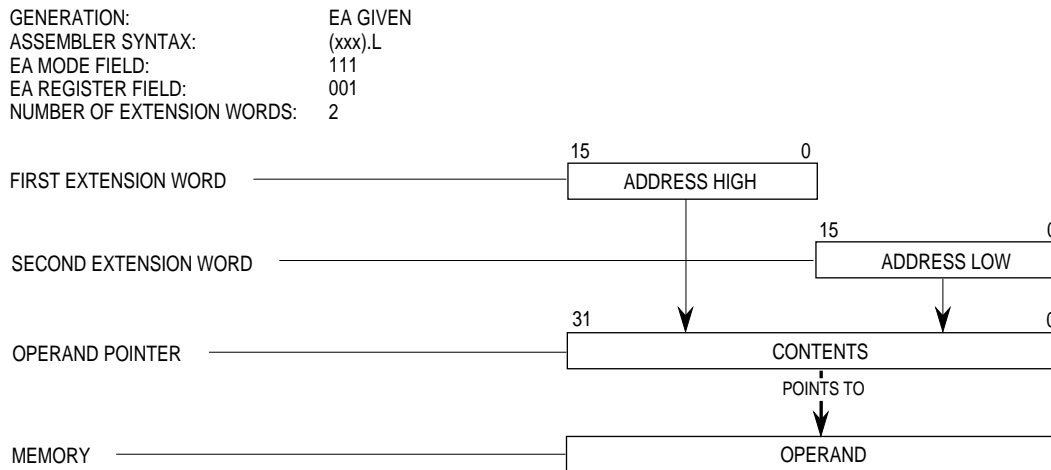
2.2.16 Absolute Short Addressing Mode

In this addressing mode, the operand is in memory, and the address of the operand is in the extension word. The 16-bit address is sign-extended to 32 bits before it is used. .



2.2.17 Absolute Long Addressing Mode

In this addressing mode, the operand is in memory, and the operand's address occupies the two extension words following the instruction word in memory. The first extension word contains the high-order part of the address; the second contains the low-order part of the address. .



2.2.18 Immediate Data

In this addressing mode, the operand is in one or two extension words. Table 2-3 lists the location of the operand within the instruction word format. The immediate data format is as follows:

| | |
|----------------------------|--|
| GENERATION: | OPERAND GIVEN |
| ASSEMBLER SYNTAX: | #<xxx> |
| EA MODE FIELD: | 111 |
| EA REGISTER FIELD: | 100 |
| NUMBER OF EXTENSION WORDS: | 1,2,4, OR 6, EXCEPT FOR PACKED DECIMAL REAL OPERANDS |

Table 2-3. Immediate Operand Location

| Operation Length | Location |
|---------------------|--|
| Byte | Low-order byte of the extension word. |
| Word | The entire extension word. |
| Long Word | High-order word of the operand is in the first extension word; the low-order word is in the second extension word. |
| Single-Precision | In two extension words. |
| Double-Precision | In four extension words. |
| Extended-Precision | In six extension words. |
| Packed-Decimal Real | In six extension words. |

2.3 EFFECTIVE ADDRESSING MODE SUMMARY

Effective addressing modes are grouped according to the use of the mode. Data addressing modes refer to data operands. Memory addressing modes refer to memory operands. Alterable addressing modes refer to alterable (writable) operands. Control addressing modes refer to memory operands without an associated size.

These categories sometimes combine to form new categories that are more restrictive. Two combined classifications are alterable memory (addressing modes that are both alterable and memory addresses) and data alterable (addressing modes that are both alterable and data). Table 2-4 lists a summary of effective addressing modes and their categories.

Table 2-4. Effective Addressing Modes and Categories

| Addressing Modes | Syntax | Mode Field | Reg. Field | Data | Memory | Control | Alterable |
|--|-------------------------|------------|------------|------|--------|---------|-----------|
| Register Direct | | | | | | | |
| Data | Dn | 000 | reg. no. | X | — | — | X |
| Address | An | 001 | reg. no. | — | — | — | X |
| Register Indirect | | | | | | | |
| Address | (An) | 010 | reg. no. | X | X | X | X |
| Address with Postincrement | (An)+ | 011 | reg. no. | X | X | — | X |
| Address with Predecrement | -(An) | 100 | reg. no. | X | X | — | X |
| Address with Displacement | (d ₁₆ ,An) | 101 | reg. no. | X | X | X | X |
| Address Register Indirect with Index | | | | | | | |
| 8-Bit Displacement | (d ₈ ,An,Xn) | 110 | reg. no. | X | X | X | X |
| Base Displacement | (bd,An,Xn) | 110 | reg. no. | X | X | X | X |
| Memory Indirect | | | | | | | |
| Postindexed | ([bd,An],Xn,od) | 110 | reg. no. | X | X | X | X |
| Preindexed | ([bd,An,Xn],od) | 110 | reg. no. | X | X | X | X |
| Program Counter Indirect with Displacement | (d ₁₆ ,PC) | 111 | 010 | X | X | X | — |
| Program Counter Indirect with Index | | | | | | | |
| 8-Bit Displacement | (d ₈ ,PC,Xn) | 111 | 011 | X | X | X | — |
| Base Displacement | (bd,PC,Xn) | 111 | 011 | X | X | X | — |
| Program Counter Memory Indirect | | | | | | | |
| Postindexed | ([bd,PC],Xn,od) | 111 | 011 | X | X | X | X |
| Preindexed | ([bd,PC,Xn],od) | 111 | 011 | X | X | X | X |
| Absolute Data Addressing | | | | | | | |
| Short | (xxx).W | 111 | 000 | X | X | X | — |
| Long | (xxx).L | 111 | 000 | X | X | X | — |
| Immediate | #<xxx> | 111 | 100 | X | X | — | — |

2.4 BRIEF EXTENSION WORD FORMAT COMPATIBILITY

Programs can be easily transported from one member of the M68000 family to another in an upward-compatible fashion. The user object code of each early member of the family, which is upward compatible with newer members, can be executed on the newer microprocessor without change. Brief extension word formats are encoded with information that allows the CPU32, MC68020, MC68030, and MC68040 to distinguish the basic M68000 family architecture's new address extensions. Figure 2-3 illustrates these brief extension word formats. The encoding for SCALE used by the CPU32, MC68020, MC68030, and MC68040 is a compatible extension of the M68000 family architecture. A value of zero for SCALE is the same encoding for both extension words. Software that uses this encoding is compatible with all processors in the M68000 family. Both brief extension word formats do not contain the other values of SCALE. Software can be easily migrated in an upward-compatible direction, with downward support only for nonscaled addressing. If the MC68000 were to execute an instruction that encoded a scaling factor, the scaling factor would be ignored and would not access the desired memory address. The earlier microprocessors do not recognize the brief extension word formats implemented by newer processors. Although they can detect illegal instructions, they do not decode invalid encodings of the brief extension word formats as exceptions.

| | | | | | | | | | | | | | | | |
|-----|----------|----|----|-----|----|---|---|----------------------|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| D/A | REGISTER | | | W/L | 0 | 0 | 0 | DISPLACEMENT INTEGER | | | | | | | |

(a) MC68000, MC68008, and MC68010

| | | | | | | | | | | | | | | | |
|-----|----------|----|----|-----|-------|---|----------------------|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| D/A | REGISTER | | | W/L | SCALE | 0 | DISPLACEMENT INTEGER | | | | | | | | |

(b) CPU32, MC68020, MC68030, and MC68040

Figure 2-3. M68000 Family Brief Extension Word Formats

2.5 FULL EXTENSION ADDRESSING MODES

The full extension word format provides additional addressing modes for the MC68020, MC68030, and MC68040. There are four elements common to these full extension addressing modes: a base register (BR), an index register (Xn), a base displacement (bd), and an outer displacement (od). Each of these four elements can be suppressed independently of each other. However, at least one element must be active and not suppressed. When an element is suppressed, it has an effective value of zero.

BR can be suppressed through the BS field of the full extension word format. The encoding of bits 0-5 in the single effective address word format (see Figure 2-2) selects BR as either the PC when using program relative addressing modes, or An when using non-program relative addressing modes. The value of the PC is the address of the extension word. For the non-program relative addressing modes, BR is the contents of a selected An.

SIZE and SCALE can be used to modify Xn. The W/L field in the full extension format selects the size of Xn as a word or long word. The SCALE field selects the scaling factor, shifts the value of the Xn left multiplying the value by 1, 2, 4, or 8, respectively, without actually changing the value. Scaling can be used to calculate the address of arrayed structures. Figure 2-4 illustrates the scaling of an Xn.

The bd and od can be either word or long word. The size of od is selected through the encoding of the I/S field in the full extension word format (refer to Table 2-2). There are two main modes of operation that use these four elements in different ways: no memory indirect action and memory indirect. The od is provided only for using memory indirect addressing modes of which there are three types: with preindex, with postindex, and with index suppressed.

SYNTAX: MOVE.B (A5, A6.L*SCALE),(A7)

WHERE

A5 = ADDRESS OF ARRAY STRUCTURE

A6 = INDEX NUMBER OF ARRAY ITEM

A7 = STACK POINTER

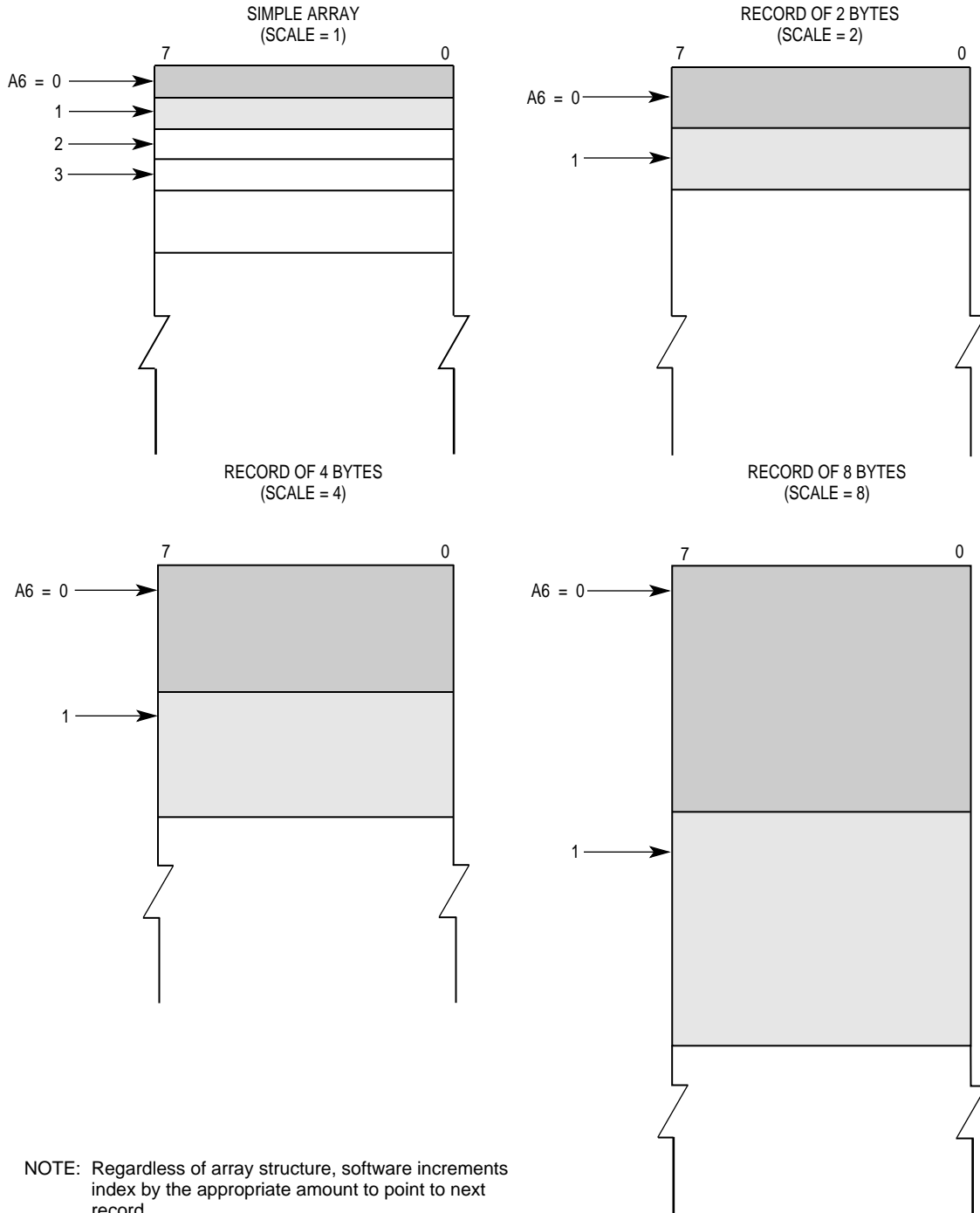


Figure 2-4. Addressing Array Items

2.5.1 No Memory Indirect Action Mode

No memory indirect action mode uses BR, Xn with its modifiers, and bd to calculate the address of the required operand. Data register indirect (Dn) and absolute address with index (bd,Xn.SIZE*SCALE) are examples of the no memory indirect action mode. Figure 2-5 illustrates the no memory indirect action mode.

| BR | Xn | bd | Addressing Mode |
|----|----|----|--|
| S | S | S | Not Applicable |
| S | S | A | Absolute Addressing Mode |
| S | A | S | Register Indirect |
| S | A | A | Register Indirect with Constant Index |
| An | S | S | Address Register Indirect |
| An | S | A | Address Register Indirect with Constant Index |
| An | A | S | Address Register Indirect with Variable Index |
| An | A | A | Address Register Indirect with Constant and Variable Index |
| PC | S | S | PC Relative |
| PC | S | A | PC Relative with Constant Index |
| PC | A | S | PC Relative with Variable Index |
| PC | A | A | PC Relative with Constant and Variable Index |

NOTE: S indicates suppressed and A indicates active.

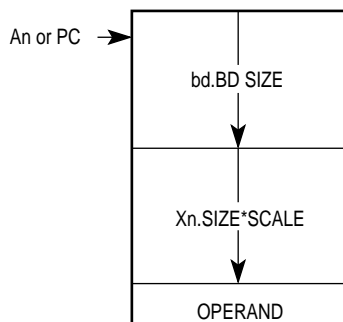


Figure 2-5. No Memory Indirect Action

2.5.2 Memory Indirect Modes

Memory indirect modes fetch two operands from memory. The BR and bd evaluate the address of the first operand, intermediate memory pointer (IMP). The value of IMP and the od evaluates the address of the second operand.

There are three types of memory indirect modes: pre-index, post-index, and index register suppressed. Xn and its modifiers can be allocated to determine either the address of the IMP (pre-index) or to the address of the second operand (post-index).

2.5.2.1 MEMORY INDIRECT WITH PREINDEX. The Xn is allocated to determine the address of the IMP. Figure 2-6 illustrates the memory indirect with pre-indexing mode.

| BR | Xn | bd | od | IMP Addressing Mode | Operand Addressing Mode |
|----|----|----|----|--|--|
| S | A | S | S | Register Indirect | Memory Pointer Directly to Data Operand |
| S | A | S | A | Register Indirect | Memory Pointer as Base with Displacement to Data Operand |
| S | A | A | S | Register Indirect with Constant Index | Memory Pointer Directly to Data Operand |
| S | A | A | A | Register Indirect with Constant Index | Memory Pointer as Base with Displacement to Data Operand |
| An | A | S | S | Address Register Indirect with Variable Index | Memory Pointer Directly to Data Operand |
| An | A | S | A | Address Register Indirect with Variable Index | Memory Pointer as Base with Displacement to Data Operand |
| An | A | A | S | Address Register Indirect with Constant and Variable Index | Memory Pointer Directly to Data Operand |
| An | A | A | A | Address Register Indirect with Constant and Variable Index | Memory Pointer as Base with Displacement to Data Operand |
| PC | A | S | S | PC Relative with Variable Index | Memory Pointer Directly to Data Operand |
| PC | A | S | A | PC Relative with Variable Index | Memory Pointer as Base with Displacement to Data Operand |
| PC | A | A | S | PC Relative with Constant and Variable Index | Memory Pointer Directly to Data Operand |
| PC | A | A | A | PC Relative with Constant and Variable Index | Memory Pointer as Base with Displacement to Data Operand |

NOTE: S indicates suppressed and A indicates active.

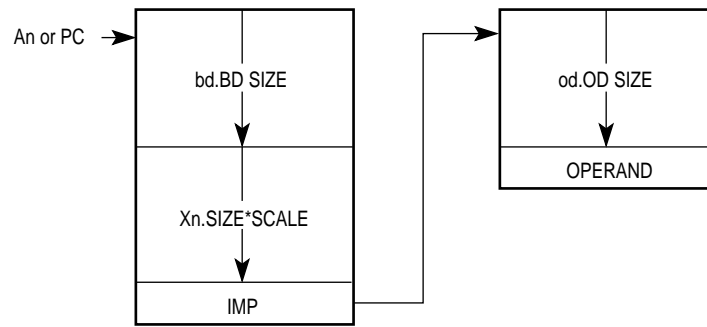


Figure 2-6. Memory Indirect with Preindex

2.5.2.2 MEMORY INDIRECT WITH POSTINDEX. The Xn is allocated to evaluate the address of the second operand. Figure 2-7 illustrates the memory indirect with post-indexing mode.

| BR | Xn | bd | od | IMP Addressing Mode | Operand Addressing Mode |
|----|----|----|----|---|---|
| S | A | S | S | — | — |
| S | A | S | A | — | — |
| S | A | A | S | Absolute Addressing Mode | Memory Pointer with Variable Index to Data Operand |
| S | A | A | A | Absolute Addressing Mode | Memory Pointer with Constant and Variable Index to Data Operand |
| An | A | S | S | Address Register Indirect | Memory Pointer with Variable Index to Data Operand |
| An | A | S | A | Address Register Indirect | Memory Pointer with Constant and Variable Index to Data Operand |
| An | A | A | S | Address Register Indirect with Constant Index | Memory Pointer with Variable Index to Data Operand |
| An | A | A | A | Address Register Indirect with Constant Index | Memory Pointer with Constant and Variable Index to Data Operand |
| PC | A | S | S | PC Relative | Memory Pointer with Variable Index to Data Operand |
| PC | A | S | A | PC Relative | Memory Pointer with Constant and Variable Index to Data Operand |
| PC | A | A | S | PC Relative with Constant Index | Memory Pointer with Variable Index to Data Operand |
| PC | A | A | A | PC Relative with Constant Index | Memory Pointer with Constant and Variable Index to Data Operand |

NOTE: S indicates suppressed and A indicates active.

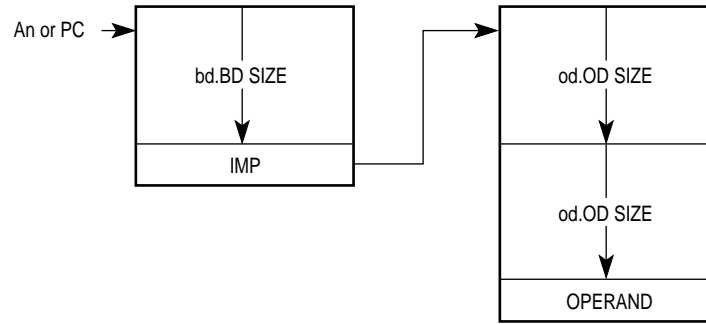


Figure 2-7. Memory Indirect with Postindex

2.5.2.3 MEMORY INDIRECT WITH INDEX SUPPRESSED. The Xn is suppressed. Figure 2-8 illustrates the memory indirect with index suppressed mode.

| BR | Xn | bd | od | IMP Addressing Mode | Operand Addressing Mode |
|----|----|----|----|---|--|
| S | S | S | S | — | — |
| S | S | S | A | — | — |
| S | S | A | S | Absolute Addressing Mode | Memory Pointer Directly to Data Operand |
| S | S | A | A | Absolute Addressing Mode | Memory Pointer as Base with Displacement to Data Operand |
| An | S | S | S | Address Register Indirect | Memory Pointer Directly to Data Operand |
| An | S | S | A | Address Register Indirect | Memory Pointer as Base with Displacement to Data Operand |
| An | S | A | S | Address Register Indirect with Constant Index | Memory Pointer Directly to Data Operand |
| An | S | A | A | Address Register Indirect with Constant Index | Memory Pointer as Base with Displacement to Data Operand |
| PC | S | S | S | PC Relative | Memory Pointer Directly to Data Operand |
| PC | S | S | A | PC Relative | Memory Pointer as Base with Displacement to Data Operand |
| PC | S | A | S | PC Relative with Constant Index | Memory Pointer Directly to Data Operand |
| PC | S | A | A | PC Relative with Constant Index | Memory Pointer as Base with Displacement to Data Operand |

NOTE: S indicates suppressed and A indicates active.

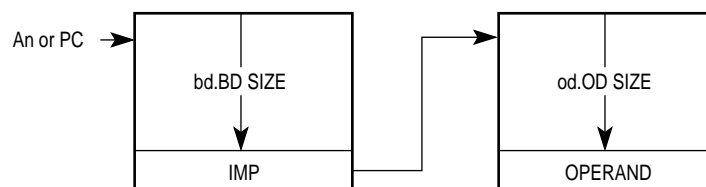


Figure 2-8. Memory Indirect with Index Suppress

2.6 OTHER DATA STRUCTURES

Stacks and queues are common data structures. The M68000 family implements a system stack and instructions that support user stacks and queues.

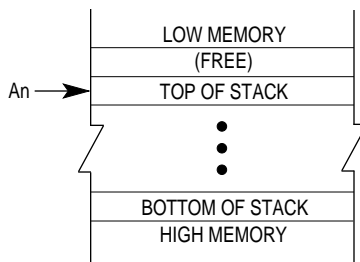
2.6.1 System Stack

Address register seven (A7) is the system stack pointer. Either the user stack pointer (USP), the interrupt stack pointer (ISP), or the master stack pointer (MSP) is active at any one time. Refer to **Section 1 Introduction** for details on these stack pointers. To keep data on the system stack aligned for maximum efficiency, the active stack pointer is automatically decremented or incremented by two for all byte-size operands moved to or from the stack. In long-word-organized memory, aligning the stack pointer on a long-word address significantly increases the efficiency of stacking exception frames, subroutine calls and returns, and other stacking operations.

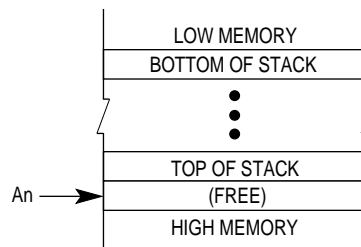
The user can implement stacks with the address register indirect with postincrement and predecrement addressing modes. With an address register the user can implement a stack that fills either from high memory to low memory or from low memory to high memory. Important considerations are:

- Use the predecrement mode to decrement the register before using its contents as the pointer to the stack.
- Use the postincrement mode to increment the register after using its contents as the pointer to the stack.
- Maintain the stack pointer correctly when byte, word, and long-word items mix in these stacks.

To implement stack growth from high memory to low memory, use $-(A_n)$ to push data on the stack and $(A_n) +$ to pull data from the stack. For this type of stack, after either a push or a pull operation, the address register points to the top item on the stack.



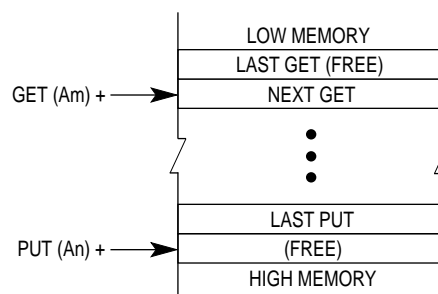
To implement stack growth from low memory to high memory, use $(An) +$ to push data on the stack and $-(An)$ to pull data from the stack. After either a push or pull operation, the address register points to the next available space on the stack. .



2.6.2 Queues

The user can implement queues, groups of information waiting to be processed, with the address register indirect with postincrement or predecrement addressing modes. Using a pair of address registers, the user implements a queue that fills either from high memory to low memory or from low memory to high memory. Two registers are used because the queues get pushed from one end and pulled from the other. One address register contains the put pointer; the other register the get pointer. To implement growth of the queue from low memory to high memory, use the put address register to put data into the queue and the get address register to get data from the queue.

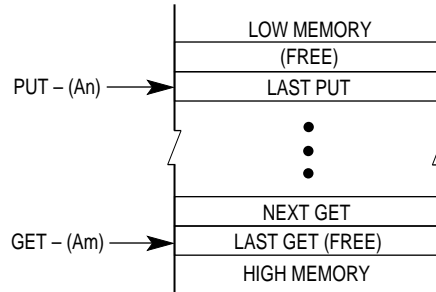
After a put operation, the put address register points to the next available space in the queue; the unchanged get address register points to the next item to be removed from the queue. After a get operation, the get address register points to the next item to be removed from the queue; the unchanged put address register points to the next available space in the queue. .



To implement the queue as a circular buffer, the relevant address register should be checked and adjusted. If necessary, do this before performing the put or get operation. Subtracting the buffer length (in bytes) from the register adjusts the address register. To implement growth of the queue from high memory to low memory, use the put address register indirect to put data into the queue and get address register indirect to get data from the queue.

Addressing Capabilities

After a put operation, the put address register points to the last item placed in the queue; the unchanged get address register points to the last item removed from the queue. After a get operation, the get address register points to the last item placed in the queue.



To implement the queue as a circular buffer, the get or put operation should be performed first. Then the relevant address register should be checked and adjusted, if necessary. Adding the buffer length (in bytes) to the address register contents adjusts the address register.

SECTION 3

INSTRUCTION SET SUMMARY

This section briefly describes the M68000 family instruction set, using Motorola's assembly language syntax and notation. It includes instruction set details such as notation and format, selected instruction examples, and an integer condition code discussion. The section concludes with a discussion of floating-point details such as computational accuracy, conditional test definitions, an explanation of the operation table, and a discussion of not-a-numbers (NaNs) and postprocessing.

3.1 INSTRUCTION SUMMARY

Instructions form a set of tools that perform the following types of operations:

| | |
|---------------------------------|-------------------------------|
| Data Movement | Program Control |
| Integer Arithmetic | System Control |
| Logical Operations | Cache Maintenance |
| Shift and Rotate Operations | Multiprocessor Communications |
| Bit Manipulation | Memory Management |
| Bit Field Manipulation | Floating-Point Arithmetic |
| Binary-Coded Decimal Arithmetic | |

The following paragraphs describe in detail the instruction for each type of operation. Table 3-1 lists the notations used throughout this manual. In the operand syntax statements of the instruction definitions, the operand on the right is the destination operand.

Table 3-1. Notational Conventions

| Single- And Double Operand Operations | |
|--|---|
| + | Arithmetic addition or postincrement indicator. |
| - | Arithmetic subtraction or predecrement indicator. |
| × | Arithmetic multiplication. |
| ÷ | Arithmetic division or conjunction symbol. |
| ~ | Invert; operand is logically complemented. |
| ∧ | Logical AND |
| ∨ | Logical OR |
| ⊕ | Logical exclusive OR |
| → | Source operand is moved to destination operand. |
| ←→ | Two operands are exchanged. |
| <op> | Any double-operand operation. |
| <operand>tested | Operand is compared to zero and the condition codes are set appropriately. |
| sign-extended | All bits of the upper portion are made equal to the high-order bit of the lower portion. |
| Other Operations | |
| TRAP | Equivalent to Format #Offset Word → (SSP); SSP - 2 → SSP; PC → (SSP); SSP - 4 → SSP; SR → (SSP); SSP - 2 → SSP; (Vector) → PC |
| STOP | Enter the stopped state, waiting for interrupts. |
| <operand> ₁₀ | The operand is BCD; operations are performed in decimal. |
| If <condition> then <operations> else <operations> | Test the condition. If true, the operations after “then” are performed. If the condition is false and the optional “else” clause is present, the operations after “else” are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example. |
| Register Specifications | |
| An | Any Address Register n (example: A3 is address register 3) |
| Ax, Ay | Source and destination address registers, respectively. |
| Dc | Data register D7–D0, used during compare. |
| Dh, Dl | Data register’s high- or low-order 32 bits of product. |
| Dn | Any Data Register n (example: D5 is data register 5) |
| Dr, Dq | Data register’s remainder or quotient of divide. |
| Du | Data register D7–D0, used during update. |
| Dx, Dy | Source and destination data registers, respectively. |
| MRn | Any Memory Register n. |
| Rn | Any Address or Data Register |
| Rx, Ry | Any source and destination registers, respectively. |
| Xn | Index Register |

Table 3-1. Notational Conventions (Continued)

| Data Format And Type | |
|---------------------------------|---|
| + inf | Positive Infinity |
| <fmt> | Operand Data Format: Byte (B), Word (W), Long (L), Single (S), Double (D), Extended (X), or Packed (P). |
| B, W, L | Specifies a signed integer data type (twos complement) of byte, word, or long word. |
| D | Double-precision real data format (64 bits). |
| k | A twos complement signed integer (–64 to +17) specifying a number's format to be stored in the packed decimal format. |
| P | Packed BCD real data format (96 bits, 12 bytes). |
| S | Single-precision real data format (32 bits). |
| X | Extended-precision real data format (96 bits, 16 bits unused). |
| – inf | Negative Infinity |
| Subfields and Qualifiers | |
| #<xxx> or #<data> | Immediate data following the instruction word(s). |
| () | Identifies an indirect address in a register. |
| [] | Identifies an indirect address in memory. |
| bd | Base Displacement |
| ccc | Index into the MC68881/MC68882 Constant ROM |
| d _n | Displacement Value, n Bits Wide (example: d ₁₆ is a 16-bit displacement). |
| LSB | Least Significant Bit |
| LSW | Least Significant Word |
| MSB | Most Significant Bit |
| MSW | Most Significant Word |
| od | Outer Displacement |
| SCALE | A scale factor (1, 2, 4, or 8 for no-word, word, long-word, or quad-word scaling, respectively). |
| SIZE | The index register's size (W for word, L for long word). |
| {offset:width} | Bit field selection. |
| Register Names | |
| CCR | Condition Code Register (lower byte of status register) |
| DFC | Destination Function Code Register |
| FPcr | Any Floating-Point System Control Register (FPCR, FPSR, or FPIAR) |
| FPm, FPn | Any Floating-Point Data Register specified as the source or destination, respectively. |
| IC, DC, IC/DC | Instruction, Data, or Both Caches |
| MMUSR | MMU Status Register |
| PC | Program Counter |
| Rc | Any Non Floating-Point Control Register |
| SFC | Source Function Code Register |
| SR | Status Register |

Table 3-1. Notational Conventions (Concluded)

| Register Codes | |
|-----------------------|--|
| * | General Case |
| C | Carry Bit in CCR |
| cc | Condition Codes from CCR |
| FC | Function Code |
| N | Negative Bit in CCR |
| U | Undefined, Reserved for Motorola Use. |
| V | Overflow Bit in CCR |
| X | Extend Bit in CCR |
| Z | Zero Bit in CCR |
| — | Not Affected or Applicable. |
| Stack Pointers | |
| ISP | Supervisor/Interrupt Stack Pointer |
| MSP | Supervisor/Master Stack Pointer |
| SP | Active Stack Pointer |
| SSP | Supervisor (Master or Interrupt) Stack Pointer |
| USP | User Stack Pointer |
| Miscellaneous | |
| <ea> | Effective Address |
| <label> | Assemble Program Label |
| <list> | List of registers, for example D3–D0. |
| LB | Lower Bound |
| m | Bit m of an Operand |
| m–n | Bits m through n of Operand |
| UB | Upper Bound |

3.1.1 Data Movement Instructions

The MOVE and FMOVE instructions with their associated addressing modes are the basic means of transferring and storing addresses and data. MOVE instructions transfer byte, word, and long-word operands from memory to memory, memory to register, register to memory, and register to register. MOVE instructions transfer word and long-word operands and ensure that only valid address manipulations are executed. In addition to the general MOVE instructions, there are several special data movement instructions: MOVE16, MOVEM, MOVEP, MOVEQ, EXG, LEA, PEA, LINK, and UNLK. The MOVE16 instruction is an MC68040 extension to the M68000 instruction set.

The FMOVE instructions move operands into, out of, and between floating-point data registers. FMOVE also moves operands to and from the floating-point control register (FPCR), floating-point status register (FPSR), and floating-point instruction address register (FPIAR). For operands moved into a floating-point data register, FSMOVE and FDMOVE explicitly select single- and double-precision rounding of the result, respectively. FMOVEM moves any combination of either floating-point data registers or floating-point control registers. Table 3-2 lists the general format of these integer and floating-point data movement instructions.

Table 3-2. Data Movement Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|-------------------------|---|---|--|
| EXG | Rn, Rn | 32 | Rn ← → Rn |
| FMOVE | FPm,FPn <ea>,FPn FPm,<ea> <ea>,FPcr FPcr,<ea> | X B, W, L, S, D, X, P B, W, L, S, D, X, P 32 32 | Source → Destination |
| FSMOVE, FDMOVE | FPm,FPn <ea>,FPn | X B, W, L, S, D, X | Source → Destination; round destination to single or double precision. |
| FMOVEM | <ea>,<list> ¹ <ea>,Dn <list> ¹ ,<ea> Dn,<ea> | 32, X X 32, X X | Listed Registers → Destination Source → Listed Registers |
| LEA | <ea>,An | 32 | <ea> → An |
| LINK | An,#<d> | 16, 32 | SP - 4 → SP; An → (SP); SP → An, SP + D → SP |
| MOVE MOVE16 MOVEA | <ea>,<ea> <ea>,<ea> <ea>,An | 8, 16, 32 16 bytes 16, 32 → 32 | Source → Destination Aligned 16-Byte Block → Destination |
| MOVEM | list,<ea> <ea>,list | 16, 32 16, 32 → 32 | Listed Registers → Destination Source → Listed Registers |
| MOVEP | Dn, (d ₁₆ ,An) (d ₁₆ ,An),Dn | 16, 32 | Dn 31-24 → (An + d _n); Dn 23-16 → (An + d _n + 2); Dn 15-8 → (An + d _n + 4); Dn 7-0 → (An + d _n + 6) (An + d _n) → Dn 31-24; (An + d _n + 2) → Dn 23-16; (An + d _n + 4) → Dn 15-8; (An + d _n + 6) → Dn 7-0 |
| MOVEQ | #<data>,Dn | 8 → 32 | Immediate Data → Destination |
| PEA | <ea> | 32 | SP - 4 → SP; <ea> → (SP) |
| UNLK | An | 32 | An → SP; (SP) → An; SP + 4 → SP |

NOTE: A register list includes any combination of the eight floating-point data registers or any combination of three control registers (FPCR, FPSR, and FPIAR). If a register list mask resides in a data register, only floating-point data registers may be specified.

3.1.2 Integer Arithmetic Instructions

The integer arithmetic operations include four basic operations: ADD, SUB, MUL, and DIV. They also include CMP, CMPM, CMP2, CLR, and NEG. The instruction set includes ADD, CMP, and SUB instructions for both address and data operations with all operand sizes valid for data operations. Address operands consist of 16 or 32 bits. The CLR and NEG instructions apply to all sizes of data operands. Signed and unsigned MUL and DIV instructions include:

- Word multiply to produce a long-word product.
- Long-word multiply to produce a long-word or quad-word product.
- Long word divided by a word divisor (word quotient and word remainder).
- Long word or quad word divided by a long-word divisor (long-word quotient and long-word remainder).

A set of extended instructions provides multiprecision and mixed-size arithmetic: ADDX, SUBX, EXT, and NEGX. Refer to Table 3-3 for a summary of the integer arithmetic operations. In Table 3-3, X refers to the X-bit in the CCR.

Table 3-3. Integer Arithmetic Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|-------------|----------------------------------|--|--|
| ADD | Dn,<ea> | 8, 16, 32 | Source + Destination → Destination |
| ADDA | <ea>,Dn <ea>,An | 8, 16, 32 16, 32 | |
| ADDI | #<data>,<ea> | 8, 16, 32 | Immediate Data + Destination → Destination |
| ADDQ | #<data>,<ea> | 8, 16, 32 | |
| ADDX | Dn,Dn -(An), -(An) | 8, 16, 32 8, 16, 32 | Source + Destination + X → Destination |
| CLR | <ea> | 8, 16, 32 | 0 → Destination |
| CMP | <ea>,Dn | 8, 16, 32 | Destination – Source |
| CMPA | <ea>,An | 16, 32 | |
| CMPI | #<data>,<ea> | 8, 16, 32 | Destination – Immediate Data |
| CMPM | (An)+,(An)+ | 8, 16, 32 | Destination – Source |
| CMP2 | <ea>,Rn | 8, 16, 32 | Lower Bound → Rn → Upper Bound |
| DIVS/DIVU | <ea>,Dn <ea>,Dr–Dq | 32 ÷ 16 → 16,16 64 ÷ 32 → 32,32 | Destination ÷ Source → Destination (Signed or Unsigned Quotient, Remainder) |
| DIVSL/DIVUL | <ea>,Dq <ea>,Dr–Dq | 32 ÷ 32 → 32 32 ÷ 32 → 32,32 | |
| EXT | Dn | 8 → 16 | Sign-Extended Destination → Destination |
| EXTB | Dn | 16 → 32 8 → 32 | |
| MULS/MULU | <ea>,Dn <ea>,Dl <ea>,Dh–Dl | 16 x 16 → 32 32 x 32 → 32 32 x 32 → 64 | Source x Destination → Destination (Signed or Unsigned) |
| NEG | <ea> | 8, 16, 32 | 0 – Destination → Destination |
| NEGX | <ea> | 8, 16, 32 | 0 – Destination – X → Destination |
| SUB | <ea>,Dn | 8, 16, 32 | Destination = Source → Destination |
| SUBA | Dn,<ea> <ea>,An | 8, 16, 32 16, 32 | |
| SUBI | #<data>,<ea> | 8, 16, 32 | Destination – Immediate Data → Destination |
| SUBQ | #<data>,<ea> | 8, 16, 32 | |
| SUBX | Dn,Dn -(An), -(An) | 8, 16, 32 8, 16, 32 | Destination – Source – X → Destination |

3.1.3 Logical Instructions

The logical operation instructions (AND, OR, EOR, and NOT) perform logical operations with all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provides these logical operations with all sizes of immediate data. Table 3-4 summarizes the logical operations.

Table 3-4. Logical Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|-------------|--------------------|------------------------|---|
| AND | <ea>,Dn Dn,<ea> | 8, 16, 32 8, 16, 32 | Source \wedge Destination \rightarrow Destination |
| ANDI | #<data>,<ea> | 8, 16, 32 | Immediate Data \wedge Destination \rightarrow Destination |
| EOR | Dn,<ea> | 8, 16, 32 | Source \oplus Destination \rightarrow Destination |
| EORI | #<data>,<ea> | 8, 16, 32 | Immediate Data \oplus Destination \rightarrow Destination |
| NOT | <ea> | 8, 16, 32 | \sim Destination \rightarrow Destination |
| OR | <ea>,Dn Dn,<ea> | 8, 16, 32 | Source \vee Destination \rightarrow Destination |
| ORI | #<data>,<ea> | 8, 16, 32 | Immediate Data \vee Destination \rightarrow Destination |

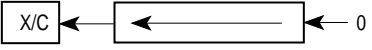
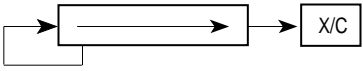
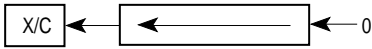
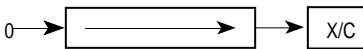
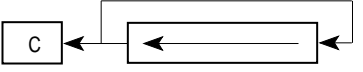
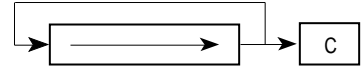
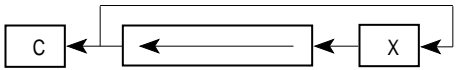
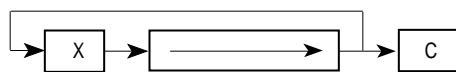
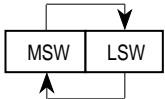
3.1.4 Shift and Rotate Instructions

The ASR, ASL, LSR, and LSL instructions provide shift operations in both directions. The ROR, ROL, ROXR, and ROXL instructions perform rotate (circular shift) operations, with and without the CCR extend bit (X-bit). All shift and rotate operations can be performed on either registers or memory.

Register shift and rotate operations shift all operand sizes. The shift count can be specified in the instruction operation word (to shift from 1 – 8 places) or in a register (modulo 64 shift count).

Memory shift and rotate operations shift word operands one bit position only. The SWAP instruction exchanges the 16-bit halves of a register. Fast byte swapping is possible by using the ROR and ROL instructions with a shift count of eight, enhancing the performance of the shift/rotate instructions. Table 3-5 is a summary of the shift and rotate operations. In Table 3-5, C and X refer to the C-bit and X-bit in the CCR.

Table 3-5. Shift and Rotate Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|-------------|------------------------------|------------------------------|---|
| ASL | Dn, Dn # <data>, Dn ea | 8, 16, 32 8, 16, 32 16 |  |
| ASR | Dn, Dn # <data>, Dn ea | 8, 16, 32 8, 16, 32 16 |  |
| LSL | Dn, Dn # <data>, Dn ea | 8, 16, 32 8, 16, 32 16 |  |
| LSR | Dn, Dn # <data>, Dn ea | 8, 16, 32 8, 16, 32 16 |  |
| ROL | Dn, Dn # <data>, Dn ea | 8, 16, 32 8, 16, 32 16 |  |
| ROR | Dn, Dn # <data>, Dn ea | 8, 16, 32 8, 16, 32 16 |  |
| ROXL | Dn, Dn # <data>, Dn ea | 8, 16, 32 8, 16, 32 16 |  |
| ROXR | Dn, Dn # <data>, Dn ea | 8, 16, 32 8, 16, 32 16 |  |
| SWAP | Dn | 32 |  |

NOTE: X indicates the extend bit and C the carry bit in the CCR.

3.1.5 Bit Manipulation Instructions

BTST, BSET, BCLR, and BCHG are bit manipulation instructions. All bit manipulation operations can be performed on either registers or memory. The bit number is specified either as immediate data or in the contents of a data register. Register operands are 32 bits long, and memory operands are 8 bits long. Table 3-6 summarizes bit manipulation operations; Z refers to the zero bit of the CCR.

Table 3-6. Bit Manipulation Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|-------------|-------------------------|----------------|--|
| BCHG | Dn,<ea> #<data>,<ea> | 8, 32 8, 32 | ~ (<Bit Number> of Destination) → Z → Bit of Destination |
| BCLR | Dn,<ea> #<data>,<ea> | 8, 32 8, 32 | ~ (<Bit Number> of Destination) → Z; 0 → Bit of Destination |
| BSET | Dn,<ea> #<data>,<ea> | 8, 32 8, 32 | ~ (<Bit Number> of Destination) → Z; 1 → Bit of Destination |
| BTST | Dn,<ea> #<data>,<ea> | 8, 32 8, 32 | ~ (<Bit Number> of Destination) → Z |

3.1.6 Bit Field Instructions

The M68000 family architecture supports variable-length bit field operations on fields of up to 32 bits. The BFINS instruction inserts a value into a bit field. BFEXTU and BFEXTS extract a value from the field. BFFFO finds the first set bit in a bit field. Also included are instructions analogous to the bit manipulation operations: BFTST, BFSET, BFCLR, and BFCHG. Table 3-7 summarizes bit field operations.

Table 3-7. Bit Field Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|-------------|-------------------------|--------------|--|
| BFCHG | <ea> {offset:width} | 1–32 | ~ Field → Field |
| BFCLR | <ea> {offset:width} | 1–32 | 0's → Field |
| BFEXTS | <ea> {offset:width}, Dn | 1–32 | Field → Dn; Sign-Extended |
| BFEXTU | <ea> {offset:width}, Dn | 1–32 | Field → Dn; Zero-Extended |
| BFFFO | <ea> {offset:width}, Dn | 1–32 | Scan for First Bit Set in Field; Offset → Dn. |
| BFINS | Dn,<ea> {offset:width} | 1–32 | Dn → Field |
| BFSET | <ea> {offset:width} | 1–32 | 1's → Field |
| BFTST | <ea> {offset:width} | 1–32 | Field MSB → N; ~ (OR of All Bits in Field) → Z |

NOTE: All bit field instructions set the CCR N and Z bits as shown for BFTST before performing the specified operation.

3.1.7 Binary-Coded Decimal Instructions

Five instructions support operations on binary-coded decimal (BCD) numbers. The arithmetic operations on packed BCD numbers are ABCD, SBCD, and NBCD. PACK and UNPK instructions aid in the conversion of byte-encoded numeric data, such as ASCII or EBCDIC strings to BCD data and vice versa. Table 3-8 summarizes BCD operations. In Table 3-8 X refers to the X-bit in the CCR.

Table 3-8. Binary-Coded Decimal Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|-------------|---------------------------------------|------------------|--|
| ABCD | Dn,Dn -(An), -(An) | 8 8 | Source ₁₀ + Destination ₁₀ + X → Destination |
| NBCD | <ea> | 8 | 0 – Destination ₁₀ – X → Destination |
| PACK | -(An), -(An) #<data> Dn,Dn,#<data> | 16 → 8 16 → 8 | Unpackaged Source + Immediate Data → Packed Destination |
| SBCD | Dn,Dn -(An), -(An) | 8 8 | Destination ₁₀ – Source ₁₀ – X → Destination |
| UNPK | -(An),-(An) #<data> Dn,Dn,#<data> | 8 → 16 8 → 16 | Packed Source → Unpacked Source Unpacked Source + Immediate Data → Unpacked Destination |

3.1.8 Program Control Instructions

A set of subroutine call and return instructions and conditional and unconditional branch instructions perform program control operations. Also included are test operand instructions (TST and FTST), which set the integer or floating-point condition codes for use by other program and system control instructions. NOP forces synchronization of the internal pipelines. Table 3-9 summarizes these instructions.

Table 3-9. Program Control Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|---|----------------|--------------------------|---|
| Integer and Floating-Point Conditional | | | |
| Bcc, FBcc | <label> | 8, 16, 32 | If Condition True, Then PC + d _n → PC |
| DBcc, FDBcc | Dn,<label> | 16 | If Condition False, Then Dn - 1 → Dn If Dn → -1, Then PC + d _n → PC |
| Scc, FScc | <ea> | 8 | If Condition True, Then 1's → Destination; Else 0's → Destination |
| Unconditional | | | |
| BRA | <label> | 8, 16, 32 | PC + d _n → PC |
| BSR | <label> | 8, 16, 32 | SP - 4 → SP; PC → (SP); PC + d _n → PC |
| JMP | <ea> | none | Destination → PC |
| JSR | <ea> | none | SP - 4 → SP; PC → (SP); Destination → PC |
| NOP | none | none | PC + 2 → PC (Integer Pipeline Synchronized) |
| FNOP | none | none | PC + 4 → PC (FPU Pipeline Synchronized) |
| Returns | | | |
| RTD | #<data> | 16 | (SP) → PC; SP + 4 + d _n → SP |
| RTR | none | none | (SP) → CCR; SP + 2 → SP; (SP) → PC; SP + 4 → SP |
| RTS | none | none | (SP) → PC; SP + 4 → SP |
| Test Operand | | | |
| TST | <ea> | 8, 16, 32 | Set Integer Condition Codes |
| FTST | <ea> FPn | B, W, L, S, D, X, P X | Set Floating-Point Condition Codes |

Letters cc in the integer instruction mnemonics Bcc, DBcc, and Scc specify testing one of the following conditions:

- | | |
|------------------|--------------------------|
| CC—Carry clear | GE—Greater than or equal |
| LS—Lower or same | PL—Plus |
| CS—Carry set | GT—Greater than |
| LT—Less than | T—Always true* |
| EQ—Equal | HI—Higher |
| MI—Minus | VC—Overflow clear |
| F—Never true* | LE—Less than or equal |
| NE—Not equal | VS—Overflow set |

*Not applicable to the Bcc instructions.

3.1.9 System Control Instructions

Privileged and trapping instructions as well as instructions that use or modify the CCR provide system control operations. FSAVE and FRESTORE save and restore the nonuser visible portion of the FPU during context switches in a virtual memory or multitasking system. The conditional trap instructions, which use the same conditional tests as their corresponding program control instructions, allow an optional 16- or 32-bit immediate operand to be included as part of the instruction for passing parameters to the operating system. These instructions cause the processor to flush the instruction pipe. Table 3-10 summarizes these instructions. See 3.2 Integer Unit Condition Code Computation for more details on condition codes.

Table 3-10. System Control Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|--------------------------------|--------------------|----------------|--|
| Privileged | | | |
| ANDI to SR | #<data>,SR | 16 | Immediate Data \wedge SR \rightarrow SR |
| EORI to SR | #<data>,SR | 16 | Immediate Data \oplus SR \rightarrow SR |
| FRESTORE | <ea> | none | State Frame \rightarrow Internal Floating-Point Registers |
| FSAVE | <ea> | none | Internal Floating-Point Registers \rightarrow State Frame |
| MOVE to SR | <ea>,SR | 16 | Source \rightarrow SR |
| MOVE from SR | SR,<ea> | 16 | SR \rightarrow Destination |
| MOVE USP | USP,An An,USP | 32 32 | USP \rightarrow An An \rightarrow USP |
| MOVEC | Rc,Rn Rn,Rc | 32 32 | Rc \rightarrow Rn Rn \rightarrow Rc |
| MOVES | Rn,<ea> <ea>,Rn | 8, 16, 32 | Rn \rightarrow Destination Using DFC Source Using SFC \rightarrow Rn |
| ORI to SR | #<data>,SR | 16 | Immediate Data \vee SR \rightarrow SR |
| RESET | none | none | Assert Reset Output |
| RTE | none | none | (SP) \rightarrow SR; SP + 2 \rightarrow SP; (SP) \rightarrow PC; SP + 4 \rightarrow SP; Restore Stack According to Format |
| STOP | #<data> | 16 | Immediate Data \rightarrow SR; STOP |
| Trap Generating | | | |
| BKPT | #<data> | none | Run Breakpoint Cycle |
| CHK | <ea>,Dn | 16, 32 | If Dn < 0 or Dn > (<ea>), Then CHK Exception |
| CHK2 | <ea>,Rn | 8, 16, 32 | If Rn < Lower Bound or Rn > Upper Bound, Then CHK Exception |
| ILLEGAL | none | none | SSP - 2 \rightarrow SSP; Vector Offset \rightarrow (SSP); SSP - 4 \rightarrow SSP; PC \rightarrow (SSP); SSP - 2 \rightarrow SSP; SR \rightarrow (SSP); Illegal Instruction Vector Address \rightarrow PC |
| TRAP | #<data> | none | SSP - 2 \rightarrow SSP; Format and Vector Offset \rightarrow (SSP) SSP - 4 \rightarrow SSP; PC \rightarrow (SSP); SSP - 2 \rightarrow SSP; SR \rightarrow (SSP); Vector Address \rightarrow PC |
| TRAPcc | none #<data> | none 16, 32 | If cc True, Then Trap Exception |
| FTRAPcc | none #<data> | none 16, 32 | If Floating-Point cc True, Then Trap Exception |
| TRAPV | none | none | If V, Then Take Overflow Trap Exception |
| Condition Code Register | | | |
| ANDI to SR | #<data>,CCR | 8 | Immediate Data \wedge CCR \rightarrow CCR |
| EORI to SR | #<data>,CCR | 8 | Immediate Data \oplus CCR \rightarrow CCR |
| MOVE to SR | <ea>,CCR | 16 | Source \rightarrow CCR |
| MOVE from SR | CCR,<ea> | 16 | CCR \rightarrow Destination |
| ORI to SR | #<data>,CCR | 8 | Immediate Data \vee CCR \rightarrow CCR |

Letters cc in the TRAPcc and FTRAPcc specify testing for a condition.

3.1.10 Cache Control Instructions (MC68040)

The cache instructions provide maintenance functions for managing the instruction and data caches. CINV invalidates cache entries in both caches, and CPUSH pushes dirty data from the data cache to update memory. Both instructions can operate on either or both caches and can select a single cache line, all lines in a page, or the entire cache. Table 3-11 summarizes these instructions.

Table 3-11. Cache Control Operation Format

| Instruction | Operand Syntax | Operand Size | Operation |
|----------------------------|---------------------------------------|----------------------|---|
| CINVL | caches,(An) | none | Invalidate cache line |
| CINVP | caches, (An) | none | Invalidate cache page |
| CINVA | caches | none | Invalidate entire cache |
| CPUSHL CPUSHP CPUSHA | caches,(An) caches, (An) caches | none none none | Push selected dirty data cache lines, then invalidate selected cache lines |

3.1.11 Multiprocessor Instructions

The TAS, CAS, and CAS2 instructions coordinate the operations of processors in multiprocessing systems. These instructions use read-modify-write bus cycles to ensure uninterrupted updating of memory. Coprocessor instructions control the coprocessor operations. Table 3- 12 summarizes these instructions.

Table 3-12. Multiprocessor Operations

| Instruction | Operand Syntax | Operand Size | Operation |
|--------------------------|--------------------------------|----------------|---|
| Read-Write-Modify | | | |
| CAS | Dc,Du,<ea> | 8, 16, 32 | Destination – Dc → CC If Z, Then Du → Destination Else Destination → Dc |
| CAS2 | Dc1–Dc2, Du1–Du2, (Rn)–(Rn) | 16, 32 | Dual Operand CAS |
| TAS | <ea> | 8 | Destination – 0; Set Condition Codes; 1 → Destination [7] |
| Coprocessor | | | |
| cpBcc | <label> | 16, 32 | If cpcc True, Then PC + d _n → PC |
| cpDBcc | <label>,Dn | 16 | If cpcc False, Then Dn – 1 → Dn If Dn ≠ –1, Then PC + d _n → PC |
| cpGEN | User Defined | User Defined | Operand → Coprocessor |
| cpRESTORE | <ea> | none | Restore Coprocessor State from <ea> |
| cpSAVE | <ea> | none | Save Coprocessor State at <ea> |
| cpScc | <ea> | 8 | If cpcc True, Then 1's → Destination; Else 0's → Destination |
| cpTRAPcc | none #<data> | none 16, 32 | If cpcc True, Then TRAPcc Exception |

3.1.12 Memory Management Unit (MMU) Instructions

The PFLUSH instructions flush the address translation caches (ATCs) and can optionally select only nonglobal entries for flushing. PTEST performs a search of the address translation tables, stores the results in the MMU status register, and loads the entry into the ATC. Table 3-13 summarizes these instructions.

Table 3-13. MMU Operation Format

| Instruction | Processor | Operand Syntax | Operand Size | Operation |
|-------------|-------------------------------|----------------------|--------------|---|
| PBcc | MC68851 | <label> | none | Branch on PMMU Condition |
| PDBcc | MC68851 | Dn,<label> | none | Test, Decrement, and Branch |
| PFLUSHA | MC68030 MC68040 MC68851 | none | none | Invalidate All ATC Entries |
| PFLUSH | MC68040 | (An) | none | Invalidate ATC Entries at Effective Address |
| PFLUSHN | MC68040 | (An) | none | Invalidate Nonglobal ATC Entries at Effective Address |
| PFLUSHAN | MC68040 | none | none | Invalidate All Nonglobal ATC Entries |
| PFLUSHS | MC68851 | none | none | Invalidate All Shared/Global ATC Entries |
| PFLUSHR | MC68851 | <ea> | none | Invalidate ATC and RPT Entries |
| PLOAD | MC68030 MC68851 | FC,<ea> | none | Load an Entry into the ATC |
| PMOVE | MC68030 MC68851 | MRn,<ea> <ea>,MRn | 8,16,32,64 | Move to/from MMU Registers |
| PRESTORE | MC68851 | <ea> | none | PMMU Restore Function |
| PSAVE | MC68851 | <ea> | none | PMMU Save Function |
| PScC | MC68851 | <ea> | 8 | Set on PMMU Condition |
| PTEST | MC68030 MC68040 MC68851 | (An) | none | Information About Logical Address → MMU Status Register |
| PTRAPcc | MC68851 | #<data> | 16,32 | Trap on PMMU Condition |

3.1.13 Floating-Point Arithmetic Instructions

The following paragraphs describe the floating-point instructions, organized into two categories of operation: dyadic (requiring two operands) and monadic (requiring one operand).

The dyadic floating-point instructions provide several arithmetic functions that require two input operands, such as add and subtract. For these operations, the first operand can be located in memory, an integer data register, or a floating-point data register. The second operand is always located in a floating-point data register. The results of the operation store in the register specified as the second operand. All FPU operations support all data formats. Results are rounded to either extended-, single-, or double-precision format. Table 3-14 gives the general format of dyadic instructions, and Table 3-15 lists the available operations.

Table 3-14. Dyadic Floating-Point Operation Format

| Instruction | Operand Syntax | Operand Format | Operation |
|-------------|---------------------|--------------------------|-----------------------------|
| F<dop> | <ea>,FPn FPm,FPn | B, W, L, S, D, X, P X | FPn <Function> Source → FPn |

NOTE: < dop > is any one of the dyadic operation specifiers.

Table 3-15. Dyadic Floating-Point Operations

| Instruction | Operation |
|--------------------|-----------------------------------|
| FADD, FSADD, FDADD | Add |
| FCMP | Compare |
| FDIV, FSDIV, FDDIV | Divide |
| FMOD | Modulo Remainder |
| FMUL, FSMUL, FDMUL | Multiply |
| FREM | IEEE Remainder |
| FSCALE | Scale Exponent |
| FSUB, FSSUB, FDSUB | Subtract |
| FSGLDIV, FSGLMUL | Single-Precision Divide, Multiply |

The monadic floating-point instructions provide several arithmetic functions requiring only one input operand. Unlike the integer counterparts to these functions (e.g., NEG < ea >), a source and a destination can be specified. The operation is performed on the source operand and the result is stored in the destination, which is always a floating-point data register. When the source is not a floating-point data register, all data formats are supported. The data format is always extended precision for register-to-register operations. Table 3-16 lists the general format of these instructions, and Table 3-17 lists the available operations.

Table 3-16. Monadic Floating-Point Operation Format

| Instruction | Operand Syntax | Operand Format | Operation |
|-------------|----------------------------|-------------------------------|---|
| F<mop> | <ea>,FPn FPm,FPn FPn | B, W, L, S, D, X, P X X | Source → Function → FPn FPn → Function → FPn |

NOTE: < mop > is any one of the monadic operation specifiers.

Table 3-17. Monadic Floating-Point Operations

| Instruction | Operation | Instruction | Operation |
|-------------|---------------------------------------|-------------|--------------------|
| FABS | Absolute Value | FLOGN | $\ln(x)$ |
| FACOS | Arc Cosine | FLOGNP1 | $\ln(x + 1)$ |
| FASIN | Arc Sine | FLOG10 | $\log_{10}(x)$ |
| FATAN | Hyperbolic Art Tangent | FLOG2 | $\log_2(x)$ |
| FCOS | Cosine | FNEG | Negate |
| FCOSH | Hyperbolic Cosine | FSIN | Sine |
| FETOX | e^x | FSINH | Hyperbolic Sine |
| FETOXM1 | $e^x - 1$ | FSQRT | Square Root |
| FGETEXP | Extract Exponent | FTAN | Tangent |
| FGETMAN | Extract Mantissa | FTANH | Hyperbolic Tangent |
| FINT | Extract Integer Part | FTENTOX | 10^x |
| FINTRZ | Extract Integer Part, Rounded-to-Zero | FTWOTOX | 2^x |

3.2 INTEGER UNIT CONDITION CODE COMPUTATION

Many integer instructions affect the CCR to indicate the instruction's results. Program and system control instructions also use certain combinations of these bits to control program and system flow. The condition codes meet consistency criteria across instructions, uses, and instances. They also meet the criteria of meaningful results, where no change occurs unless it provides useful information. Refer to **Section 1 Introduction** for details concerning the CCR.

Table 3-18 lists the integer condition code computations for instructions and Table 3-19 lists the condition names, encodings, and tests for the conditional branch and set instructions. The test associated with each condition is a logical formula using the current states of the condition codes. If this formula evaluates to one, the condition is true. If the formula evaluates to zero, the condition is false. For example, the T condition is always true, and the EQ condition is true only if the Z-bit condition code is currently true.

Table 3-18. Integer Unit Condition Code Computations

| Operations | X | N | Z | V | C | Special Definition |
|---|---|---|---|---|---|---|
| ABCD | * | U | ? | U | ? | C = Decimal Carry Z = $Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$ |
| ADD, ADDI, ADDQ | * | * | * | ? | ? | V = $Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ C = $Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$ |
| ADDX | * | * | ? | ? | ? | V = $Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ C = $Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$ Z = $Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$ |
| AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, EXTB, NOT, TAS, TST | — | * | * | 0 | 0 | |
| CHK | — | * | U | U | U | |
| CHK2, CMP2 | — | U | ? | U | ? | Z = $(R = LB) \vee (R = UB)$ C = $(LB \leq UB) \wedge (IR < LB) \vee (R > UB)$ $\vee (UB < LB) \wedge (R > UB) \wedge (R < LB)$ |
| SUB, SUBI, SUBQ | * | * | * | ? | ? | V = $\overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ C = $Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ |
| SUBX | * | * | ? | ? | ? | V = $\overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ C = $Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ Z = $Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$ |
| CAS, CAS2, CMP, CMPA, CMPI, CMPM | — | * | * | ? | ? | V = $\overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ C = $Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ |
| DIVS, DUVU | — | * | * | ? | 0 | V = Division Overflow |
| MULS, MULU | — | * | * | ? | 0 | V = Multiplication Overflow |
| SBCD, NBCD | * | U | ? | U | ? | C = Decimal Borrow Z = $Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$ |
| NEG | * | * | * | ? | ? | V = $Dm \wedge Rm$ C = $Dm \vee Rm$ |
| NEGX | * | * | ? | ? | ? | V = $Dm \wedge Rm$ C = $Dm \vee Rm$ Z = $Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$ |
| BTST, BCHG, BSET, BCLR | — | — | ? | — | — | Z = \overline{Dn} |
| BFTST, BFCHG, BFSET, BFCLR | — | ? | ? | 0 | 0 | N = Dm Z = $\overline{Dn} \wedge \overline{Dm-1} \wedge \dots \wedge \overline{D0}$ |
| BFEXTS, BFEXTU, BFFFO | — | ? | ? | 0 | 0 | N = Sm Z = $Sm \wedge \overline{Sm-1} \wedge \dots \wedge \overline{S0}$ |
| BFINS | — | ? | ? | 0 | 0 | N = Dm Z = $\overline{Dm} \wedge \overline{Dm-1} \wedge \dots \wedge \overline{D0}$ |
| ASL | * | * | * | ? | ? | V = $Dm \wedge \overline{Dm-1} \vee \dots \vee \overline{Dm-r} \vee \overline{Dm} \wedge$ $(\overline{Dm-1} \vee \dots \vee \overline{Dm-r})$ C = $\overline{Dm-r+1}$ |
| ASL (r = 0) | — | * | * | 0 | 0 | |
| LSL, ROXL | * | * | * | 0 | ? | C = $Dm - r + 1$ |

Table 3-18. Integer Unit Condition Code Computations (Continued)

| Operations | X | N | Z | V | C | Special Definition |
|------------------|---|---|---|---|---|------------------------|
| LSR (r = 0) | — | * | * | 0 | 0 | |
| ROXL (r = 0) | — | * | * | 0 | ? | X = C |
| ROL | — | * | * | 0 | ? | C = D _{m-r+1} |
| ROL (r = 0) | — | * | * | 0 | 0 | |
| ASR, LSR, ROXR | * | * | * | 0 | ? | C = D _{r-1} |
| ASR, LSR (r = 0) | — | * | * | 0 | 0 | |
| ROXR (r = 0) | — | * | * | 0 | ? | X = C |
| ROR | — | * | * | 0 | ? | C = D _{r-1} |
| ROR (r = 0) | — | * | * | 0 | 0 | |

? = Other—See Special Definition

N = Result Operand (MSB)

Z = $\overline{R_m} \wedge \dots \wedge \overline{R_0}$

S_m = Source Operand (MSB)

D_m = Destination Operand (MSB)

R_m = Result Operand (MSB)

$\overline{R_m}$ = Not Result Operand (MSB)

R = Register Tested

r = Shift Count

Table 3-19. Conditional Tests

| Mnemonic | Condition | Encoding | Test |
|----------|------------------|----------|--|
| T* | True | 0000 | 1 |
| F* | False | 0001 | 0 |
| HI | High | 0010 | $\overline{C} \wedge \overline{Z}$ |
| LS | Low or Same | 0011 | C V Z |
| CC(HI) | Carry Clear | 0100 | C |
| CS(LO) | Carry Set | 0101 | C |
| NE | Not Equal | 0110 | Z |
| EQ | Equal | 0111 | Z |
| VC | Overflow Clear | 1000 | V |
| VS | Overflow Set | 1001 | V |
| PL | Plus | 1010 | N |
| MI | Minus | 1011 | N |
| GE | Greater or Equal | 1100 | $N \wedge V \vee \overline{N} \wedge \overline{V}$ |
| LT | Less Than | 1101 | $N \wedge \overline{V} \vee \overline{N} \wedge V$ |
| GT | Greater Than | 1110 | $N \wedge V \wedge \overline{Z} \vee \overline{N} \wedge \overline{V} \wedge \overline{Z}$ |
| LE | Less or Equal | 1111 | $Z \vee N \wedge \overline{V} \vee \overline{N} \wedge V$ |

NOTES:

\overline{N} = Logical Not N

\overline{V} = Logical Not V

\overline{Z} = Logical Not Z

*Not available for the Bcc instruction.

3.3 INSTRUCTION EXAMPLES

The following paragraphs provide examples of how to use selected instructions.

3.3.1 Using the Cas and Cas2 Instructions

The CAS instruction compares the value in a memory location with the value in a data register, and copies a second data register into the memory location if the compared values are equal. This provides a means of updating system counters, history information, and globally shared pointers. The instruction uses an indivisible read-modify-write cycle. After CAS reads the memory location, no other instruction can change that location before CAS has written the new value. This provides security in single-processor systems, in multitasking environments, and in multiprocessor environments. In a single-processor system, the operation is protected from instructions of an interrupt routine. In a multitasking environment, no other task can interfere with writing the new value of a system variable. In a multiprocessor environment, the other processors must wait until the CAS instruction completes before accessing a global pointer.

3.3.2 Using the Moves Instruction

This instruction moves the byte, word, or long-word operand from the specified general register to a location within the address space specified by the destination function code (DFC) register. It also moves the byte, word, or long-word operand from a location within the address space specified by the source function code (SFC) register to the specified general register.

3.3.3 Nested Subroutine Calls

The LINK instruction pushes an address onto the stack, saves the stack address at which the address is stored, and reserves an area of the stack. Using this instruction in a series of subroutine calls results in a linked list of stack frames.

The UNLK instruction removes a stack frame from the end of the list by loading an address into the stack pointer and pulling the value at that address from the stack. When the operand of the instruction is the address of the link address at the bottom of a stack frame, the effect is to remove the stack frame from the stack and from the linked list.

3.3.4 Bit Field Instructions

One of the data types provided by the MC68030 is the bit field, consisting of as many as 32 consecutive bits. An offset from an effective address and a width value defines a bit field. The offset is a value in the range of -231 through $231 - 1$ from the most significant bit (bit 7) at the effective address. The width is a positive number, 1 through 32. The most significant bit of a bit field is bit 0. The bits number in a direction opposite to the bits of an integer.

The instruction set includes eight instructions that have bit field operands. The insert bit field (BFINS) instruction inserts a bit field stored in a register into a bit field. The extract bit field signed (BFEXTS) instruction loads a bit field into the least significant bits of a register and

extends the sign to the left, filling the register. The extract bit field unsigned (BFEXTU) also loads a bit field, but zero fills the unused portion of the destination register.

The set bit field (BFSET) instruction sets all the bits of a field to ones. The clear bit field (BFCLR) instruction clears a field. The change bit field (BFCHG) instruction complements all the bits in a bit field. These three instructions all test the previous value of the bit field, setting the condition codes accordingly. The test bit field (BFTST) instruction tests the value in the field, setting the condition codes appropriately without altering the bit field. The find first one in bit field (BFFFO) instruction scans a bit field from bit 0 to the right until it finds a bit set to one and loads the bit offset of the first set bit into the specified data register. If no bits in the field are set, the field offset and the field width is loaded into the register.

An important application of bit field instructions is the manipulation of the exponent field in a floating-point number. In the IEEE standard format, the most significant bit is the sign bit of the mantissa. The exponent value begins at the next most significant bit position; the exponent field does not begin on a byte boundary. The extract bit field (BFEXTU) instruction and the BFTST instruction are the most useful for this application, but other bit field instructions can also be used.

Programming of input and output operations to peripherals requires testing, setting, and inserting of bit fields in the control registers of the peripherals. This is another application for bit field instructions. However, control register locations are not memory locations; therefore, it is not always possible to insert or extract bit fields of a register without affecting other fields within the register.

Another widely used application for bit field instructions is bit- mapped graphics. Because byte boundaries are ignored in these areas of memory, the field definitions used with bit field instructions are very helpful.

3.3.5 Pipeline Synchronization with the Nop Instruction

Although the no operation (NOP) instruction performs no visible operation, it serves an important purpose. It forces synchronization of the integer unit pipeline by waiting for all pending bus cycles to complete. All previous integer instructions and floating-point external operand accesses complete execution before the NOP begins. The NOP instruction does not synchronize the FPU pipeline—floating- point instructions with floating-point register operand destinations can be executing when the NOP begins. NOP is considered a change of flow instruction and traps for trace on change of flow. A single- cycle nonsynchronizing operation can be affected with the TRAPF instruction.

3.4 FLOATING-POINT INSTRUCTION DETAILS

The following paragraphs describe the operation tables used in the instruction descriptions and the conditional tests that can be used to change program flow based on floating-point conditions. Details on NaNs and floating-point condition codes are also discussed. The IEEE 754 standard specifies that each data format must support add, subtract, multiply, divide, remainder, square root, integer part, and compare. In addition to these arithmetic

functions, software supports remainder and integer part; the FPU also supports the nontranscendental operations of absolute value, negate, and test.

Most floating-point instruction descriptions include an operation table. This table lists the resulting data types for the instruction based on the operand,s input. Table 3-20 is an operation table example for the FADD instruction. The operation table lists the source operand type along the top, and the destination operand type along the side. In-range numbers are normalized, denormalized, unnormalized real numbers, or integers that are converted to normalized or denormalized extended-precision numbers upon entering the FPU.

Table 3-20. Operation Table Example (FADD Instruction)

| DESTINATION | SOURCE ¹ | | |
|-------------|---------------------|---------------------------|--|
| | + In Range | - + Zero - | + Infinity - |
| In Range | + ADD - | ADD | +inf -inf |
| Zero | + ADD - | + 0.0 0.0 ² | 0.0 ² -0.0 +inf -inf |
| Infinity | + +inf - -inf | +inf -inf | +inf NAN ³ NAN ³ -inf |

NOTES:

- 1.If either operand is a NAN, refer to **1.6.5 NANs** for more information.
- 2.Returns +0.0 in rounding modes RN, RZ, and RP; returns -0.0 in RM.
- 3.Sets the OPERR bit in the FPSR exception byte.

For example, Table 3-20 illustrates that if both the source and destination operand are positive zero, the result is also a positive zero. If the source operand is a positive zero and the destination operand is an in-range number, then the ADD algorithm is executed to obtain the result. If a label such as ADD appears in the table, it indicates that the FPU performs the indicated operation and returns the correct result. Since the result of such an operation is undefined, a NAN is returned as the result, and the OPERR bit is set in the FPSR EXC byte.

In addition to the data types covered in the operation tables for each floating-point instruction, NANs can also be used as inputs to an arithmetic operation. The operation tables do not contain a row and column for NANs because NANs are handled the same way for all operations. If either operand, but not both operands, of an operation is a nonsignaling NAN, then that NAN is returned as the result. If both operands are nonsignaling NANs, then the destination operand nonsignaling NAN is returned as the result.

If either operand to an operation is a signaling NAN (SNAN), then the SNAN bit is set in the FPSR EXC byte. If the SNAN exception enable bit is set in the FPCR ENABLE byte, then the exception is taken and the destination is not modified. If the SNAN exception enable bit is not set, setting the SNAN bit in the operand to a one converts the SNAN to a nonsignaling NAN. The operation then continues as described in the preceding paragraph for nonsignaling NANs.

3.5 FLOATING-POINT COMPUTATIONAL ACCURACY

Representing a real number in a binary format of finite precision is problematic. If the number cannot be represented exactly, a round-off error occurs. Furthermore, when two of these inexact numbers are used in a calculation, the result becomes even more inexact. The IEEE 754 standard defines the error bounds for calculating binary floating-point values so that the result obtained by any conforming device can be predicted exactly for a particular precision and rounding mode. The error bound defined by the IEEE 754 standard is one-half unit in the last place of the destination data format in the RN mode, and one unit in last place in the other rounding modes. The operation's data format must have the same input values, rounding mode, and precision. The standard also specifies the maximum allowable error that can be introduced during a calculation and the manner in which rounding of the result is performed.

The single- and double-precision formats provide emulation for devices that only support those precisions. The execution speed of all instructions is the same whether using single- or double-precision rounding. When using these two data formats, the FPU produces the same results as any other device that conforms to the IEEE standard but does not support extended precision. The results are the same when performing the same operation in extended precision and storing the results in single- or double-precision format.

The FPU performs all floating-point internal operations in extended-precision. It supports mixed-mode arithmetic by converting single- and double-precision operands to extended-precision values before performing the specified operation. The FPU converts all memory data formats to the extended-precision data format and stores the value in a floating-point register or uses it as the source operand for an arithmetic operation. The FPU also converts extended-precision data formats in a floating-point data register to any data format and either stores it in a memory destination or in an integer data register.

Additionally if the external operand is a denormalized number, the number is normalized before an operation is performed. However, an external denormalized number moved into a floating-point data register is stored as a denormalized number. The number is first normalized and then denormalized before it is stored in the designated floating-point data register. This method simplifies the handling of all other data formats and types.

If an external operand is an unnormalized number, the number is normalized before it is used in an arithmetic operation. If the external operand is an unnormalized zero (i.e., with a mantissa of all zeros), the number is converted to a normalized zero before the specified operation is performed. The regular use of unnormalized inputs not only defeats the purpose of the IEEE 754 standard, but also can produce gross inaccuracies in the results.

3.5.1 Intermediate Result

All FPU calculations use an intermediate result. When the FPU performs any operation, the calculation is carried out using extended-precision inputs, and the intermediate result is calculated as if to produce infinite precision. After the calculation is complete, the intermediate result is rounded to the selected precision and stored in the destination.

Figure 3-1 illustrates the intermediate result format. The intermediate result's exponent for some dyadic operations (i.e., multiply and divide) can easily overflow or underflow the 15-bit exponent of the destination floating-point register. To simplify the overflow and underflow detection, intermediate results in the FPU maintain a 16-bit (17 bits for the MC68881 and MC68882), two's complement, integer exponent. Detection of an overflow or underflow intermediate result always converts the 16-bit exponent into a 15-bit biased exponent before being stored in a floating-point data register. The FPU internally maintains the 67-bit mantissa for rounding purposes. The mantissa is always rounded to 64 bits (or less, depending on the selected rounding precision) before it is stored in a floating-point data register.

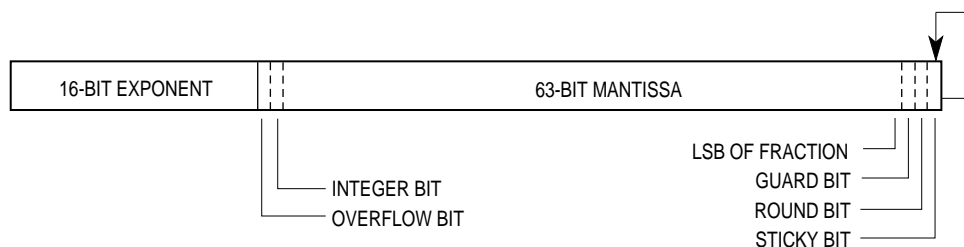


Figure 3-1. Intermediate Result Format

If the destination is a floating-point data register, the result is in the extended-precision format and is rounded to the precision specified by the FPSR PREC bits before being stored. All mantissa bits beyond the selected precision are zero. If the single- or double-precision mode is selected, the exponent value is in the correct range even if it is stored in extended-precision format. If the destination is a memory location, the FPSR PREC bits are ignored. In this case, a number in the extended-precision format is taken from the source floating-point data register, rounded to the destination format precision, and then written to memory.

Depending on the selected rounding mode or destination data format in effect, the location of the least significant bit of the mantissa and the locations of the guard, round, and sticky bits in the 67-bit intermediate result mantissa varies. The guard and round bits are always calculated exactly. The sticky bit is used to create the illusion of an infinitely wide intermediate result. As the arrow illustrates in Figure 3-1, the sticky bit is the logical OR of all the bits in the infinitely precise result to the right of the round bit. During the calculation stage of an arithmetic operation, any non-zero bits generated that are to the right of the round bit set the sticky bit to one. Because of the sticky bit, the rounded intermediate result for all required IEEE arithmetic operations in the RN mode is in error by no more than one half unit in the last place.

3.5.2 Rounding the Result

The FPU supports the four rounding modes specified by the IEEE 754 standard. These modes are round to nearest (RN), round toward zero (RZ), round toward plus infinity (RP), and round toward minus infinity (RM). The RM and RP rounding modes are often referred to as "directed rounding modes" and are useful in interval arithmetic. Rounding is accomplished through the intermediate result. Single-precision results are rounded to a 24-bit boundary; double-precision results are rounded to a 53-bit boundary; and extended-precision results are rounded to a 64-bit boundary. Table 3-21 lists the encodings for the FPCR that denote the rounding and precision modes.

Table 3-21. FPCR Encodings

| Rounding Mode (RND Field) | Encoding | | Rounding Precision (PREC Field) |
|------------------------------|----------|---|------------------------------------|
| To Nearest (RN) | 0 | 0 | Extend (X) |
| To Zero (RZ) | 0 | 1 | Single (S) |
| To Minus Infinity (RM) | 1 | 0 | Double (D) |
| To Plus Infinity (RP) | 1 | 1 | Undefined |

Rounding the intermediate result's mantissa to the specified precision and checking the 16-bit intermediate exponent to ensure that it is within the representable range of the selected rounding precision accomplishes range control. Range control is a method used to assure correct emulation of a device that only supports single- or double- precision arithmetic. If the intermediate result's exponent exceeds the range of the selected precision, the exponent value appropriate for an underflow or overflow is stored as the result in the 16-bit extended-precision format exponent. For example, if the data format and rounding mode is single precision RM and the result of an arithmetic operation overflows the magnitude of the single-precision format, the largest normalized single-precision value is stored as an extended-precision number in the destination floating-point data register (i.e., an unbiased 15-bit exponent of \$00FF and a mantissa of \$FFFFFF0000000000). If an infinity is the appropriate result for an underflow or overflow, the infinity value for the destination data format is stored as the result (i.e., an exponent with the maximum value and a mantissa of zero).

Figure 3-2 illustrates the algorithm that the FPU uses to round an intermediate result to the selected rounding precision and destination data format. If the destination is a floating-point register, either the selected rounding precision specified by the FPCR PREC status byte or by the instruction itself determines the rounding boundary. For example, FSADD and FDADD specify single- and double-precision rounding regardless of the precision specified in the FPCR PREC status byte. If the destination is external memory or an integer data register, the destination data format determines the rounding boundary. If the rounded result of an operation is not exact, then the INEX2 bit is set in the FPSR EXC status byte.

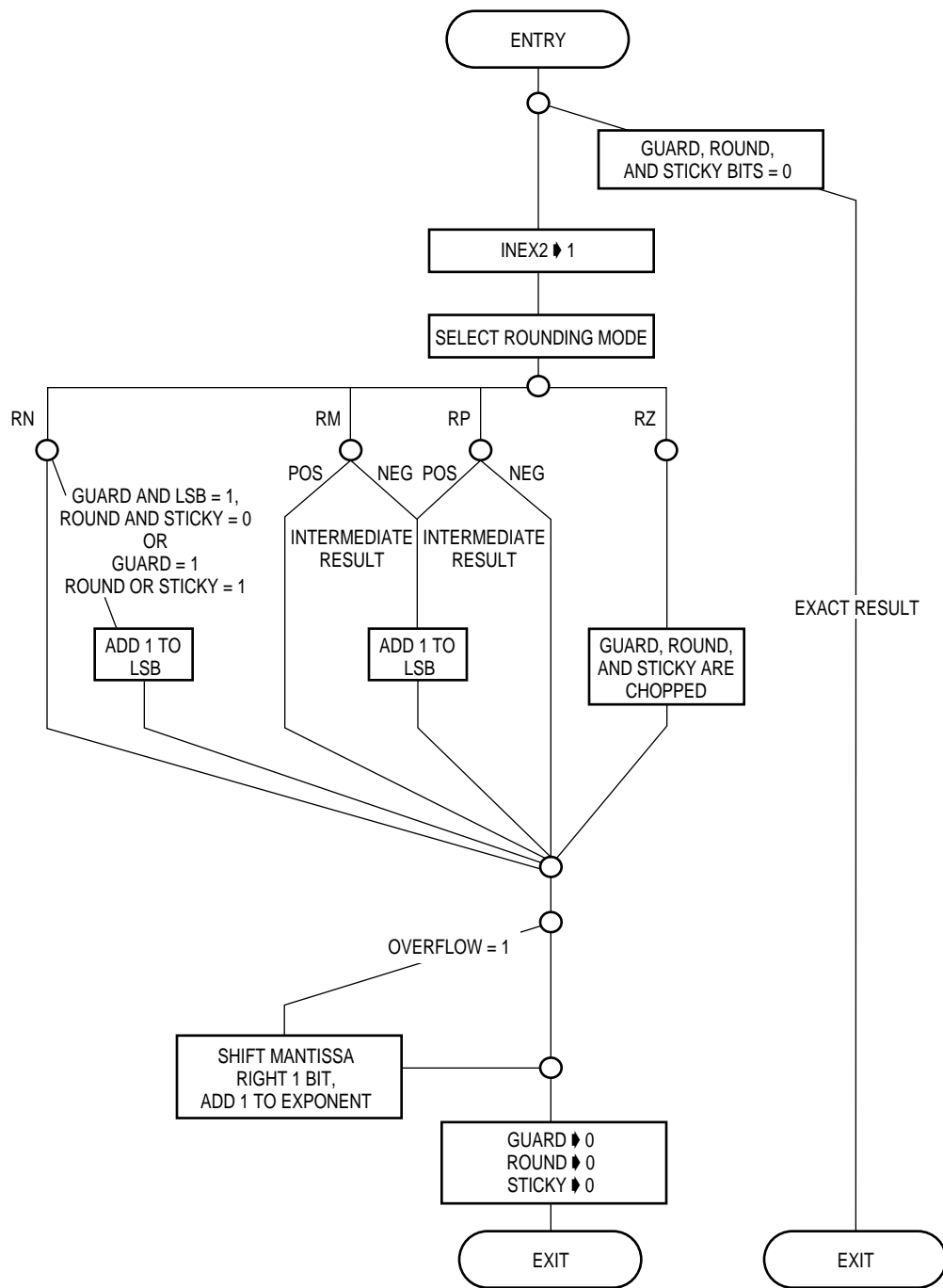


Figure 3-2. Rounding Algorithm Flowchart

The three additional bits beyond the extended-precision format, the difference between the intermediate result's 67-bit mantissa and the storing result's 64-bit mantissa, allow the FPU to perform all calculations as though it were performing calculations using a float engine with infinite bit prec. The result is always correct for the specified destination's data format before performing rounding (unless an overflow or underflow error occurs). The specified rounding operation then produces a number that is as close as possible to the infinitely precise

intermediate value and still representable in the The following tie-case example shows how the 67-bit mantissa allows the FPU to meet the error bound of the IEEE specification:

| Result | Integer | 63-Bit Fraction | Guard | Round | Sticky |
|--------------------|---------|-----------------|-------|-------|--------|
| Intermediate | x | xxx...x00 | 1 | 0 | 0 |
| Rounded-to-Nearest | x | xxx...x00 | 0 | 0 | 0 |

The LSB of the rounded result does not increment though the guard bit is set in the intermediate result. The IEEE 754 standard specifies that tie cases should be handled in this manner. If the destination data format is extended and there is a difference between the infinitely precise intermediate result and the round-to-nearest result, the relative difference is 2^{-64} (the value of the guard bit). This error is equal to half of the least significant bit's value and is the worst case error that can be introduced when using the RN mode. Thus, the term one-half unit in the last place correctly identifies the error bound for this operation. This error specification is the relative error present in the result; the absolute error bound is equal to $2^{\text{exponent}} \times 2^{-64}$. The following example shows the error bound for the other rounding modes:

| Result | Integer | 63-Bit Fraction | Guard | Round | Sticky |
|--------------------|---------|-----------------|-------|-------|--------|
| Intermediate | x | xxx...x00 | 1 | 1 | 1 |
| Rounded-to-Nearest | x | xxx...x00 | 0 | 0 | 0 |

The difference between the infinitely precise result and the rounded result is $2^{-64} + 2^{-65} + 2^{-66}$, which is slightly less than 2^{-63} (the value of the LSB). Thus, the error bound for this operation is not more than one unit in the last place. For all arithmetic operations, the FPU meets these error bounds, providing accurate and repeatable results.

3.6 FLOATING-POINT POSTPROCESSING

Most operations end with a postprocessing step. The FPU provides two steps in postprocessing. First, the condition code bits in the FPSR are set or cleared at the end of each arithmetic operation or move operation to a single floating-point data register. The condition code bits are consistently set based on the result of the operation. Second, the FPU supports 32 conditional tests that allow floating-point conditional instructions to test floating-point conditions in exactly the same way as the integer conditional instructions test the integer condition code. The combination of consistently set condition code bits and the simple programming of conditional instructions gives the processor a very flexible, high-performance method of altering program flow based on floating-point results. While reading the summary for each instruction, it should be assumed that an instruction performs postprocessing unless the summary specifically states that the instruction does not do so. The following paragraphs describe postprocessing in detail.

3.6.1 Underflow, Round, Overflow

During the calculation of an arithmetic result, the FPU arithmetic logic unit (ALU) has more precision and range than the 80-bit extended precision format. However, the final result of these operations is an extended-precision floating-point value. In some cases, an intermediate result becomes either smaller or larger than can be represented in extended precision. Also, the operation can generate a larger exponent or more bits of precision than can be represented in the chosen rounding precision. For these reasons, every arithmetic instruction ends by rounding the result and checking for overflow and underflow.

At the completion of an arithmetic operation, the intermediate result is checked to see if it is too small to be represented as a normalized number in the selected precision. If so, the underflow (UNFL) bit is set in the FPSR EXC byte. It is also denormalized unless denormalization provides a zero value. Denormalizing a number causes a loss of accuracy, but a zero is not returned unless absolutely necessary. If a number is grossly underflowed, the FPU returns a zero or the smallest denormalized number with the correct sign, depending on the rounding mode in effect.

If no underflow occurs, the intermediate result is rounded according to the user-selected rounding precision and rounding mode. After rounding, the inexact bit (INEX2) is set appropriately. Lastly, the magnitude of the result is checked to see if it is too large to be represented in the current rounding precision. If so, the overflow (OVFL) bit is set and a correctly signed infinity or correctly signed largest normalized number is returned, depending on the rounding mode in effect.

3.6.2 Conditional Testing

Unlike the integer arithmetic condition codes, an instruction either always sets the floating-point condition codes in the same way or it does not change them at all. Therefore, the instruction descriptions do not include floating-point condition code settings. The following paragraphs describe how floating-point condition codes are set for all instructions that modify condition codes.

The condition code bits differ slightly from the integer condition codes. Unlike the operation type dependent integer condition codes, examining the result at the end of the operation sets or clears the floating-point condition codes accordingly. The M68000 family integer condition codes bits N and Z have this characteristic, but the V and C bits are set differently for different instructions. The data type of the operation's result determines how the four condition code bits are set. Table 3-22 lists the condition code bit setting for each data type. Loading the FPCC with one of the other combinations and executing a conditional instruction can produce an unexpected branch condition.

Table 3-22. FPCC Encodings

| Data Type | N | Z | I | NAN |
|------------------------------|---|---|---|-----|
| + Normalized or Denormalized | 0 | 0 | 0 | 0 |
| – Normalized or Denormalized | 1 | 0 | 0 | 0 |
| + 0 | 0 | 1 | 0 | 0 |
| – 0 | 1 | 1 | 0 | 0 |
| + Infinity | 0 | 0 | 1 | 0 |
| – Infinity | 1 | 0 | 1 | 0 |
| + NAN | 0 | 0 | 0 | 1 |
| – NAN | 1 | 0 | 0 | 1 |

The inclusion of the NAN data type in the IEEE floating-point number system requires each conditional test to include the NAN condition code bit in its Boolean equation. Because a comparison of a NAN with any other data type is unordered (i.e., it is impossible to determine if a NAN is bigger or smaller than an in-range number), the compare instruction sets the NAN condition code bit when an unordered compare is attempted. All arithmetic instructions also set the NAN bit if the result of an operation is a NAN. The conditional instructions interpret the NAN condition code bit equal to one as the unordered condition.

The IEEE 754 standard defines four conditions: equal to (EQ), greater than (GT), less than (LT), and unordered (UN). In addition, the standard only requires the generation of the condition codes as a result of a floating-point compare operation. The FPU can test these conditions at the end of any operation affecting the condition codes. For purposes of the floating-point conditional branch, set byte on condition, decrement and branch on condition, and trap on condition instructions, the processor logically combines the four FPCC condition codes to form 32 conditional tests. There are three main categories of conditional tests: IEEE nonaware tests, IEEE aware tests, and miscellaneous. The set of IEEE nonaware tests is best used:

- when porting a program from a system that does not support the IEEE standard to a conforming system, or
- when generating high-level language code that does not support IEEE floating-point concepts (i.e., the unordered condition).

The 32 conditional tests are separated into two groups; 16 that cause an exception if an unordered condition is present when the conditional test is attempted and 16 that do not cause an exception. An unordered condition occurs when one or both of the operands in a floating-point compare operation. The inclusion of the unordered condition in floating-point branches destroys the familiar trichotomy relationship (greater than, equal, less than) that exists for integers. For example, the opposite of floating-point branch greater than (FBGT) is not floating-point branch less than or equal (FBLE). Rather, the opposite condition is floating-point branch not greater than (FBNGT). If the result of the previous instruction was unordered, FBNGT is true; whereas, both FBGT and FBLE would be false since unordered fails both of these tests (and sets BSUN). Compiler programmers should be particularly careful of the lack of trichotomy in the floating-point branches since it is common for compilers to invert the sense of conditions.

When using the IEEE nonaware tests, the user receives a BSUN exception whenever a branch is attempted and the NAN condition code bit is set, unless the branch is an FBEQ or an FBNE. If the BSUN exception is enabled in the FPCR, the exception causes another exception. Therefore, the IEEE nonaware program is interrupted if an unexpected condition occurs. Compilers and programmers who are knowledgeable of the IEEE 754 standard should use the IEEE aware tests in programs that contain ordered and unordered conditions. Since the ordered or unordered attribute is explicitly included in the conditional test, the BSUN bit is not set in the FPSR EXC byte when the unordered condition occurs. Table 3-23 summarizes the conditional mnemonics, definitions, equations, predicates, and whether the BSUN bit is set in the FPSR EXC byte for the 32 floating-point conditional tests. The equation column lists the combination of FPCC bits for each test in the form of an equation. All condition codes with an overbar indicate cleared bits; all other bits are set.

Table 3-23. Floating-Point Conditional Tests

| Mnemonic | Definition | Equation | Predicate | BSUN Bit Set |
|----------------------------|-------------------------------|--|-----------|--------------|
| IEEE Nonaware Tests | | | | |
| EQ | Equal | Z | 000001 | No |
| NE | Not Equal | Z | 001110 | No |
| GT | Greater Than | $\overline{NAN} \vee \overline{Z} \vee \overline{N}$ | 010010 | Yes |
| NGT | Not Greater Than | $NAN \vee Z \vee N$ | 011101 | Yes |
| GE | Greater Than or Equal | $Z \vee (\overline{NAN} \vee \overline{N})$ | 010011 | Yes |
| NGE | Not Greater Than or Equal | $NAN \vee (N \wedge \overline{Z})$ | 011100 | Yes |
| LT | Less Than | $N \wedge (\overline{NAN} \vee \overline{Z})$ | 010100 | Yes |
| NLT | Not Less Than | $NAN \vee (Z \vee \overline{N})$ | 011011 | Yes |
| LE | Less Than or Equal | $Z \vee (N \wedge \overline{NAN})$ | 010101 | Yes |
| NLE | Not Less Than or Equal | $NAN \vee (\overline{N} \vee \overline{Z})$ | 011010 | Yes |
| GL | Greater or Less Than | $\overline{NAN} \vee \overline{Z}$ | 010110 | Yes |
| NGL | Not Greater or Less Than | $NAN \vee Z$ | 011001 | Yes |
| GLE | Greater, Less or Equal | NAN | 010111 | Yes |
| NGLE | Not Greater, Less or Equal | NAN | 011000 | Yes |
| IEEE Aware Tests | | | | |
| EQ | Equal | Z | 000001 | No |
| NE | Not Equal | Z | 001110 | No |
| OGT | Ordered Greater Than | $\overline{NAN} \vee \overline{Z} \vee \overline{N}$ | 000010 | No |
| ULE | Unordered or Less or Equal | $NAN \vee Z \vee N$ | 001101 | No |
| OGE | Ordered Greater Than or Equal | $Z \vee (\overline{NAN} \vee \overline{N})$ | 000011 | No |
| ULT | Unordered or Less Than | $NAN \vee (N \wedge \overline{Z})$ | 001100 | No |
| OLT | Ordered Less Than | $N \wedge (\overline{NAN} \vee \overline{Z})$ | 000100 | No |
| UGE | Unordered or Greater or Equal | $NAN \vee Z \vee N$ | 001011 | No |
| OLE | Ordered Less Than or Equal | $Z \vee (N \wedge \overline{NAN})$ | 000101 | No |
| UGT | Unordered or Greater Than | $NAN \vee (\overline{N} \vee \overline{Z})$ | 001010 | No |
| OGL | Ordered Greater or Less Than | $\overline{NAN} \vee \overline{Z}$ | 000110 | No |
| UEQ | Unordered or Equal | $NAN \vee Z$ | 001001 | No |
| OR | Ordered | NAN | 000111 | No |
| UN | Unordered | NAN | 001000 | No |
| Miscellaneous Tests | | | | |
| F | False | False | 000000 | No |
| T | True | True | 001111 | No |
| SF | Signaling False | False | 010000 | Yes |
| ST | Signaling True | True | 011111 | Yes |
| SEQ | Signaling Equal | Z | 010001 | Yes |
| SNE | Signaling Not Equal | Z | 011110 | Yes |

3.7 INSTRUCTION DESCRIPTIONS

Section 4, 5, 6, and 7 contain detailed information about each instruction in the M68000 family instruction set. Each section arranges the instruction in alphabetical order by instruction mnemonic and includes descriptions of the instruction's notation and format. Figure 3-3 illustrates the format of the instruction descriptions. Note that the illustration is an amalgamation of the various parts that make up an instruction description. Instruction descriptions for the integer unit differ slightly from those for the floating-point unit; i.e. there are no operation tables included for integer unit instruction descriptions.

The size attribute line specifies the size of the operands of an instruction. When an instruction uses operands of more than one size, the mnemonic of the instruction includes a suffix such as:

- .B—Byte Operands
- .W—Word Operands
- .L—Long-Word Operands
- .S—Single-Precision Real Operands
- .D—Double-Precision Real Operands
- .X—Extended-Precision Real Operands
- .P—Packed BCD Real Operands

The instruction format specifies the bit pattern and fields of the operation and command words, and any other words that are always part of the instruction. The effective address extensions are not explicitly illustrated. The extension words, if any, follow immediately after the illustrated portions of the instructions.

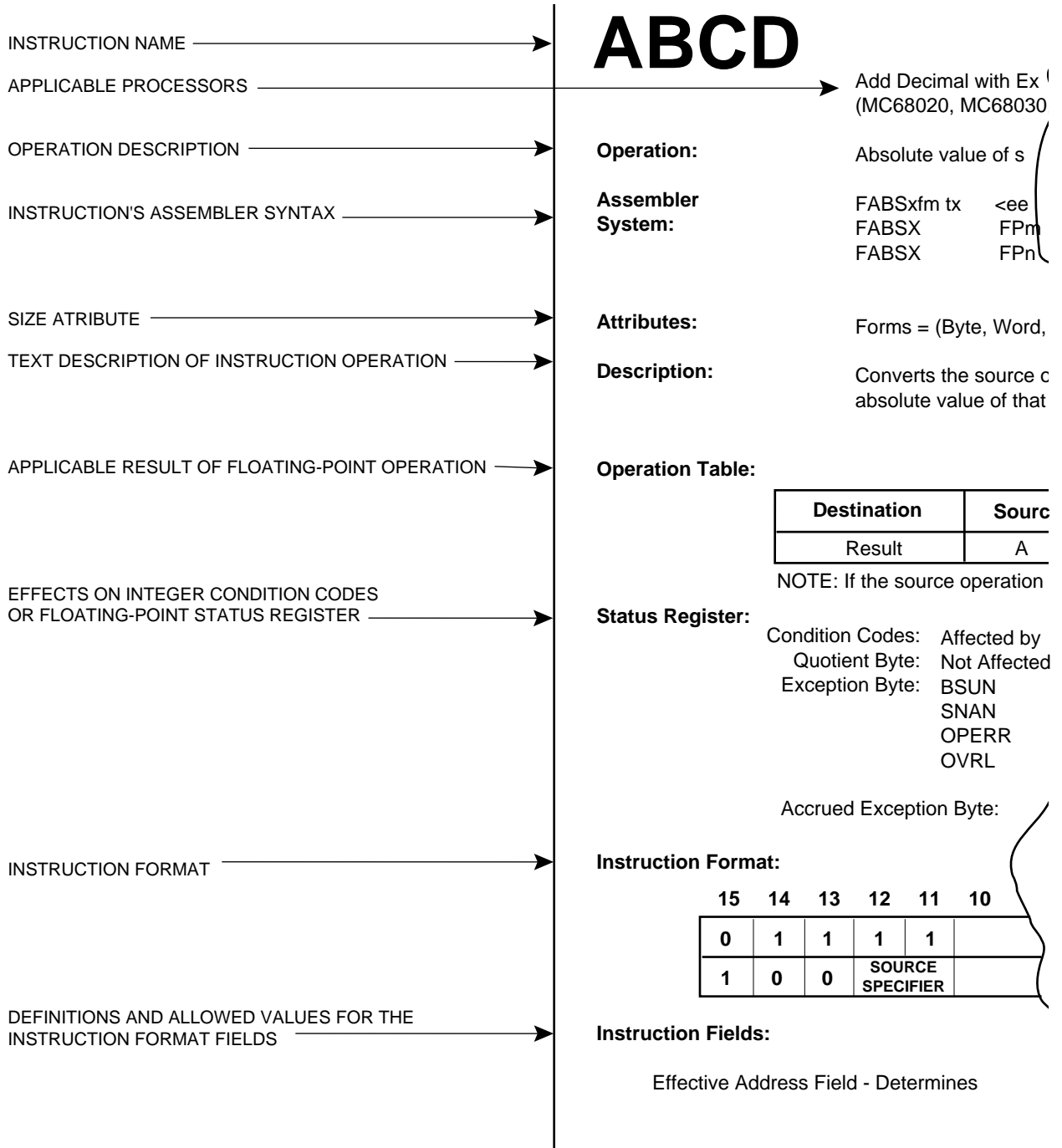


Figure 3-3. Instruction Description Format

SECTION 4

INTEGER INSTRUCTIONS

This section contains detailed information about the integer instructions for the M68000 family. A detailed discussion of each instruction description is arranged in alphabetical order by instruction mnemonic.

Each instruction description identifies the differences among the M68000 family for that instruction. Noted under the title of the instruction are all specific processors that apply to that instruction—for example:

Test Bit Field and Change (MC68030, MC68040)

The MC68HC000 is identical to the MC68000 except for power dissipation; therefore, all instructions that apply to the MC68000 also apply to the MC68HC000. All references to the MC68000, MC68020, and MC68030 include references to the corresponding embedded controllers, MC68EC000, MC68EC020, and MC68EC030. All references to the MC68040 include the MC68LC040 and MC68EC040. This referencing applies throughout this section unless otherwise specified.

Identified within the paragraphs are the specific processors that use different instruction fields, instruction formats, etc.—for example:

MC68020, MC68030, and MC68040 only

| | | | | | |
|------------|-----|----------------|--------------|-----|-----|
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn)** | 111 | 011 |
|------------|-----|----------------|--------------|-----|-----|

**Can be used with CPU32 processor

Appendix A Processor Instruction Summary provides a listing of all processors and the instructions that apply to them for quick reference.

ABCD**Add Decimal with Extend
(M68000 Family)****ABCD****Operation:** Source10 + Destination10 + X → Destination**Assembler** ABCD Dy,Dx**Syntax:** ABCD – (Ay), – (Ax)**Attributes:** Size = (Byte)**Description:** Adds the source operand to the destination operand along with the extend bit, and stores the result in the destination location. The addition is performed using binary-coded decimal arithmetic. The operands, which are packed binary-coded decimal numbers, can be addressed in two different ways:

1. Data Register to Data Register: The operands are contained in the data registers specified in the instruction.
2. Memory to Memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | U | * | U | * |

X — Set the same as the carry bit.

N — Undefined.

Z — Cleared if the result is nonzero; unchanged otherwise.

V — Undefined.

C — Set if a decimal carry was generated; cleared otherwise.

NOTE

Normally, the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

ABCD**Add Decimal with Extend
(M68000 Family)****ABCD****Instruction Format:**

| | | | | | | | | | | | | | | | | |
|----|----|----|----|-------------|----|---|---|---|---|---|---|---|-----|-------------|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 0 | 0 | REGISTER Rx | | | | 1 | 0 | 0 | 0 | 0 | R/M | REGISTER Ry | | |

Instruction Fields:

Register Rx field—Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

R/M field—Specifies the operand addressing mode.

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Ry field—Specifies the source register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

ADD

Add (M68000 Family)

ADD

Operation: Source + Destination → Destination

Assembler Syntax: ADD < ea > ,Dn

Syntax: ADD Dn, < ea >

Attributes: Size = (Byte, Word, Long)

Description: Adds the source operand to the destination operand using binary addition and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The mode of the instruction indicates which operand is the source and which is the destination, as well as the operand size.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | * | * |

X — Set the same as the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.

C — Set if a carry is generated; cleared otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|--------|---|---|-------------------|------|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 1 | REGISTER | | | OPMODE | | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | | MODE | | REGISTER | | |

ADD**Add
(M68000 Family)****ADD****Instruction Fields:**

Register field—Specifies any of the eight data registers.

Opmode field

| Byte | Word | Long | Operation |
|------|------|------|----------------------|
| 000 | 001 | 010 | < ea > + Dn → Dn |
| 100 | 101 | 110 | Dn + < ea > → < ea > |

Effective Address field—Determines addressing mode.

- a. If the location specified is a source operand, all addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An* | 001 | reg. number:An | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | 111 | 100 |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|-----|-----|
| (bd,An,Xn)** | 110 | reg. number:An | (bd,PC,Xn)† | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

*Word and long only

**Can be used with CPU32.

ADD**Add
(M68000 Family)****ADD**

- b. If the location specified is a destination operand, only memory alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32

NOTE

The Dn mode is used when the destination is a data register; the destination < ea > mode is invalid for a data register.

ADDA is used when the destination is an address register. ADDI and ADDQ are used when the source is immediate data. Most assemblers automatically make this distinction.

ADDA

Add Address (M68000 Family)

ADDA

Operation: Source + Destination → Destination

Assembler

Syntax: ADDA < ea > , An

Attributes: Size = (Word, Long)

Description: Adds the source operand to the destination address register and stores the result in the address register. The size of the operation may be specified as word or long. The entire destination address register is used regardless of the operation size.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|--------|---|---|-------------------|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 1 | REGISTER | | | | OPMODE | | | EFFECTIVE ADDRESS | | | | |
| | | | | | | | | | | | MODE | | REGISTER | | |

Instruction Fields:

Register field—Specifies any of the eight address registers. This is always the destination.

Opmode field—Specifies the size of the operation.

011— Word operation; the source operand is sign-extended to a long operand and the operation is performed on the address register using all 32 bits.

111— Long operation.

ADDA**Add Address
(M68000 Family)****ADDA**

Effective Address field—Specifies the source operand. All addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | 001 | reg. number:An |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32

ADDI

Add Immediate (M68000 Family)

ADDI

Operation: Immediate Data + Destination → Destination

Assembler

Syntax: ADDI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Adds the immediate data to the destination operand and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | * | * |

X — Set the same as the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.

C — Set if a carry is generated; cleared otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|------------------|----|----|----|----|----|---|---|------|---|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| 16-BIT WORD DATA | | | | | | | | | | 8-BIT BYTE DATA | | | | | |
| 32-BIT LONG DATA | | | | | | | | | | | | | | | |

ADDI

Add Immediate (M68000 Family)

ADDI

Instruction Fields:

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32

Immediate field—Data immediately following the instruction.

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is the next two immediate words.

ADDQ

Add Quick (M68000 Family)

ADDQ

Operation: Immediate Data + Destination → Destination

Assembler

Syntax: ADDQ # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Adds an immediate value of one to eight to the operand at the destination location. The size of the operation may be specified as byte, word, or long. Word and long operations are also allowed on the address registers. When adding to address registers, the condition codes are not altered, and the entire destination address register is used regardless of the operation size.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | * | * |

X — Set the same as the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Set if a carry occurs; cleared otherwise.

The condition codes are not affected when the destination is an address register.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|------|----|---|---|------|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | DATA | | | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

ADDQ**Add Quick
(M68000 Family)****ADDQ****Instruction Fields:**

Data field—Three bits of immediate data representing eight values (0 – 7), with the immediate value zero representing a value of eight.

Size field—Specifies the size of the operation.

00— Byte operation

01— Word operation

10— Long operation

Effective Address field—Specifies the destination location. Only alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | 001 | reg. number:An |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn**) | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)† | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Word and long only.

**Can be used with CPU32.

ADDX

Add Extended (M68000 Family)

ADDX

Operation: Source + Destination + X → Destination

Assembler Syntax: ADDX Dy,Dx

Syntax: ADDX – (Ay), – (Ax)

Attributes: Size = (Byte, Word, Long)

Description: Adds the source operand and the extend bit to the destination operand and stores the result in the destination location. The operands can be addressed in two different ways:

1. Data register to data register—The data registers specified in the instruction contain the operands.
2. Memory to memory—The address registers specified in the instruction address the operands using the predecrement addressing mode.

The size of the operation can be specified as byte, word, or long.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | * | * | * | * |

X — Set the same as the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Cleared if the result is nonzero; unchanged otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Set if a carry is generated; cleared otherwise.

NOTE

Normally, the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

ADDX**Add Extended
(M68000 Family)****ADDX****Instruction Format:**

| | | | | | | | | | | | | | | | |
|----|----|----|----|-------------|----|---|---|------|---|---|---|-----|-------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 1 | REGISTER Rx | | | 1 | SIZE | | 0 | 0 | R/M | REGISTER Ry | | |

Instruction Fields:

Register Rx field—Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

Size field—Specifies the size of the operation.

00 — Byte operation

01 — Word operation

10 — Long operation

R/M field—Specifies the operand address mode.

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Ry field—Specifies the source register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

AND

AND Logical (M68000 Family)

AND

Operation: Source L Destination → Destination

Assembler Syntax: AND < ea > ,Dn

Syntax: AND Dn, < ea >

Attributes: Size = (Byte, Word, Long)

Description: Performs an AND operation of the source operand with the destination operand and stores the result in the destination location. The size of the operation can be specified as byte, word, or long. The contents of an address register may not be used as an operand.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the result is set; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|--------|---|---|---------------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 0 | REGISTER | | | OPMODE | | | EFFECTIVE ADDRESS MODE | | | REGISTER | | |

Instruction Fields:

Register field—Specifies any of the eight data registers.

Opmode field

| Byte | Word | Long | Operation |
|------|------|------|-----------------------------|
| 000 | 001 | 010 | < ea > \wedge Dn → Dn |
| 100 | 101 | 110 | Dn \wedge < ea > → < ea > |

AND**AND Logical
(M68000 Family)****AND**

Effective Address field—Determines addressing mode.

- a. If the location specified is a source operand, only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

AND

AND Logical (M68000 Family)

AND

- b. If the location specified is a destination operand, only memory alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NOTE

The Dn mode is used when the destination is a data register; the destination < ea > mode is invalid for a data register.

Most assemblers use ANDI when the source is immediate data.

ANDI

AND Immediate (M68000 Family)

ANDI

Operation: Immediate Data \wedge Destination \rightarrow Destination

Assembler

Syntax: ANDI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Performs an AND operation of the immediate data with the destination operand and stores the result in the destination location. The size of the operation can be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the result is set; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|------------------|----|----|----|----|----|---|---|------|---|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| 16-BIT WORD DATA | | | | | | | | | | 8-BIT BYTE DATA | | | | | |
| 32-BIT LONG DATA | | | | | | | | | | | | | | | |

ANDI

AND Immediate (M68000 Family)

ANDI

Instruction Fields:

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32

Immediate field—Data immediately following the instruction.

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is the next two immediate words.

ANDI to CCR

CCR AND Immediate (M68000 Family)

ANDI to CCR

Operation: Source \wedge CCR \rightarrow CCR

Assembler

Syntax: ANDI # < data > ,CCR

Attributes: Size = (Byte)

Description: Performs an AND operation of the immediate operand with the condition codes and stores the result in the low-order byte of the status register.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | * | * | * | * |

X — Cleared if bit 4 of immediate operand is zero; unchanged otherwise.

N — Cleared if bit 3 of immediate operand is zero; unchanged otherwise.

Z — Cleared if bit 2 of immediate operand is zero; unchanged otherwise.

V — Cleared if bit 1 of immediate operand is zero; unchanged otherwise.

C — Cleared if bit 0 of immediate operand is zero; unchanged otherwise.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|-----------------|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| | | | | | | | | 8-BIT BYTE DATA | | | | | | | |

ASL, ASR

Arithmetic Shift (M68000 Family)

ASL, ASR

Operation: Destination Shifted By Count → Destination

Assembler ASd Dx,Dy

Syntax: ASd # < data > ,Dy
ASd < ea >
where d is direction, L or R

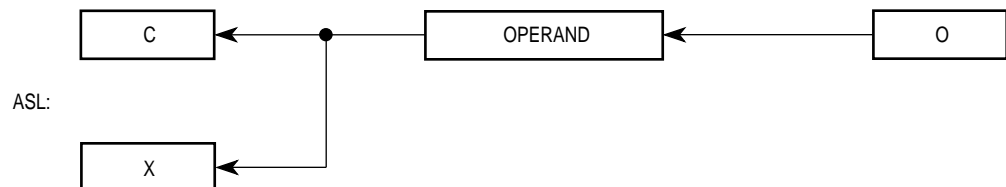
Attributes: Size = (Byte, Word, Long)

Description: Arithmetically shifts the bits of the operand in the direction (L or R) specified. The carry bit receives the last bit shifted out of the operand. The shift count for the shifting of a register may be specified in two different ways:

1. Immediate—The shift count is specified in the instruction (shift range, 1 – 8).
2. Register—The shift count is the value in the data register specified in instruction modulo 64.

The size of the operation can be specified as byte, word, or long. An operand in memory can be shifted one bit only, and the operand size is restricted to a word.

For ASL, the operand is shifted left; the number of positions shifted is the shift count. Bits shifted out of the high-order bit go to both the carry and the extend bits; zeros are shifted into the low-order bit. The overflow bit indicates if any sign changes occur during the shift.

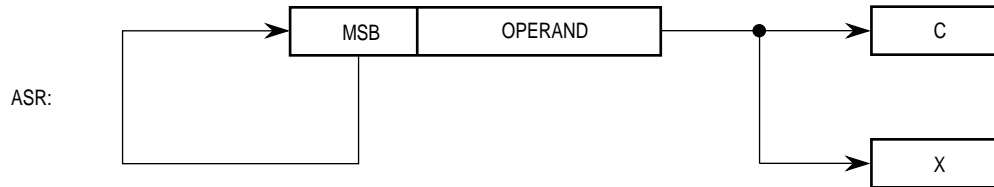


ASL, ASR

Arithmetic Shift (M68000 Family)

ASL, ASR

For ASR, the operand is shifted right; the number of positions shifted is the shift count. Bits shifted out of the low-order bit go to both the carry and the extend bits; the sign bit (MSB) is shifted into the high-order bit.



Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | * | * | * | * |

- X — Set according to the last bit shifted out of the operand; unaffected for a shift count of zero.
- N — Set if the most significant bit of the result is set; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if the most significant bit is changed at any time during the shift operation; cleared otherwise.
- C — Set according to the last bit shifted out of the operand; cleared for a shift count of zero.

Instruction Format:

REGISTER SHIFTS

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|--------------------|----|----|------|-----|---|---|----------|---|---|---|---|
| 1 | 1 | 1 | 0 | COUNT? REGISTER | | dr | SIZE | i/r | 0 | 0 | REGISTER | | | | |

Instruction Fields:

Count/Register field—Specifies shift count or register that contains the shift count:

If $i/r = 0$, this field contains the shift count. The values 1 – 7 represent counts of 1 – 7; a value of zero represents a count of eight.

If $i/r = 1$, this field specifies the data register that contains the shift count (modulo 64).

ASL, ASR

Arithmetic Shift (M68000 Family)

ASL, ASR

dr field—Specifies the direction of the shift.

- 0 — Shift right
- 1 — Shift left

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

i/r field

- If i/r = 0, specifies immediate shift count.
- If i/r = 1, specifies register shift count.

Register field—Specifies a data register to be shifted.

Instruction Format:

MEMORY SHIFTS

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|----|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | dr | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

dr field—Specifies the direction of the shift.

- 0 — Shift right
- 1 — Shift left

ASL, ASR

Arithmetic Shift (M68000 Family)

ASL, ASR

Effective Address field—Specifies the operand to be shifted. Only memory alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

Bcc**Branch Conditionally
(M68000 Family)****Bcc**

Operation: If Condition True
Then $PC + d_n \rightarrow PC$

**Assembler
Syntax:** Bcc < label >

Attributes: Size = (Byte, Word, Long*)
*(MC68020, MC68030, and MC68040 only)

Description: If the specified condition is true, program execution continues at location (PC + displacement). The program counter contains the address of the instruction word for the Bcc instruction plus two. The displacement is a twos-complement integer that represents the relative distance in bytes from the current program counter to the destination program counter. If the 8-bit displacement field in the instruction word is zero, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (\$FF), the 32-bit displacement (long word immediately following the instruction) is used. Condition code cc specifies one of the following conditional tests (refer to Table 3-19 for more information on these conditional tests):

| Mnemonic | Condition | Mnemonic | Condition |
|----------|------------------|----------|----------------|
| CC(HI) | Carry Clear | LS | Low or Same |
| CS(LO) | Carry Set | LT | Less Than |
| EQ | Equal | MI | Minus |
| GE | Greater or Equal | NE | Not Equal |
| GT | Greater Than | PL | Plus |
| HI | High | VC | Overflow Clear |
| LE | Less or Equal | VS | Overflow Set |

Condition Codes:

Not affected.

Bcc**Branch Conditionally
(M68000 Family)****Bcc****Instruction Format:**

| | | | | | | | | | | | | | | | |
|--|----|----|----|-----------|----|---|---|--------------------|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 0 | CONDITION | | | | 8-BIT DISPLACEMENT | | | | | | | |
| 16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00 | | | | | | | | | | | | | | | |
| 32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF | | | | | | | | | | | | | | | |

Instruction Fields:

Condition field—The binary code for one of the conditions listed in the table.

8-Bit Displacement field—Two's complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

16-Bit Displacement field—Used for the displacement when the 8-bit displacement field contains \$00.

32-Bit Displacement field—Used for the displacement when the 8-bit displacement field contains \$FF.

NOTE

A branch to the immediately following instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains \$00 (zero offset).

BCHG

Test a Bit and Change (M68000 Family)

BCHG

Operation: TEST (< number > of Destination) → Z;
 TEST (< number > of Destination) → < bit number > of Destination

Assembler BCHG Dn, < ea >

Syntax: BCHG # < data > , < ea >

Attributes: Size = (Byte, Long)

Description: Tests a bit in the destination operand and sets the Z condition code appropriately, then inverts the specified bit in the destination. When the destination is a data register, any of the 32 bits can be specified by the modulo 32-bit number. When the destination is a memory location, the operation is a byte operation, and the bit number is modulo 8. In all cases, bit zero refers to the least significant bit. The bit number for this operation may be specified in either of two ways:

1. Immediate—The bit number is specified in a second word of the instruction.
2. Register—The specified data register contains the bit number.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | — | * | — | — |

X — Not affected.

N — Not affected.

Z — Set if the bit tested is zero; cleared otherwise.

V — Not affected.

C — Not affected.

BCHG**Test a Bit and Change
(M68000 Family)****BCHG****Instruction Format:**

BIT NUMBER DYNAMIC, SPECIFIED IN A REGISTER

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | REGISTER | | | 1 | 0 | 1 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |

Instruction Fields:

Register field—Specifies the data register that contains the bit number.

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn* | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)† | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Long only; all others are byte only.

**Can be used with CPU32.

BCHG**Test a Bit and Change
(M68000 Family)****BCHG****Instruction Format:**

BIT NUMBER STATIC, SPECIFIED AS IMMEDIATE DATA

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------------|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | BIT NUMBER | | | | | | | |

Instruction Fields:

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn* | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|-----------------|
| (bd,PC,Xn)† | — | (bd,An,Xn)** |
| ([bd,PC,Xn],od) | — | ([bd,An,Xn],od) |
| ([bd,PC],Xn,od) | — | ([bd,An],Xn,od) |

*Long only; all others are byte only.

**Can be used with CPU32.

Bit Number field—Specifies the bit number.

BCLR**Test a Bit and Clear
(M68000 Family)****BCLR**

Operation: TEST (< bit number > of Destination) → Z; 0 → < bit number > of Destination

Assembler BCLR Dn, < ea >

Syntax: BCLR # < data > , < ea >

Attributes: Size = (Byte, Long)

Description: Tests a bit in the destination operand and sets the Z condition code appropriately, then clears the specified bit in the destination. When a data register is the destination, any of the 32 bits can be specified by a modulo 32-bit number. When a memory location is the destination, the operation is a byte operation, and the bit number is modulo 8. In all cases, bit zero refers to the least significant bit. The bit number for this operation can be specified in either of two ways:

1. Immediate—The bit number is specified in a second word of the instruction.
2. Register—The specified data register contains the bit number.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | — | * | — | — |

X — Not affected.

N — Not affected.

Z — Set if the bit tested is zero; cleared otherwise.

V — Not affected.

C — Not affected.

BCLR**Test a Bit and Clear
(M68000 Family)****BCLR****Instruction Format:**

BIT NUMBER DYNAMIC, SPECIFIED IN A REGISTER

| | | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0 | 0 | 0 | 0 | REGISTER | | | | 1 | 1 | 0 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |

Instruction Fields:

Register field—Specifies the data register that contains the bit number.

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn* | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)† | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Long only; all others are byte only.

**Can be used with CPU32.

BCLR**Test a Bit and Clear
(M68000 Family)****BCLR****Instruction Format:**

BIT NUMBER STATIC, SPECIFIED AS IMMEDIATE DATA

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------------|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | BIT NUMBER | | | | | | | |

Instruction Fields:

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn* | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)† | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Long only; all others are byte only.

**Can be used with CPU32.

Bit Number field—Specifies the bit number.

BFCHG**Test Bit Field and Change
(MC68020, MC68030, MC68040)****BFCHG**

Operation: TEST (< bit field > of Destination) → < bit field > of Destination

Assembler

Syntax: BFCHG < ea > {offset:width}

Attributes: Unsized

Description: Sets the condition codes according to the value in a bit field at the specified effective address, then complements the field.

A field offset and a field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the field is set; cleared otherwise.

Z — Set if all bits of the field are zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | | |
|----|----|----|----|----|--------|---|---|---|---|-------------------|-------|----------|---|---|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | | |
| 0 | 0 | 0 | 0 | Do | OFFSET | | | | | Dw | WIDTH | | | | | |

NOTE

For the MC68020, MC68030, and MC68040, all bit field instructions access only those bytes in memory that contain some portion of the bit field. The possible accesses are byte, word, 3-byte, long word, and long word with byte (for a 5-byte access).

BFCHG**Test Bit Field and Change
(MC68020, MC68030, MC68040)****BFCHG****Instruction Fields:**

Effective Address field—Specifies the base location for the bit field. Only data register direct or control alterable addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

Do field—Determines how the field offset is specified.

- 0 — The offset field contains the bit field offset.
- 1 — Bits 8 – 6 of the extension word specify a data register that contains the offset; bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.

If Do = 0, the offset field is an immediate operand; the operand value is in the range 0 – 31.

If Do = 1, the offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.

- 0 — The width field contains the bit field width.
- 1 — Bits 2 – 0 of the extension word specify a data register that contains the width; bits 3 – 4 are zero.

Width field—Specifies the field width, depending on Dw.

If Dw = 0, the width field is an immediate operand; an operand value in the range 1 – 31 specifies a field width of 1 – 31, and a value of zero specifies a width of 32.

If Dw = 1, the width field specifies a data register that contains the width. The value is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value of zero specifies a width of 32.

BFCLR**Test Bit Field and Clear**
(MC68020, MC68030, MC68040)**BFCLR****Operation:** 0 → < bit field > of Destination**Assembler****Syntax:** BFCLR < ea > {offset:width}**Attributes:** Unsized**Description:** Sets condition codes according to the value in a bit field at the specified effective address and clears the field.

The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the field is set; cleared otherwise.

Z — Set if all bits of the field are zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | | |
|----|----|----|----|----|--------|---|---|---|---|-------------------|-------|----------|---|---|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | | |
| 0 | 0 | 0 | 0 | Do | OFFSET | | | | | Dw | WIDTH | | | | | |

BFCLR**Test Bit Field and Clear
(MC68020, MC68030, MC68040)****BFCLR****Instruction Fields:**

Effective Address field—Specifies the base location for the bit field. Only data register direct or control alterable addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

Do field—Determines how the field offset is specified.

- 0 — The offset field contains the bit field offset.
- 1 — Bits 8 – 6 of the extension word specify a data register that contains the offset; bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.

If Do = 0, the offset field is an immediate operand; the operand value is in the range of 0 – 31.

If Do = 1, the offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.

- 0 — The width field contains the bit field width.
- 1 — Bits 2 – 0 of the extension word specify a data register that contains the width; bits 3 – 4 are zero.

Width field—Specifies the field width, depending on Dw.

If Dw = 0, the width field is an immediate operand; operand values in the range of 1 – 31 specify a field width of 1 – 31, and a value of zero specifies a width of 32.

If Dw = 1, the width field specifies a data register that contains the width. The value is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value of zero specifies a width of 32.

BFEXTS

Extract Bit Field Signed (MC68020, MC68030, MC68040)

BFEXTS

Operation: < bit field > of Source → Dn

Assembler

Syntax: BFEXTS < ea > {offset:width},Dn

Attributes: Unsized

Description: Extracts a bit field from the specified effective address location, sign extends to 32 bits, and loads the result into the destination data register. The field offset and field width select the bit field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the field is set; cleared otherwise.

Z — Set if all bits of the field are zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----------|----|----|----|----|--------|---|---|---|----|-------------------|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| REGISTER | | | | Do | OFFSET | | | | Dw | WIDTH | | | | | |

BFEXTS**Extract Bit Field Signed
(MC68020, MC68030, MC68040)****BFEXTS****Instruction Fields:**

Effective Address field—Specifies the base location for the bit field. Only data register direct or control addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

Register field—Specifies the destination register.

Do field—Determines how the field offset is specified.

- 0 — The offset field contains the bit field offset.
- 1 — Bits 8 – 6 of the extension word specify a data register that contains the offset; bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.

If Do = 0, the offset field is an immediate operand; the operand value is in the range of 0 – 31.

If Do = 1, the offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.

- 0 — The width field contains the bit field width.
- 1 — Bits 2 – 0 of the extension word specify a data register that contains the width; bits 4 – 3 are zero.

BFEXTS

Extract Bit Field Signed (MC68020, MC68030, MC68040)

BFEXTS

Width field—Specifies the field width, depending on Dw.

If Dw = 0, the width field is an immediate operand; operand values in the range of 1 – 31 specify a field width of 1 – 31, and a value of zero specifies a width of 32.

If Dw = 1, the width field specifies a data register that contains the width. The value is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value of zero specifies a width of 32.

BFEXTU

Extract Bit Field Unsigned
(MC68020, MC68030, MC68040)

BFEXTU

Operation: < bit offset > of Source → Dn

Assembler

Syntax: BFEXTU < ea > {offset:width},Dn

Attributes: Unsized

Description: Extracts a bit field from the specified effective address location, zero extends to 32 bits, and loads the results into the destination data register. The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the source field is set; cleared otherwise.

Z — Set if all bits of the field are zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | | |
|----------|----------|----|----|--------|--------|---|---|---|----|-------------------|----------|---|---|---|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | EFFECTIVE ADDRESS | | | | | | |
| REGISTER | | | Do | OFFSET | | | | | Dw | MODE | REGISTER | | | | | |
| 0 | REGISTER | | | Do | OFFSET | | | | | Dw | WIDTH | | | | | |

BFEXTU

Extract Bit Field Unsigned (MC68020, MC68030, MC68040)

BFEXTU

Instruction Fields:

Effective Address field—Specifies the base location for the bit field. Only data register direct or control addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

Register field—Specifies the destination data register.

Do field—Determines how the field offset is specified.

- 0 — The offset field contains the bit field offset.
- 1 — Bits 8 – 6 of the extension word specify a data register that contains the offset; bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.

If Do = 0, the offset field is an immediate operand; the operand value is in the range of 0 – 31.

If Do = 1, the offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.

- 0 — The width field contains the bit field width.
- 1 — Bits 2 – 0 of the extension word specify a data register that contains the width; bits 4 – 3 are zero.

BFEXTU

**Extract Bit Field Unsigned
(MC68020, MC68030, MC68040)**

BFEXTU

Width field—Specifies the field width, depending on Dw.

If Dw = 0, the width field is an immediate operand; operand values in the range of 1 – 31 specify a field width of 1 – 31, and a value of zero specifies a width of 32.

If Dw = 1, the width field specifies a data register that contains the width. The value is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value of zero specifies a width of 32.

BFFFO**Find First One in Bit Field
(MC68020, MC68030, MC68040)****BFFFO****Operation:** < bit offset > of Source Bit Scan → Dn**Assembler****Syntax:** BFFFO < ea > {offset:width},Dn**Attributes:** Unsized

Description: Searches the source operand for the most significant bit that is set to a value of one. The bit offset of that bit (the bit offset in the instruction plus the offset of the first one bit) is placed in Dn. If no bit in the bit field is set to one, the value in Dn is the field offset plus the field width. The instruction sets the condition codes according to the bit field value. The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the field is set; cleared otherwise.

Z — Set if all bits of the field are zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----------|----|----|--------|---|---|---|-------------------|-------|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | REGISTER | | Do | OFFSET | | | | Dw | WIDTH | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

BFFFO**Find First One in Bit Field
(MC68020, MC68030, MC68040)****BFFFO****Instruction Fields:**

Effective Address field—Specifies the base location for the bit field. Only data register direct or control addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

Register field—Specifies the destination data register operand.

Do field—Determines how the field offset is specified.

- 0 — The offset field contains the bit field offset.
- 1 — Bits 8 – 6 of the extension word specify a data register that contains the offset; bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.

If Do = 0, the offset field is an immediate operand; the operand value is in the range of 0 – 31.

If Do = 1, the offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.

- 0 — The width field contains the bit field width.
- 1 — Bits 2 – 0 of the extension word specify a data register that contains the width; bits 4 – 3 are zero.

BFFFO**Find First One in Bit Field
(MC68020, MC68030, MC68040)****BFFFO**

Width field—Specifies the field width, depending on Dw.

If Dw = 0, the width field is an immediate operand; operand values in the range of 1 – 31 specify a field width of 1 – 31, and a value of zero specifies a width of 32.

If Dw = 1, the width field specifies a data register that contains the width. The value is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value of zero specifies a width of 32.

BFINS**Insert Bit Field**
(MC68020, MC68030, MC68040)**BFINS****Operation:** Dn → < bit field > of Destination**Assembler****Syntax:** BFINS Dn, < ea > {offset:width}**Attributes:** Unsized**Description:** Inserts a bit field taken from the low-order bits of the specified data register into a bit field at the effective address location. The instruction sets the condition codes according to the inserted value. The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.**Condition Codes:**

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the field is set; cleared otherwise.

Z — Set if all bits of the field are zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | | |
|----------|----------|----|----|----|--------|---|---|---|---|-------------------|----------|---|---|---|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | EFFECTIVE ADDRESS | | | | | | |
| REGISTER | | | | Do | OFFSET | | | | | Dw | REGISTER | | | | | |
| 0 | REGISTER | | | Do | OFFSET | | | | | Dw | WIDTH | | | | | |

BFINS**Insert Bit Field**
(MC68020, MC68030, MC68040)**BFINS****Instruction Fields:**

Effective Address field—Specifies the base location for the bit field. Only data register direct or control alterable addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

Register field—Specifies the source data register operand.

Do field—Determines how the field offset is specified.

0 — The offset field contains the bit field offset.

1 — Bits 8 – 6 of the extension word specify a data register that contains the offset; bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.

If Do = 0, the offset field is an immediate operand; the operand value is in the range of 0 – 31.

If Do = 1, the offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.

0 — The width field contains the bit field width.

1 — Bits 2 – 0 of the extension word specify a data register that contains the width; bits 4 – 3 are zero.

BFINS

Insert Bit Field (MC68020, MC68030, MC68040)

BFINS

Width field—Specifies the field width, depending on Dw.

If Dw = 0, the width field is an immediate operand; operand values in the range of 1 – 31 specify a field width of 1 – 31, and a value of zero specifies a width of 32.

If Dw = 1, the width field specifies a data register that contains the width. The value is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value of zero specifies a width of 32.

BFSET**Test Bit Field and Set**
(MC68020, MC68030, MC68040)**BFSET****Operation:** 1 → < bit field > of Destination**Assembler****Syntax:** BFSET < ea > {offset:width}**Attributes:** Unsized**Description:** Sets the condition codes according to the value in a bit field at the specified effective address, then sets each bit in the field.

The field offset and the field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the field is set; cleared otherwise.

Z — Set if all bits of the field are zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|--------|---|---|---|---|-------------------|-------|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| 0 | 0 | 0 | 0 | Do | OFFSET | | | | | Dw | WIDTH | | | | |

BFSET**Test Bit Field and Set
(MC68020, MC68030, MC68040)****BFSET****Instruction Fields:**

Effective Address field—Specifies the base location for the bit field. Only data register direct or control alterable addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

Do field—Determines how the field offset is specified.

- 0 — The offset field contains the bit field offset.
- 1 — Bits 8 – 6 of the extension word specify a data register that contains the offset; bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.

If Do = 0, the offset field is an immediate operand; the operand value is in the range of 0 – 31.

If Do = 1, the offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.

- 0 — The width field contains the bit field width.
- 1 — Bits 2 – 0 of the extension word specify a data register that contains the width; bits 4 – 3 are zero.

Width field—Specifies the field width, depending on Dw.

If Dw = 0, the width field is an immediate operand; operand values in the range of 1 – 31 specify a field width of 1 – 31, and a value of zero specifies a width of 32.

If Dw = 1, the width field specifies a data register that contains the width. The value is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value of zero specifies a width of 32.

BFTST

Test Bit Field (MC68020, MC68030, MC68040)

BFTST

Operation: < bit field > of Destination

Assembler

Syntax: BFTST < ea > {offset:width}

Attributes: Unsized

Description: Sets the condition codes according to the value in a bit field at the specified effective address location. The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the field is set; cleared otherwise.

Z — Set if all bits of the field are zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | | |
|----|----|----|----|----|--------|---|---|---|---|-------------------|-------|----------|---|---|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | | |
| 0 | 0 | 0 | 0 | Do | OFFSET | | | | | Dw | WIDTH | | | | | |

BFTST**Test Bit Field
(MC68020, MC68030, MC68040)****BFTST****Instruction Fields:**

Effective Address field—Specifies the base location for the bit field. Only data register direct or control addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

Do field—Determines how the field offset is specified.

- 0 — The offset field contains the bit field offset.
- 1 — Bits 8 – 6 of the extension word specify a data register that contains the offset; bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.

If Do = 0, the offset field is an immediate operand; the operand value is in the range of 0 – 31.

If Do = 1, the offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.

- 0 — The width field contains the bit field width.
- 1 — Bits 2 – 0 of the extension word specify a data register that contains the width; bits 4 – 3 are zero.

Width field—Specifies the field width, depending on Dw.

If Dw = 0, the width field is an immediate operand, operand values in the range of 1 – 31 specify a field width of 1 – 31, and a value of zero specifies a width of 32.

If Dw = 1, the width field specifies a data register that contains the width. The value is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value of zero specifies a width of 32.

BKPT**Breakpoint****BKPT**

(MC68EC000, MC68010, MC68020, MC68030, MC68040, CPU32)

Operation: Run Breakpoint Acknowledge Cycle; TRAP As Illegal Instruction**Assembler****Syntax:** BKPT # < data >**Attributes:** Unsized

Description: For the MC68010, a breakpoint acknowledge bus cycle is run with function codes driven high and zeros on all address lines. Whether the breakpoint acknowledge bus cycle is terminated with \overline{DTACK} , \overline{BERR} , or \overline{VPA} , the processor always takes an illegal instruction exception. During exception processing, a debug monitor can distinguish different software breakpoints by decoding the field in the BKPT instruction. For the MC68000 and MC68008, the breakpoint cycle is not run, but an illegal instruction exception is taken.

For the MC68020, MC68030, and CPU32, a breakpoint acknowledge bus cycle is executed with the immediate data (value 0 – 7) on bits 2 – 4 of the address bus and zeros on bits 0 and 1 of the address bus. The breakpoint acknowledge bus cycle accesses the CPU space, addressing type 0, and provides the breakpoint number specified by the instruction on address lines A2 – A4. If the external hardware terminates the cycle with \overline{DSACKx} or \overline{STERM} , the data on the bus (an instruction word) is inserted into the instruction pipe and is executed after the breakpoint instruction. The breakpoint instruction requires a word to be transferred so, if the first bus cycle accesses an 8-bit port, a second bus cycle is required. If the external logic terminates the breakpoint acknowledge bus cycle with \overline{BERR} (i.e., no instruction word available), the processor takes an illegal instruction exception.

For the MC68040, this instruction executes a breakpoint acknowledge bus cycle. Regardless of the cycle termination, the MC68040 takes an illegal instruction exception.

For more information on the breakpoint instruction refer to the appropriate user's manual on bus operation.

This instruction supports breakpoints for debug monitors and real-time hardware emulators.

BKPT

Breakpoint
 (MC68EC000, MC68010, MC68020,
 MC68030, MC68040, CPU32)

BKPT**Condition Codes:**

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|--------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | VECTOR | | |

Instruction Field:

Vector field—Contains the immediate data, a value in the range of 0 – 7. This is the breakpoint number.

BRA**Branch Always
(M68000 Family)****BRA****Operation:** $PC + d_n \rightarrow PC$ **Assembler****Syntax:** BRA < label >**Attributes:** Size = (Byte, Word, Long*)
*(MC68020, MC68030, MC68040 only)

Description: Program execution continues at location (PC) + displacement. The program counter contains the address of the instruction word of the BRA instruction plus two. The displacement is a twos complement integer that represents the relative distance in bytes from the current program counter to the destination program counter. If the 8-bit displacement field in the instruction word is zero, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (\$FF), the 32-bit displacement (long word immediately following the instruction) is used.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|---|---|--------------------|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 8-BIT DISPLACEMENT | | | | | | | |
| 16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00 | | | | | | | | | | | | | | | |
| 32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF | | | | | | | | | | | | | | | |

Instruction Fields:

8-Bit Displacement field—Twos complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed.

16-Bit Displacement field—Used for a larger displacement when the 8-bit displacement is equal to \$00.

32-Bit Displacement field—Used for a larger displacement when the 8-bit displacement is equal to \$FF.

NOTE

A branch to the immediately following instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains \$00 (zero offset).

BSET**Test a Bit and Set
(M68000 Family)****BSET**

Operation: TEST (< bit number > of Destination) → Z; 1 → < bit number > of Destination

Assembler BSET Dn, < ea >

Syntax: BSET # < data > , < ea >

Attributes: Size = (Byte, Long)

Description: Tests a bit in the destination operand and sets the Z condition code appropriately, then sets the specified bit in the destination operand. When a data register is the destination, any of the 32 bits can be specified by a modulo 32-bit number. When a memory location is the destination, the operation is a byte operation, and the bit number is modulo 8. In all cases, bit zero refers to the least significant bit. The bit number for this operation can be specified in either of two ways:

1. Immediate—The bit number is specified in the second word of the instruction.
2. Register—The specified data register contains the bit number.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | — | * | — | — |

X — Not affected.

N — Not affected.

Z — Set if the bit tested is zero; cleared otherwise.

V — Not affected.

C — Not affected.

BSET**Test a Bit and Set
(M68000 Family)****BSET****Instruction Format:**

BIT NUMBER DYNAMIC, SPECIFIED IN A REGISTER

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | REGISTER | | | 1 | 1 | 1 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |

Instruction Fields:

Register field—Specifies the data register that contains the bit number.

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn* | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)† | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Long only; all others are byte only.

**Can be used with CPU32.

BSET**Test a Bit and Set
(M68000 Family)****BSET****Instruction Format:**

BIT NUMBER STATIC, SPECIFIED AS IMMEDIATE DATA

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|------------|---|---|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | BIT NUMBER | | | | | | | | |

Instruction Fields:

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn* | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)† | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Long only; all others are byte only.

**Can be used with CPU32.

Bit Number field—Specifies the bit number.

BSR**Branch to Subroutine
(M68000 Family)****BSR****Operation:** $SP - 4 \rightarrow SP; PC \rightarrow (SP); PC + d_n \rightarrow PC$ **Assembler****Syntax:** BSR < label >**Attributes:** Size = (Byte, Word, Long*)
*(MC68020, MC68030, MC68040 only)**Description:** Pushes the long-word address of the instruction immediately following the BSR instruction onto the system stack. The program counter contains the address of the instruction word plus two. Program execution then continues at location (PC) + displacement. The displacement is a twos complement integer that represents the relative distance in bytes from the current program counter to the destination program counter. If the 8-bit displacement field in the instruction word is zero, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (\$FF), the 32-bit displacement (long word immediately following the instruction) is used.**Condition Codes:**

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|---|---|--------------------|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 8-BIT DISPLACEMENT | | | | | | | |
| 16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00 | | | | | | | | | | | | | | | |
| 32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF | | | | | | | | | | | | | | | |

BSR

Branch to Subroutine (M68000 Family)

BSR

Instruction Fields:

8-Bit Displacement field—Two's complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed.

16-Bit Displacement field—Used for a larger displacement when the 8-bit displacement is equal to \$00.

32-Bit Displacement field—Used for a larger displacement when the 8-bit displacement is equal to \$FF.

NOTE

A branch to the immediately following instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains \$00 (zero offset).

BTST**Test a Bit
(M68000 Family)****BTST**

Operation: TEST (< bit number > of Destination) → Z

Assembler BTST Dn, < ea >

Syntax: BTST # < data > , < ea >

Attributes: Size = (Byte, Long)

Description: Tests a bit in the destination operand and sets the Z condition code appropriately. When a data register is the destination, any of the 32 bits can be specified by a modulo 32-bit number. When a memory location is the destination, the operation is a byte operation, and the bit number is modulo 8. In all cases, bit zero refers to the least significant bit. The bit number for this operation can be specified in either of two ways:

1. Immediate—The bit number is specified in a second word of the instruction.
2. Register—The specified data register contains the bit number.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | — | * | — | — |

X — Not affected.

N — Not affected.

Z — Set if the bit tested is zero; cleared otherwise.

V — Not affected.

C — Not affected.

BTST**Test a Bit
(M68000 Family)****BTST****Instruction Format:**

BIT NUMBER DYNAMIC, SPECIFIED IN A REGISTER

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | REGISTER | | | 1 | 0 | 0 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |

Instruction Fields:

Register field—Specifies the data register that contains the bit number.

Effective Address field—Specifies the destination location. Only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn* | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)† | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Long only; all others are byte only.

**Can be used with CPU32.

BTST**Test a Bit
(M68000 Family)****BTST****Instruction Format:**

BIT NUMBER STATIC, SPECIFIED AS IMMEDIATE DATA

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------------|---|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | BIT NUMBER | | | | | | | |

Instruction Fields:

Effective Address field—Specifies the destination location. Only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn) | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

Bit Number field—Specifies the bit number.

CALLM

Call Module (MC68020)

CALLM

Operation: Save Current Module State on Stack; Load New Module State from Destination

Assembler

Syntax: CALLM # < data > , < ea >

Attributes: Unsized

Description: The effective address of the instruction is the location of an external module descriptor. A module frame is created on the top of the stack, and the current module state is saved in the frame. The immediate operand specifies the number of bytes of arguments to be passed to the called module. A new module state is loaded from the descriptor addressed by the effective address.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|----------------|---|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ARGUMENT COUNT | | | | | | | |

CALLM

Call Module (MC68020)

CALLM

Instruction Fields:

Effective Address field—Specifies the address of the module descriptor. Only control addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | — | — | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

Argument Count field—Specifies the number of bytes of arguments to be passed to the called module. The 8-bit field can specify from 0 to 255 bytes of arguments. The same number of bytes is removed from the stack by the RTM instruction.

CAS CAS2

Compare and Swap with Operand (MC68020, MC68030, MC68040)

CAS CAS2

Operation: CAS Destination – Compare Operand → cc;
 If Z, Update Operand → Destination
 Else Destination → Compare Operand
 CAS2 Destination 1 – Compare 1 → cc;
 If Z, Destination 2 – Compare 2 → cc
 If Z, Update 1 → Destination 1; Update 2 → Destination 2
 Else Destination 1 → Compare 1; Destination 2 → Compare 2

Assembler Syntax: CAS Dc,Du, < ea >
 CAS2 Dc1:Dc2,Du1:Du2,(Rn1):(Rn2)

Attributes: Size = (Byte^{*}, Word, Long)

Description: CAS compares the effective address operand to the compare operand (Dc). If the operands are equal, the instruction writes the update operand (Du) to the effective address operand; otherwise, the instruction writes the effective address operand to the compare operand (Dc).

CAS2 compares memory operand 1 (Rn1) to compare operand 1 (Dc1). If the operands are equal, the instruction compares memory operand 2 (Rn2) to compare operand 2 (Dc2). If these operands are also equal, the instruction writes the update operands (Du1 and Du2) to the memory operands (Rn1 and Rn2). If either comparison fails, the instruction writes the memory operands (Rn1 and Rn2) to the compare operands (Dc1 and Dc2).

Both operations access memory using locked or read-modify-write transfer sequences, providing a means of synchronizing several processors.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | * |

- X — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if an overflow is generated; cleared otherwise.
- C — Set if a borrow is generated; cleared otherwise.

*. CAS2 cannot use byte operands.

CAS CAS2

Compare and Swap with Operand (MC68020, MC68030, MC68040)

CAS CAS2

Instruction Format:

CAS

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|------|---|----|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | SIZE | | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | Du | | | 0 | 0 | 0 | Dc | | |

Instruction Fields:

Size field—Specifies the size of the operation.

- 01 — Byte operation
- 10 — Word operation
- 11 — Long operation

Effective Address field—Specifies the location of the memory operand. Only memory alterable addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | — | — | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

Du field—Specifies the data register that contains the update value to be written to the memory operand location if the comparison is successful.

Dc field—Specifies the data register that contains the value to be compared to the memory operand.

CAS CAS2

Compare and Swap with Operand (MC68020, MC68030, MC68040)

CAS CAS2

Instruction Format:

CAS2

| | | | | | | | | | | | | | | | |
|------|-----|----|----|----|------|---|-----|---|---|---|---|---|-----|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | SIZE | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| D/A1 | Rn1 | | | 0 | 0 | 0 | Du1 | | | 0 | 0 | 0 | Dc1 | | |
| D/A2 | Rn2 | | | 0 | 0 | 0 | Du2 | | | 0 | 0 | 0 | Dc2 | | |

Instruction Fields:

Size field—Specifies the size of the operation.

- 10 — Word operation
- 11 — Long operation

D/A1, D/A2 fields—Specify whether Rn1 and Rn2 reference data or address registers, respectively.

- 0 — The corresponding register is a data register.
- 1 — The corresponding register is an address register.

Rn1, Rn2 fields—Specify the numbers of the registers that contain the addresses of the first and second memory operands, respectively. If the operands overlap in memory, the results of any memory update are undefined.

Du1, Du2 fields—Specify the data registers that contain the update values to be written to the first and second memory operand locations if the comparison is successful.

Dc1, Dc2 fields—Specify the data registers that contain the test values to be compared to the first and second memory operands, respectively. If Dc1 and Dc2 specify the same data register and the comparison fails, memory operand 1 is stored in the data register.

NOTE

The CAS and CAS2 instructions can be used to perform secure update operations on system control data structures in a multiprocessing environment.

In the MC68040 if the operands are not equal, the destination or destination 1 operand is written back to memory to complete the locked access for CAS or CAS2, respectively.

CHK**Check Register Against Bounds
(M68000 Family)****CHK**

Operation: If $D_n < 0$ or $D_n > \text{Source}$
Then TRAP

Assembler

Syntax: CHK < ea > ,Dn

Attributes: Size = (Word, Long*)
*(MC68020, MC68030, MC68040 only)

Description: Compares the value in the data register specified in the instruction to zero and to the upper bound (effective address operand). The upper bound is a twos complement integer. If the register value is less than zero or greater than the upper bound, a CHK instruction exception (vector number 6) occurs.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | U | U | U |

X — Not affected.

N — Set if $D_n < 0$; cleared if $D_n > \text{effective address operand}$; undefined otherwise.

Z — Undefined.

V — Undefined.

C — Undefined.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|------|---|---|-------------------|------|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | REGISTER | | | SIZE | | 0 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | | MODE | | REGISTER | | |

CHK**Check Register Against Bounds
(M68000 Family)****CHK****Instruction Fields:**

Register field—Specifies the data register that contains the value to be checked.

Size field—Specifies the size of the operation.

11— Word operation

10— Long operation

Effective Address field—Specifies the upper bound operand. Only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

CHK2

Check Register Against Bounds (MC68020, MC68030, MC68040, CPU32)

CHK2

Operation: If $R_n < LB$ or $R_n > UB$
Then TRAP

Assembler

Syntax: CHK2 < ea > ,Rn

Attributes: Size = (Byte, Word, Long)

Description: Compares the value in R_n to each bound. The effective address contains the bounds pair: the upper bound following the lower bound. For signed comparisons, the arithmetically smaller value should be used as the lower bound. For unsigned comparisons, the logically smaller value should be the lower bound.

The size of the data and the bounds can be specified as byte, word, or long. If R_n is a data register and the operation size is byte or word, only the appropriate low-order part of R_n is checked. If R_n is an address register and the operation size is byte or word, the bounds operands are sign-extended to 32 bits, and the resultant operands are compared to the full 32 bits of A_n .

If the upper bound equals the lower bound, the valid range is a single value. If the register value is less than the lower bound or greater than the upper bound, a CHK instruction exception (vector number 6) occurs.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | U | * | U | * |

X — Not affected.

N — Undefined.

Z — Set if R_n is equal to either bound; cleared otherwise.

V — Undefined.

C — Set if R_n is out of bounds; cleared otherwise.

CHK2**Check Register Against Bounds
(MC68020, MC68030, MC68040, CPU32)****CHK2****Instruction Format:**

| | | | | | | | | | | | | | | | |
|-----|----------|----|----|----|------|------|---|----------|---|-------------------|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | SIZE | | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | MODE | | REGISTER | | | | | | | |
| D/A | REGISTER | | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Instruction Fields:

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the location of the bounds operands. Only control addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | — | — |
| – (An) | — | — |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |
| (bd,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

D/A field—Specifies whether an address register or data register is to be checked.

- 0 — Data register
- 1 — Address register

Register field—Specifies the address or data register that contains the value to be checked.

CLR**Clear an Operand
(M68000 Family)****CLR****Operation:** 0 → Destination**Assembler****Syntax:** CLR < ea >**Attributes:** Size = (Byte, Word, Long)**Description:** Clears the destination operand to zero. The size of the operation may be specified as byte, word, or long.**Condition Codes:**

| X | N | Z | V | C |
|---|---|---|---|---|
| — | 0 | 1 | 0 | 0 |

X — Not affected.

N — Always cleared.

Z — Always set.

V — Always cleared.

C — Always cleared.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|------|---|-------------------|---|---|----------|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

CLR

Clear an Operand (M68000 Family)

CLR

Instruction Fields:

Size field—Specifies the size of the operation.

- 00— Byte operation
- 01— Word operation
- 10— Long operation

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NOTE

In the MC68000 and MC68008 a memory location is read before it is cleared.

CMP

Compare (M68000 Family)

CMP

Operation: Destination – Source → cc

Assembler

Syntax: CMP < ea > , Dn

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand from the destination data register and sets the condition codes according to the result; the data register is not changed. The size of the operation can be byte, word, or long.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | * | * |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Set if a borrow occurs; cleared otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|--------|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 1 | 1 | REGISTER | | | OPMODE | | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

Register field—Specifies the destination data register.

Opmode field

| Byte | Word | Long | Operation |
|------|------|------|-------------|
| 000 | 001 | 010 | Dn – < ea > |

CMP**Compare
(M68000 Family)****CMP**

Effective Address field—Specifies the source operand. All addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An* | 001 | reg. number:An |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)† | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Word and Long only.

**Can be used with CPU32.

NOTE

CMPA is used when the destination is an address register. CMPI is used when the source is immediate data. CMPM is used for memory-to-memory compares. Most assemblers automatically make the distinction.

CMPA

Compare Address (M68000 Family)

CMPA

Operation: Destination – Source → cc

Assembler

Syntax: CMPA < ea > , An

Attributes: Size = (Word, Long)

Description: Subtracts the source operand from the destination address register and sets the condition codes according to the result; the address register is not changed. The size of the operation can be specified as word or long. Word length source operands are sign-extended to 32 bits for comparison.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | * |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.

C — Set if a borrow is generated; cleared otherwise.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----------|----|---|--------|---|---|---------------------------|---|---|----------|---|---|
| 1 | 0 | 1 | 1 | REGISTER | | | OPMODE | | | EFFECTIVE ADDRESS MODE | | | REGISTER | | |

CMPA

Compare Address (M68000 Family)

CMPA

Instruction Fields:

Register field—Specifies the destination address register.

Opmode field—Specifies the size of the operation.

011— Word operation; the source operand is sign-extended to a long operand, and the operation is performed on the address register using all 32 bits.

111— Long operation.

Effective Address field—Specifies the source operand. All addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | 001 | reg. number:An |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn) | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

CMPI

Compare Immediate (M68000 Family)

CMPI

Operation: Destination – Immediate Data → cc

Assembler

Syntax: CMPI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the immediate data from the destination operand and sets the condition codes according to the result; the destination location is not changed. The size of the operation may be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | * | * |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Set if a borrow occurs; cleared otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|------------------|----|----|----|----|----|---|---|-----------------|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |
| 16-BIT WORD DATA | | | | | | | | 8-BIT BYTE DATA | | | | | | | |
| 32-BIT LONG DATA | | | | | | | | | | | | | | | |

CMPI

Compare Immediate (M68000 Family)

CMPI

Instruction Fields:

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the destination operand. Only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|--------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC)* | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn)* | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|-----|-----|
| (bd,An,Xn)** | 110 | reg. number:An | (bd,PC,Xn)† | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

*PC relative addressing modes do not apply to MC68000, MC68008, or MC6801.

**Can be used with CPU32.

Immediate field—Data immediately following the instruction.

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is the next two immediate words.

CMPM

Compare Memory (M68000 Family)

CMPM

Operation: Destination – Source → cc

Assembler

Syntax: CMPM (Ay) + ,(Ax) +

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand from the destination operand and sets the condition codes according to the results; the destination location is not changed. The operands are always addressed with the postincrement addressing mode, using the address registers specified in the instruction. The size of the operation may be specified as byte, word, or long.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | * |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.

C — Set if a borrow is generated; cleared otherwise.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|-------------|----|---|---|------|---|---|---|---|-------------|---|---|
| 1 | 0 | 1 | 1 | REGISTER Ax | | | 1 | SIZE | | 0 | 0 | 1 | REGISTER Ay | | |

Instruction Fields:

Register Ax field—(always the destination) Specifies an address register in the postincrement addressing mode.

Size field—Specifies the size of the operation.

00 — Byte operation

01 — Word operation

10 — Long operation

Register Ay field—(always the source) Specifies an address register in the postincrement addressing mode.

CMP2

Compare Register Against Bounds (MC68020, MC68030, MC68040, CPU32)

CMP2

Operation: Compare $R_n < LB$ or $R_n > UB$ and Set Condition Codes

Assembler

Syntax: `CMP2 < ea > ,Rn`

Attributes: Size = (Byte, Word, Long)

Description: Compares the value in R_n to each bound. The effective address contains the bounds pair: upper bound following the lower bound. For signed comparisons, the arithmetically smaller value should be used as the lower bound. For unsigned comparisons, the logically smaller value should be the lower bound.

The size of the data and the bounds can be specified as byte, word, or long. If R_n is a data register and the operation size is byte or word, only the appropriate low-order part of R_n is checked. If R_n is an address register and the operation size is byte or word, the bounds operands are sign-extended to 32 bits, and the resultant operands are compared to the full 32 bits of A_n .

If the upper bound equals the lower bound, the valid range is a single value.

NOTE

This instruction is identical to `CHK2` except that it sets condition codes rather than taking an exception when the value in R_n is out of bounds.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | U | * | U | * |

X — Not affected.

N — Undefined.

Z — Set if R_n is equal to either bound; cleared otherwise.

V — Undefined.

C — Set if R_n is out of bounds; cleared otherwise.

CMP2**Compare Register Against Bounds
(MC68020, MC68030, MC68040, CPU32)****CMP2****Instruction Format:**

| | | | | | | | | | | | | | | | |
|-----|----------|----|----|----|------|---|---|------|---|-------------------|----------|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | SIZE | | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | MODE | | | REGISTER | | | | |
| D/A | REGISTER | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Instruction Fields:

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the location of the bounds pair. Only control addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | — | — |
| – (An) | — | — |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |
| (bd,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

D/A field—Specifies whether an address register or data register is compared.

- 0 — Data register
- 1 — Address register

Register field—Specifies the address or data register that contains the value to be checked.

cpBcc**Branch on Coprocessor Condition
(MC68020, MC68030)****cpBcc**

Operation: If cpcc True
Then Scan PC + d_n → PC

Assembler

Syntax: cpBcc < label >

Attributes: Size = (Word, Long)

Description: If the specified coprocessor condition is true, program execution continues at location scan PC + displacement. The value of the scan PC is the address of the first displacement word. The displacement is a twos complement integer that represents the relative distance in bytes from the scan PC to the destination program counter. The displacement can be either 16 or 32 bits. The coprocessor determines the specific condition from the condition field in the operation word.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|--|----|----|----|----------------|----|---|---|---|------|-----------------------|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | COPROCESSOR ID | | | 0 | 1 | SIZE | COPROCESSOR CONDITION | | | | | |
| OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS | | | | | | | | | | | | | | | |
| WORD OR | | | | | | | | | | | | | | | |
| LONG-WORD DISPLACEMENT | | | | | | | | | | | | | | | |

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation. Coprocessor ID of 000 results in an F-line exception for the MC68030.

Size field—Specifies the size of the displacement.

- 0 — The displacement is 16 bits.
- 1 — The displacement is 32 bits.

Coprocessor Condition field—Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which provides directives to the main processor for processing this instruction.

16-Bit Displacement field—The displacement value occupies 16 bits.

32-Bit Displacement field—The displacement value occupies 32 bits.

cpDBcc

**Test Coprocessor Condition
Decrement and Branch
(MC68020, MC68030)**

cpDBcc

Operation: If cpcc False
Then ($D_n - 1 \rightarrow D_n$; If $D_n \neq -1$ Then Scan $PC + d_n \rightarrow PC$)

Assembler

Syntax: cpDBcc D_n , < label >

Attributes: Size = (Word)

Description: If the specified coprocessor condition is true, execution continues with the next instruction. Otherwise, the low-order word in the specified data register is decremented by one. If the result is equal to -1 , execution continues with the next instruction. If the result is not equal to -1 , execution continues at the location indicated by the value of the scan PC plus the sign-extended 16-bit displacement. The value of the scan PC is the address of the displacement word. The displacement is a twos complement integer that represents the relative distance in bytes from the scan PC to the destination program counter. The coprocessor determines the specific condition from the condition word that follows the operation word.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | | |
|--|----|----|----|-------------------|----|---|---|---|---|-----------------------|---|---|---|----------|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 1 | COPROCESSOR ID | | | | 0 | 0 | 1 | 0 | 0 | 1 | REGISTER | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | COPROCESSOR CONDITION | | | | | | |
| OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS | | | | | | | | | | | | | | | | |
| 16-BIT DISPLACEMENT | | | | | | | | | | | | | | | | |

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation; coprocessor ID of 000 results in an F-line exception for the MC68030.

Register field—Specifies the data register used as the counter.

Coprocessor Condition field—Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which provides directives to the main processor for processing this instruction.

Displacement field—Specifies the distance of the branch (in bytes).

cpGEN**Coprocessor General Function
(MC68020, MC68030)****cpGEN****Operation:** Pass Command Word to Coprocessor**Assembler****Syntax:** cpGEN < parameters as defined by coprocessor >**Attributes:** Unsized**Description:** Transfers the command word that follows the operation word to the specified coprocessor. The coprocessor determines the specific operation from the command word. Usually a coprocessor defines specific instances of this instruction to provide its instruction set.**Condition Codes:**

May be modified by coprocessor; unchanged otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|--|----|----|----|-------------------|----|---|---|---|---|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | COPROCESSOR ID | | | 0 | 0 | 0 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| COPROCESSOR-DEPENDENT COMMAND WORD | | | | | | | | | | | | | | | |
| OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR-DEFINED EXTENSIONWORDS | | | | | | | | | | | | | | | |

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation; note that coprocessor ID of 000 is reserved for MMU instructions for the MC68030.

Effective Address field—Specifies the location of any operand not resident in the coprocessor. The allowable addressing modes are determined by the operation to be performed.

Coprocessor Command field—Specifies the coprocessor operation to be performed. This word is passed to the coprocessor, which in turn provides directives to the main processor for processing this instruction.

cpScc**Set on Coprocessor Condition
(MC68020, MC68030)****cpScc**

Operation: If cpcc True
 Then 1s → Destination
 Else 0s → Destination

Assembler

Syntax: cpScc < ea >

Attributes: Size = (Byte)

Description: Tests the specified coprocessor condition code. If the condition is true, the byte specified by the effective address is set to TRUE (all ones); otherwise, that byte is set to FALSE (all zeros). The coprocessor determines the specific condition from the condition word that follows the operation word.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|--|----|----|----|-------------------|----|---|---|---|---|--------------------------------------|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | COPROCESSOR ID | | | 0 | 0 | 1 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | COPROCESSOR CONDITION | | | | | |
| OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR-DEFINED EXTENSIONWORDS | | | | | | | | | | | | | | | |

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation. Coprocessor ID of 000 results in an F-line exception for the MC68030.

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |
| (bd,An,Xn) | 110 | reg. number:An | (bd,PC,Xn) | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

Coprocessor Condition field—Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which in turn provides directives to the main processor for processing this instruction.

cpTRAPcc Trap on Coprocessor Condition cpTRAPcc (MC68020, MC68030)

Operation: If cpcc True
Then TRAP

Assembler Syntax: cpTRAPcc
cpTRAPcc # < data >

Attributes: Unsized or Size = (Word, Long)

Description: Tests the specified coprocessor condition code; if the selected coprocessor condition is true, the processor initiates a cpTRAPcc exception, vector number 7. The program counter value placed on the stack is the address of the next instruction. If the selected condition is not true, no operation is performed, and execution continues with the next instruction. The coprocessor determines the specific condition from the condition word that follows the operation word. Following the condition word is a user-defined data operand specified as immediate data to be used by the trap handler.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|--|----|----|----|----------------|----|---|---|---|---|-----------------------|---|---|--------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | COPROCESSOR ID | | | 0 | 0 | 1 | 1 | 1 | 1 | OPMODE | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | COPROCESSOR CONDITION | | | | | |
| OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS | | | | | | | | | | | | | | | |
| OPTIONAL WORD | | | | | | | | | | | | | | | |
| OR LONG-WORD OPERAND | | | | | | | | | | | | | | | |

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation; coprocessor ID of 000 results in an F-line exception for the MC68030.

Opmode field—Selects the instruction form.

010—Instruction is followed by one operand word.

011—Instruction is followed by two operand words.

100—Instruction has no following operand words.

Coprocessor Condition field—Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which provides directives to the main processor for processing this instruction.

DBcc**Test Condition, Decrement, and Branch
(M68000 Family)****DBcc**

Operation: If Condition False
Then ($D_n - 1 \rightarrow D_n$; If $D_n \neq -1$ Then $PC + d_n \rightarrow PC$)

Assembler

Syntax: DBcc Dn, < label >

Attributes: Size = (Word)

Description: Controls a loop of instructions. The parameters are a condition code, a data register (counter), and a displacement value. The instruction first tests the condition for termination; if it is true, no operation is performed. If the termination condition is not true, the low-order 16 bits of the counter data register decrement by one. If the result is -1 , execution continues with the next instruction. If the result is not equal to -1 , execution continues at the location indicated by the current value of the program counter plus the sign-extended 16-bit displacement. The value in the program counter is the address of the instruction word of the DBcc instruction plus two. The displacement is a twos complement integer that represents the relative distance in bytes from the current program counter to the destination program counter. Condition code cc specifies one of the following conditional tests (refer to Table 3-19 for more information on these conditional tests):

| Mnemonic | Condition | Mnemonic | Condition |
|----------|------------------|----------|----------------|
| CC(HI) | Carry Clear | LS | Low or Same |
| CS(LO) | Carry Set | LT | Less Than |
| EQ | Equal | MI | Minus |
| F | False | NE | Not Equal |
| GE | Greater or Equal | PL | Plus |
| GT | Greater Than | T | True |
| HI | High | VC | Overflow Clear |
| LE | Less or Equal | VS | Overflow Set |

Condition Codes:

Not affected.

DBcc**Test Condition, Decrement, and Branch
(M68000 Family)****DBcc****Instruction Format:**

| | | | | | | | | | | | | | | | |
|---------------------|----|----|----|-----------|----|---|---|---|---|---|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | CONDITION | | | | 1 | 1 | 0 | 0 | 1 | REGISTER | | |
| 16-BIT DISPLACEMENT | | | | | | | | | | | | | | | |

Instruction Fields:

Condition field—The binary code for one of the conditions listed in the table.

Register field—Specifies the data register used as the counter.

Displacement field—Specifies the number of bytes to branch.

NOTE

The terminating condition is similar to the UNTIL loop clauses of high-level languages. For example: DBMI can be stated as "decrement and branch until minus".

Most assemblers accept DBRA for DBF for use when only a count terminates the loop (no condition is tested).

A program can enter a loop at the beginning or by branching to the trailing DBcc instruction. Entering the loop at the beginning is useful for indexed addressing modes and dynamically specified bit operations. In this case, the control index count must be one less than the desired number of loop executions. However, when entering a loop by branching directly to the trailing DBcc instruction, the control count should equal the loop execution count. In this case, if a zero count occurs, the DBcc instruction does not branch, and the main loop is not executed.

DIVS, DIVSL

Signed Divide
(M68000 Family)

DIVS, DIVSL

Operation: Destination \div Source \rightarrow Destination

Assembler Syntax: DIVS.W < ea > ,Dn32/16 \rightarrow 16r – 16q
 *DIVS.L < ea > ,Dq 32/32 \rightarrow 32q
 *DIVS.L < ea > ,Dr:Dq 64/32 \rightarrow 32r – 32q
 *DIVSL.L < ea > ,Dr:Dq 32/32 \rightarrow 32r – 32q
 *Applies to MC68020, MC68030, MC68040, CPU32 only

Attributes: Size = (Word, Long)

Description: Divides the signed destination operand by the signed source operand and stores the signed result in the destination. The instruction uses one of four forms. The word form of the instruction divides a long word by a word. The result is a quotient in the lower word (least significant 16 bits) and a remainder in the upper word (most significant 16 bits). The sign of the remainder is the same as the sign of the dividend.

The first long form divides a long word by a long word. The result is a long quotient; the remainder is discarded.

The second long form divides a quad word (in any two data registers) by a long word. The result is a long-word quotient and a long-word remainder.

The third long form divides a long word by a long word. The result is a long-word quotient and a long-word remainder.

Two special conditions may arise during the operation:

1. Division by zero causes a trap.
2. Overflow may be detected and set before the instruction completes. If the instruction detects an overflow, it sets the overflow condition code, and the operands are unaffected.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | 0 |

X—Not affected.

N — Set if the quotient is negative; cleared otherwise; undefined if overflow or divide by zero occurs.

Z — Set if the quotient is zero; cleared otherwise; undefined if overflow or divide by zero occurs.

V — Set if division overflow occurs; undefined if divide by zero occurs; cleared otherwise.

C — Always cleared.

DIVS, DIVSLSigned Divide
(M68000 Family)**DIVS, DIVSL****Instruction Format:**

WORD

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---------------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 0 | REGISTER | | | 1 | 1 | 1 | EFFECTIVE ADDRESS MODE | | | REGISTER | | |

Instruction Fields:

Register field—Specifies any of the eight data registers. This field always specifies the destination operand.

Effective Address field—Specifies the source operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

NOTE

Overflow occurs if the quotient is larger than a 16-bit signed integer.

DIVS, DIVSLSigned Divide
(M68000 Family)**DIVS, DIVSL****Instruction Format:**

LONG

| | | | | | | | | | | | | | | | |
|----|-------------|----|----|----|------|---|---|---|---|-------------------|---|---|-------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |
| 0 | REGISTER Dq | | | 1 | SIZE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | REGISTER Dr | | |

Instruction Fields:

Effective Address field—Specifies the source operand. Only data alterable addressing modes can be used as listed in the following tables:

MC68020, MC68030, and MC68040 only

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |
| (bd,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

Register Dq field—Specifies a data register for the destination operand. The low-order 32 bits of the dividend comes from this register, and the 32-bit quotient is loaded into this register.

Size field—Selects a 32- or 64-bit division operation.

0 — 32-bit dividend is in register Dq.

1 — 64-bit dividend is in Dr – Dq.

DIVS, DIVSL**Signed Divide
(M68000 Family)****DIVS, DIVSL**

Register Dr field—After the division, this register contains the 32-bit remainder. If Dr and Dq are the same register, only the quotient is returned. If the size field is 1, this field also specifies the data register that contains the high-order 32 bits of the dividend.

NOTE

Overflow occurs if the quotient is larger than a 32-bit signed integer.

DIVU, DIVUL

Unsigned Divide
(M68000 Family)

DIVU, DIVUL

Operation: Destination \div Source \rightarrow Destination

Assembler Syntax: DIVU.W < ea > ,Dn32/16 \rightarrow 16r – 16q
 *DIVU.L < ea > ,Dq 32/32 \rightarrow 32q
 *DIVU.L < ea > ,Dr:Dq 64/32 \rightarrow 32r – 32q
 *DIVUL.L < ea > ,Dr:Dq 32/32 \rightarrow 32r – 32q
 *Applies to MC68020, MC68030, MC68040, CPU32 only.

Attributes: Size = (Word, Long)

Description: Divides the unsigned destination operand by the unsigned source operand and stores the unsigned result in the destination. The instruction uses one of four forms. The word form of the instruction divides a long word by a word. The result is a quotient in the lower word (least significant 16 bits) and a remainder in the upper word (most significant 16 bits).

The first long form divides a long word by a long word. The result is a long quotient; the remainder is discarded.

The second long form divides a quad word (in any two data registers) by a long word. The result is a long-word quotient and a long-word remainder.

The third long form divides a long word by a long word. The result is a long-word quotient and a long-word remainder.

Two special conditions may arise during the operation:

1. Division by zero causes a trap.
2. Overflow may be detected and set before the instruction completes. If the instruction detects an overflow, it sets the overflow condition code, and the operands are unaffected.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | 0 |

X — Not affected.

N — Set if the quotient is negative; cleared otherwise; undefined if overflow or divide by zero occurs.

Z — Set if the quotient is zero; cleared otherwise; undefined if overflow or divide by zero occurs.

V — Set if division overflow occurs; cleared otherwise; undefined if divide by zero occurs.

C — Always cleared.

DIVU, DIVUL**Unsigned Divide
(M68000 Family)****DIVU, DIVUL****Instruction Format:**

WORD

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 0 | REGISTER | | | 0 | 1 | 1 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |

Instruction Fields:

Register field—Specifies any of the eight data registers; this field always specifies the destination operand.

Effective Address field—Specifies the source operand. Only data addressing modes can be used as listed in the following tables:

MC68020, MC68030, and MC68040 only

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | 111 | 100 |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|-----|-----|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

**Can be used with CPU32.

NOTE

Overflow occurs if the quotient is larger than a 16-bit signed integer.

DIVU, DIVULUnsigned Divide
(M68000 Family)**DIVU, DIVUL****Instruction Format:**

LONG

| | | | | | | | | | | | | | | | |
|----|-------------|----|----|----|------|---|---|---|---|-------------------|---|---|-------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |
| 0 | REGISTER Dq | | | 0 | SIZE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | REGISTER Dr | | |

Instruction Fields:

Effective Address field—Specifies the source operand. Only data addressing modes can be used as listed in the following tables:

MC68020, MC68030, and MC68040 only

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |
| (bd,An,Xn)* | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |
| (bd,PC,Xn)* | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

Register Dq field—Specifies a data register for the destination operand. The low-order 32 bits of the dividend comes from this register, and the 32-bit quotient is loaded into this register.

Size field—Selects a 32- or 64-bit division operation.

0 — 32-bit dividend is in register Dq.

1 — 64-bit dividend is in Dr – Dq.

DIVU, DIVUL**Unsigned Divide
(M68000 Family)****DIVU, DIVUL**

Register Dr field—After the division, this register contains the 32-bit remainder. If Dr and Dq are the same register, only the quotient is returned. If the size field is 1, this field also specifies the data register that contains the high-order 32 bits of the dividend.

NOTE

Overflow occurs if the quotient is larger than a 32-bit unsigned integer.

EOR

Exclusive-OR Logical (M68000 Family)

EOR

Operation: Source \oplus Destination \rightarrow Destination

Assembler

Syntax: EOR Dn, < ea >

Attributes: Size = (Byte, Word, Long)

Description: Performs an exclusive-OR operation on the destination operand using the source operand and stores the result in the destination location. The size of the operation may be specified to be byte, word, or long. The source operand must be a data register. The destination operand is specified in the effective address field.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the result is set; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

WORD

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|--------|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 1 | 1 | REGISTER | | | OPMODE | | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

Register field—Specifies any of the eight data registers.

Opmode field

| Byte | Word | Long | Operation |
|------|------|------|---|
| 100 | 101 | 110 | < ea > \oplus Dn \rightarrow < ea > |

EOR**Exclusive-OR Logical
(M68000 Family)****EOR**

Effective Address field—Specifies the destination ope data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NOTE

Memory-to-data-register operations are not allowed. Most assemblers use EORI when the source is immediate data.

EORI

Exclusive-OR Immediate (M68000 Family)

EORI

Operation: Immediate Data \oplus Destination \rightarrow Destination

Assembler

Syntax: EORI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Performs an exclusive-OR operation on the destination operand using the immediate data and the destination operand and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the result is set; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|------------------|----|----|----|----|----|---|---|-----------------|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |
| 16-BIT WORD DATA | | | | | | | | 8-BIT BYTE DATA | | | | | | | |
| 32-BIT LONG DATA | | | | | | | | | | | | | | | |

EORI**Exclusive-OR Immediate
(M68000 Family)****EORI****Instruction Fields:**

Size field—Specifies the size of the operation.

00— Byte operation

01— Word operation

10— Long operation

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn) | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

Immediate field—Data immediately following the instruction.

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is next two immediate words.

EORI to CCR

Exclusive-OR Immediate to Condition Code (M68000 Family)

EORI to CCR

Operation: Source \oplus CCR \rightarrow CCR

Assembler

Syntax: EORI # < data > ,CCR

Attributes: Size = (Byte)

Description: Performs an exclusive-OR operation on the condition code register using the immediate operand and stores the result in the condition code register (low-order byte of the status register). All implemented bits of the condition code register are affected.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | * | * | * | * |

X — Changed if bit 4 of immediate operand is one; unchanged otherwise.

N — Changed if bit 3 of immediate operand is one; unchanged otherwise.

Z — Changed if bit 2 of immediate operand is one; unchanged otherwise.

V — Changed if bit 1 of immediate operand is one; unchanged otherwise.

C — Changed if bit 0 of immediate operand is one; unchanged otherwise.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|-----------------|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| | | | | | | | | 8-BIT BYTE DATA | | | | | | | |

EXG**Exchange Registers
(M68000 Family)****EXG**

Operation: Rx \longleftrightarrow Ry

Assembler Syntax: EXG Dx,Dy
EXG Ax,Ay EXG Dx,Ay

Attributes: Size = (Long)

Description: Exchanges the contents of two 32-bit registers. The instruction performs three types of exchanges.

1. Exchange data registers.
2. Exchange address registers.
3. Exchange a data register and an address register.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|-------------|----|---|---|--------|---|---|---|-------------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 0 | REGISTER Rx | | | 1 | OPMODE | | | | REGISTER Ry | | | |

Instruction Fields:

Register Rx field—Specifies either a data register or an address register depending on the mode. If the exchange is between data and address registers, this field always specifies the data register.

Opmode field—Specifies the type of exchange.

01000—Data registers

01001—Address registers

10001—Data register and address register

Register Ry field—Specifies either a data register or an address register depending on the mode. If the exchange is between data and address registers, this field always specifies the address register.

EXT, EXTB

**Sign-Extend
(M68000 Family)**

EXT, EXTB

Operation: Destination Sign-Extended → Destination

Assembler EXT.W Dnextend byte to word

Syntax: EXT.L Dnextend word to long word

EXTB.L Dnextend byte to long word (MC68020, MC68030
MC68040, CPU32)

Attributes: Size = (Word, Long)

Description: Extends a byte in a data register to a word or a long word, or a word in a data register to a long word, by replicating the sign bit to the left. If the operation extends a byte to a word, bit 7 of the designated data register is copied to bits 15 – 8 of that data register. If the operation extends a word to a long word, bit 15 of the designated data register is copied to bits 31 – 16 of the data register. The EXTB form copies bit 7 of the designated register to bits 31 – 8 of the data register.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|--------|---|---|---|---|---|----------|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | OPMODE | | | 0 | 0 | 0 | REGISTER | | |

Instruction Fields:

Opmode field—Specifies the size of the sign-extension operation.

010—Sign-extend low-order byte of data register to word.

011—Sign-extend low-order word of data register to long.

111—Sign-extend low-order byte of data register to long.

Register field—Specifies the data register is to be sign-extended.

ILLEGAL**Take Illegal Instruction Trap
(M68000 Family)****ILLEGAL**

Operation: *SSP – 2 → SSP; Vector Offset → (SSP);
 SSP – 4 → SSP; PC → (SSP);
 SSP – 2 → SSP; SR → (SSP);
 Illegal Instruction Vector Address → PC

*The MC68000 and MC68008 cannot write the vector offset and format code to the system stack.

Assembler

Syntax: ILLEGAL

Attributes: Unsized

Description: Forces an illegal instruction exception, vector number 4. All other illegal instruction bit patterns are reserved for future extension of the instruction set and should not be used to force an exception.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

JMP

Jump (M68000 Family)

JMP

Operation: Destination Address → PC

Assembler

Syntax: JMP < ea >

Attributes: Unsized

Description: Program execution continues at the effective address specified by the instruction. The addressing mode for the effective address must be a control addressing mode.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Field:

Effective Address field—Specifies the address of the next instruction. Only control addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | — | — |
| – (An) | — | — |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

JSR

Jump to Subroutine (M68000 Family)

JSR

Operation: SP – 4 → Sp; PC → (SP); Destination Address → PC

Assembler

Syntax: JSR < ea >

Attributes: Unsized

Description: Pushes the long-word address of the instruction immediately following the JSR instruction onto the system stack. Program execution then continues at the address specified in the instruction.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Field:

Effective Address field—Specifies the address of the next instruction. Only control addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | — | — |
| – (An) | — | — |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

LEA**Load Effective Address
(M68000 Family)****LEA****Operation:** < ea > → An**Assembler****Syntax:** LEA < ea > ,An**Attributes:** Size = (Long)**Description:** Loads the effective address into the specified address register. All 32 bits of the address register are affected by this instruction.**Condition Codes:**

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | REGISTER | | | 1 | 1 | 1 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |

Instruction Fields:

Register field—Specifies the address register to be updated with the effective address.

Effective Address field—Specifies the address to be loaded into the address register. Only control addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | — | — |
| – (An) | — | — |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn) | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

LINK**Link and Allocate
(M68000 Family)****LINK**

Operation: $SP - 4 \rightarrow SP; An \rightarrow (SP); SP \rightarrow An; SP + d_n \rightarrow SP$

Assembler

Syntax: LINK An, # < displacement >

Attributes: Size = (Word, Long*)

*MC68020, MC68030, MC68040 and CPU32 only.

Description: Pushes the contents of the specified address register onto the stack. Then loads the updated stack pointer into the address register. Finally, adds the displacement value to the stack pointer. For word-size operation, the displacement is the sign-extended word following the operation word. For long size operation, the displacement is the long word following the operation word. The address register occupies one long word on the stack. The user should specify a negative displacement in order to allocate stack area.

Condition Codes:

Not affected.

Instruction Format:

WORD

| | | | | | | | | | | | | | | | |
|-------------------|----|----|----|----|----|---|---|---|---|---|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | REGISTER | | |
| WORD DISPLACEMENT | | | | | | | | | | | | | | | |

Instruction Format:

LONG

| | | | | | | | | | | | | | | | |
|-------------------------|----|----|----|----|----|---|---|---|---|---|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | REGISTER | | |
| HIGH-ORDER DISPLACEMENT | | | | | | | | | | | | | | | |
| LOW-ORDER DISPLACEMENT | | | | | | | | | | | | | | | |

LINK

Link and Allocate (M68000 Family)

LINK

Instruction Fields:

Register field—Specifies the address register for the link.

Displacement field—Specifies the twos complement integer to be added to the stack pointer.

NOTE

LINK and UNLK can be used to maintain a linked list of local data and parameter areas on the stack for nested subroutine calls.

LSL, LSR

Logical Shift (M68000 Family)

LSL, LSR

Operation: Destination Shifted By Count → Destination

Assembler Syntax: LSd Dx,Dy
LSd # < data > ,Dy
LSd < ea >
where d is direction, L or R

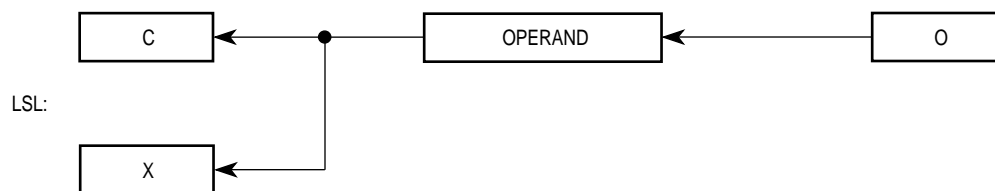
Attributes: Size = (Byte, Word, Long)

Description: Shifts the bits of the operand in the direction specified (L or R). The carry bit receives the last bit shifted out of the operand. The shift count for the shifting of a register is specified in two different ways:

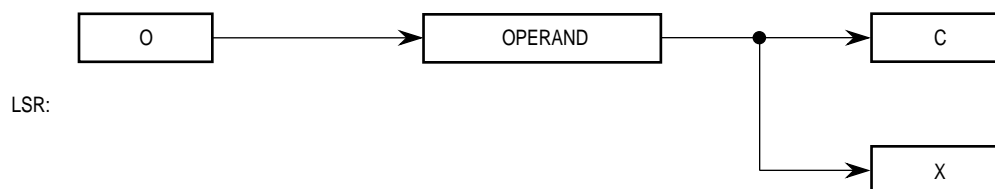
1. Immediate—The shift count (1 – 8) is specified in the instruction.
2. Register—The shift count is the value in the data register specified in the instruction modulo 64.

The size of the operation for register destinations may be specified as byte, word, or long. The contents of memory, < ea > , can be shifted one bit only, and the operand size is restricted to a word.

The LSL instruction shifts the operand to the left the number of positions specified as the shift count. Bits shifted out of the high-order bit go to both the carry and the extend bits; zeros are shifted into the low-order bit.



The LSR instruction shifts the operand to the right the number of positions specified as the shift count. Bits shifted out of the low-order bit go to both the carry and the extend bits; zeros are shifted into the high-order bit.



LSL, LSR

Logical Shift
(M68000 Family)

LSL, LSR

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | 0 | * |

X — Set according to the last bit shifted out of the operand; unaffected for a shift count of zero.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Set according to the last bit shifted out of the operand; cleared for a shift count of zero.

Instruction Format:

REGISTER SHIFTS

| | | | | | | | | | | | | | | | |
|----|----|----|----|--------------------|----|----|------|---|-----|---|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | COUNT/ REGISTER | | dr | SIZE | | i/r | 0 | 1 | REGISTER | | | |

Instruction Fields:

Count/Register field

If $i/r = 0$, this field contains the shift count. The values 1 – 7 represent shifts of 1 – 7; value of zero specifies a shift count of eight.

If $i/r = 1$, the data register specified in this field contains the shift count (modulo 64).

dr field—Specifies the direction of the shift.

0 — Shift right

1 — Shift left

Size field—Specifies the size of the operation.

00 — Byte operation

01 — Word operation

10 — Long operation i/r field

If $i/r = 0$, specifies immediate shift count.

If $i/r = 1$, specifies register shift count.

Register field—Specifies a data register to be shifted.

LSL, LSR

Logical Shift (M68000 Family)

LSL, LSR

Instruction Format:

MEMORY SHIFTS

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|----|---|---|--------------------------------------|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | dr | 1 | 1 | EFFECTIVE ADDRESS MODE REGISTER | | | | | |

Instruction Fields:

dr field—Specifies the direction of the shift.

0 — Shift right

1 — Shift left

Effective Address field—Specifies the operand to be shifted. Only memory alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | — | — | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|---|---|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

MOVE

Move Data from Source to Destination (M68000 Family)

MOVE

Operation: Source → Destination

Assembler

Syntax: MOVE < ea > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Moves the data at the source to the destination location and sets the condition codes according to the data. The size of the operation may be specified as byte, word, or long. Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|------|----|-------------|----|---|------|--------|---|------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | SIZE | | DESTINATION | | | | SOURCE | | | | | | | |
| | | | | REGISTER | | | MODE | | | MODE | | | REGISTER | | |

Instruction Fields:

Size field—Specifies the size of the operand to be moved.

01 — Byte operation

11 — Word operation

10 — Long operation

MOVE**Move Data from Source to Destination
(M68000 Family)****MOVE**

Destination Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

MOVE**Move Data from Source to Destination
(M68000 Family)****MOVE**

Source Effective Address field—Specifies the source operand. All addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | 001 | reg. number:An |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)** | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)** | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*For byte size operation, address register direct is not allowed.

**Can be used with CPU32.

NOTE

Most assemblers use MOVEA when the destination is an address register.

MOVEQ can be used to move an immediate 8-bit value to a data register.

MOVEA

Move Address (M68000 Family)

MOVEA

Operation: Source → Destination

Assembler

Syntax: MOVEA < ea > ,An

Attributes: Size = (Word, Long)

Description: Moves the contents of the source to the destination address register. The size of the operation is specified as word or long. Word-size source operands are sign-extended to 32-bit quantities.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|------|----------------------|----|----|---|---|------|---|-----------------|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | SIZE | DESTINATION REGISTER | | 0 | 0 | 1 | MODE | | SOURCE REGISTER | | | | | |

Instruction Fields:

Size field—Specifies the size of the operand to be moved.

11 — Word operation; the source operand is sign-extended to a long operand and all 32 bits are loaded into the address register.

10 — Long operation.

Destination Register field—Specifies the destination address register.

MOVEA

Move Address (M68000 Family)

MOVEA

Effective Address field—Specifies the location of the source operand. All addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | 001 | reg. number:An |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

MOVE from CCR

**Move from the
Condition Code Register**
(MC68010, MC68020, MC68030, MC68040, CPU32)

MOVE from CCR

Operation: CCR → Destination

Assembler

Syntax: MOVE CCR, < ea >

Attributes: Size = (Word)

Description: Moves the condition code bits (zero-extended to word size) to the destination location. The operand size is a word. Unimplemented bits are read as zeros.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

MOVE from CCR

**Move from the
Condition Code Register**
(MC68010, MC68020, MC68030, MC68040, CPU32)

MOVE from CCR

Instruction Field:

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NOTE

MOVE from CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

**MOVE
to CCR****Move to Condition Code Register
(M68000 Family)****MOVE
to CCR****Operation:** Source → CCR**Assembler****Syntax:** MOVE < ea > ,CCR**Attributes:** Size = (Word)**Description:** Moves the low-order byte of the source operand to the condition code register. The upper byte of the source operand is ignored; the upper byte of the status register is not altered.**Condition Codes:**

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | * | * |

X — Set to the value of bit 4 of the source operand.

N — Set to the value of bit 3 of the source operand.

Z — Set to the value of bit 2 of the source operand.

V — Set to the value of bit 1 of the source operand.

C — Set to the value of bit 0 of the source operand.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

MOVE to CCR

Move to Condition Code Register (M68000 Family)

MOVE to CCR

Instruction Field:

Effective Address field—Specifies the location of the source operand. Only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

NOTE

MOVE to CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

MOVE from SR

Move from the Status Register
(MC68000, MC68008)

MOVE from SR

Operation: SR → Destination

Assembler

Syntax: MOVE SR, < ea >

Attributes: Size = (Word)

Description: Moves the data in the status register to the destination location. The destination is word length. Unimplemented bits are read as zeros.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

NOTE

Use the MOVE from CCR instruction to access only the condition codes. Memory destination is read before it is written to.

MOVE16

Move 16-Byte Block (MC68040)

MOVE16

Operation: Source Block → Destination Block

Assembler Syntax: MOVE16 (Ax) + ,(Ay) +
 MOVE16 (xxx).L,(An)
 MOVE16 (xxx).L,(An) +
 MOVE16 (An),(xxx).L
 MOVE16 (An) + ,(xxx).L

Attributes: Size = (Line)

Description: Moves the source line to the destination line. The lines are aligned to 16-byte boundaries. Applications for this instruction include coprocessor communications, memory initialization, and fast block copy operations.

MOVE16 has two formats. The postincrement format uses the postincrement addressing mode for both source and destination; whereas, the absolute format specifies an absolute long address for either the source or destination.

Line transfers are performed using burst reads and writes, which begin with the long word pointed to by the effective address of the source and destination, respectively. An address register used in the postincrement addressing mode is incremented by 16 after the transfer.

Example: MOVE16 (A0) + \$FE802 A0 = \$1400F

The line at address \$14000 is read into a temporary holding register by a burst read transfer starting with long-word \$14000. Address values in A0 of \$14000 – \$1400F cause the same line to be read, starting at different long words. The line is then written to the line at address \$FE800 beginning with long-word \$FE800 after the instruction A0 contains \$1401F.

Source line at \$14000:

| | | | |
|-------------|-------------|-------------|-------------|
| \$14000 | \$14004 | \$14008 | \$1400C |
| LONG WORD 0 | LONG WORD 1 | LONG WORD 2 | LONG WORD 3 |

Destination line at \$FE8000:

| | | | |
|-------------|-------------|-------------|-------------|
| \$FE800 | \$FE804 | \$FE808 | \$FE80C |
| LONG WORD 0 | LONG WORD 1 | LONG WORD 2 | LONG WORD 3 |

MOVE16

Move 16-Byte Block (MC68040)

MOVE16

Condition Codes:

Not affected.

Instruction Format:

POSTINCREMENT SOURCE AND DESTINATION

| | | | | | | | | | | | | | | | |
|----|-------------|----|----|----|----|---|---|---|---|---|---|---|-------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | REGISTER Ax | | |
| 1 | REGISTER Ay | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Instruction Fields:

Register Ax—Specifies a source address register for the postincrement addressing mode.

Register Ay—Specifies a destination address register for the postincrement addressing mode.

Instruction Format:

Absolute Long Address Source or Destination

| | | | | | | | | | | | | | | | |
|--------------------|----|----|----|----|----|---|---|---|---|---|--------|---|-------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | OPMODE | | REGISTER Ay | | |
| HIGH-ORDER ADDRESS | | | | | | | | | | | | | | | |
| LOW-ORDER ADDRESS | | | | | | | | | | | | | | | |

Instruction Fields:

Opmode field—Specifies the addressing modes used for source and destination:

| Opmode | Source | Destination | Assembler Syntax |
|--------|---------|-------------|------------------------|
| 0 0 | (Ay) + | (xxx).L | MOVE16 (Ay) + ,(xxx).L |
| 0 1 | (xxx).L | (Ay) + | MOVE16 (xxx).L,(Ay) + |
| 1 0 | (Ay) | (xxx).L | MOVE16 (Ay),(xxx).L |
| 1 1 | (xxx).L | (Ay) | MOVE16 (xxx).L,(Ay) |

Register Ay—Specifies an address register for the indirect and postincrement addressing mode used as a source or destination.

32-Bit Address field—Specifies the absolute address used as a source or destination.

MOVEM

Move Multiple Registers (M68000 Family)

MOVEM

Operation: Registers → Destination; Source → Registers

Assembler MOVEM < list > , < ea >

Syntax: MOVEM < ea > , < list >

Attributes: Size = (Word, Long)

Description: Moves the contents of selected registers to or from consecutive memory locations starting at the location specified by the effective address. A register is selected if the bit in the mask field corresponding to that register is set. The instruction size determines whether 16 or 32 bits of each register are transferred. In the case of a word transfer to either address or data registers, each word is sign-extended to 32 bits, and the resulting long word is loaded into the associated register.

Selecting the addressing mode also selects the mode of operation of the MOVEM instruction, and only the control modes, the predecrement mode, and the postincrement mode are valid. If the effective address is specified by one of the control modes, the registers are transferred starting at the specified address, and the address is incremented by the operand length (2 or 4) following each transfer. The order of the registers is from D0 to D7, then from A0 to A7.

If the effective address is specified by the predecrement mode, only a register-to-memory operation is allowed. The registers are stored starting at the specified address minus the operand length (2 or 4), and the address is decremented by the operand length following each transfer. The order of storing is from A7 to A0, then from D7 to D0. When the instruction has completed, the decremented address register contains the address of the last operand stored. For the MC68020, MC68030, MC68040, and CPU32, if the addressing register is also moved to memory, the value written is the initial register value decremented by the size of the operation. The MC68000 and MC68010 write the initial register value (not decremented).

If the effective address is specified by the postincrement mode, only a memory-to-register operation is allowed. The registers are loaded starting at the specified address; the address is incremented by the operand length (2 or 4) following each transfer. The order of loading is the same as that of control mode addressing. When the instruction has completed, the incremented address register contains the address of the last operand loaded plus the operand length. If the addressing register is also loaded from memory, the memory value is ignored and the register is written with the postincremented effective address.

MOVEM

Move Multiple Registers (M68000 Family)

MOVEM

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|--------------------|----|----|----|----|----|---|---|---|------|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | dr | 0 | 0 | 1 | SIZE | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| REGISTER LIST MASK | | | | | | | | | | | | | | | |

Instruction Fields:

dr field—Specifies the direction of the transfer.

- 0 — Register to memory.
- 1 — Memory to register.

Size field—Specifies the size of the registers being transferred.

- 0 — Word transfer
- 1 — Long transfer

Effective Address field—Specifies the memory address for the operation. For register-to-memory transfers, only control alterable addressing modes or the predecrement addressing mode can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | — | — |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

MOVEM**Move Multiple Registers
(M68000 Family)****MOVEM**

For memory-to-register transfers, only control addressing modes or the postincrement addressing mode can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | — | — |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

Register List Mask field—Specifies the registers to be transferred. The low-order bit corresponds to the first register to be transferred; the high-order bit corresponds to the last register to be transferred. Thus, for both control modes and postincrement mode addresses, the mask correspondence is:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

For the predecrement mode addresses, the mask correspondence is reversed:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 |

MOVEP

Move Peripheral Data (M68000 Family)

MOVEP

Operation: Source → Destination

Assembler Syntax: MOVEP Dx,(d16,Ay)

Syntax: MOVEP (d16,Ay),Dx

Attributes: Size = (Word, Long)

Description: Moves data between a data register and alternate bytes within the address space starting at the location specified and incrementing by two. The high-order byte of the data register is transferred first, and the low-order byte is transferred last. The memory address is specified in the address register indirect plus 16-bit displacement addressing mode. This instruction was originally designed for interfacing 8-bit peripherals on a 16-bit data bus, such as the MC68000 bus. Although supported by the MC68020, MC68030, and MC68040, this instruction is not useful for those processors with an external 32-bit bus.

Example: Long transfer to/from an even address.

Byte Organization in Register

| | | | | | | | |
|------------|----|-----------|----|-----------|---|-----------|---|
| 31 | 24 | 23 | 16 | 15 | 8 | 7 | 0 |
| HIGH ORDER | | MID UPPER | | MID LOWER | | LOW ORDER | |

Byte Organization in 16-Bit Memory (Low Address at Top)

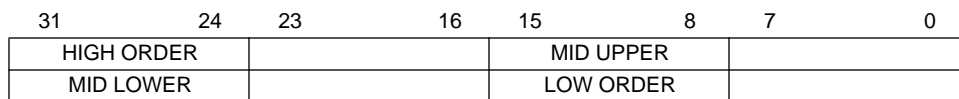
| | | | |
|------------|---|---|---|
| 15 | 8 | 7 | 0 |
| HIGH ORDER | | | |
| MID UPPER | | | |
| MID LOWER | | | |
| LOW ORDER | | | |

MOVEP

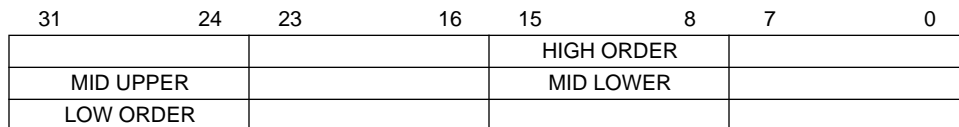
Move Peripheral Data (M68000 Family)

MOVEP

Byte Organization in 32-Bit Memory

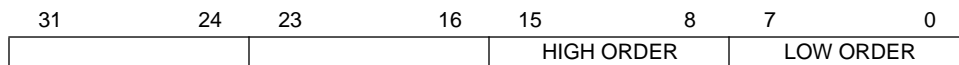


or

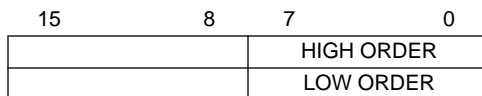


Example: Word transfer to/from (odd address).

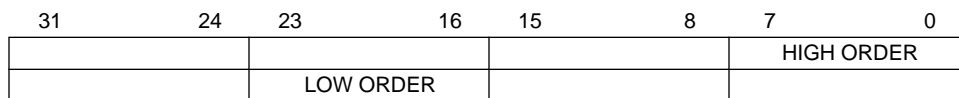
Byte Organization in Register



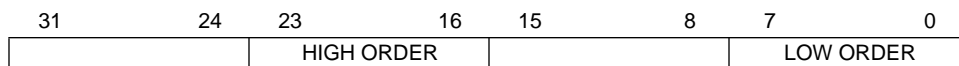
Byte Organization in 16-Bit Memory (Low Address at Top)



Byte Organization in 32-Bit Memory



or



MOVEP

Move Peripheral Data (M68000 Family)

MOVEP

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|---------------|----|---|---|--------|---|---|---|---|---|------------------|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0 | 0 | 0 | 0 | DATA REGISTER | | | | OPMODE | | | 0 | 0 | 1 | ADDRESS REGISTER | | |
| 16-BIT DISPLACEMENT | | | | | | | | | | | | | | | | |

Instruction Fields:

Data Register field—Specifies the data register for the instruction.

Opmode field—Specifies the direction and size of the operation.

100—Transfer word from memory to register.

101—Transfer long from memory to register.

110—Transfer word from register to memory.

111—Transfer long from register to memory.

Address Register field—Specifies the address register which is used in the address register indirect plus displacement addressing mode.

Displacement field—Specifies the displacement used in the operand address.

MOVEQ

Move Quick
(M68000 Family)

MOVEQ

Operation: Immediate Data → Destination

Assembler

Syntax: MOVEQ # < data > ,Dn

Attributes: Size = (Long)

Description: Moves a byte of immediate data to a 32-bit data register. The data in an 8-bit field within the operation word is sign-extended to a long operand in the data register as it is transferred.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----------|----|---|---|------|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | REGISTER | | | 0 | DATA | | | | | | | |

Instruction Fields:

Register field—Specifies the data register to be loaded.

Data field—Eight bits of data, which are sign-extended to a long operand.

MULS

Signed Multiply (M68000 Family)

MULS

Operation: Source x Destination → Destination

Assembler Syntax: MULS.W <ea>, Dn16 x 16 → 32

*MULS.L <ea>, DI 32 x 32 → 32

*MULS.L <ea>, Dh – DI 32 x 32 → 64

*Applies to MC68020, MC68030, MC68040, CPU32

Attributes: Size = (Word, Long)

Description: Multiplies two signed operands yielding a signed result. This instruction has a word operand form and a long operand form.

In the word form, the multiplier and multiplicand are both word operands, and the result is a long-word operand. A register operand is the low-order word; the upper word of the register is ignored. All 32 bits of the product are saved in the destination data register.

In the long form, the multiplier and multiplicand are both long-word operands, and the result is either a long word or a quad word. The long-word result is the low-order 32 bits of the quad-word result; the high-order 32 bits of the product are discarded.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | 0 |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if overflow; cleared otherwise.

C — Always cleared.

NOTE

Overflow ($V = 1$) can occur only when multiplying 32-bit operands to yield a 32-bit result. Overflow occurs if the high-order 32 bits of the quad-word product are not the sign extension of the low-order 32 bits.

MULS

Signed Multiply (M68000 Family)

MULS

Instruction Format:

WORD

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---------------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 0 | REGISTER | | | 1 | 1 | 1 | EFFECTIVE ADDRESS MODE | | | REGISTER | | |

Instruction Fields:

Register field—Specifies a data register as the destination.

Effective Address field—Specifies the source operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

MULS

Signed Multiply (M68000 Family)

MULS

Instruction Format:

LONG

| | | | | | | | | | | | | | | | |
|----|-------------|----|----|------|------|----------|---|---|---|-------------------|---|---|-------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | EFFECTIVE ADDRESS | | | | | |
| | | | | MODE | | REGISTER | | | | | | | | | |
| 0 | REGISTER DI | | | 1 | SIZE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | REGISTER Dh | | |

Instruction Fields:

Effective Address field—Specifies the source operand. Only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | 111 | 100 |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|-----|-----|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

Register DI field—Specifies a data register for the destination operand. The 32-bit multiplicand comes from this register, and the low-order 32 bits of the product are loaded into this register.

Size field—Selects a 32- or 64-bit product.

0 — 32-bit product to be returned to register DI.

1 — 64-bit product to be returned to Dh – DI.

Register Dh field—If size is one, specifies the data register into which the high-order 32 bits of the product are loaded. If Dh = DI and size is one, the results of the operation are undefined. Otherwise, this field is unused.

MULU

Unsigned Multiply (M68000 Family)

MULU

Operation: Source x Destination → Destination

Assembler Syntax: MULU.W <ea>,Dn16 x 16 → 32
*MULU.L <ea>,DI 32 x 32 → 32
*MULU.L <ea>,Dh – DI 32 x 32 → 64

*Applies to MC68020, MC68030, MC68040, CPU32 only

Attributes: Size = (Word, Long)

Description: Multiplies two unsigned operands yielding an unsigned result. This instruction has a word operand form and a long operand form.

In the word form, the multiplier and multiplicand are both word operands, and the result is a long-word operand. A register operand is the low-order word; the upper word of the register is ignored. All 32 bits of the product are saved in the destination data register.

In the long form, the multiplier and multiplicand are both long-word operands, and the result is either a long word or a quad word. The long-word result is the low-order 32 bits of the quad-word result; the high-order 32 bits of the product are discarded.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | 0 |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if overflow; cleared otherwise.

C — Always cleared.

NOTE

Overflow ($V = 1$) can occur only when multiplying 32-bit operands to yield a 32-bit result. Overflow occurs if any of the high-order 32 bits of the quad-word product are not equal to zero.

MULU

Unsigned Multiply (M68000 Family)

MULU

Instruction Format:

WORD

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 0 | REGISTER | | | | 0 | 1 | 1 | EFFECTIVE ADDRESS MODE REGISTER | | | | |

Instruction Fields:

Register field—Specifies a data register as the destination.

Effective Address field—Specifies the source operand. Only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

MULU

Unsigned Multiply (M68000 Family)

MULU

Instruction Format:

LONG

| | | | | | | | | | | | | | | | |
|----|-------------|----|----|----|------|---|---|---|---|-------------------|---|---|-------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |
| 0 | REGISTER DI | | | 0 | SIZE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | REGISTER Dh | | |

Instruction Fields:

Effective Address field—Specifies the source operand. Only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | 111 | 100 |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|-----|-----|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

Register DI field—Specifies a data register for the destination operand. The 32-bit multiplicand comes from this register, and the low-order 32 bits of the product are loaded into this register.

Size field—Selects a 32- or 64-bit product.

0 — 32-bit product to be returned to register DI.

1 — 64-bit product to be returned to Dh – DI.

Register Dh field—If size is one, specifies the data register into which the high-order 32 bits of the product are loaded. If Dh = DI and size is one, the results of the operation are undefined. Otherwise, this field is unused.

NBCD**Negate Decimal with Extend
(M68000 Family)****NBCD**

Operation: $0 - \text{Destination}_{10} - X \rightarrow \text{Destination}$

Assembler

Syntax: NBCD < ea >

Attributes: Size = (Byte)

Description: Subtracts the destination operand and the extend bit from zero. The operation is performed using binary-coded decimal arithmetic. The packed binary-coded decimal result is saved in the destination location. This instruction produces the tens complement of the destination if the extend bit is zero or the nines complement if the extend bit is one. This is a byte operation only.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | U | * | U | * |

X — Set the same as the carry bit.

N — Undefined.

Z — Cleared if the result is nonzero; unchanged otherwise.

V — Undefined.

C — Set if a decimal borrow occurs; cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of the operation. This allows successful tests for zero results upon completion of multiple-precision operations.

NBCD**Negate Decimal with Extend
(M68000 Family)****NBCD****Instruction Format:**

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NEG

Negate (M68000 Family)

NEG

Operation: 0 – Destination → Destination

Assembler

Syntax: NEG < ea >

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the destination operand from zero and stores the result in the destination location. The size of the operation is specified as byte, word, or long.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | * | * |

X — Set the same as the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Cleared if the result is zero; set otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

NEG

Negate (M68000 Family)

NEG

Instruction Fields:

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn) | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NEGX

Negate with Extend (M68000 Family)

NEGX

Operation: $0 - \text{Destination} - X \rightarrow \text{Destination}$

Assembler

Syntax: NEGX < ea >

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the destination operand and the extend bit from zero. Stores the result in the destination location. The size of the operation is specified as byte, word, or long.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | * | * | * | * |

X — Set the same as the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Cleared if the result is nonzero; unchanged otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Set if a borrow occurs; cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of the operation. This allows successful tests for zero results upon completion of multiple-precision operations.

NEGX**Negate with Extend
(M68000 Family)****NEGX****Instruction Format:**

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

Size field—Specifies the size of the operation.

00 — Byte operation

01 — Word operation

10 — Long operation

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NOP

No Operation
(M68000 Family)

NOP

Operation: None

**Assembler
Syntax:** NOP

Attributes: Unsized

Description: Performs no operation. The processor state, other than the program counter, is unaffected. Execution continues with the instruction following the NOP instruction. The NOP instruction does not begin execution until all pending bus cycles have completed. This synchronizes the pipeline and prevents instruction overlap.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

NOT**Logical Complement
(M68000 Family)****NOT****Operation:** \sim Destination \rightarrow Destination**Assembler****Syntax:** NOT < ea >**Attributes:** Size = (Byte, Word, Long)**Description:** Calculates the ones complement of the destination operand and stores the result in the destination location. The size of the operation is specified as byte, word, or long.**Condition Codes:**

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

NOT**Logical Complement
(M68000 Family)****NOT****Instruction Fields:**

Size field—Specifies the size of the operation.

00— Byte operation

01— Word operation

10— Long operation

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

OR**Inclusive-OR Logical
(M68000 Family)****OR****Operation:** Source V Destination → Destination**Assembler Syntax:** OR < ea > ,Dn**Syntax:** OR Dn, < ea >**Attributes:** Size = (Byte, Word, Long)**Description:** Performs an inclusive-OR operation on the source operand and the destination operand and stores the result in the destination location. The size of the operation is specified as byte, word, or long. The contents of an address register may not be used as an operand.**Condition Codes:**

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the result is set; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|--------|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 0 | REGISTER | | | OPMODE | | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

Register field—Specifies any of the eight data registers.

Opmode field

| Byte | Word | Long | Operation |
|------|------|------|----------------------|
| 000 | 001 | 010 | < ea > V Dn → Dn |
| 100 | 101 | 110 | Dn V < ea > → < ea > |

OR

Inclusive-OR Logical (M68000 Family)

OR

Effective Address field—If the location specified is a source operand, only data addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

OR**Inclusive-OR Logical
(M68000 Family)****OR**

If the location specified is a destination operand, only memory alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NOTE

If the destination is a data register, it must be specified using the destination Dn mode, not the destination < ea > mode.

Most assemblers use ORI when the source is immediate data.

ORI

Inclusive-OR (M68000 Family)

ORI

Operation: Immediate Data V Destination → Destination

Assembler

Syntax: ORI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Performs an inclusive-OR operation on the immediate data and the destination operand and stores the result in the destination location. The size of the operation is specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the result is set; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------------|----|----|----|----|----|---|---|-----------------|---|-------------------|---|----------|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| 16-BIT WORD DATA | | | | | | | | 8-BIT BYTE DATA | | | | | | | |
| 32-BIT LONG DATA | | | | | | | | | | | | | | | |

ORI**Inclusive-OR
(M68000 Family)****ORI****Instruction Fields:**

Size field—Specifies the size of the operation.

00— Byte operation

01— Word operation

10— Long operation

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|---|---|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

Immediate field—Data immediately following the instruction.

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is the next two immediate words.

**ORI
to CCR****Inclusive-OR Immediate
to Condition Codes
(M68000 Family)****ORI
to CCR****Operation:** Source V CCR → CCR**Assembler****Syntax:** ORI # < data > ,CCR**Attributes:** Size = (Byte)**Description:** Performs an inclusive-OR operation on the immediate operand and the condition codes and stores the result in the condition code register (low-order byte of the status register). All implemented bits of the condition code register are affected.**Condition Codes:**

| X | N | Z | V | C |
|---|---|---|---|---|
| * | * | * | * | * |

X — Set if bit 4 of immediate operand is one; unchanged otherwise.

N — Set if bit 3 of immediate operand is one; unchanged otherwise.

Z — Set if bit 2 of immediate operand is one; unchanged otherwise.

V — Set if bit 1 of immediate operand is one; unchanged otherwise.

C — Set if bit 0 of immediate operand is one; unchanged otherwise.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|-----------------|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| | | | | | | | | 8-BIT BYTE DATA | | | | | | | |

PACK

Pack
(MC68020, MC68030, MC68040)

PACK

Operation: Source (Unpacked BCD) + Adjustment → Destination (Packed BCD)

Assembler Syntax: PACK – (Ax), – (Ay),# < adjustment >

Syntax: PACK Dx,Dy,# < adjustment >

Attributes: Unsized

Description: Adjusts and packs the lower four bits of each of two bytes into a single byte.

When both operands are data registers, the adjustment is added to the value contained in the source register. Bits 11 – 8 and 3 – 0 of the intermediate result are concatenated and placed in bits 7 – 0 of the destination register. The remainder of the destination register is unaffected.

Source:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | x | x | x | a | b | c | d | x | x | x | x | e | f | g | h |
| Dx | | | | | | | | | | | | | | | |

Add Adjustment Word:

| | | | | | | | | | | | | | | | |
|------------------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|---|
| 15 | | | | | | | | | | | | | | | 0 |
| 16-BIT EXTENSION | | | | | | | | | | | | | | | |

Resulting in:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x' | x' | x' | x' | a' | b' | c' | d' | x' | x' | x' | x' | e' | f' | g' | h' |

Destination:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|----|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | u | u | u | u | u | u | u | a' | b' | c' | d' | e' | f' | g' | h' |
| Dy | | | | | | | | | | | | | | | |

When the predecrement addressing mode is specified, two bytes from the source are fetched and concatenated. The adjustment word is added to the concatenated bytes. Bits 3 – 0 of each byte are extracted. These eight bits are concatenated to form a new byte which is then written to the destination.

PACK

Pack
(MC68020, MC68030, MC68040)

PACK

Source:

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | x | x | x | a | b | c | d |
| x | x | x | x | e | f | g | h |
| Ax | | | | | | | |

Concatenated Word:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | x | x | x | a | b | c | d | x | x | x | x | e | f | g | h |

Add Adjustment Word:

| | | |
|------------------|--|---|
| 15 | | 0 |
| 16-BIT EXTENSION | | |

Destination:

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| a' | b' | c' | d' | e' | f' | g' | h' |
| Ay | | | | | | | |

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|------------------------------|----|----|----|----------------|----|---|---|---|---|---|---|-----|----------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 0 | REGISTER Dy/Ay | | | 1 | 0 | 1 | 0 | 0 | R/M | REGISTER Dx/Ax | | |
| 16-BIT ADJUSTMENT EXTENSION: | | | | | | | | | | | | | | | |

PACK

Pack
(MC68020, MC68030, MC68040)

PACK

Instruction Fields:

Register Dy/Ay field—Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register in the predecrement addressing mode.

R/M field—Specifies the operand addressing mode.

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Dx/Ax field—Specifies the source register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register in the predecrement addressing mode.

Adjustment field—Immediate data word that is added to the source operand. This word is zero to pack ASCII or EBCDIC codes. Other values can be used for other codes.

PEA

Push Effective Address (M68000 Family)

PEA

Operation: $SP - 4 \rightarrow SP; \langle ea \rangle \rightarrow (SP)$

Assembler

Syntax: PEA < ea >

Attributes: Size = (Long)

Description: Computes the effective address and pushes it onto the stack. The effective address is a long address.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Field:

Effective Address field—Specifies the address to be pushed onto the stack. Only control addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | — | — | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | — | — | | | |
| – (An) | — | — | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|-----|-----|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

ROL, ROR

Rotate (Without Extend) (M68000 Family)

ROL, ROR

Operation: Destination Rotated By < count > → Destination

Assembler ROd Dx,Dy

Syntax: ROd # < data > ,Dy ROd < ea > where d is direction, L or R

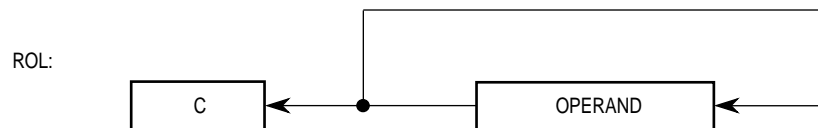
Attributes: Size = (Byte, Word, Long)

Description: Rotates the bits of the operand in the direction specified (L or R). The extend bit is not included in the rotation. The rotate count for the rotation of a register is specified in either of two ways:

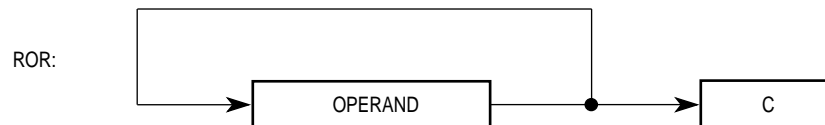
1. Immediate—The rotate count (1 – 8) is specified in the instruction.
2. Register—The rotate count is the value in the data register specified in the instruction, modulo 64.

The size of the operation for register destinations is specified as byte, word, or long. The contents of memory, (ROd < ea >), can be rotated one bit only, and operand size is restricted to a word.

The ROL instruction rotates the bits of the operand to the left; the rotate count determines the number of bit positions rotated. Bits rotated out of the high-order bit go to the carry bit and also back into the low-order bit.



The ROR instruction rotates the bits of the operand to the right; the rotate count determines the number of bit positions rotated. Bits rotated out of the low-order bit go to the carry bit and also back into the high-order bit.



ROL,ROR

Rotate (Without Extend)
(M68000 Family)

ROL,ROR

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | * |

X — Not affected.

N — Set if the most significant bit of the result is set; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Set according to the last bit rotated out of the operand; cleared when the rotate count is zero.

Instruction Format:

REGISTER ROTATE

| | | | | | | | | | | | | | | | |
|----|----|----|----|--------------------|----|---|----|------|---|-----|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | COUNT/ REGISTER | | | dr | SIZE | | i/r | 1 | 1 | REGISTER | | |

Instruction Fields:

Count/Register field:

If $i/r = 0$, this field contains the rotate count. The values 1 – 7 represent counts of 1 – 7, and zero specifies a count of eight.

If $i/r = 1$, this field specifies a data register that contains the rotate count (modulo 64).

dr field—Specifies the direction of the rotate.

0 — Rotate right

1 — Rotate left

Size field—Specifies the size of the operation.

00 — Byte operation

01 — Word operation

10 — Long operation

i/r field—Specifies the rotate count location.

If $i/r = 0$, immediate rotate count.

If $i/r = 1$, register rotate count.

Register field—Specifies a data register to be rotated.

ROL, ROR**Rotate (Without Extend)
(M68000 Family)****ROL, ROR****Instruction Format:**

| MEMORY ROTATE | | | | | | | | | | | | | | | | | |
|---------------|----|----|----|----|----|---|----|---|---|-------------------|---|----------|---|---|---|--|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | dr | 1 | 1 | EFFECTIVE ADDRESS | | | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | | | |

Instruction Fields:

dr field—Specifies the direction of the rotate.

0 — Rotate right

1 — Rotate left

Effective Address field—Specifies the operand to be rotated. Only memory alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | — | — | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|---|---|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

ROXL, ROXR

Rotate with Extend
(M68000 Family)

ROXL, ROXR

Operation: Destination Rotated With X By Count → Destination

Assembler ROXd Dx,Dy

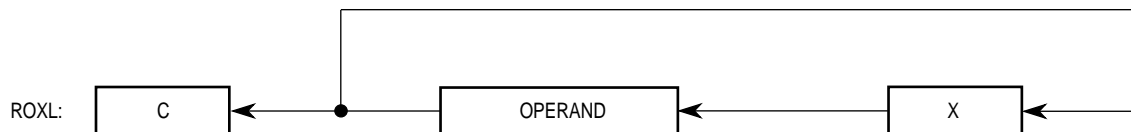
Syntax: ROXd # < data > ,Dy
ROXd < ea >
where d is direction, L or R

Attributes: Size = (Byte, Word, Long)

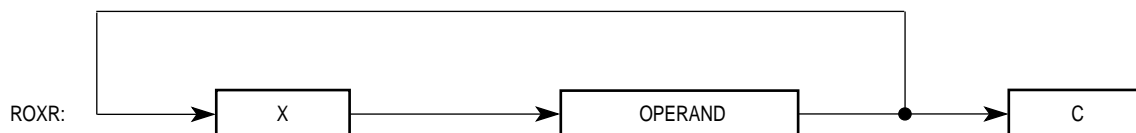
Description: Rotates the bits of the operand in the direction specified (L or R). The extend bit is included in the rotation. The rotate count for the rotation of a register is specified in either of two ways:

1. Immediate—The rotate count (1 – 8) is specified in the instruction.
2. Register—The rotate count is the value in the data register specified in the instruction, modulo 64.

The size of the operation for register destinations is specified as byte, word, or long. The contents of memory, < ea > , can be rotated one bit only, and operand size is restricted to a word. The ROXL instruction rotates the bits of the operand to the left; the rotate count determines the number of bit positions rotated. Bits rotated out of the high-order bit go to the carry bit and the extend bit; the previous value of the extend bit rotates into the low-order bit.



The ROXR instruction rotates the bits of the operand to the right; the rotate count determines the number of bit positions rotated. Bits rotated out of the low-order bit go to the carry bit and the extend bit; the previous value of the extend bit rotates into the high-order bit.



ROXL, ROXRRotate with Extend
(M68000 Family)**ROXL, ROXR****Condition Codes:**

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | 0 | * |

X — Set to the value of the last bit rotated out of the operand; unaffected when the rotate count is zero.

N — Set if the most significant bit of the result is set; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Set according to the last bit rotated out of the operand; when the rotate count is zero, set to the value of the extend bit.

Instruction Format:

| | | | | | | | | | | | | | | | |
|-----------------|----|----|----|--------------------|----|----|------|---|-----|---|---|----------|---|---|---|
| REGISTER ROTATE | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | COUNT/ REGISTER | | dr | SIZE | | i/r | 1 | 0 | REGISTER | | | |

Instruction Fields:

Count/Register field:

If $i/r = 0$, this field contains the rotate count. The values 1 – 7 represent counts of 1 – 7, and zero specifies a count of eight.

If $i/r = 1$, this field specifies a data register that contains the rotate count (modulo 64).

dr field—Specifies the direction of the rotate.

0 — Rotate right

1 — Rotate left

ROXL, ROXR

**Rotate with Extend
(M68000 Family)**

ROXL, ROXR

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

i/r field—Specifies the rotate count location.

- If i/r = 0, immediate rotate count.
- If i/r = 1, register rotate count.

Register field—Specifies a data register to be rotated.

Instruction Format:

| MEMORY ROTATE | | | | | | | | | | | | | | | | |
|---------------|----|----|----|----|----|---|----|---|---|-------------------|---|---|----------|---|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | dr | 1 | 1 | EFFECTIVE ADDRESS | | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | | |

Instruction Fields:

dr field—Specifies the direction of the rotate.

- 0 — Rotate right
- 1 — Rotate left

Effective Address field—Specifies the operand to be rotated. Only memory alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | — | — | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|---|---|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

RTD**Return and Deallocate**
(MC68010, MC68020, MC68030, MC68040, CPU32)**RTD****Operation:** (SP) → PC; SP + 4 + d_n → SP**Assembler****Syntax:** RTD # < displacement >**Attributes:** Unsized**Description:** Pulls the program counter value from the stack and adds the sign-extended 16-bit displacement value to the stack pointer. The previous program counter value is lost.**Condition Codes:**

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 16-BIT DISPLACEMENT | | | | | | | | | | | | | | | |

Instruction Field:

Displacement field—Specifies the twos complement integer to be sign-extended and added to the stack pointer.

RTM**Return from Module
(MC68020)****RTM**

Operation: Reload Saved Module State from Stack

Assembler

Syntax: RTM Rn

Attributes: Unsized

Description: A previously saved module state is reloaded from the top of stack. After the module state is retrieved from the top of the stack, the caller's stack pointer is incremented by the argument count value in the module state.

Condition Codes:

Set according to the content of the word on the stack.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|-----|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | D/A | REGISTER | | |

Instruction Fields:

D/A field—Specifies whether the module data pointer is in a data or an address register.

0 — the register is a data register

1 — the register is an address register

Register field—Specifies the register number for the module data area pointer to be restored from the saved module state. If the register specified is A7 (SP), the updated value of the register reflects the stack pointer operations, and the saved module data area pointer is lost.

RTR**Return and Restore Condition Codes
(M68000 Family)****RTR****Operation:** (SP) → CCR; SP + 2 → SP; (SP) → PC; SP + 4 → SP**Assembler****Syntax:** RTR**Attributes:** Unsized**Description:** Pulls the condition code and program counter values from the stack. The previous condition code and program counter values are lost. The supervisor portion of the status register is unaffected.**Condition Codes:**

Set to the condition codes from the stack.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

RTS**Return from Subroutine
(M68000 Family)****RTS****Operation:** (SP) → PC; SP + 4 → SP**Assembler****Syntax:** RTS**Attributes:** Unsized**Description:** Pulls the program counter value from the stack. The previous program counter value is lost.**Condition Codes:**

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

SBCD**Subtract Decimal with Extend
(M68000 Family)****SBCD**

Operation: Destination₁₀ – Source₁₀ – X → Destination

Assembler SBCD Dx,Dy

Syntax: SBCD – (Ax), – (Ay)

Attributes: Size = (Byte)

Description: Subtracts the source operand and the extend bit from the destination operand and stores the result in the destination location. The subtraction is performed using binary-coded decimal arithmetic; the operands are packed binary-coded decimal numbers. The instruction has two modes:

1. Data register to data register—the data registers specified in the instruction contain the operands.
2. Memory to memory—the address registers specified in the instruction access the operands from memory using the predecrement addressing mode.

This operation is a byte operation only.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | U | * | U | * |

X — Set the same as the carry bit.

N — Undefined.

Z — Cleared if the result is nonzero; unchanged otherwise.

V — Undefined.

C — Set if a borrow (decimal) is generated; cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

SBCD**Subtract Decimal with Extend
(M68000 Family)****SBCD****Instruction Format:**

| | | | | | | | | | | | | | | | | |
|----|----|----|----|----------------|----|---|---|---|---|---|---|---|-----|----------------|---|--|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 0 | 0 | 0 | REGISTER Dy/Ay | | | | 1 | 0 | 0 | 0 | 0 | R/M | REGISTER Dx/Ax | | |

Instruction Fields:

Register Dy/Ay field—Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

R/M field—Specifies the operand addressing mode.

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Dx/Ax field—Specifies the source register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

Scc**Set According to Condition
(M68000 Family)****Scc**

Operation: If Condition True
Then 1s → Destination
Else 0s → Destination

Assembler

Syntax: Scc < ea >

Attributes: Size = (Byte)

Description: Tests the specified condition code; if the condition is true, sets the byte specified by the effective address to TRUE (all ones). Otherwise, sets that byte to FALSE (all zeros). Condition code cc specifies one of the following conditional tests (refer to Table 3-19 for more information on these conditional tests):

| Mnemonic | Condition | Mnemonic | Condition |
|----------|------------------|----------|----------------|
| CC(HI) | Carry Clear | LS | Low or Same |
| CS(LO) | Carry Set | LT | Less Than |
| EQ | Equal | MI | Minus |
| F | False | NE | Not Equal |
| GE | Greater or Equal | PL | Plus |
| GT | Greater Than | T | True |
| HI | High | VC | Overflow Clear |
| LE | Less or Equal | VS | Overflow Set |

Condition Codes:

Not affected.

Scc**Set According to Condition
(M68000 Family)****Scc****Instruction Format:**

| | | | | | | | | | | | | | | | |
|----|----|----|----|-----------|----|---|---|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | CONDITION | | | | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

Condition field—The binary code for one of the conditions listed in the table.

Effective Address field—Specifies the location in which the TRUE/FALSE byte is to be stored. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|---|---|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NOTE

A subsequent NEG.B instruction with the same effective address can be used to change the Scc result from TRUE or FALSE to the equivalent arithmetic value (TRUE = 1, FALSE = 0). In the MC68000 and MC68008, a memory destination is read before it is written.

SUB

Subtract (M68000 Family)

SUB

Operation: Destination – Source → Destination

Assembler Syntax: SUB < ea > ,Dn

Syntax: SUB Dn, < ea >

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand from the destination operand and stores the result in the destination. The size of the operation is specified as byte, word, or long. The mode of the instruction indicates which operand is the source, which is the destination, and which is the operand size.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | * | * |

X — Set to the value of the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.

C — Set if a borrow is generated; cleared otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|--------|---|---|-------------------|------|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 1 | REGISTER | | | OPMODE | | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | | MODE | | REGISTER | | |

SUB**Subtract
(M68000 Family)****SUB****Instruction Fields:**

Register field—Specifies any of the eight data registers.

Opmode field

| Byte | Word | Long | Operation |
|------|------|------|---|
| 000 | 001 | 010 | $D_n - \langle ea \rangle \rightarrow D_n$ |
| 100 | 101 | 110 | $\langle ea \rangle - D_n \rightarrow \langle ea \rangle$ |

Effective Address field—Determines the addressing mode. If the location specified is a source operand, all addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An* | 001 | reg. number:An | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | 111 | 100 |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|-----|-----|
| (bd,An,Xn)** | 110 | reg. number:An | (bd,PC,Xn)** | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

*For byte-sized operation, address register direct is not allowed.

**Can be used with CPU32.

SUB**Subtract
(M68000 Family)****SUB**

If the location specified is a destination operand, only memory alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | — | — |
| An | — | — |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | — | — |
| | | |
| | | |
| (d ₁₆ ,PC) | — | — |
| (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|---|---|
| (bd,PC,Xn)* | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

NOTE

If the destination is a data register, it must be specified as a destination Dn address, not as a destination < ea > address.

Most assemblers use SUBA when the destination is an address register and SUBI or SUBQ when the source is immediate data.

SUBA

Subtract Address (M68000 Family)

SUBA

Operation: Destination – Source → Destination

Assembler

Syntax: SUBA < ea > ,An

Attributes: Size = (Word, Long)

Description: Subtracts the source operand from the destination address register and stores the result in the address register. The size of the operation is specified as word or long. Word-sized source operands are sign-extended to 32-bit quantities prior to the subtraction.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------|----|---|--------|---|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 1 | REGISTER | | | OPMODE | | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

Register field—Specifies the destination, any of the eight address registers.

Opmode field—Specifies the size of the operation.

011— Word operation. The source operand is sign-extended to a long operand and the operation is performed on the address register using all 32 bits.

111— Long operation.

SUBA

Subtract Address (M68000 Family)

SUBA

Effective Address field—Specifies the source operand. All addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register |
|-------------------------|------|----------------|
| Dn | 000 | reg. number:Dn |
| An | 001 | reg. number:An |
| (An) | 010 | reg. number:An |
| (An) + | 011 | reg. number:An |
| – (An) | 100 | reg. number:An |
| (d ₁₆ ,An) | 101 | reg. number:An |
| (d ₈ ,An,Xn) | 110 | reg. number:An |

| Addressing Mode | Mode | Register |
|-------------------------|------|----------|
| (xxx).W | 111 | 000 |
| (xxx).L | 111 | 001 |
| #<data> | 111 | 100 |
| | | |
| | | |
| (d ₁₆ ,PC) | 111 | 010 |
| (d ₈ ,PC,Xn) | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | |
|-----------------|-----|----------------|
| (bd,An,Xn)* | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

| | | |
|-----------------|-----|-----|
| (bd,PC,Xn)* | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

*Can be used with CPU32.

SUBI**Subtract Immediate
(M68000 Family)****SUBI****Operation:** Destination – Immediate Data → Destination**Assembler****Syntax:** SUBI # < data > , < ea >**Attributes:** Size = (Byte, Word, Long)**Description:** Subtracts the immediate data from the destination operand and stores the result in the destination location. The size of the operation is specified as byte, word, or long. The size of the immediate data matches the operation size.**Condition Codes:**

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | * | * |

X — Set to the value of the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Set if a borrow occurs; cleared otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|------------------|----|----|----|----|----|---|---|-----------------|---|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |
| 16-BIT WORD DATA | | | | | | | | 8-BIT BYTE DATA | | | | | | | |
| 32-BIT LONG DATA | | | | | | | | | | | | | | | |

SUBI**Subtract Immediate
(M68000 Family)****SUBI****Instruction Fields:**

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the destination operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|---|---|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

Immediate field—Data immediately following the instruction.

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is the next two immediate words.

SUBQ

Subtract Quick (M68000 Family)

SUBQ

Operation: Destination – Immediate Data → Destination

Assembler

Syntax: SUBQ # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the immediate data (1 – 8) from the destination operand. The size of the operation is specified as byte, word, or long. Only word and long operations can be used with address registers, and the condition codes are not affected. When subtracting from address registers, the entire destination address register is used, despite the operation size.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| * | * | * | * | * |

X — Set to the value of the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Set if a borrow occurs; cleared otherwise.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|------|----|---|---|------|---|-------------------|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | DATA | | | 1 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

SUBQ**Subtract Quick
(M68000 Family)****SUBQ****Instruction Fields:**

Data field—Three bits of immediate data; 1 – 7 represent immediate values of 1 – 7, and zero represents eight.

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the destination location. Only alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An* | 001 | reg. number:An | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|---|---|
| (bd,An,Xn)** | 110 | reg. number:An | (bd,PC,Xn)** | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

*Word and long only.

**Can be used with CPU32.

SUBX

Subtract with Extend (M68000 Family)

SUBX

Operation: Destination – Source – X → Destination

Assembler SUBX Dx,Dy

Syntax: SUBX – (Ax), – (Ay)

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand and the extend bit from the destination operand and stores the result in the destination

location. The instruction has two modes:

1. Data register to data register—the data registers specified in the instruction contain the operands.
2. Memory to memory—the address registers specified in the instruction access the operands from memory using the predecrement addressing mode.

The size of the operand is specified as byte, word, or long.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| * | * | * | * | * |

X — Set to the value of the carry bit.

N — Set if the result is negative; cleared otherwise.

Z — Cleared if the result is nonzero; unchanged otherwise.

V — Set if an overflow occurs; cleared otherwise.

C — Set if a borrow occurs; cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

SUBX**Subtract with Extend
(M68000 Family)****SUBX****Instruction Format:**

| | | | | | | | | | | | | | | | |
|----|----|----|----|----------------|----|---|---|------|---|---|---|-----|----------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 1 | REGISTER Dy/Ay | | | 1 | SIZE | | 0 | 0 | R/M | REGISTER Dx/Ax | | |

Instruction Fields:

Register Dy/Ay field—Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

Size field—Specifies the size of the operation.

00 — Byte operation

01 — Word operation

10 — Long operation

R/M field—Specifies the operand addressing mode.

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Dx/Ax field—Specifies the source register:

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

SWAP

Swap Register Halves (M68000 Family)

SWAP

Operation: Register 31 – 16 \leftrightarrow Register 15 – 0

Assembler

Syntax: SWAP Dn

Attributes: Size = (Word)

Description: Exchange the 16-bit words (halves) of a data register.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the 32-bit result is set; cleared otherwise.

Z — Set if the 32-bit result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|----------|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | | | REGISTER |

Instruction Field:

Register field—Specifies the data register to swap.

TAS

Test and Set an Operand (M68000 Family)

TAS

Operation: Destination Tested → Condition Codes; 1 → Bit 7 of Destination

Assembler

Syntax: TAS < ea >

Attributes: Size = (Byte)

Description: Tests and sets the byte operand addressed by the effective address field. The instruction tests the current value of the operand and sets the N and Z condition bits appropriately. TAS also sets the high-order bit of the operand. The operation uses a locked or read-modify-write transfer sequence. This instruction supports use of a flag or semaphore to coordinate several processors.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the most significant bit of the operand is currently set; cleared otherwise.

Z — Set if the operand was zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|-------------------|------|---|----------|---|---|--|
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | | |
| | | | | | | | | | | | MODE | | REGISTER | | | |

TAS**Test and Set an Operand
(M68000 Family)****TAS****Instruction Fields:**

Effective Address field—Specifies the location of the tested operand. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|-------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An | — | — | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data> | — | — |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC) | — | — |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn) | — | — |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|---|---|
| (bd,An,Xn)* | 110 | reg. number:An | (bd,PC,Xn)* | — | — |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | — | — |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | — | — |

*Can be used with CPU32.

TRAP

Trap (M68000 Family)

TRAP

Operation: 1 → S-Bit of SR
 *SSP – 2 → SSP; Format/Offset → (SSP);
 SSP – 4 → SSP; PC → (SSP); SSP – 2 → SSP;
 SR → (SSP); Vector Address → PC

*The MC68000 and MC68008 do not write vector offset or format code to the system stack.

Assembler

Syntax: TRAP # < vector >

Attributes: Unsized

Description: Causes a TRAP # < vector > exception. The instruction adds the immediate operand (vector) of the instruction to 32 to obtain the vector number. The range of vector values is 0 – 15, which provides 16 vectors.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|--------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | VECTOR | | | |

Instruction Fields:

Vector field—Specifies the trap vector to be taken.

TRAPcc

Trap on Condition (MC68020, MC68030, MC68040, CPU32)

TRAPcc

Operation: If cc
Then TRAP

Assembler Syntax: TRAPcc
TRAPcc.W # < data >
TRAPcc.L # < data >

Attributes: Unsized or Size = (Word, Long)

Description: If the specified condition is true, causes a TRAPcc exception with a vector number 7. The processor pushes the address of the next instruction word (currently in the program counter) onto the stack. If the condition is not true, the processor performs no operation, and execution continues with the next instruction. The immediate data operand should be placed in the next word(s) following the operation word and is available to the trap handler. Condition code cc specifies one of the following conditional tests (refer to Table 3-19 for more information on these conditional tests):

| Mnemonic | Condition | Mnemonic | Condition |
|----------|------------------|----------|----------------|
| CC(HI) | Carry Clear | LS | Low or Same |
| CS(LO) | Carry Set | LT | Less Than |
| EQ | Equal | MI | Minus |
| F | False | NE | Not Equal |
| GE | Greater or Equal | PL | Plus |
| GT | Greater Than | T | True |
| HI | High | VC | Overflow Clear |
| LE | Less or Equal | VS | Overflow Set |

Condition Codes:

Not affected.

TRAPcc

Trap on Condition
(MC68020, MC68030, MC68040, CPU32)

TRAPcc**Instruction Format:**

| | | | | | | | | | | | | | | | |
|---------------|----|----|----|-----------|----|---|---|---|---|---|---|---|--------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | CONDITION | | | | 1 | 1 | 1 | 1 | 1 | OPMODE | | |
| OPTIONAL WORD | | | | | | | | | | | | | | | |
| OR LONG WORD | | | | | | | | | | | | | | | |

Instruction Fields:

Condition field—The binary code for one of the conditions listed in the table.

Opmode field—Selects the instruction form.

010—Instruction is followed by word-sized operand.

011—Instruction is followed by long-word-sized operand.

100—Instruction has no operand.

TRAPV

Trap on Overflow (M68000 Family)

TRAPV

Operation: If V
Then TRAP

**Assembler
Syntax:** TRAPV

Attributes: Unsized

Description: If the overflow condition is set, causes a TRAPV exception with a vector number 7. If the overflow condition is not set, the processor performs no operation and execution continues with the next instruction.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |

TST

Test an Operand (M68000 Family)

TST

Operation: Destination Tested → Condition Codes

Assembler

Syntax: TST < ea >

Attributes: Size = (Byte, Word, Long)

Description: Compares the operand with zero and sets the condition codes according to the results of the test. The size of the operation is specified as byte, word, or long.

Condition Codes:

| | | | | |
|---|---|---|---|---|
| X | N | Z | V | C |
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the operand is negative; cleared otherwise.

Z — Set if the operand is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------|---|-------------------|---|----------|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | SIZE | | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | REGISTER | | | |

TST**Test an Operand
(M68000 Family)****TST****Instruction Fields:**

Size field—Specifies the size of the operation.

- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

Effective Address field—Specifies the addressing mode for the destination operand as listed in the following tables:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|-------------------------|------|----------------|---------------------------|------|----------|
| Dn | 000 | reg. number:Dn | (xxx).W | 111 | 000 |
| An* | 001 | reg. number:An | (xxx).L | 111 | 001 |
| (An) | 010 | reg. number:An | #<data>* | 111 | 100 |
| (An) + | 011 | reg. number:An | | | |
| – (An) | 100 | reg. number:An | | | |
| (d ₁₆ ,An) | 101 | reg. number:An | (d ₁₆ ,PC)** | 111 | 010 |
| (d ₈ ,An,Xn) | 110 | reg. number:An | (d ₈ ,PC,Xn)** | 111 | 011 |

MC68020, MC68030, and MC68040 only

| | | | | | |
|-----------------|-----|----------------|-----------------|-----|-----|
| (bd,An,Xn)*** | 110 | reg. number:An | (bd,PC,Xn)*** | 111 | 011 |
| ([bd,An,Xn],od) | 110 | reg. number:An | ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,An],Xn,od) | 110 | reg. number:An | ([bd,PC],Xn,od) | 111 | 011 |

*MC68020, MC68030, MC68040, and CPU32. Address register direct allowed only for word and long.

**PC relative addressing modes do not apply to MC68000, MC680008, or MC68010.

***Can be used with CPU32.

UNLK

Unlink (M68000 Family)

UNLK

Operation: $A_n \rightarrow SP; (SP) \rightarrow A_n; SP + 4 \rightarrow SP$

Assembler

Syntax: UNLK A_n

Attributes: Unsized

Description: Loads the stack pointer from the specified address register, then loads the address register with the long word pulled from the top of the stack.

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|----------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | REGISTER | | |

Instruction Field:

Register field—Specifies the address register for the instruction.

UNPK

Unpack BCD (MC68020, MC68030, MC68040)

UNPK

Operation: Source (Packed BCD) + Adjustment → Destination (Unpacked BCD)

Assembler Syntax: UNPACK – (Ax), – (Ay),# < adjustment >

Syntax: UNPK Dx,Dy,# < adjustment >

Attributes: Unsized

Description: Places the two binary-coded decimal digits in the source operand byte into the lower four bits of two bytes and places zero bits in the upper four bits of both bytes. Adds the adjustment value to this unpacked value. Condition codes are not altered.

When both operands are data registers, the instruction unpacks the source register contents, adds the extension word, and places the result in the destination register. The high word of the destination register is unaffected.

Source:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| u | u | u | u | u | u | u | u | a | b | c | d | e | f | g | h |
| Dx | | | | | | | | | | | | | | | |

Intermediate Expansion:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | a | b | c | d | 0 | 0 | 0 | 0 | e | f | g | h |

Add Adjustment Word:

| | | | | | | | | | | | | | | | |
|------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 16-BIT EXTENSION | | | | | | | | | | | | | | | |

Destination:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|---|---|---|---|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| v | v | v | v | a' | b' | c' | d' | w | w | w | w | e' | f' | g' | h' |
| Dy | | | | | | | | | | | | | | | |

UNPK

Unpack BCD (MC68020, MC68030, MC68040)

UNPK

When the specified addressing mode is predecrement, the instruction extracts two binary-coded decimal digits from a byte at the source address. After unpacking the digits and adding the adjustment word, the instruction writes the two bytes to the destination address. Source:

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| a | b | c | d | e | f | g | h |
| Ax | | | | | | | |

Intermediate Expansion:

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | a | b | c | d | 0 | 0 | 0 | 0 | e | f | g | h |

Add Adjustment Word:

| | | | | | | | | | | | | | | | |
|------------------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|---|
| 15 | | | | | | | | | | | | | | | 0 |
| 16-BIT EXTENSION | | | | | | | | | | | | | | | |

Destination:

| | | | | | | | |
|----|---|---|---|----|----|----|----|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| v | v | v | v | a' | b' | c' | d' |
| w | w | w | w | e' | f' | g' | h' |
| Ay | | | | | | | |

Condition Codes:

Not affected.

Instruction Format:

| | | | | | | | | | | | | | | | |
|------------------------------|----|----|----|----------------|----|---|---|---|---|---|---|-----|----------------|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 0 | REGISTER Dy/Ay | | | 1 | 1 | 0 | 0 | 0 | R/M | REGISTER Dx/Ax | | |
| 16-BIT EXTENSION: ADJUSTMENT | | | | | | | | | | | | | | | |

UNPK

Unpack BCD
(MC68020, MC68030, MC68040)

UNPK

Instruction Fields:

Register Dy/Ay field—Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register in the predecrement addressing mode.

R/M field—Specifies the operand addressing mode.

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Dx/Ax field—Specifies the data register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register in the predecrement addressing mode.

Adjustment field—Immediate data word that is added to the source operand.

Appropriate constants can be used as the adjustment to translate from binary-coded decimal to the desired code. The constant used for ASCII is \$3030; for EBCDIC, \$F0F0.

