

MOTOROLA

M68000 FAMILY

Programmer's Reference Manual

(Includes CPU32 Instructions)

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)

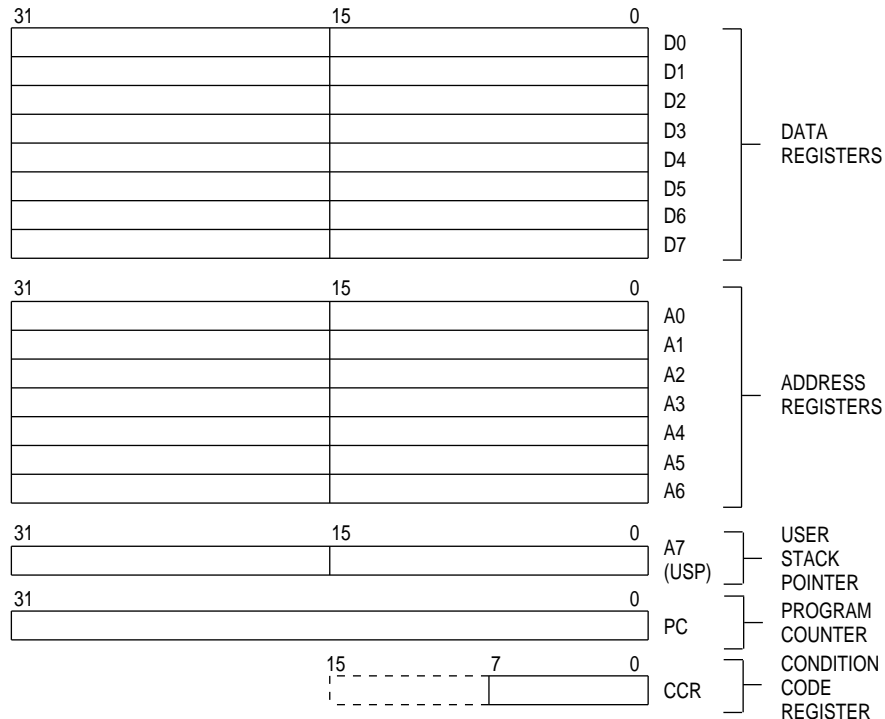


Figure 1-1. M68000 Family User Programming Model

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

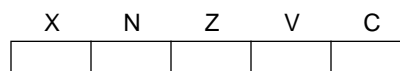
The PC contains the address of the instruction currently executing. During instruction execution and exception processing, the processor automatically increments the contents or places a new value in the PC. For some addressing modes, the PC can be used as a pointer for PC relative addressing.

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)

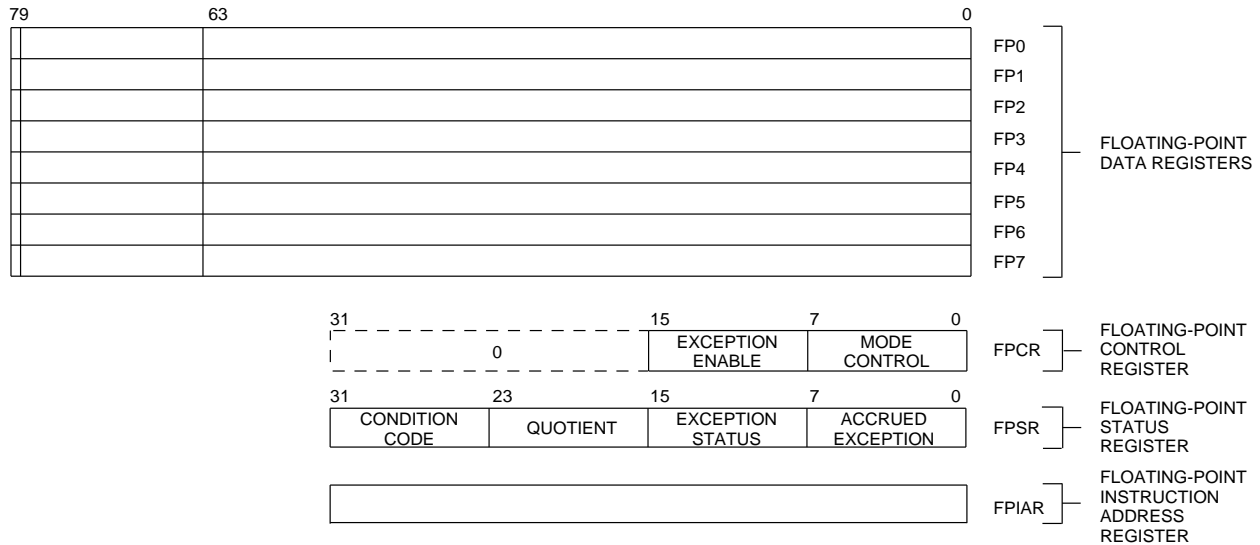


Figure 1-2. M68000 Family Floating-Point Unit User Programming Model

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).

1.7 ORGANIZATION OF DATA IN REGISTERS

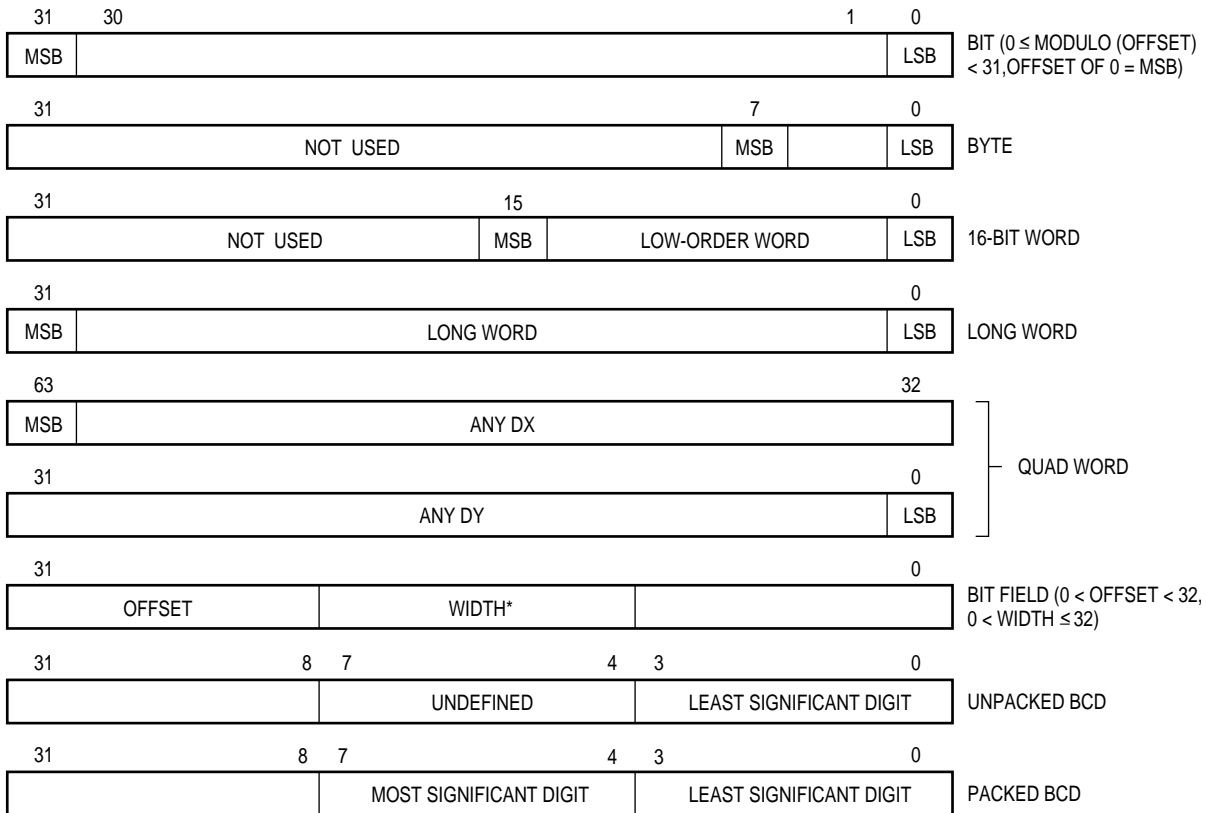
The following paragraphs describe data organization within the data, address, and control registers.

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.

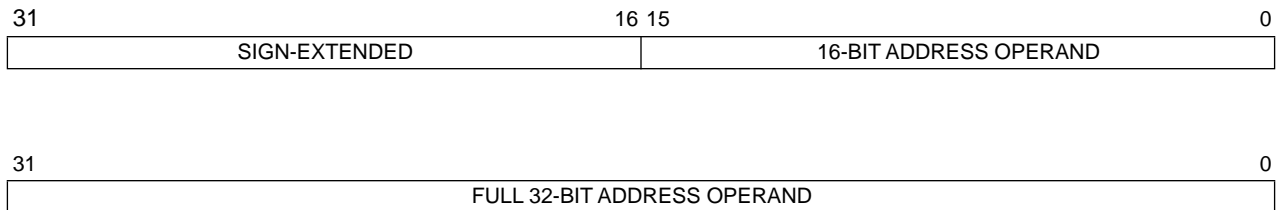


Figure 1-19. Organization of Integer Data Formats in Address Registers

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.

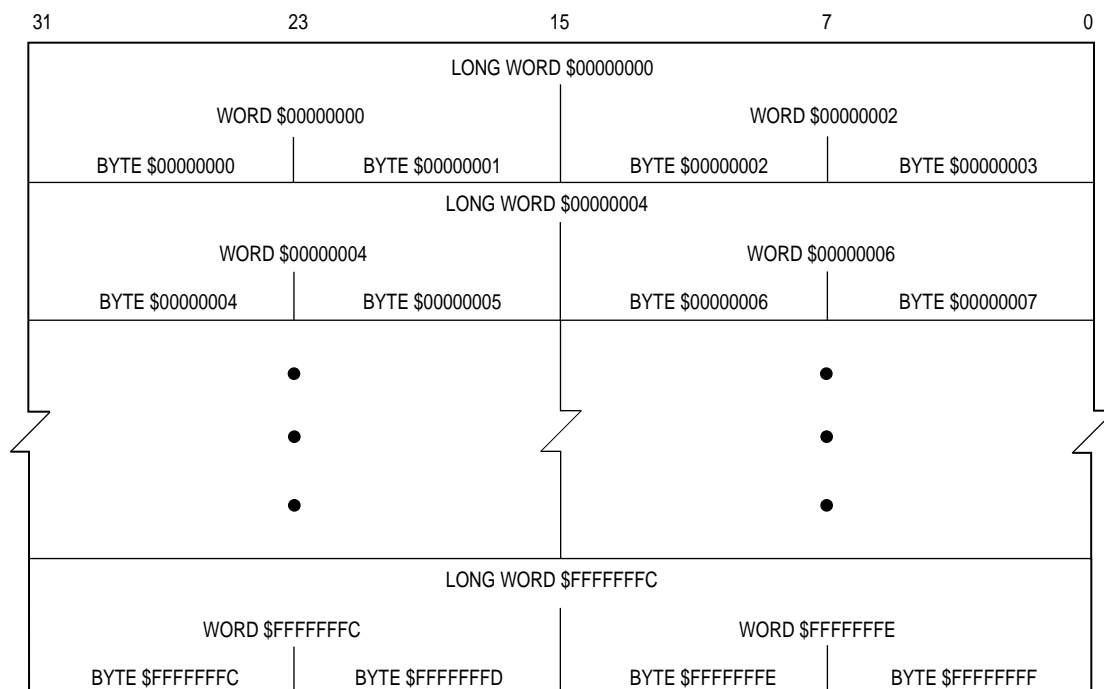


Figure 1-20. Memory Operand Addressing

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.

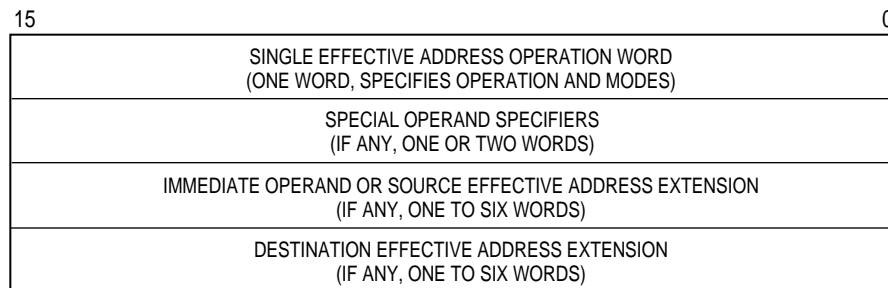


Figure 2-1. Instruction Word General Format

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.

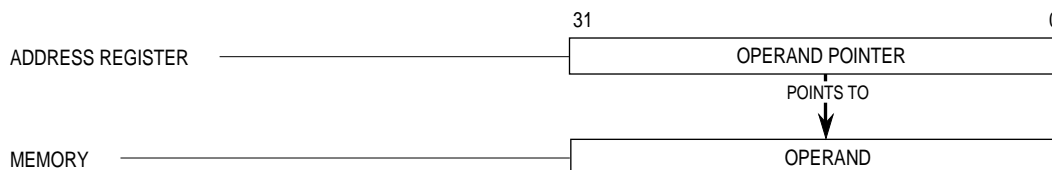
GENERATION:	EA = An
ASSEMBLER SYNTAX:	An
EA MODE FIELD:	001
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	0



2.2.3 Address Register Indirect Mode

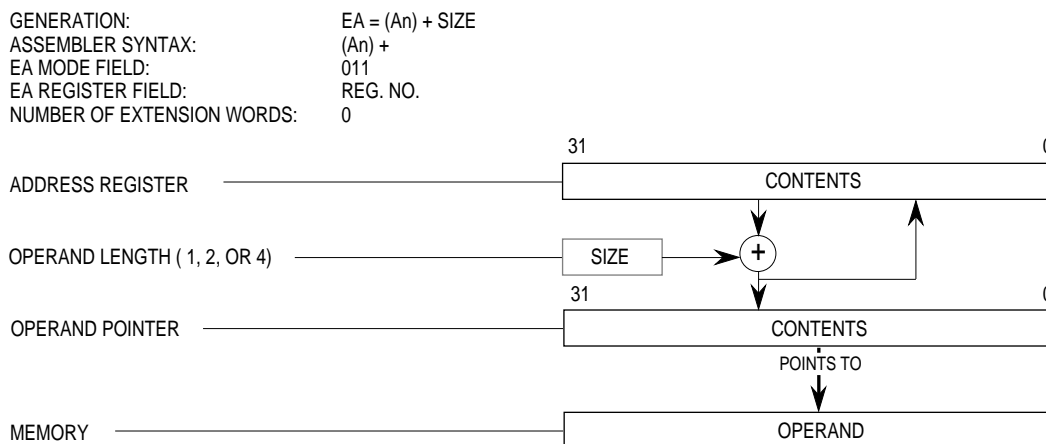
In the address register indirect mode, the operand is in memory. The effective address field specifies the address register containing the address of the operand in memory.

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



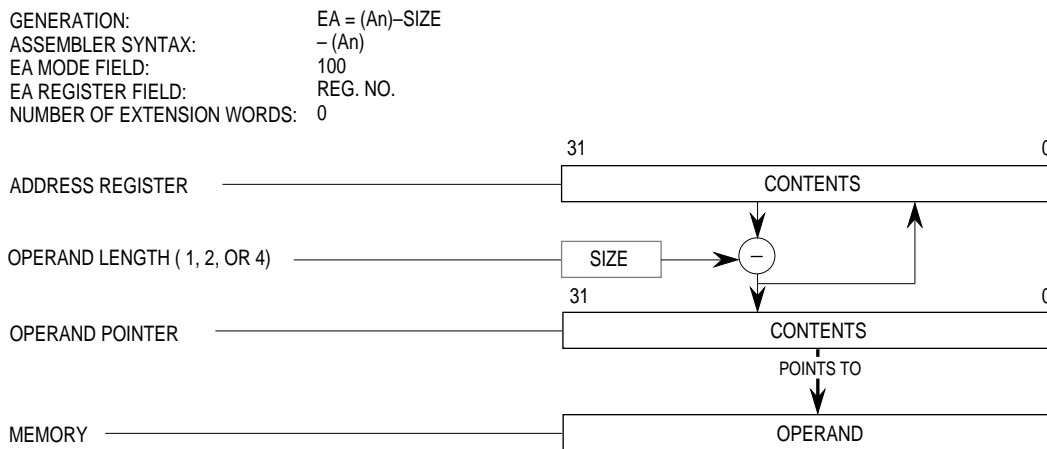
2.2.4 Address Register Indirect with Postincrement Mode

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.



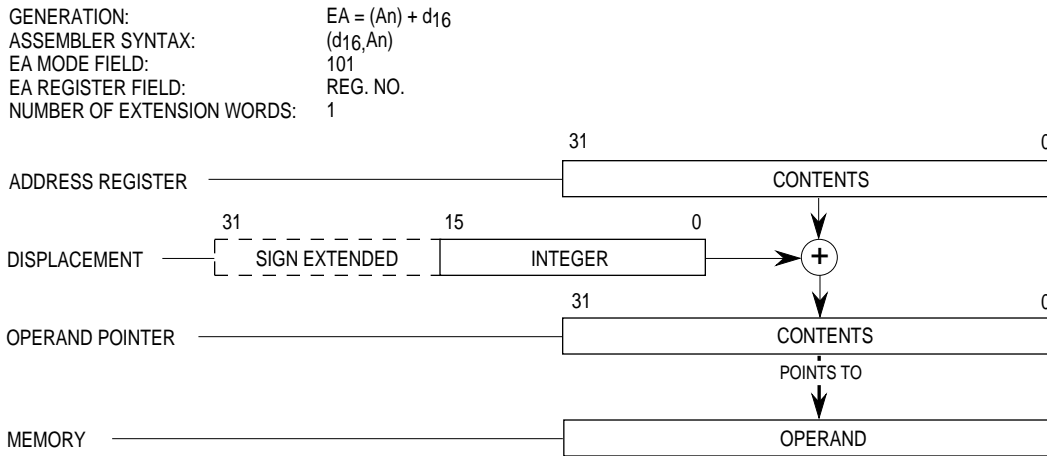
2.2.5 Address Register Indirect with Predecrement Mode

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.



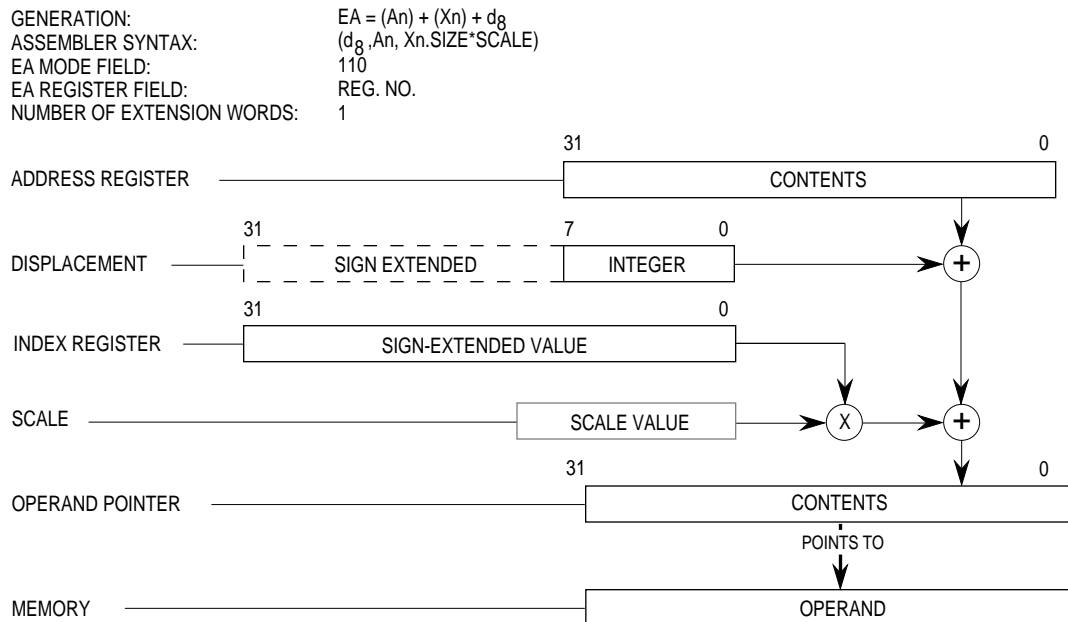
2.2.6 Address Register Indirect with Displacement Mode

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.

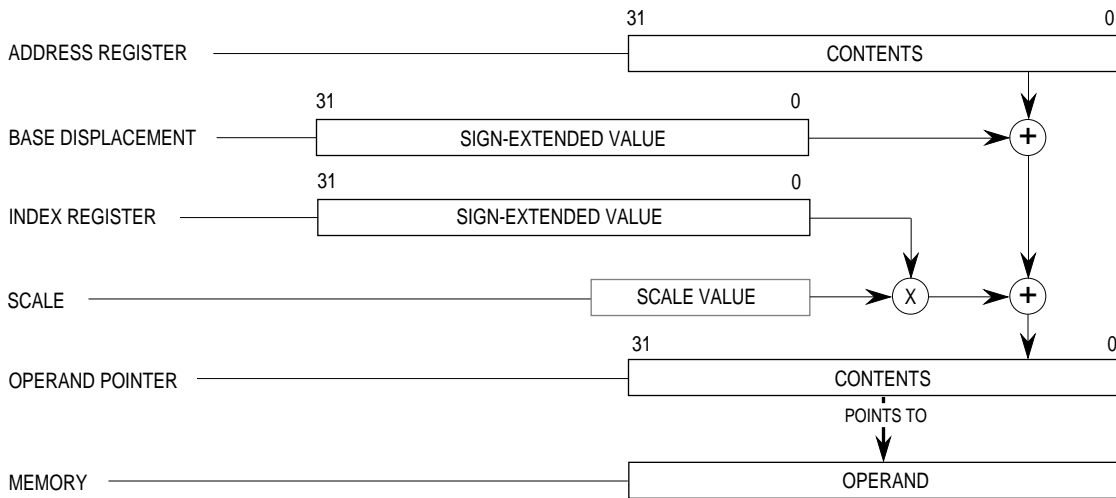


2.2.8 Address Register Indirect with Index (Base Displacement) Mode

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.

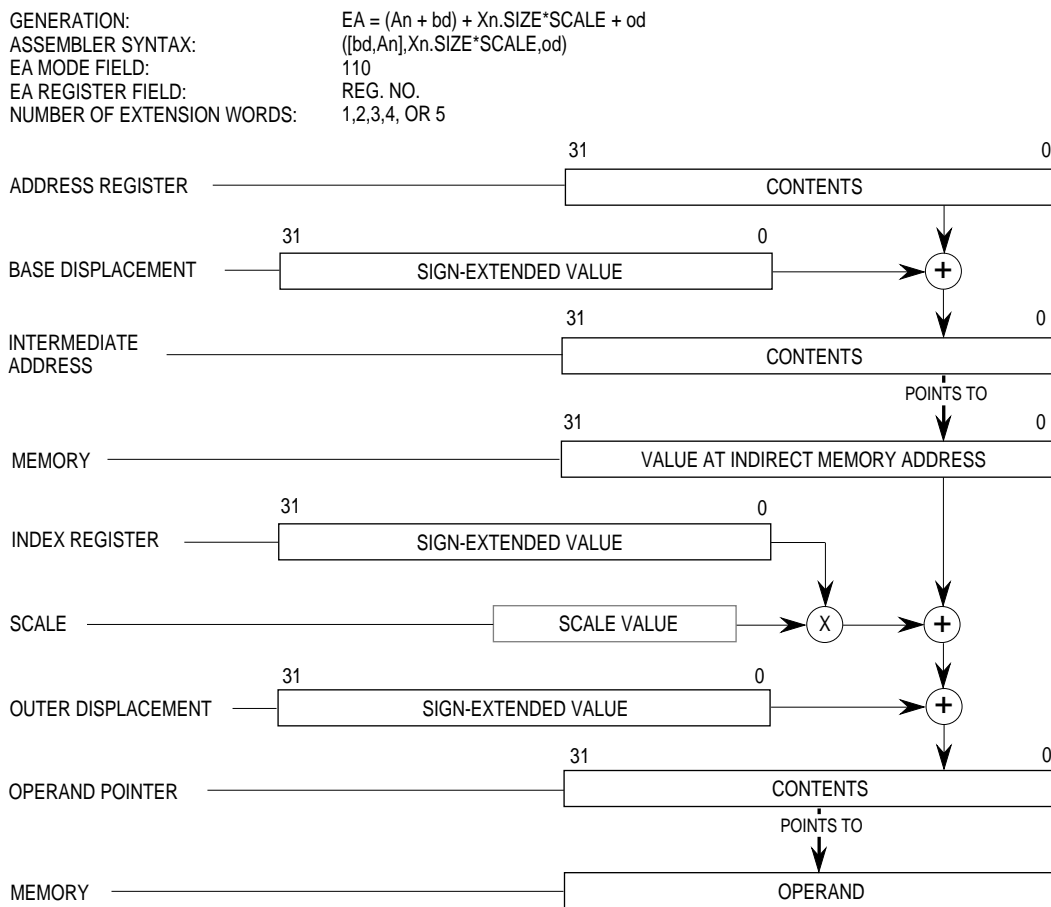
GENERATION:	$EA = (An) + (Xn) + bd$
ASSEMBLER SYNTAX:	$(bd, An, Xn, SIZE * SCALE)$
EA MODE FIELD:	110
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	1, 2, OR 3



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.

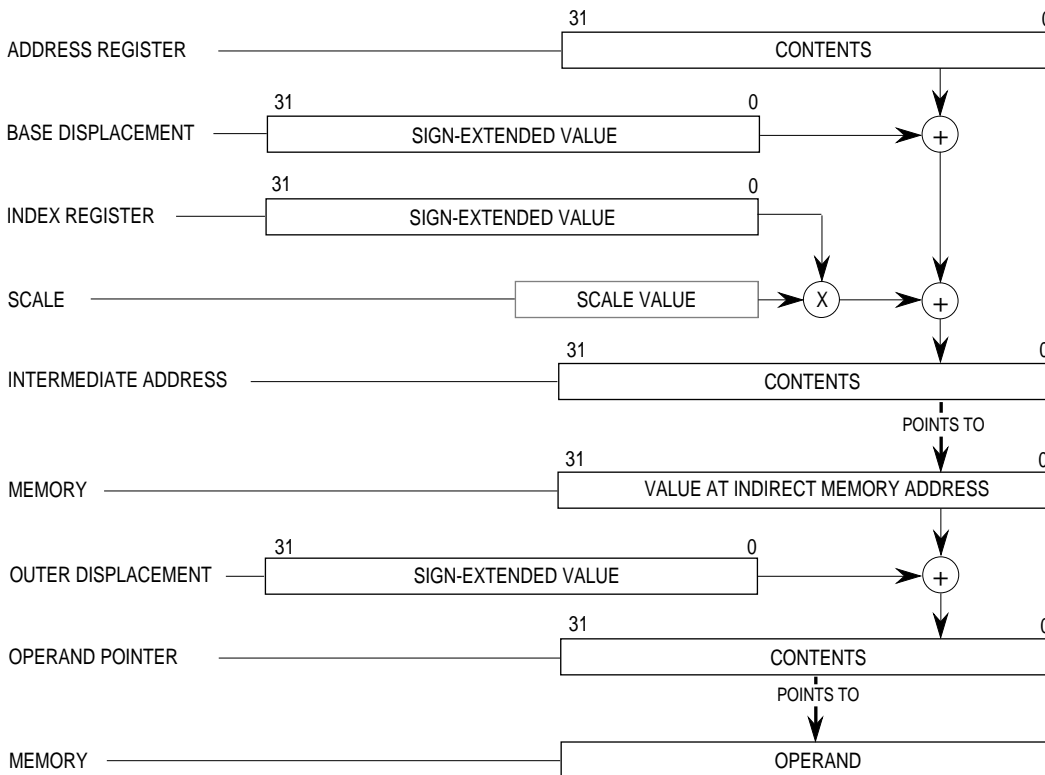


2.2.10 Memory Indirect Preindexed Mode

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.

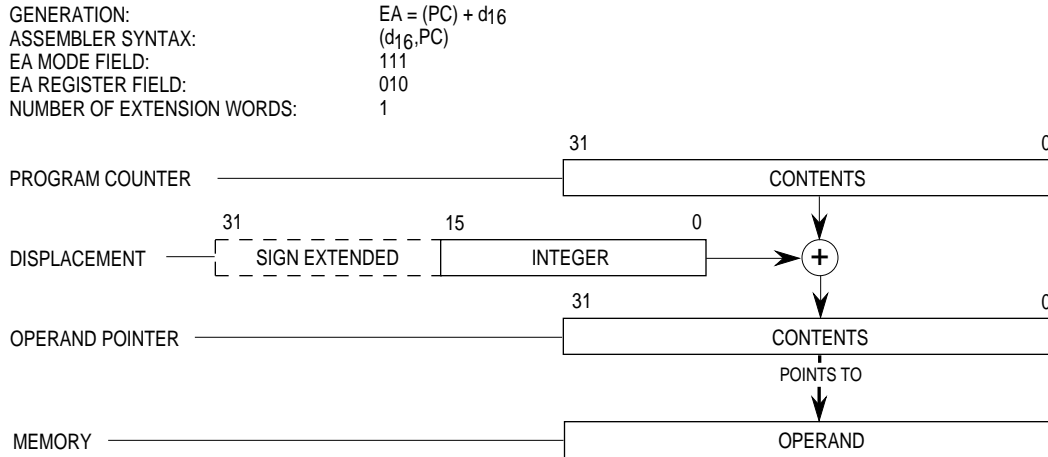
In the syntax for this mode, brackets enclose the values used to calculate the intermediate memory address. All four user-specified values are optional. Both the base and outer displacements may be null, word, or long word. When omitting a displacement or suppressing an element, its value is zero in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



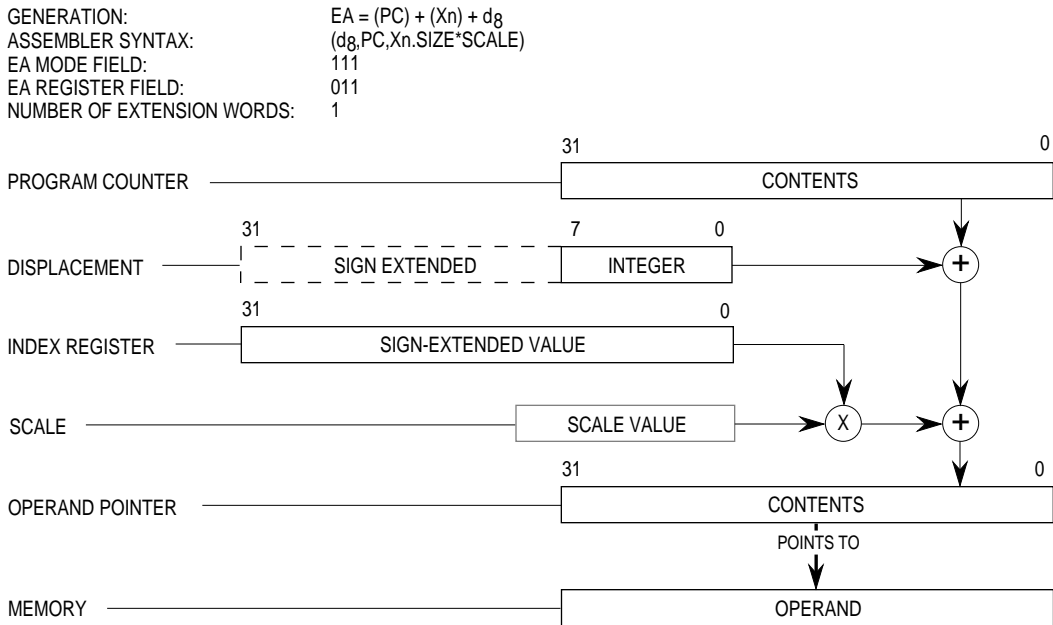
2.2.11 Program Counter Indirect with Displacement Mode

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.

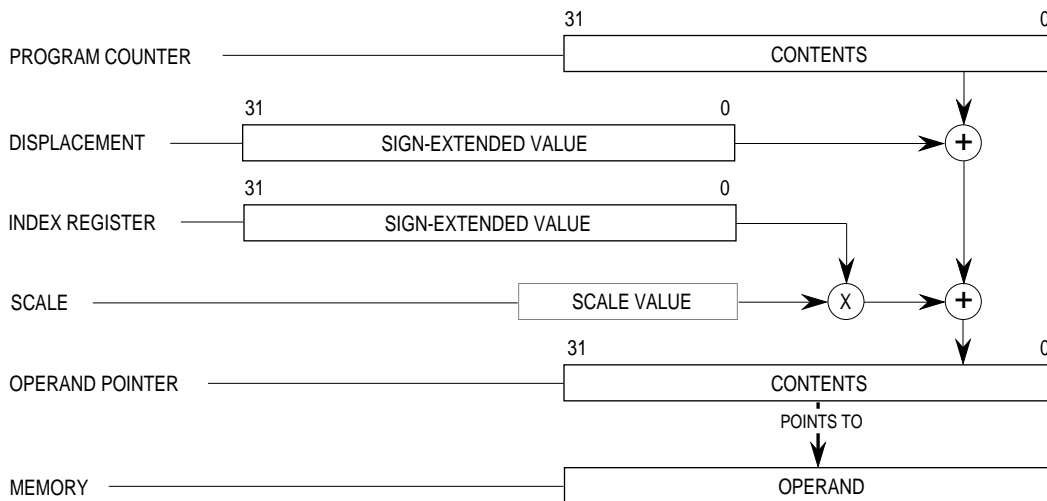


2.2.13 Program Counter Indirect with Index (Base Displacement) Mode

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.

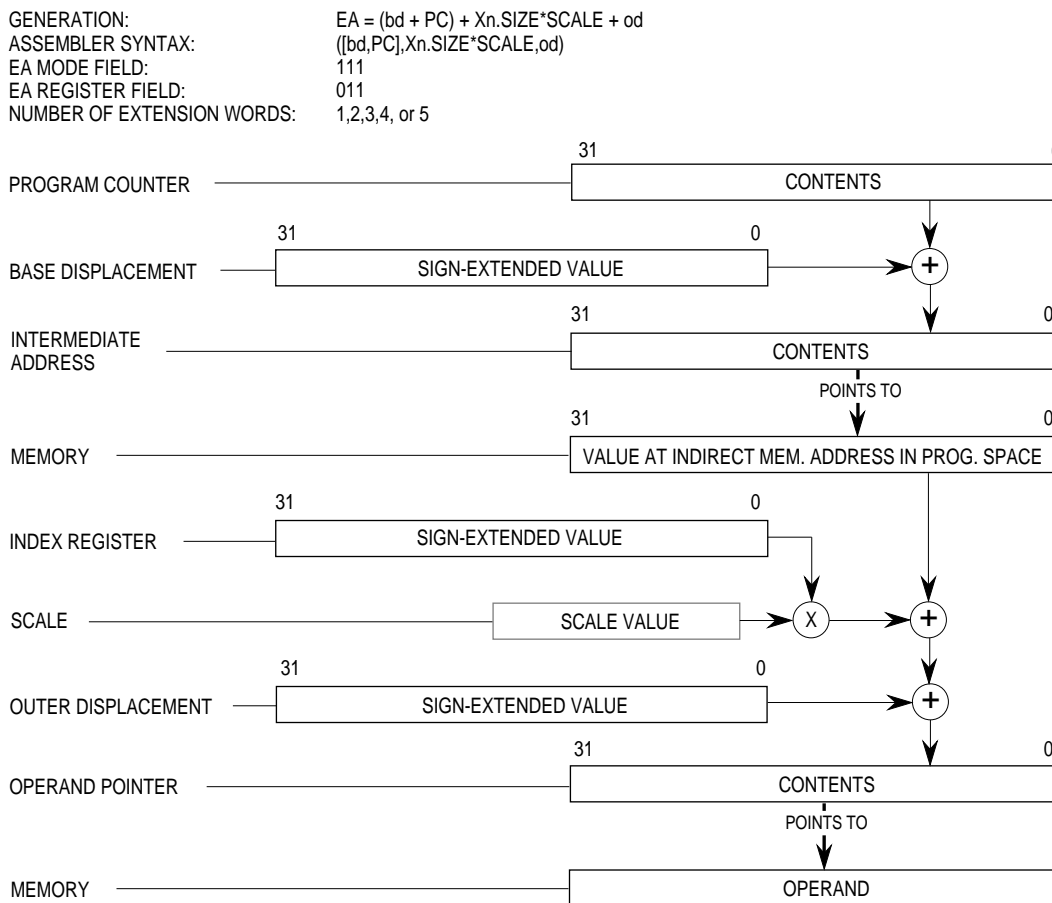
GENERATION: $EA = (PC) + (Xn) + bd$
 ASSEMBLER SYNTAX: (bd, PC, Xn, SIZE*SCALE)
 EA MODE FIELD: 111
 EA REGISTER FIELD: 011
 NUMBER OF EXTENSION WORDS: 1, 2, OR 3



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.

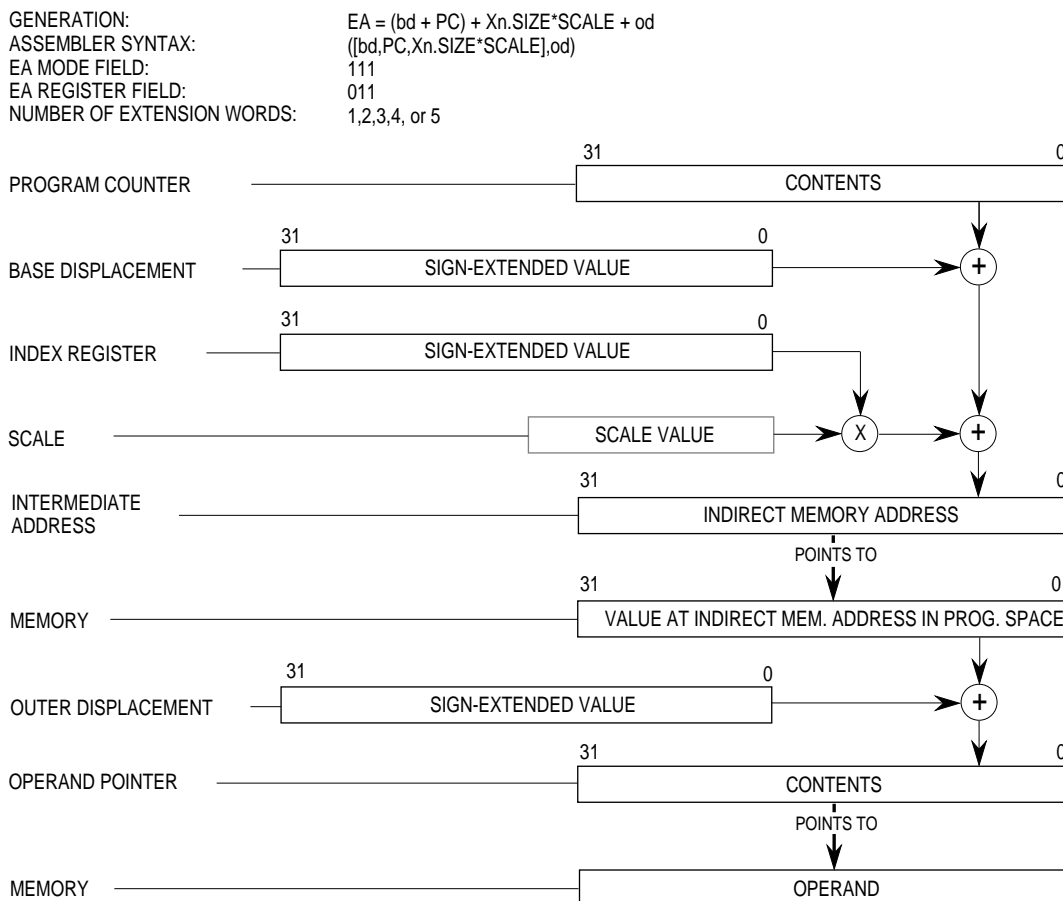
In the syntax for this mode, brackets enclose the values used to calculate the intermediate memory address. All four user-specified values are optional. The user must supply the assembler notation ZPC (a zero value PC) to show the PC is not used. This allows the user to access the program space without 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

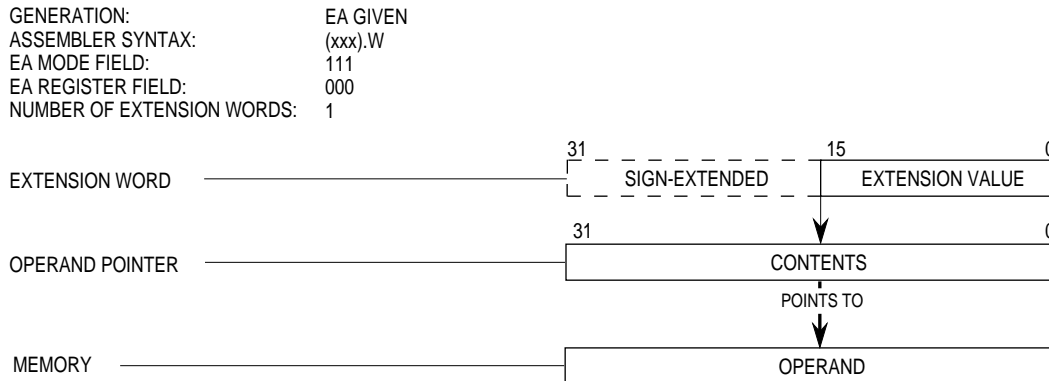
This mode is similar to the mode described in **2.2.10 Memory Indirect Preindexed Mode**, but the PC is the base register. Both the operand and operand address are in memory. The processor calculates an intermediate indirect memory address by adding the PC contents, a base displacement, and the scaled 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.



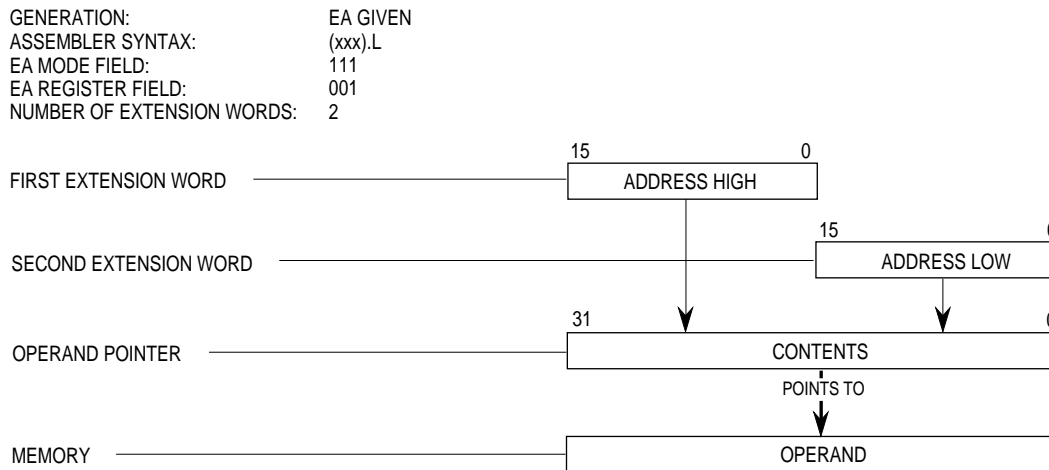
2.2.16 Absolute Short Addressing Mode

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	—	—

2.6 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.

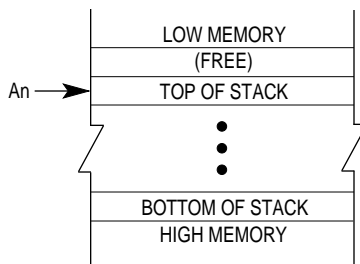
2.6.1 System Stack

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.

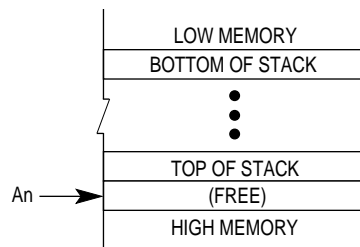
The user can implement stacks with the address register indirect with postincrement and predecrement addressing modes. With an address register the user can implement a stack that fills either from high memory to low memory or from low memory to high memory. Important considerations are:

- 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.



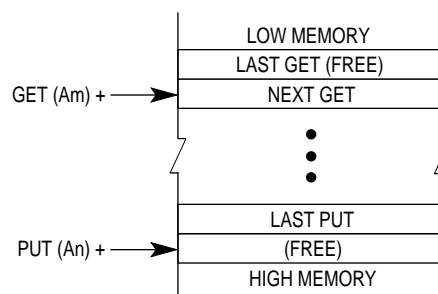
To implement stack growth from low memory to high memory, use $(An) +$ to push data on the stack and $-(An)$ to pull data from the stack. After either a push or pull operation, the address register points to the next available space on the stack. .



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.

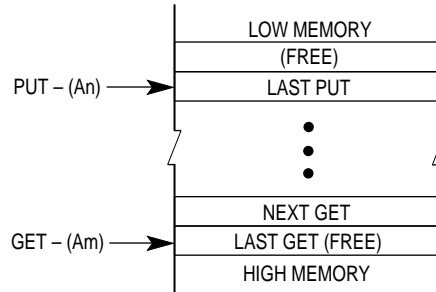
After a put operation, the put address register points to the next available space in the queue; the unchanged get address register points to the next item to be removed from the queue. After a get operation, the get address register points to the next item to be removed from the queue; the unchanged 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 \leftarrow \rightarrow 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 \rightarrow Destination
FSMOVE, FDMOVE	FPm,FPn <ea>,FPn	X B, W, L, S, D, X	Source \rightarrow 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 \rightarrow Destination Source \rightarrow Listed Registers
LEA	<ea>,An	32	<ea> \rightarrow An
LINK	An,#<d>	16, 32	SP - 4 \rightarrow SP; An \rightarrow (SP); SP \rightarrow An, SP + D \rightarrow SP
MOVE MOVE16 MOVEA	<ea>,<ea> <ea>,<ea> <ea>,An	8, 16, 32 16 bytes 16, 32 \rightarrow 32	Source \rightarrow Destination Aligned 16-Byte Block \rightarrow Destination
MOVEM	list,<ea> <ea>,list	16, 32 16, 32 \rightarrow 32	Listed Registers \rightarrow Destination Source \rightarrow Listed Registers
MOVEP	Dn, (d ₁₆ ,An) (d ₁₆ ,An),Dn	16, 32	Dn 31-24 \rightarrow (An + d _n); Dn 23-16 \rightarrow (An + d _n + 2); Dn 15-8 \rightarrow (An + d _n + 4); Dn 7-0 \rightarrow (An + d _n + 6) (An + d _n) \rightarrow Dn 31-24; (An + d _n + 2) \rightarrow Dn 23-16; (An + d _n + 4) \rightarrow Dn 15-8; (An + d _n + 6) \rightarrow Dn 7-0
MOVEQ	#<data>,Dn	8 \rightarrow 32	Immediate Data \rightarrow Destination
PEA	<ea>	32	SP - 4 \rightarrow SP; <ea> \rightarrow (SP)
UNLK	An	32	An \rightarrow SP; (SP) \rightarrow An; SP + 4 \rightarrow 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

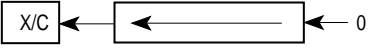
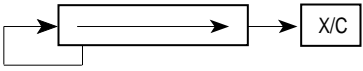
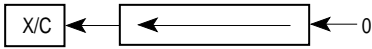
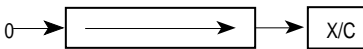
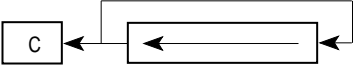
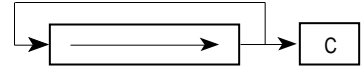
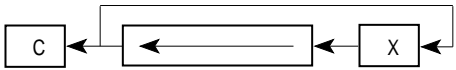
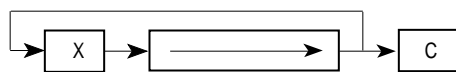
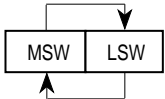
3.1.4 Shift and Rotate Instructions

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.

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, DUUVU	—	*	*	?	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.

3.7 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.

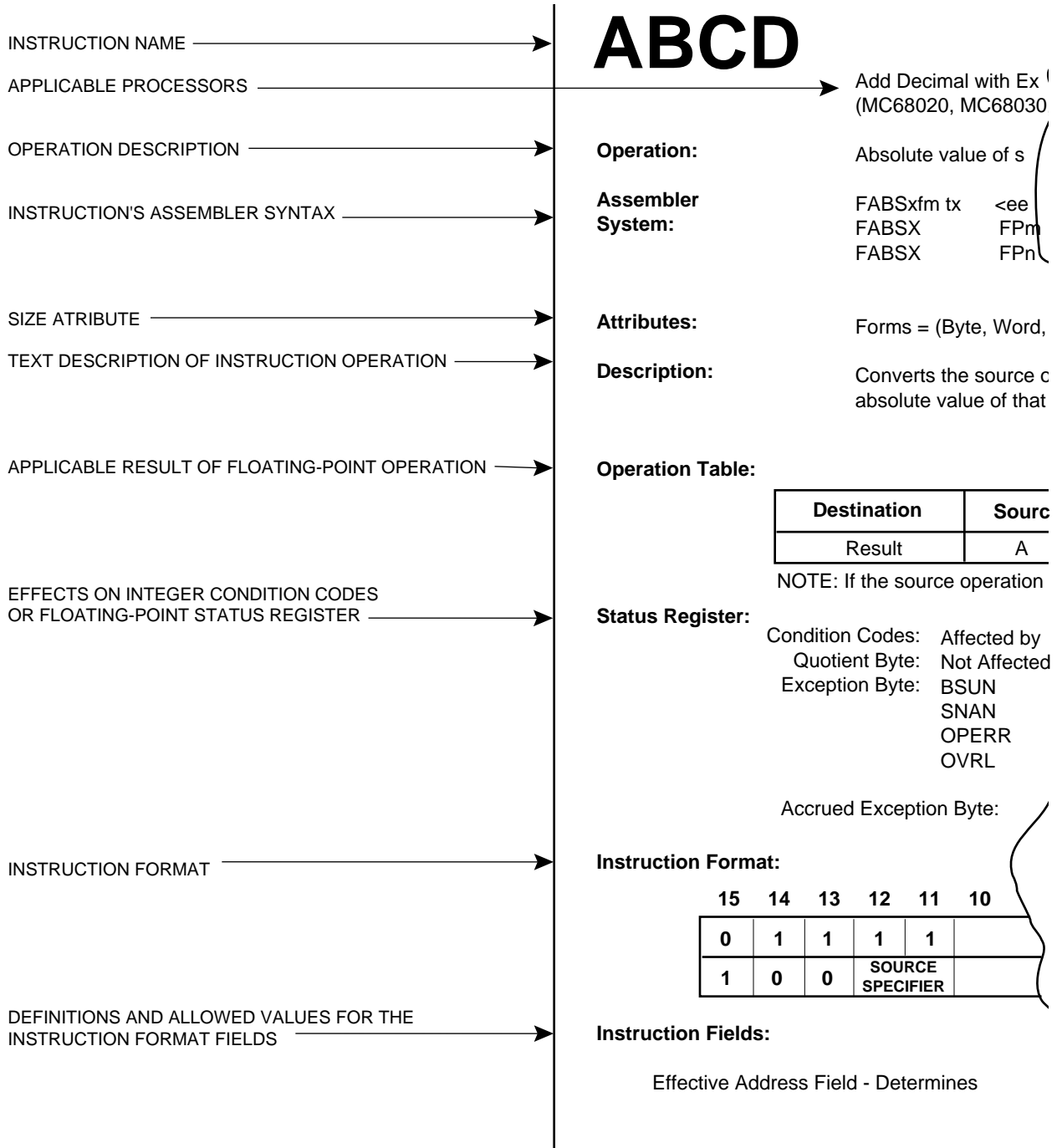


Figure 3-3. Instruction Description Format

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
0	0	0	0	0	0	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

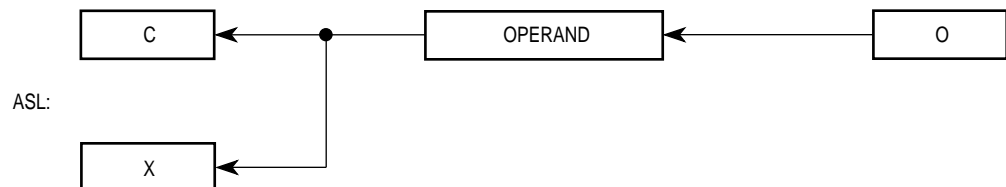
Attributes: Size = (Byte, Word, Long)

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.

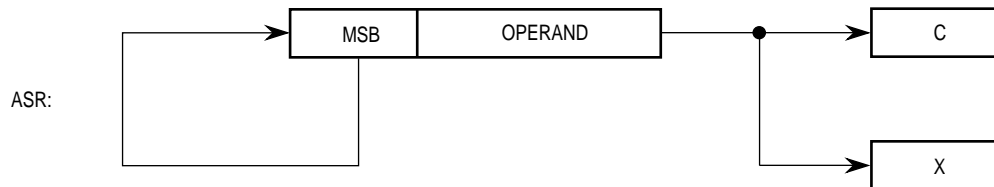


ASL, ASR

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).

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.

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.

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.

LSL, LSR

Logical Shift (M68000 Family)

LSL, LSR

Operation: Destination Shifted By Count → Destination

Assembler LSd Dx,Dy

Syntax: LSd # < data > ,Dy

LSd < ea >

where d is direction, L or R

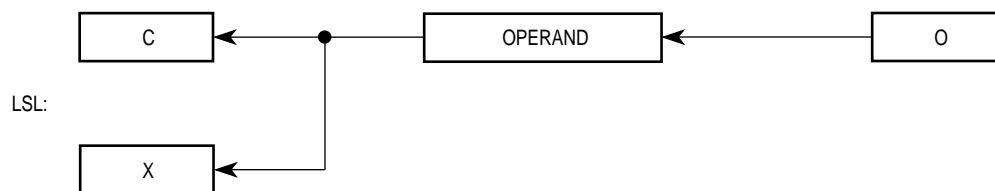
Attributes: Size = (Byte, Word, Long)

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:

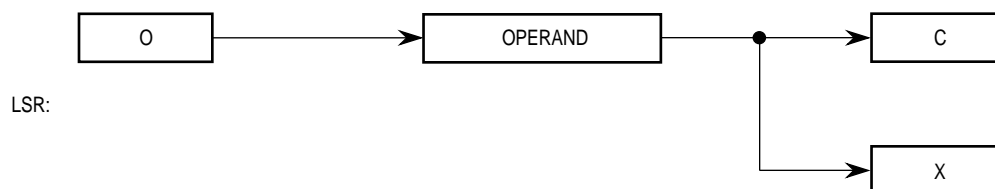
1. Immediate—The shift count (1 – 8) is specified in the instruction.
2. Register—The shift count is the value in the data register specified in the instruction modulo 64.

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.

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

A.1 MC68000, MC68008, MC68010 PROCESSORS

The following paragraphs provide information on the MC68000, MC68008, and MC68010 instruction set and addressing modes.

A.1.1 M68000, MC68008, and MC68010 Instruction Set

Table A-3 lists the instructions used with the MC68000 and MC68008 processors, and Table A-4 lists the instructions used with MC68010.

Table A-3. MC68000 and MC68008 Instruction Set

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ANDI to CCR	AND Immediate to Condition Code Register
ANDI to SR	AND Immediate to Status Register
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CHK	Check Register Against Bound
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
DBcc	Test Condition, Decrement, and Branch
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR	Logical Exclusive-OR
EORI	Logical Exclusive-OR Immediate
EORI to CCR	Exclusive-OR Immediate to Condition Code Register
EORI to SR	Exclusive-OR Immediate to Status Register
EXG	Exchange Registers
EXT	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine

**Table A-3. MC68000 and MC68008 Instruction Set
(Continued)**

Mnemonic	Description
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE to CCR	Move to Condition Code Register
MOVE from SR	Move from Status Register
MOVE to SR	Move to Status Register
MOVE USP	Move User Stack Pointer
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement
OR	Logical Inclusive-OR
ORI	Logical Inclusive-OR Immediate
ORI to CCR	Inclusive-OR Immediate to Condition Code Register
ORI to SR	Inclusive-OR Immediate to Status Register
PEA	Push Effective Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract Address
SUBI	Subtract Immediate
SUBQ	Subtract Quick
SUBX	Subtract with Extend
SWAP	Swap Register Words
TAS	Test Operand and Set
TRAP	Trap
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink

Table C-2. ASCII Code

Least Significant Digit	Most Significant Digit							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Table C-2. ASCII Code

Least Significant Digit	Most Significant Digit							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL