M68000PM/AD
REV. 1

PROGRAMMER'S REFERENCE MANUAL

(Includes **CPU32** Instructions)

MOTOROLA

M68000 FAMILY

Programmer's Reference Manual

(Includes CPU32 Instructions)

Introduction

Section 2
Addressing Capabilities

Section 3
Instruction Set Summary

Appendix A
Processor Instruction Summary

Appendix B
Exception Processing Reference

S-Record Output Format

SECTION 1
INTRODUCTION

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

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

NOTE

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

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

- 32-Bit Program Counter (PC)
- 8-Bit Condition Code Register (CCR)

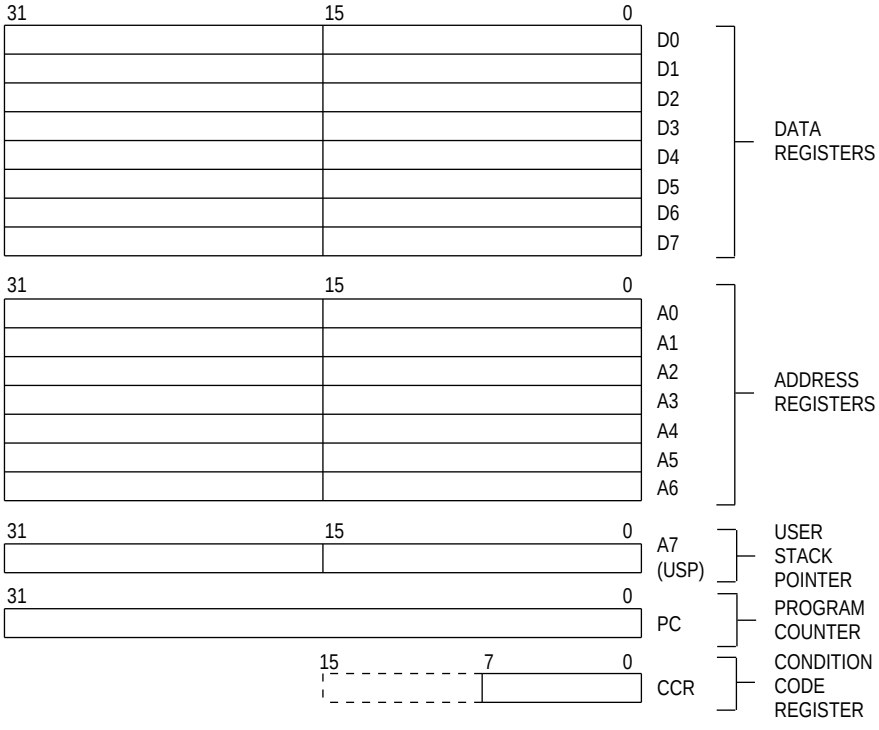


Figure 1-1. M68000 Family User Programming Model

1.1.1 Data Registers (D7 – D0)

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

1.1.2 Address Registers (A7 – A0)

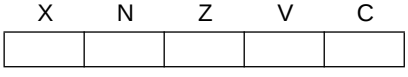
These registers can be used as software stack pointers, index registers, or base
registers. The base address registers can be used for word and long-word ope
Register A7 is used as a hardware stack pointer during stacking for subroutine c
exception handling. In the user programming model, A7 refers to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

1.1.4 Condition Code Register

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

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

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



X—Extend
Set to the value of the C-bit for arithmetic operations; otherwise not affected c 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 represen operand size; otherwise clear.

The following paragraphs describe the registers for the floating- point unit user progr
model. Figure 1-2 illustrates the M68000 family user programming model's floati
portion for the MC68040 and the MC68881/MC68882 floating-point coproces
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)

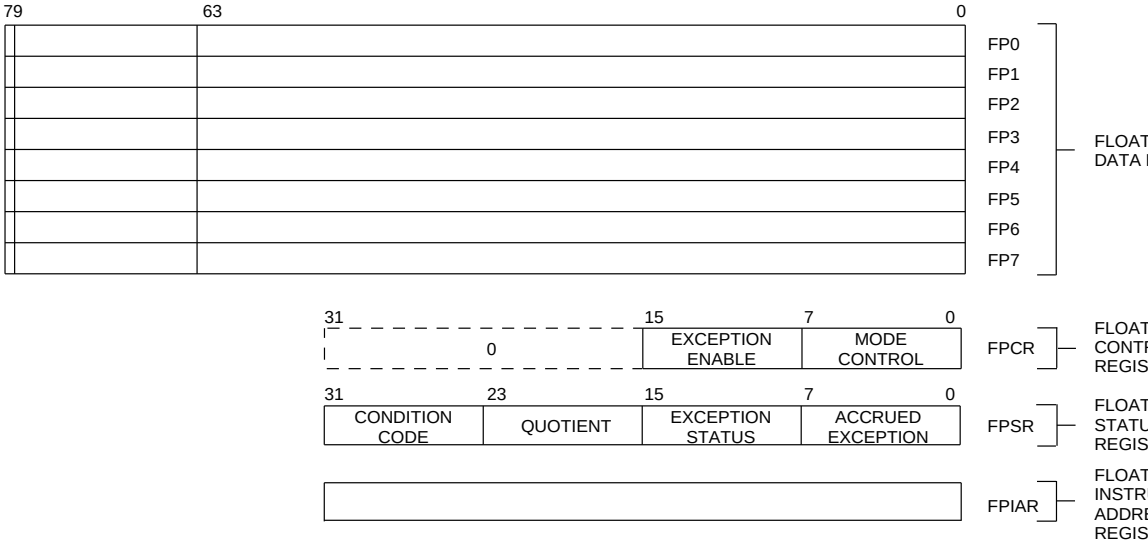


Figure 1-2. M68000 Family Floating-Point Unit User Programming Mod

1.2.1 Floating-Point Data Registers (FP7 – FP0)

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

cleared, this register provides the IEEE 754 Standard for Binary Floating-Point A
defaults.

1.2.2.1 EXCEPTION ENABLE BYTE. Each bit of the ENABLE byte (see Fig
corresponds to a floating-point exception class. The user can separately enable
each class of floating-point exceptions.

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

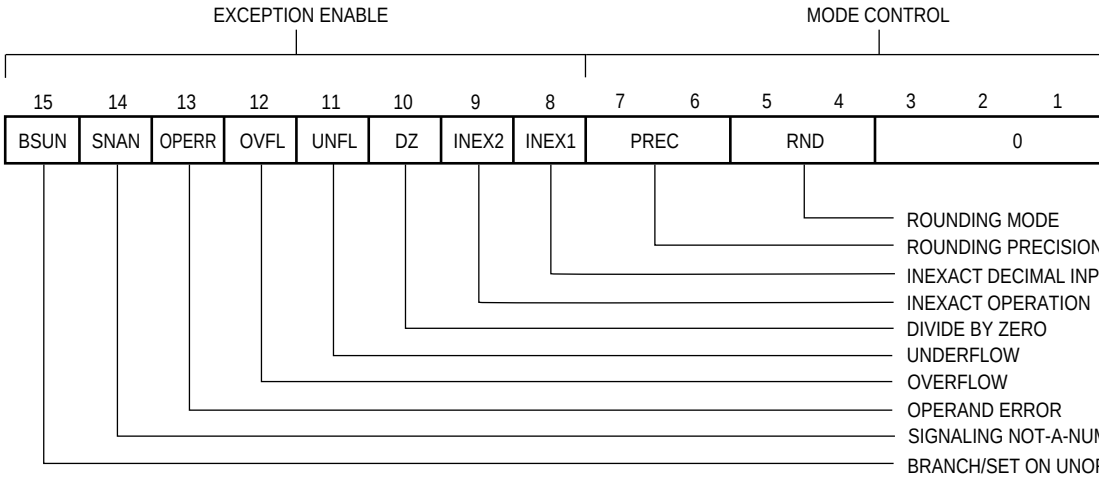


Figure 1-3. Floating-Point Control Register

1.2.3 Floating-Point Status Register (FPSR)

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

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

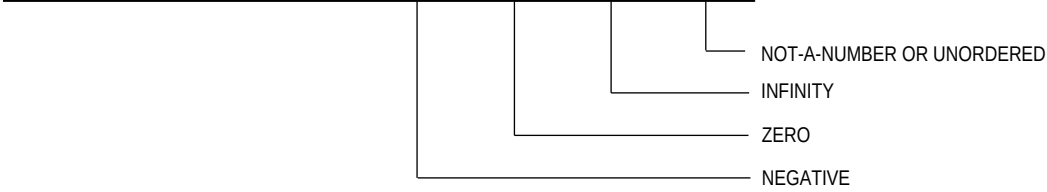


Figure 1-4. FPSR Condition Code Byte

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

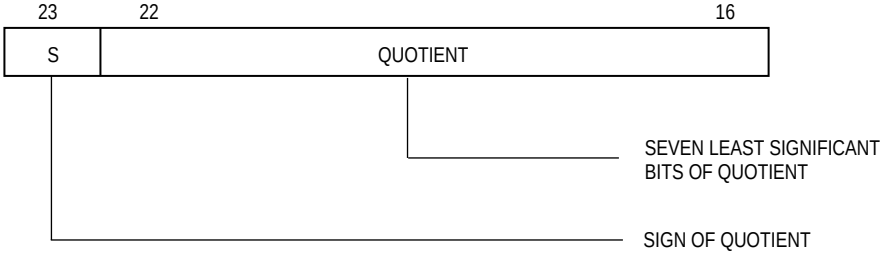


Figure 1-5. FPSR Quotient Code Byte

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

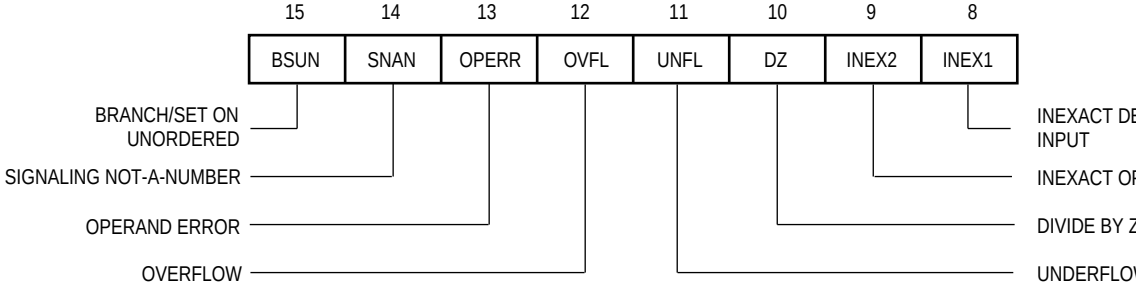


Figure 1-6. FPSR Exception Status Byte

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

.

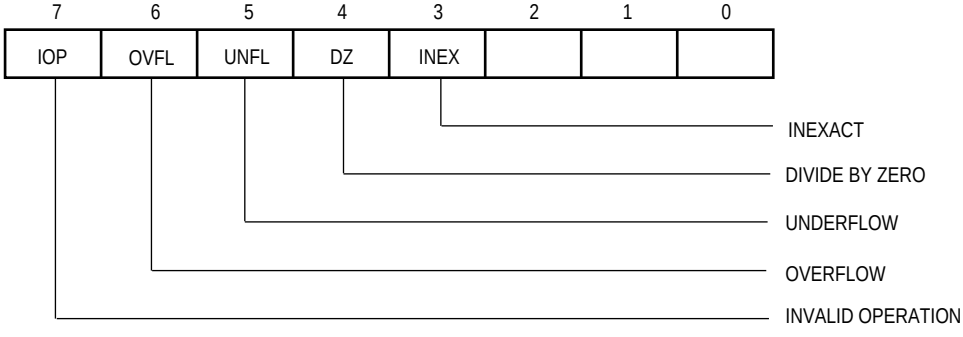


Figure 1-7. FPSR Accrued Exception Byte

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

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

instruction.

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

1.3 SUPERVISOR PROGRAMMING MODEL

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

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

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

AC	x			
CAL	x			
CRP	x	x		
DRP	x			
PCSR	x			
PMMUSR, MMUSR	x	x	x	x
SCC	x			
SRP	x	x	x	x
TC	x	x	x	x
URP			x	x
VAL	x			

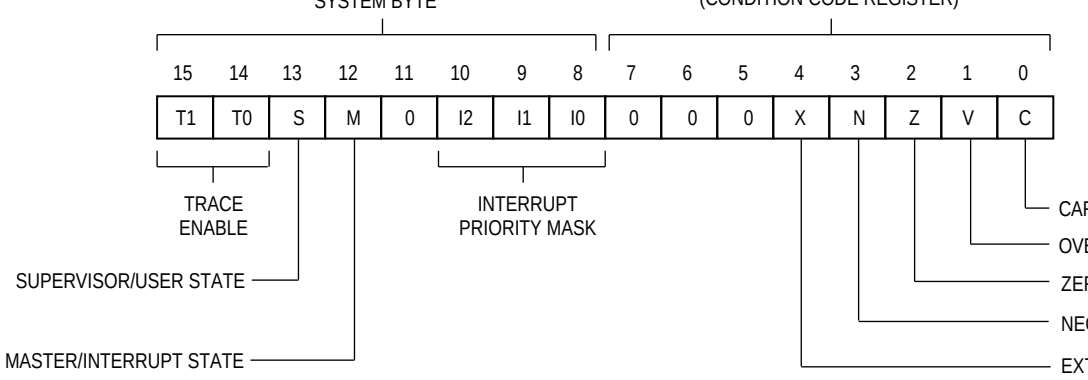
AC = Access Control Register
CAL = Current Access Level Register
CRP = CPU Root Pointer
DRP = DMA Root Pointer
PCSR = PMMU Control Register
PMMUSR = Paged Memory Management Unit Status Register
MMUSR = Memory Management Unit Status Register
SCC = Stack Change Control Register
SRP = Supervisor Root Pointer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

1.3.1 Address Register 7 (A7)

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

1.3.2 Status Register

Figure 1-8 illustrates the SR, which stores the processor status and contains the c codes that reflect the results of a previous operation. In the supervisor mode, softw access the full SR, including the interrupt priority mask and additional control bits. Th indicate the following states for the processor: one of two trace modes (T1, T0), su or user mode (S), and master or interrupt mode (M). For the MC68000, MC68 MC68008, MC68010, MC68HC000, MC68HC001, and CPU32, only one trac



T1	T0	TRACE MODE
0	0	NO TRACE
1	0	TRACE ON ANY INSTRUCTION
0	1	TRACE ON CHANGE OF FLOW
1	1	UNDEFINED

S	M	ACTIVE STACK
0	x	USP
1	0	ISP
1	1	MSP

Figure 1-8. Status Register

1.3.3 Vector Base Register (VBR)

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

1.3.4 Alternate Function Code Registers (SFC and DFC)

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

1.3.5 Acu Status Register (MC68EC030 only)

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

blocks of logical addresses that are transparently translated to corresponding addresses. These registers are independent of the on-chip MMU. For em controllers, such as the MC68EC030 and MC68EC040, the access control registe are similar in function to the TT registers but just named differently. The AC registe function are to define blocks of address space that control address space propert as cachability. The following paragraphs describe these registers.

NOTE

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

1.3.6.1 TRANSPARENT TRANSLATION/ACCESS CONTROL REGISTER FIELD THE M68030. Figure 1-9 illustrates the MC68030 transparent translation/MC6 access control register format.

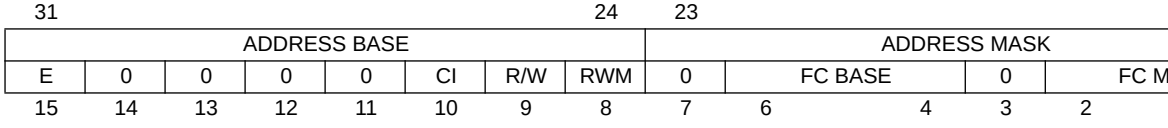


Figure 1-9. MC68030 Transparent Translation/MC68EC030 Access Control R Format

Address Base

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

Address Mask

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

1 = Caching inhibited

R/W—Read/Write
0 = Only write accesses permitted
1 = Only read accesses permitted

R/WM—Read/Write Mask
0 = R/W field used
1 = R/W field ignored

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

FC MASK—Function Code Mask
This 3-bit field contains a mask for the FC BASE field. Setting a bit in this fie
the corresponding bit of the FC BASE field to be ignored.

1.3.6.2 TRANSPARENT TRANSLATION/ACCESS CONTROL REGISTER FIEL

THE M68040. Figure 1-10 illustrates the MC68040 and MC68LC040 tra
translation/ MC68EC040 access control register format.

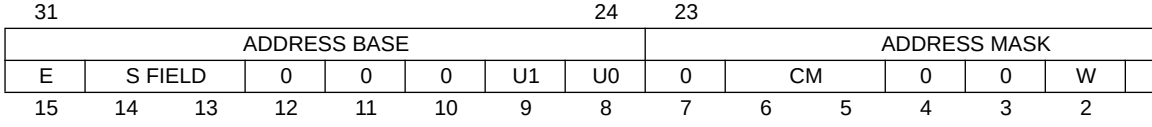


Figure 1-10. MC68040 and MC68LC040 Transparent Translation/MC68EC040 Control Register Format

Address Base
This 8-bit field is compared with address bits A31 – A24. Addresses that mat
comparison (and are otherwise eligible) are transparently translated/access co

E—Enable

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

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

S—Supervisor/User Mode

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

00 = Match only if FC2 is 0 (user mode access)
01 = Match only if FC2 is 1 (supervisor mode access)
1X = Ignore FC2 when matching

U1, U2—User Page Attributes

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

CM—Cache Mode

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

00 = Cachable, Writethrough
01 = Cachable, Copyback
10 = Noncachable, Serialized
11 = Noncachable

W—Write Protect

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

0 = Read and write accesses permitted
1 = Write accesses not permitted

1.4 INTEGER DATA FORMATS

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

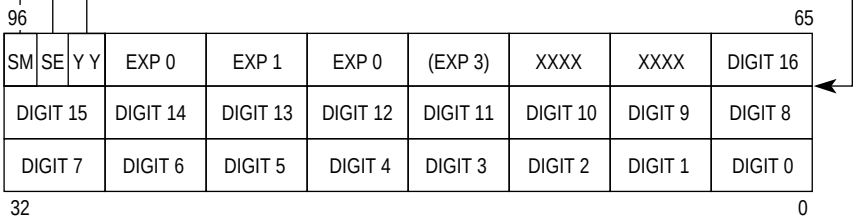
Binary-Coded Decimal (BCD)	8 Bits	Packed: 2 Digits/Byte; Unpacked: 1 Digit/Byte
Byte Integer	8 Bits	—
Word Integer	16 Bits	—
Long-Word Integer	32 Bits	—
Quad-Word Integer	64 Bits	Any Two Data Registers
16-Byte	128 Bits	Memory Only, Aligned to 16- Byte Boundary

1.5 FLOATING-POINT DATA FORMATS

The following paragraphs describe the FPU's operand data formats. The FPU
seven data formats. There are three signed binary integer formats (byte, word,
word) that are identical to those supported by the integer unit. The FPU supports t
the packed decimal real format. The MC68881 and MC68882 support this
hardware and the processors starting with the MC68040 support it in software.
also supports three binary floating- point formats (single, double, and extended p
that fully comply with the IEEE 754 standard. All references in this manual to e
precision format imply the double-extended-precision format defined by the I
standard.

1.5.1 Packed Decimal Real Format

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



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

Figure 1-11. Packed Decimal Real Format

1.5.2 Binary Floating-Point Formats

Figure 1-12 illustrates the three binary floating-point data formats. The exponent in t binary floating-point formats is an unsigned binary integer with an implied bias add When subtracting the bias from the exponent’s value, the result represents a sign complement power of two. This yields the magnitude of a normalized floating-point when multiplied by the mantissa. A program can execute a CMP instruction that co floating-point numbers in memory using biased exponents, despite the absolute ma of the exponents.

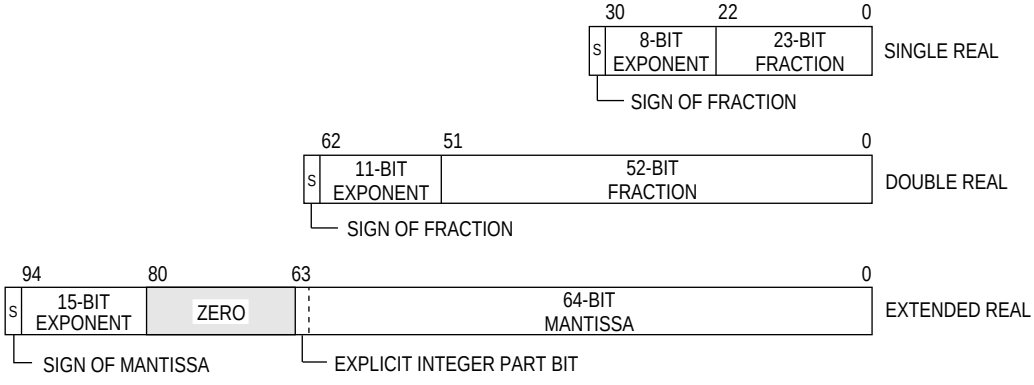


Figure 1-12. Binary Floating-Point Data Formats

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

NOTE

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

1.6 FLOATING-POINT DATA TYPES

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

There is a subtle difference between the definition of an extended- precision numb
exponent equal to zero and a single- or double-precision number with an exponen
zero. The zero exponent of a single- or double-precision number denormalizes the
definition, and the implied integer bit is zero. An extended- precision numbe
exponent of zero may have an explicit integer bit equal to one. This results in a no
number, though the exponent is equal to the minimum value. For simplicity, the
discussion treats all three floating-point formats in the same manner, where an
value of zero identifies a denormalized number. However, remember the extended-
format can deviate from this rule.

exponent can be zero.

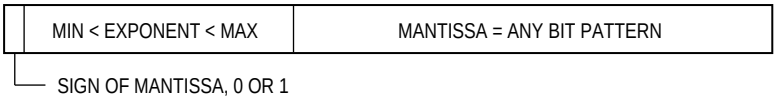


Figure 1-13. Normalized Number Format

1.6.2 Denormalized Numbers

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

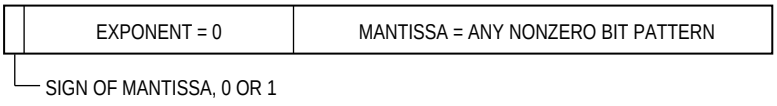


Figure 1-14. Denormalized Number Format

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

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

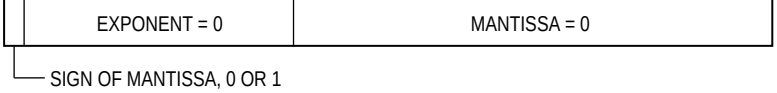


Figure 1-15. Zero Format

1.6.4 Infinities

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

.

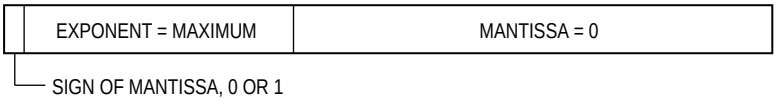


Figure 1-16. Infinity Format

1.6.5 Not-A-Numbers

When created by the FPU, NANs represent the results of operations ha mathematical interpretation, such as infinity divided by infinity. All operations in NAN operand as an input return a NAN result. When created by the user, NANs ca against unitialized variables and arrays or represent user-defined data types. For e precision NANs, the mantissa's MSB, the explicit integer bit, can be either one or Figure 1-17).

.

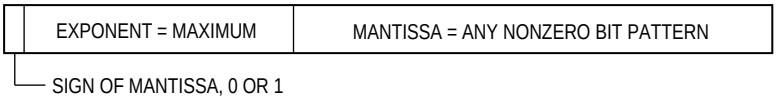


Figure 1-17. Not-A-Number Format

The FPU implements two different types of NANs identified by the value of the M mantissa for single- and double-precision, and the MSB of the mantissa minu extended-precision. If the bit is set, it is a nonsignaling NAN, otherwise, it is an S

mantissa, and storing the resulting nonsignaling NAN in the destination. Becaus
IEEE formats for NANs, the result of setting an SNAN MSB is always a nonsignalin

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

1.6.6 Data Format and Type Summary

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

Field Size In Bits	
Sign (s)	1
Biased Exponent (e)	8
Fraction (f)	23
Total	32
Interpretation of Sign	
Positive Fraction	s = 0
Negative Fraction	s = 1
Normalized Numbers	
Bias of Biased Exponent	+127 ($7F)
Range of Biased Exponent	0 < e < 255 ($FF)
Range of Fraction	Zero or Nonzero
Fraction	1.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{e-127} \times 1.f$
Denormalized Numbers	
Biased Exponent Format Minimum	0 ($00)
Bias of Biased Exponent	+126 ($7E)
Range of Fraction	Nonzero
Fraction	0.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{-126} \times 0.f$
Signed Zeros	
Biased Exponent Format Minimum	0 ($00)
Fraction	0.f = 0.0
Signed Infinities	
Biased Exponent Format Maximum	255 ($FF)
Fraction	0.f = 0.0
NANs	
Sign	Don't Care
Biased Exponent Format Maximum	255 ($FF)
Fraction	Nonzero
Representation of Fraction Nonsignaling Signaling Nonzero Bit Pattern Created by User Fraction When Created by FPCP	 0.1xxxx…xxxx 0.0xxxx…xxxx xxxxx…xxxx 11111…1111
Approximate Ranges	
Maximum Positive Normalized	3.4×10^{38}
Minimum Positive Normalized	1.2×10^{-38}
Minimum Positive Denormalized	1.4×10^{-45}

Field Size (in Bits)	
Sign (s)	1
Biased Exponent (e)	11
Fraction (f)	52
Total	64
Interpretation of Sign	
Positive Fraction	s = 0
Negative Fraction	s = 1
Normalized Numbers	
Bias of Biased Exponent	+1023 ($3FF)
Range of Biased Exponent	0 < e < 2047 ($7FF)
Range of Fraction	Zero or Nonzero
Fraction	1.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{e-1023} \times 1.f$
Denormalized Numbers	
Biased Exponent Format Minimum	0 ($000)
Bias of Biased Exponent	+1022 ($3FE)
Range of Fraction	Nonzero
Fraction	0.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{-1022} \times 0.f$
Signed Zeros	
Biased Exponent Format Minimum	0 ($00)
Fraction (Mantissa/Significand)	0.f = 0.0
Signed Infinities	
Biased Exponent Format Maximum	2047 ($7FF)
Fraction	0.f = 0.0
NANs	
Sign	0 or 1
Biased Exponent Format Maximum	255 ($7FF)
Fraction	Nonzero
Representation of Fraction Nonsignaling Signaling Nonzero Bit Pattern Created by User Fraction When Created by FPCP	 1xxxx…xxxx 0xxxx…xxxx xxxxx…xxxx 11111…1111
Approximate Ranges	
Maximum Positive Normalized	18×10^{308}
Minimum Positive Normalized	2.2×10^{-308}
Minimum Positive Denormalized	4.9×10^{-324}

Field Size (in Bits)	
Sign (s)	1
Biased Exponent (e)	15
Zero, Reserved (u)	16
Explicit Integer Bit (j)	1
Mantissa (f)	63
Total	96
Interpretation of Unused Bits	
Input	Don't Care
Output	All Zeros
Interpretation of Sign	
Positive Mantissa	s = 0
Negative Mantissa	s = 1
Normalized Numbers	
Bias of Biased Exponent	+16383 ($3FFF)
Range of Biased Exponent	0 < = e < 32767 ($7FFF)
Explicit Integer Bit	1
Range of Mantissa	Zero or Nonzero
Mantissa (Explicit Integer Bit and Fraction)	1.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{e-16383} \times 1.f$
Denormalized Numbers	
Biased Exponent Format Minimum	0 ($0000)
Bias of Biased Exponent	+16383 ($3FFF)
Explicit Integer Bit	0
Range of Mantissa	Nonzero
Mantissa (Explicit Integer Bit and Fraction)	0.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{-16383} \times 0.f$
Signed Zeros	
Biased Exponent Format Minimum	0 ($0000)
Mantissa (Explicit Integer Bit and Fraction)	0.0
Signed Infinities	
Biased Exponent Format Maximum	32767 ($7FFF)
Explicit Integer Bit	Don't Care
Mantissa (Explicit Integer Bit and Fraction)	x.000…0000

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

Table 1-7. Packed Decimal Real Format Summary

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

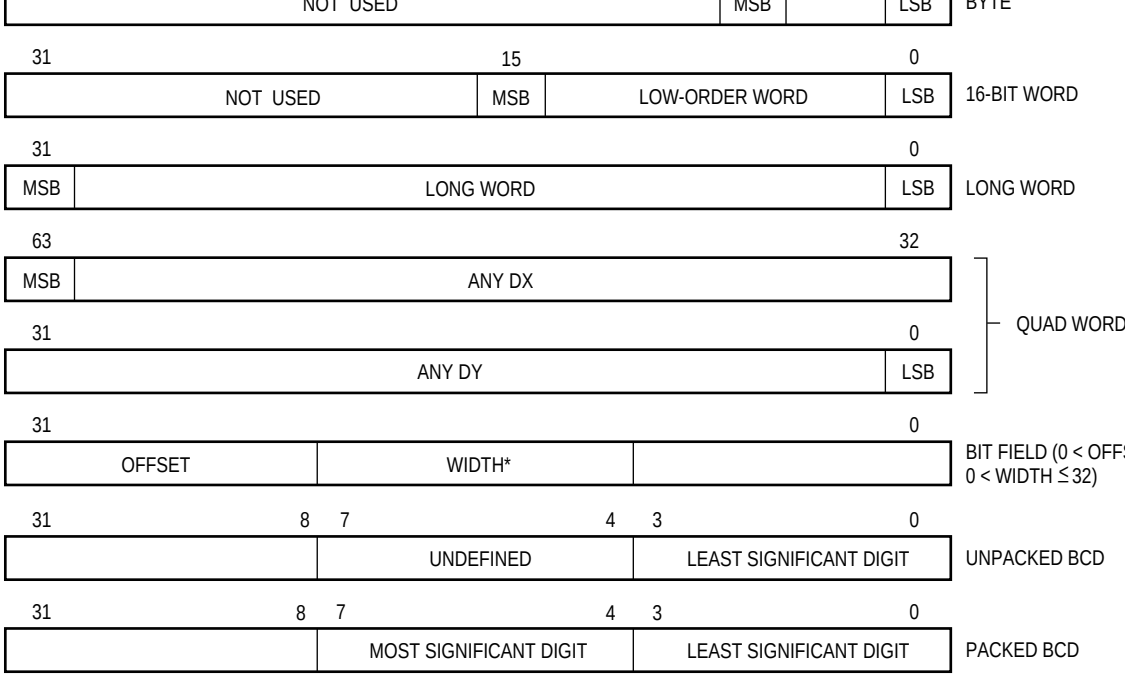
A packed decimal real data format with the SE and both Y bits set, an exponent
and a nonzero 16-bit decimal fraction is a NAN. When the FPU uses this format, the
of the NAN is moved bit- by-bit into the extended-precision mantissa of a floating-p
register. The exponent of the register is set to signify a NAN, and no conversion occ
MSB of the most significant digit in the decimal fraction (the MSB of digit 15) is a do
as in extended-precision NANs, and the MSB of minus one of digit 15 is the SNAN
NAN bit is a zero, then it is an SNAN.

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

Each integer data register is 32 bits wide. Byte and word operands occupy the low
16-bit portions of integer data registers, respectively. Long- word operands occupy
32 bits of integer data registers. A data register that is either a source or destinatio
only uses or changes the appropriate lower 8 or 16 bits (in byte or word op
respectively). The remaining high-order portion does not change and goes unu
address of the least significant bit (LSB) of a long-word integer is zero, and the M
For bit fields, the address of the MSB is zero, and the LSB is the width of the regis
one (the offset). If the width of the register plus the offset is greater than 32, th
wraps around within the register. Figure 1-18 illustrates the organization of var
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
operation (signed and unsigned). Quad words may be organized in any two int
registers without restrictions on order or pairing. There are no explicit instructio
management of this data format, although the MOVEM instruction can be used t
quad word into or out of registers.

Binary-coded decimal (BCD) data represents decimal numbers in binary form.
there are many BCD codes, the BCD instructions of the M68000 family support two
packed and unpacked. In these formats, the LSBs consist of a binary number h
numeric value of the corresponding decimal number. In the unpacked BCD form
defines one decimal number that has four LSBs containing the binary value
undefined MSBs. Each byte of the packed BCD format contains two decimal num
least significant four bits contain the least significant decimal number and
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 ca
used for byte-size operands. When an address register is a source operand, either
order word or the entire long-word operand is used, depending upon the operati
When an address register is the destination operand, the entire register becomes a
despite the operation size. If the source operand is a word size, it is sign-extended t
and then used in the operation to an address register destination. Address regis
primarily for addresses and address computation support. The instruction set
instructions that add to, compare, and move the contents of address registers. Fig
illustrates the organization of addresses in address registers.

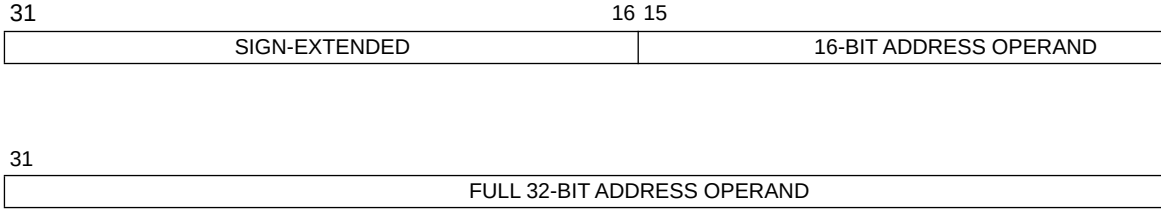


Figure 1-19. Organization of Integer Data Formats in Address Register

alternate function code registers, supervisor function code (SFC) and data func
(DFC), are 32-bit registers with only bits 0P2 implemented. These bits contain the
space values for the read or write operands of MOVES, PFLUSH, and PTEST ins
Values transfer to and from the SFC and DFC by using the MOVEC instruction. T
long-word transfers; the upper 29 bits are read as zeros and are ignored when wr

1.7.2 Organization of Integer Data Formats in Memory

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

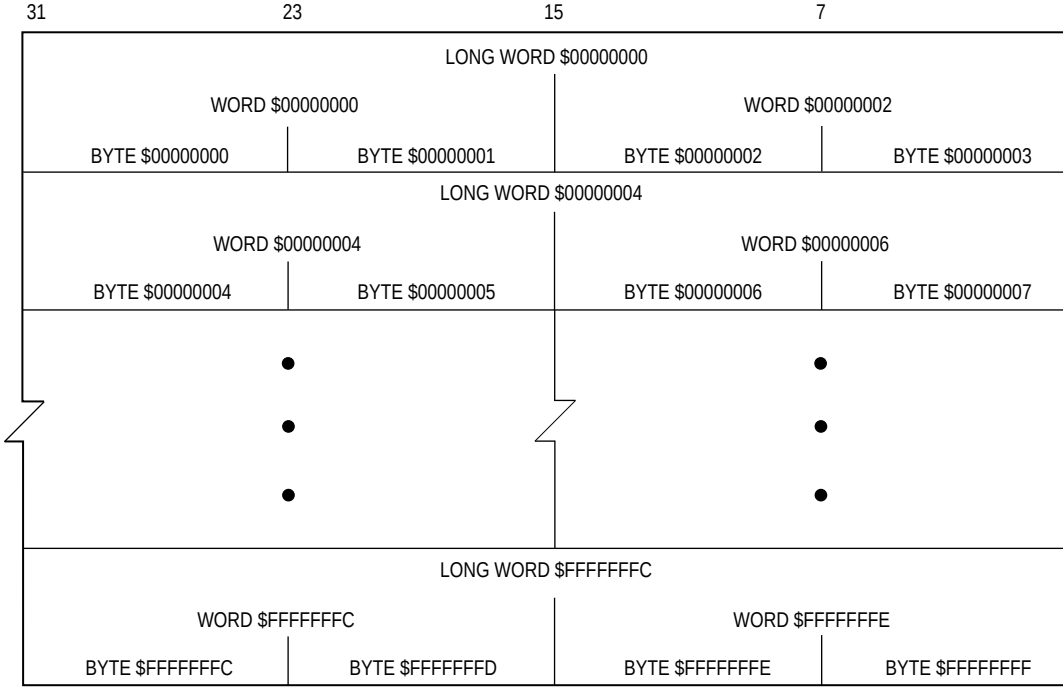


Figure 1-20. Memory Operand Addressing

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

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

A 16-byte block operand, supported by the MOVE16 instruction, has a block of 1 aligned to a 16-byte boundary. An address that can point to any byte in the block s this operand.

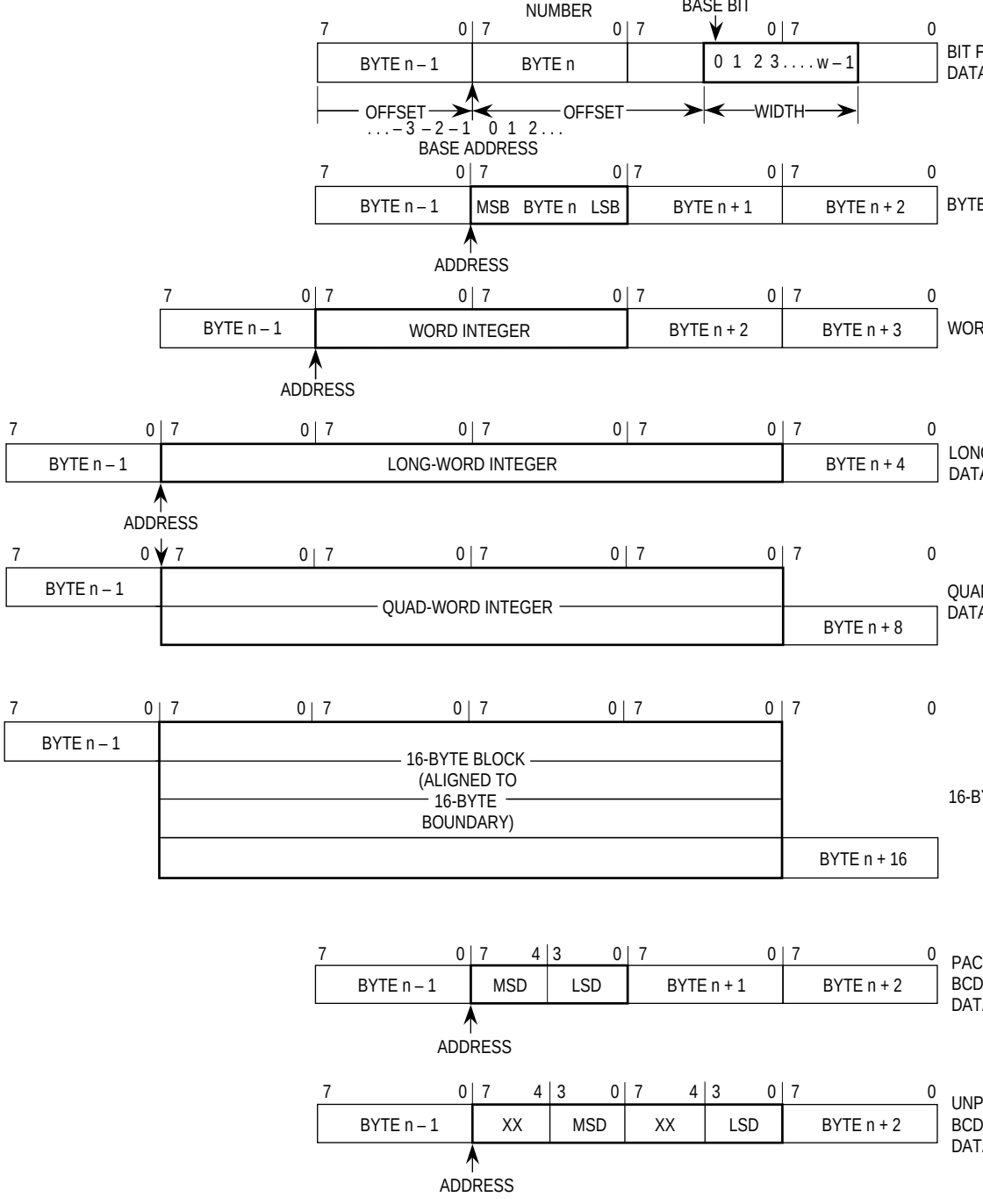


Figure 1-21. Memory Organization for Integer Operands

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

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

NOTES:
* = Data Format/Type Supported by On-Chip MC68040 FPU Hardware
† = Data Format/Type Supported by Software (MC68040FPSP)

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

.

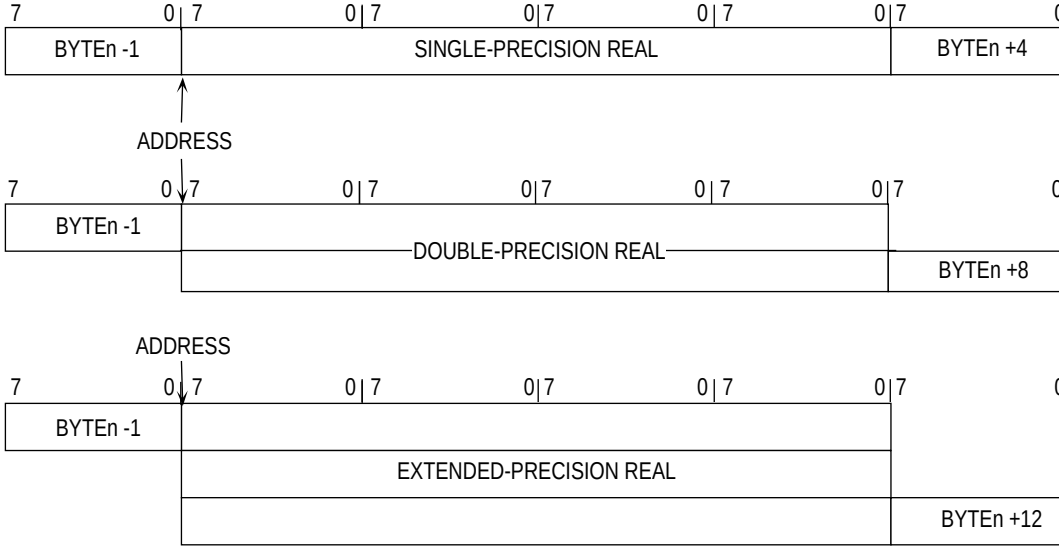


Figure 1-22. Organization of FPU Data Formats in Memory

SECTION 2
ADDRESSING CAPABILITIES

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

2.1 INSTRUCTION FORMAT

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

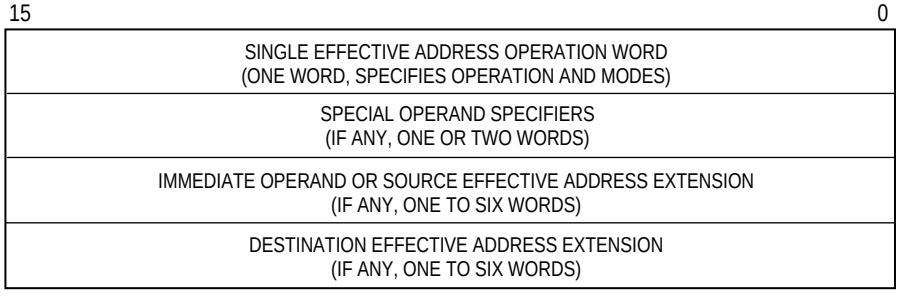


Figure 2-1. Instruction Word General Format

The single effective address operation word format is the basic instruction word (se
2-2). The encoding of the mode field selects the addressing mode. The regis
contains the general register number or a value that selects the addressing mode w
mode field contains opcode 111. Some indexed or indirect addressing modes
combination of the simple effective address operation word followed by a brief e
word. Other indexed or indirect addressing modes consist of the simple effective
operation word and a full extension word. The longest instruction is a MOVE instruc
a full extension word for both the source and destination effective addresses and eig
extension words. It also contains 32-bit base displacements and 32-bit outer displa
for both source and destination addresses. Figure 2-2 illustrates the three formats
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
X	X	X	X	X	X	X	X	X	X	EFFECTIVE ADDRESS				
										MODE			REGIS	

BRIEF EXTENSION WORD FORMAT

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

Figure 2-2. Instruction Word Specification Formats

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 (I the index/indirect selection (I/IS) field determine the type of indexing and indire Table 2-2 lists the index and indirect operations corresponding to all combinations I/IS values.

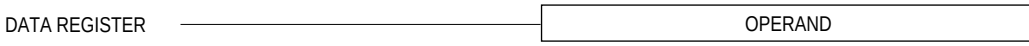
0	010	Indirect Preindexed with Word Outer Displacement
0	011	Indirect Preindexed with Long Outer Displacement
0	100	Reserved
0	101	Indirect Postindexed with Null Outer Displacement
0	110	Indirect Postindexed with Word Outer Displacement
0	111	Indirect Postindexed with Long Outer Displacement
1	000	No 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 ins
defines the location of every operand for the function. Instructions specify an
location in one of three ways. A register field within an instruction can specify the re
be used; an instruction's effective address field can contain addressing mode info
or the instruction's definition can imply the use of a specific register. Other fields w
instruction specify whether the register selected is an address or data register and
register is to be used. **Section 1 Introduction** contains detailed register descriptio

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

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

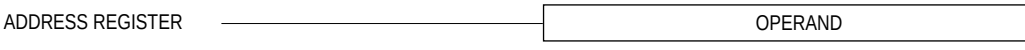


2.2.2 Address Register Direct Mode

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

.

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

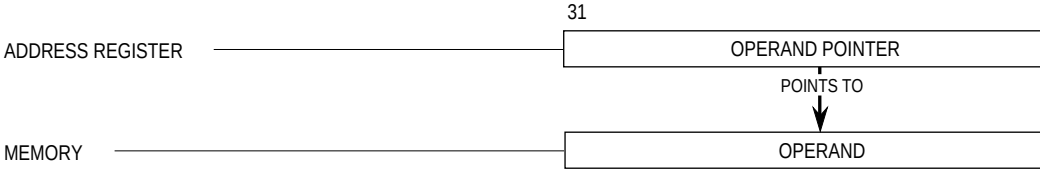


2.2.3 Address Register Indirect Mode

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

.

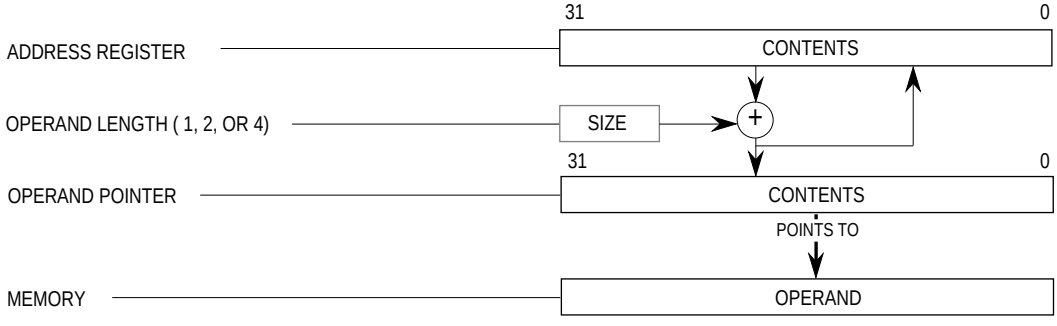
GENERATION:	EA = (An)
ASSEMBLER SYNTAX:	(An)
EA MODE FIELD:	010
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	0



may support incrementing for any operand size, up to 255 bytes. If the address re
the stack pointer and the operand size is byte, the address is incremented by two
the stack pointer aligned to a word boundary.

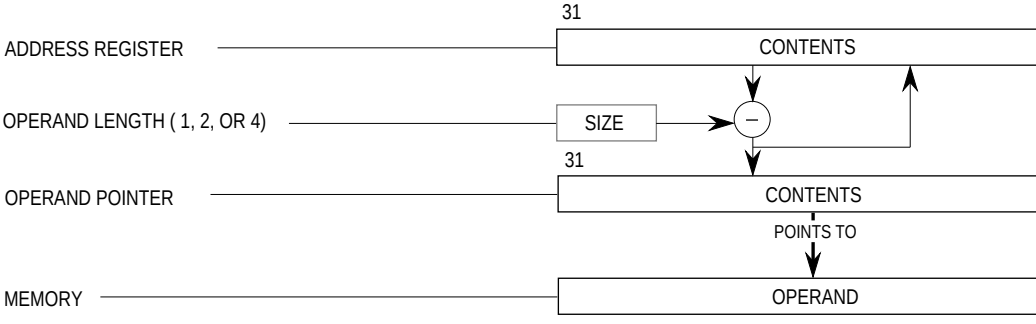
.

GENERATION:	EA = (An) + SIZE
ASSEMBLER SYNTAX:	(An) +
EA MODE FIELD:	011
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	0



support decrementing for any operand size up to 255 bytes. If the address regis
stack pointer and the operand size is byte, the address is decremented by two to
stack pointer aligned to a word boundary.

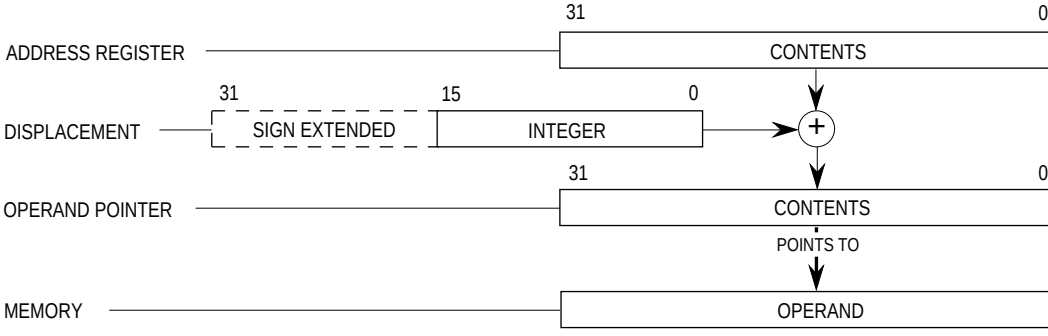
GENERATION:	EA = (An)–SIZE
ASSEMBLER SYNTAX:	– (An)
EA MODE FIELD:	100
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	0



address calculations.

.

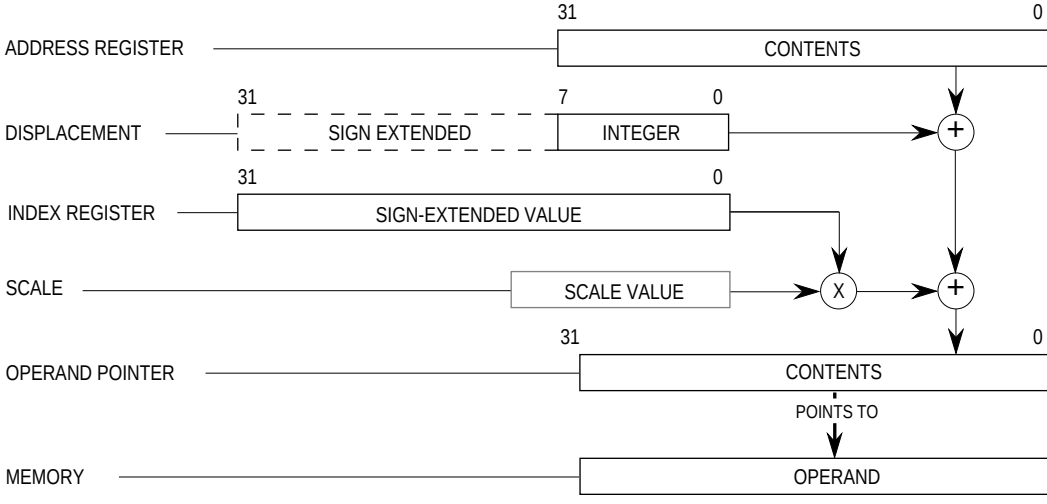
GENERATION:	EA = (An) + d_{16}
ASSEMBLER SYNTAX:	(d_{16},An)
EA MODE FIELD:	101
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	1



eight bits; and the index register's sign-extended contents (possibly scaled). The u
specify the address register, the displacement, and the index register in this mode

.

GENERATION:	EA = (An) + (Xn) + d_8
ASSEMBLER SYNTAX:	(d_8 ,An, Xn.SIZE*SCALE)
EA MODE FIELD:	110
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	1

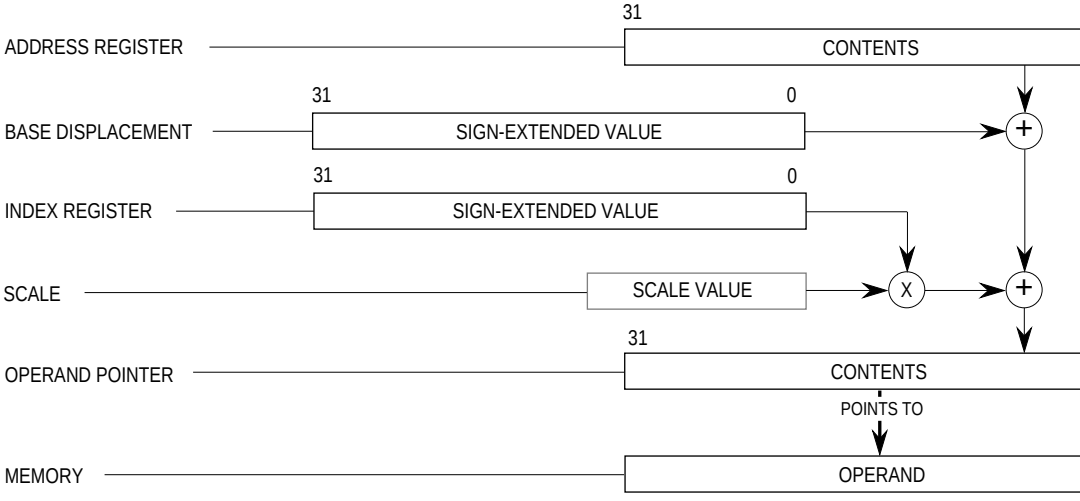


index register.

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

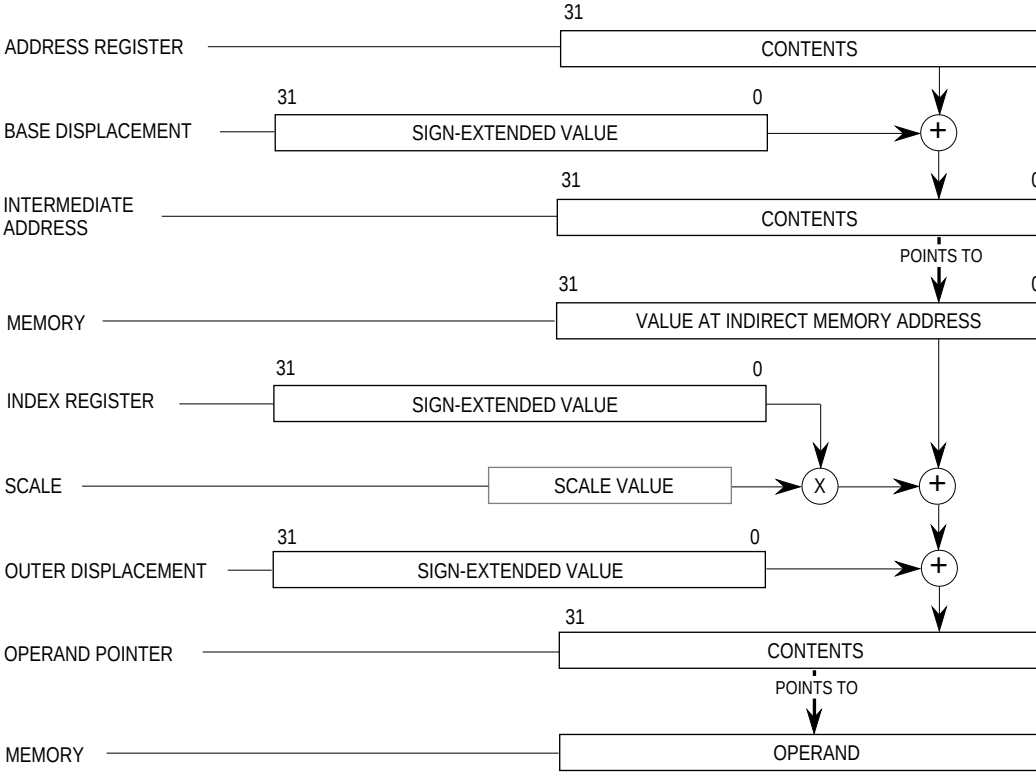


displacements and the index register contents are sign-extended to 32 bits.

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

.

GENERATION:	EA = (An + bd) + 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

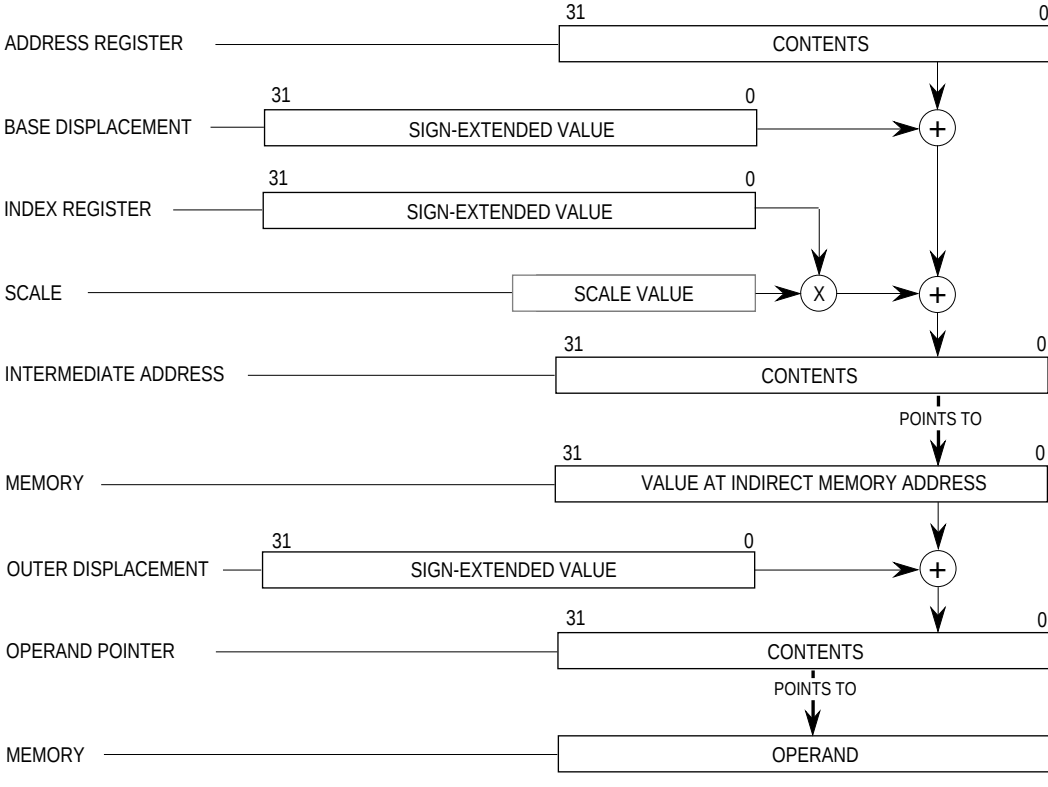


and the index register contents are sign-extended to 32 bits.

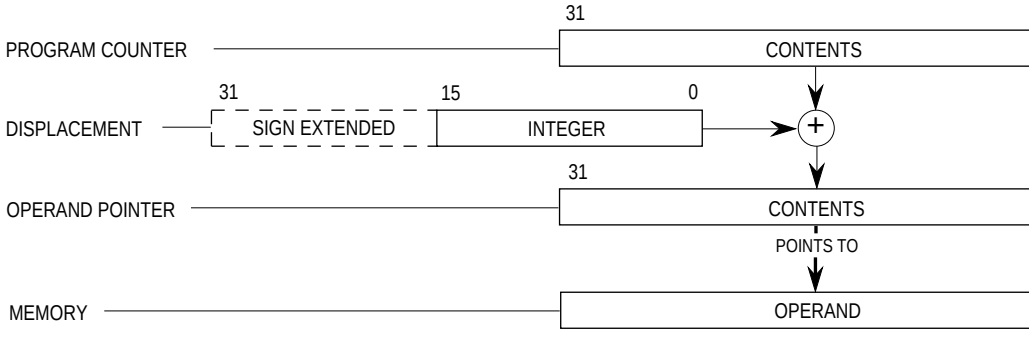
In the syntax for this mode, brackets enclose the values used to calculate the inter memory address. All four user-specified values are optional. Both the base ar displacements may be null, word, or long word. When omitting a displace suppressing an element, its value is zero in the effective address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



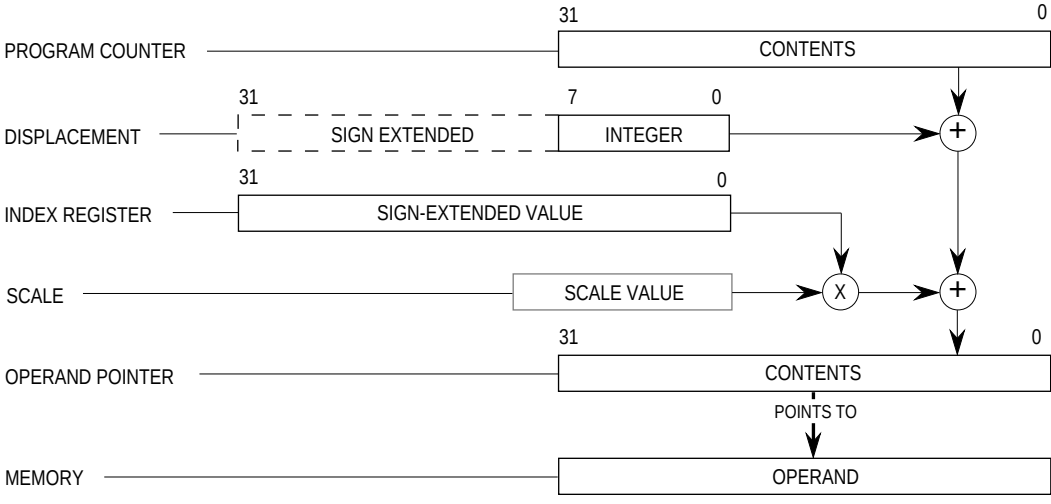
GENERATION:	EA = (PC) + d_{16}
ASSEMBLER SYNTAX:	(d_{16},PC)
EA MODE FIELD:	111
EA REGISTER FIELD:	010
NUMBER OF EXTENSION WORDS:	1



index operand. The value in the PC is the address of the extension word. This is a reference allowed only for reads. The user must include the displacement, the PC, index register when specifying this addressing mode.

.

GENERATION:	EA = (PC) + (Xn) + d_8
ASSEMBLER SYNTAX:	(d_8,PC,Xn.SIZE*SCALE)
EA MODE FIELD:	111
EA REGISTER FIELD:	011
NUMBER OF EXTENSION WORDS:	1

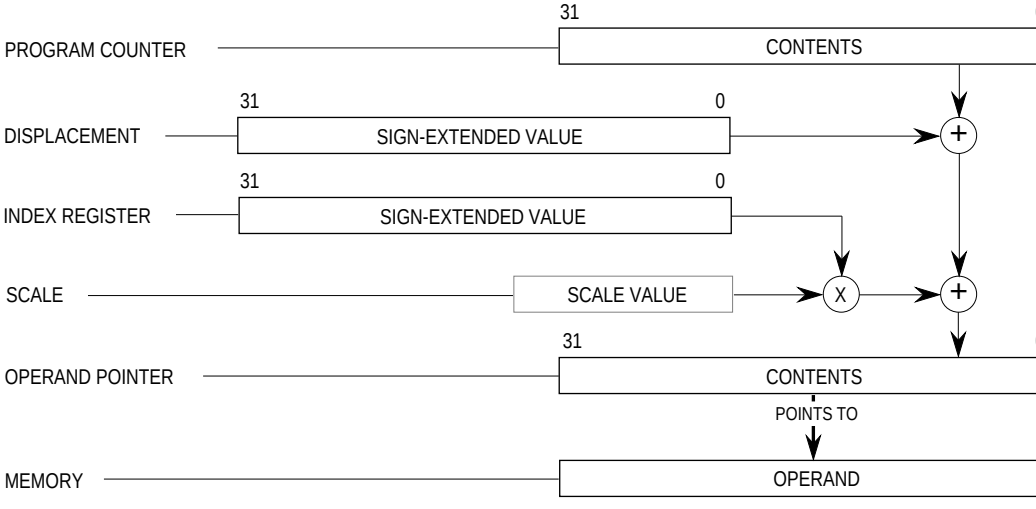


displacement, and the scaled contents of the sign-extended index register. The va
PC is the address of the first extension word. This is a program reference allowe
reads.

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

.

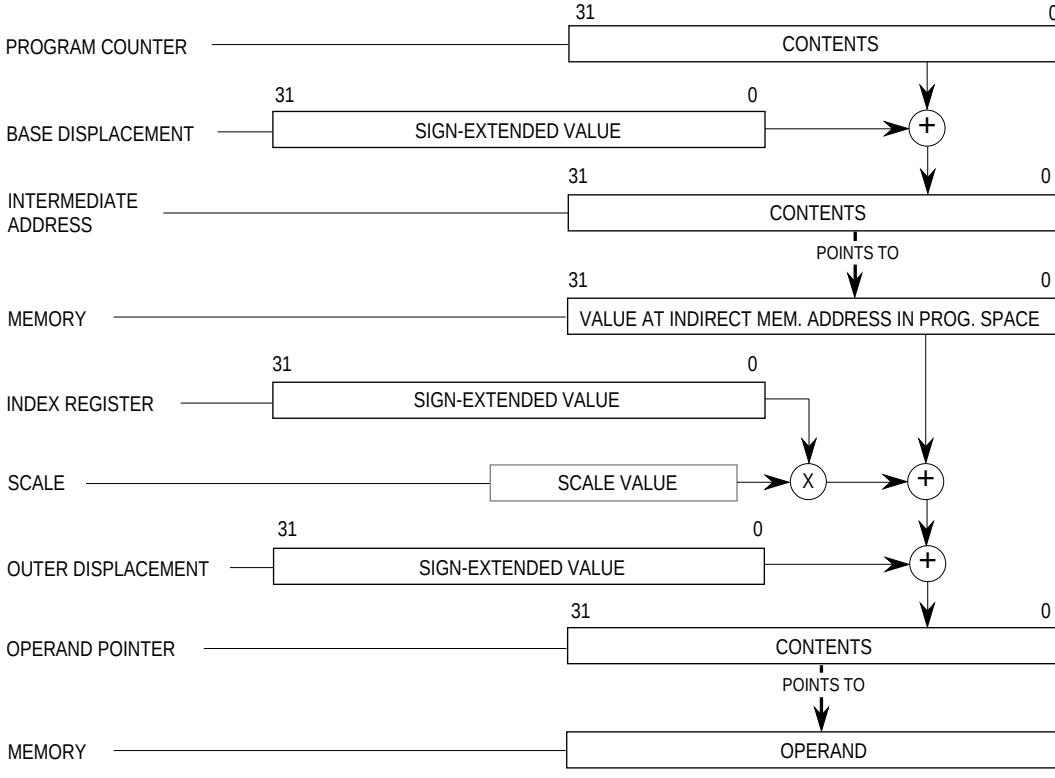
GENERATION:	EA = (PC) + (Xn) + bd
ASSEMBLER SYNTAX:	(bd, PC, Xn. SIZE*SCALE)
EA MODE FIELD:	111
EA REGISTER FIELD:	011
NUMBER OF EXTENSION WORDS:	1,2, OR 3



adds the scaled contents of the index register and the optional outer displacement
the effective address. The value of the PC used in the calculation is the address of
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 inter
memory address. All four user-specified values are optional. The user must su
assembler notation ZPC (a zero value PC) to show the PC is not used. This allows
to access the program space without using the PC in calculating the effective addre
the base and outer displacements may be null, word, or long word. When on
displacement or suppressing an element, its value is zero in the effective
calculation.

GENERATION:	EA = (bd + PC) + Xn.SIZE*SCALE + od
ASSEMBLER SYNTAX:	([bd,PC],Xn.SIZE*SCALE,od)
EA MODE FIELD:	111
EA REGISTER FIELD:	011
NUMBER OF EXTENSION WORDS:	1,2,3,4, or 5

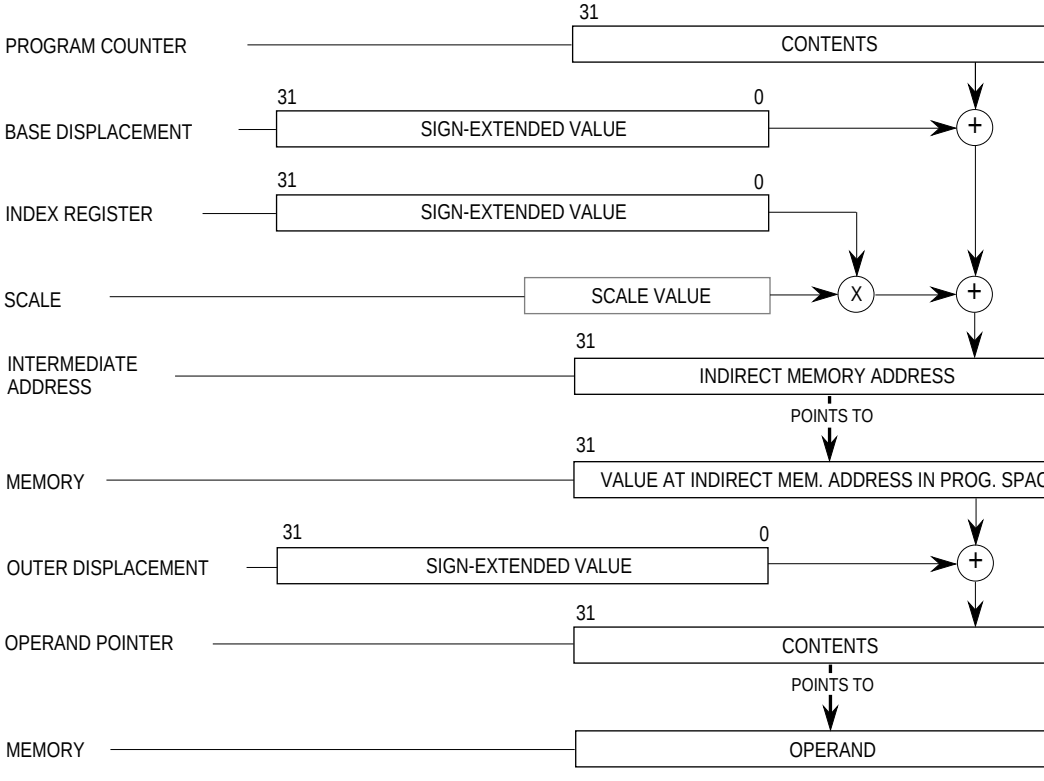


a long word at immediate indirect memory address and adds the optio
displacement to yield the effective address. The value of the PC is the address o
extension word. This is a program reference allowed only for reads.

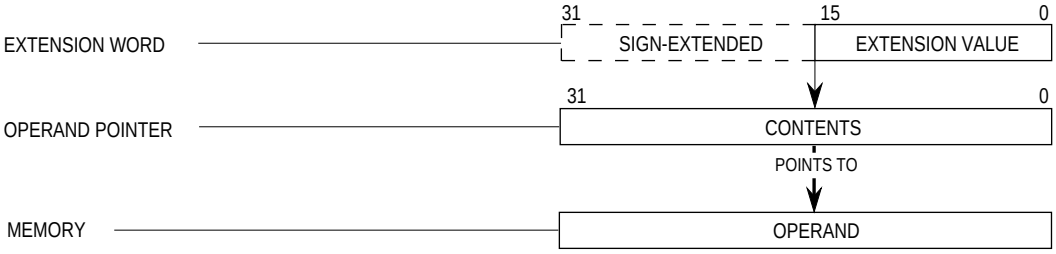
In the syntax for this mode, brackets enclose the values used to calculate the inte
memory address. All four user-specified values are optional. The user must s
assembler notation ZPC showing that the PC is not used. This allows the user to a
program space without using the PC in calculating the effective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

.

GENERATION: EA = (bd + PC) + Xn.SIZE*SCALE + od
ASSEMBLER SYNTAX: ([bd,PC,Xn.SIZE*SCALE],od)
EA MODE FIELD: 111
EA REGISTER FIELD: 011
NUMBER OF EXTENSION WORDS: 1,2,3,4, or 5



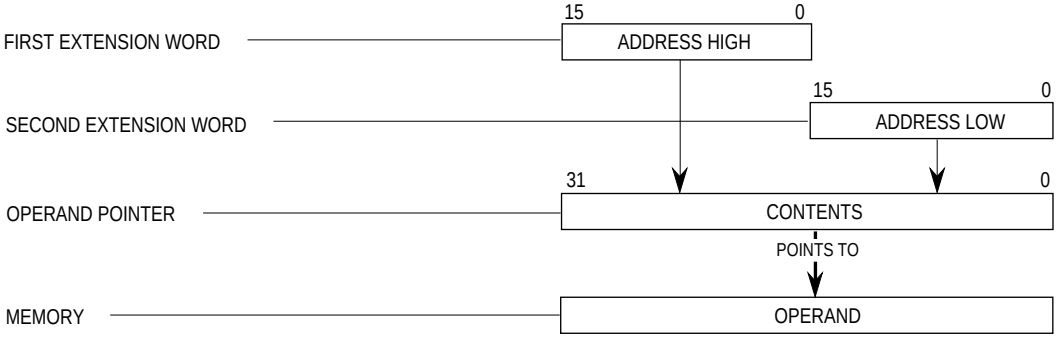
GENERATION:	EA GIVEN
ASSEMBLER SYNTAX:	(xxx).W
EA MODE FIELD:	111
EA REGISTER FIELD:	000
NUMBER OF EXTENSION WORDS:	1



2.2.17 Absolute Long Addressing Mode

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

GENERATION:	EA GIVEN
ASSEMBLER SYNTAX:	(xxx).L
EA MODE FIELD:	111
EA REGISTER FIELD:	001
NUMBER OF EXTENSION WORDS:	2



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-orde 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.

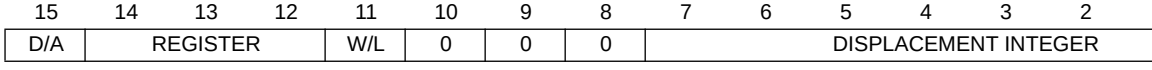
2.3 EFFECTIVE ADDRESSING MODE SUMMARY

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

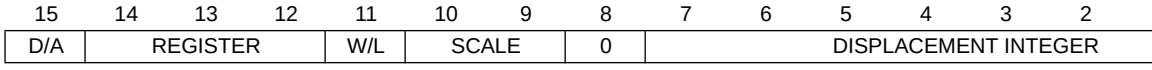
These categories sometimes combine to form new categories that are more restric combined classifications are alterable memory (addressing modes that are both and memory addresses) and data alterable (addressing modes that are both alte data). Table 2-4 lists a summary of effective addressing modes and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

Address	An	001	reg. no.	—	—	—	
Register Indirect							
Address	(An)	010	reg. no.	X	X	X	
Address with Postincrement	(An)+	011	reg. no.	X	X	—	
Address with Predecrement	–(An)	100	reg. no.	X	X	—	
Address with Displacement	(d_{16},An)	101	reg. no.	X	X	X	
Address Register Indirect with Index							
8-Bit Displacement	(d_8,An,Xn)	110	reg. no.	X	X	X	
Base Displacement	(bd,An,Xn)	110	reg. no.	X	X	X	
Memory Indirect							
Postindexed	([bd,An],Xn,od)	110	reg. no.	X	X	X	
Preindexed	([bd,An,Xn],od)	110	reg. no.	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	
Preindexed	([bd,PC,Xn],od)	111	011	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	—	

that allows the CPU32, MC68020, MC68030, and MC68040 to distinguish the basi
family architecture's new address extensions. Figure 2-3 illustrates these brief
word formats. The encoding for SCALE used by the CPU32, MC68020, MC68
MC68040 is a compatible extension of the M68000 family architecture. A value o
SCALE is the same encoding for both extension words. Software that uses this en
compatible with all processors in the M68000 family. Both brief extension word fo
not contain the other values of SCALE. Software can be easily migrated in an
compatible direction, with downward support only for nonscaled addressing. If the
were to execute an instruction that encoded a scaling factor, the scaling factor
ignored and would not access the desired memory address. The earlier microproc
not recognize the brief extension word formats implemented by newer processors.
they can detect illegal instructions, they do not decode invalid encodings of
extension word formats as exceptions.



(a) MC68000, MC68008, and MC68010



(b) CPU32, MC68020, MC68030, and MC68040

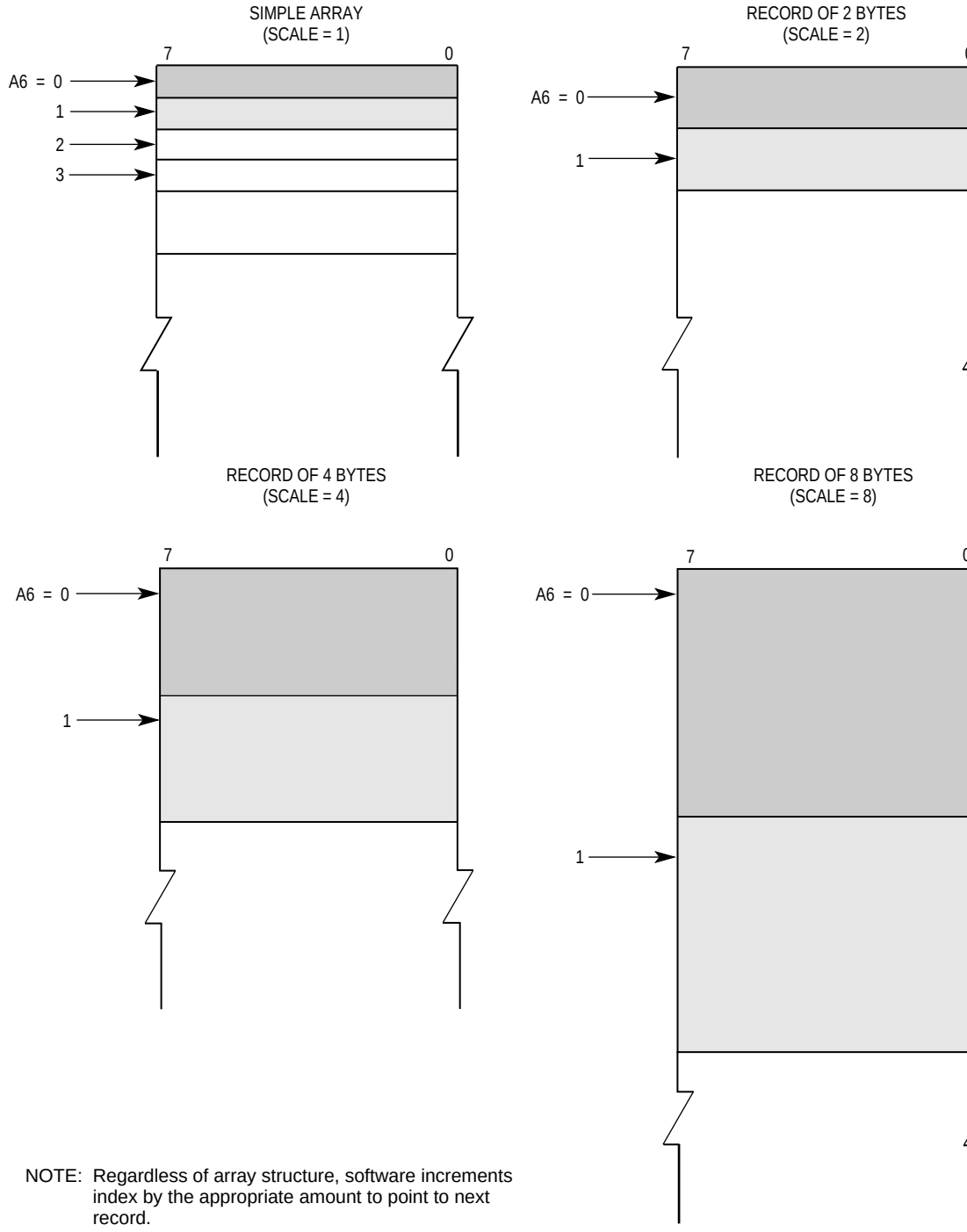
Figure 2-3. M68000 Family Brief Extension Word Formats

independently of each other. However, at least one element must be active
suppressed. When an element is suppressed, it has an effective value of zero.

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

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

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



NOTE: Regardless of array structure, software increments index by the appropriate amount to point to next record.

Figure 2-4. Addressing Array Items

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

NOTE: S indicates suppressed and A indicates active.

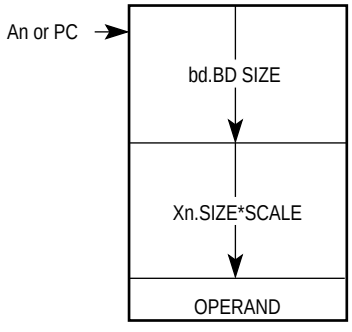


Figure 2-5. No Memory Indirect Action

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

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

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

NOTE: S indicates suppressed and A indicates active.

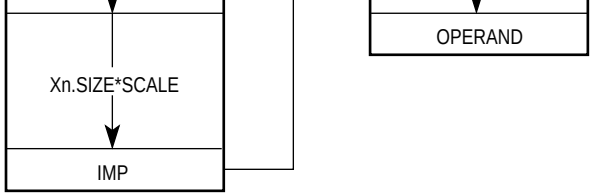


Figure 2-6. Memory Indirect with Preindex

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

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

NOTE: S indicates suppressed and A indicates active.

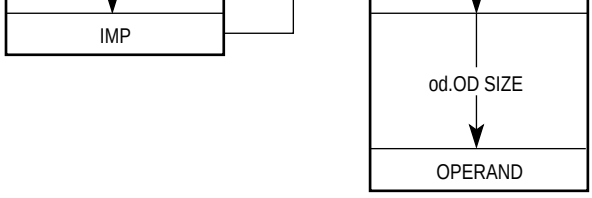


Figure 2-7. Memory Indirect with Postindex

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

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

NOTE: S indicates suppressed and A indicates active.

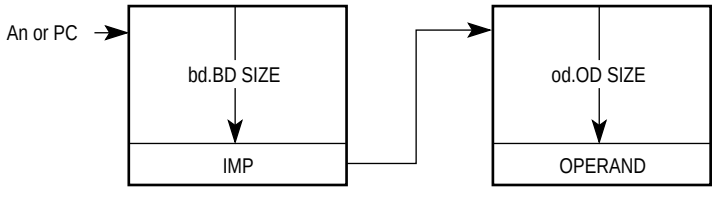


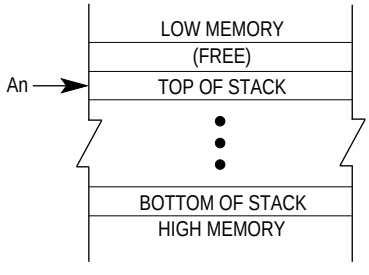
Figure 2-8. Memory Indirect with Index Suppress

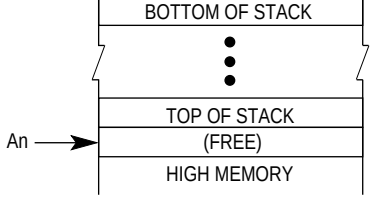
Address register seven (A7) is the system stack pointer. Either the user stack pointe
the interrupt stack pointer (ISP), or the master stack pointer (MSP) is active at any c
Refer to **Section 1 Introduction** for details on these stack pointers. To keep data
system stack aligned for maximum efficiency,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

The user can implement stacks with the address register indirect with postincrem
predecrement addressing modes. With an address register the user can implemen
that fills either from high memory to low memory or from low memory to high m
Important consideration are:

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

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

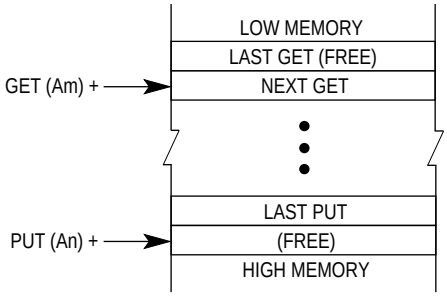




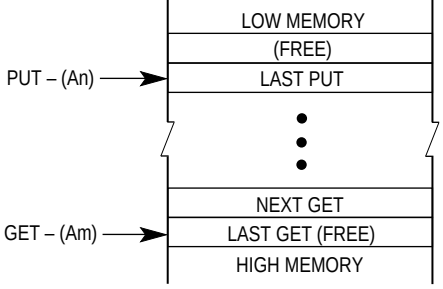
2.6.2 Queues

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

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



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



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

SECTION 3
INSTRUCTION SET SUMMARY

This section briefly describes the M68000 family instruction set, using Motorola,s language syntax and notation. It includes instruction set details such as notation an selected instruction examples, and an integer condition code discussion. The concludes with a discussion of floating-point details such as computational conditional test definitions, an explanation of the operation table, and a discussion numbers (NANs) and postprocessing.

3.1 INSTRUCTION SUMMARY

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

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

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

×	Arithmetic multiplication.
÷	Arithmetic division or conjunction symbol.
~	Invert; operand is logically complemented.
Λ	Logical AND
V	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); 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 fals optional “else”clause is present, the operations after “else”are performed. If the conditio and else is omitted, the instruction performs no operation. Refer to the Bcc instruction d 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

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 st 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, respe
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

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

and ensure that only valid address manipulations are executed. In addition to th
MOVE instructions, there are several special data movement instructions: I
MOVEM, MOVEP, MOVEQ, EXG, LEA, PEA, LINK, and UNLK. The MOVE16 inst
an MC68040 extension to the M68000 instruction set.

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

	FPm,<ea> <ea>,FPcr FPcr,<ea>	B, W, L, S, D, X, P 32 32	
FSMOVE, FDMOVE	FPm,FPn <ea>,FPn	X B, W, L, S, D, X	Source → Destination; round destination to sing double precision.
FMOVEM	<ea>,<list>[1] <ea>,Dn <list>[1],<ea> Dn,<ea>	32, X X 32, X X	Listed Registers → Destination Source → Listed Registers
LEA	<ea>,An	32	<ea> → An
LINK	An,#<d>	16, 32	SP – 4 → SP; An → (SP); SP → An, SP + D →
MOVE MOVE16 MOVEA	<ea>,<ea> <ea>,<ea> <ea>,An	8, 16, 32 16 bytes 16, 32 → 32	Source → Destination Aligned 16-Byte Block → Destination
MOVEM	list,<ea> <ea>,list	16, 32 16, 32 → 32	Listed Registers → Destination Source → Listed Registers
MOVEP	Dn, (d_{16},An) (d_{16},An),Dn	16, 32	Dn 31–24 → (An + d_n); Dn 23–16 → (An + d_n + Dn 15–8 → (An + d_n + 4); Dn 7–0 → (An + d_n + (An + d_n) → Dn 31–24; (An + d_n + 2) → Dn 23– (An + d_n + 4) → Dn 15–8; (An + d_n + 6) → Dn 7
MOVEQ	#<data>,Dn	8 → 32	Immediate Data → Destination
PEA	<ea>	32	SP – 4 → SP; <ea> → (SP)
UNLK	An	32	An → SP; (SP) → An; SP + 4 → SP

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

3.1.2 Integer Arithmetic Instructions

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

Instruction	Operand Syntax	Operand Size	Operation
ADD ADDA	Dn,<ea> <ea>,Dn <ea>,An	8, 16, 32 8, 16, 32 16, 32	Source + Destination → Destination
ADDI ADDQ	#<data>,<ea> #<data>,<ea>	8, 16, 32 8, 16, 32	Immediate Data + Destination → Destinati
ADDX	Dn,Dn –(An), –(An)	8, 16, 32 8, 16, 32	Source + Destination + X → Destination
CLR	<ea>	8, 16, 32	0 → Destination
CMP CMPA	<ea>,Dn <ea>,An	8, 16, 32 16, 32	Destination – Source
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 DIVSL/DIVUL	<ea>,Dn <ea>,Dr–Dq <ea>,Dq <ea>,Dr–Dq	32 ÷16 → 16,16 64 ÷32 → 32,32 32 ÷32 → 32 32 ÷32 → 32,32	Destination ÷Source → Destination (Signed or Unsigned Quotient, Remainder
EXT EXTB	Dn Dn Dn	8 → 16 16 → 32 8 → 32	Sign-Extended Destination → Destination
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 SUBA	<ea>,Dn Dn,<ea> <ea>,An	8, 16, 32 8, 16, 32 16, 32	Destination = Source → Destination
SUBI SUBQ	#<data>,<ea> #<data>,<ea>	8, 16, 32 8, 16, 32	Destination – Immediate Data → Destinati
SUBX	Dn,Dn –(An), –(An)	8, 16, 32 8, 16, 32	Destination – Source – X → Destination

Table 3-4. Logical Operation Format

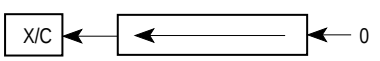
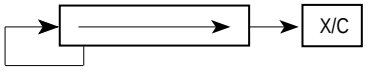
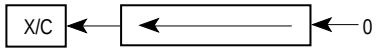
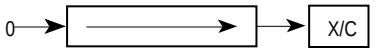
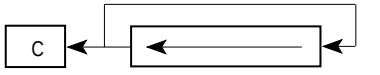
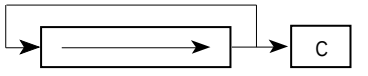
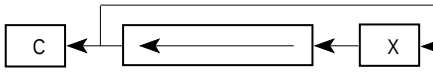
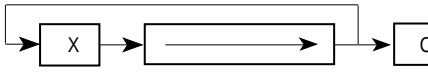
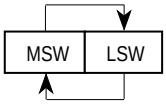
Instruction	Operand Syntax	Operand Size	Operation
AND	<ea>,Dn Dn,<ea>	8, 16, 32 8, 16, 32	Source Λ Destination → Destination
ANDI	#<data>,<ea>	8, 16, 32	Immediate Data Λ Destination → Destination
EOR	Dn,<ea>	8, 16, 32	Source ⊕ Destination → Destination
EORI	#<data>,<ea>	8, 16, 32	Immediate Data ⊕ Destination → Destination
NOT	<ea>	8, 16, 32	~ Destination → Destination
OR	<ea>,Dn Dn,<ea>	8, 16, 32	Source V Destination → Destination
ORI	#<data>,<ea>	8, 16, 32	Immediate Data V Destination → Destination

3.1.4 Shift and Rotate Instructions

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

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

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

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.

operations; Z refers to the zero bit of the CCR.

Table 3-6. Bit Manipulation Operation Format

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

3.1.6 Bit Field Instructions

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

Table 3-7. Bit Field Operation Format

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

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

Table 3- 8 X refers to the X-bit in the CCR.

Table 3-8. Binary-Coded Decimal Operation Format

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

3.1.8 Program Control Instructions

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

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

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

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

*Not applicable to the Bcc instructions.

3.1.9 System Control Instructions

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

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

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

summarizes these instructions.

Table 3-11. Cache Control Operation Format

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

3.1.11 Multiprocessor Instructions

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

Table 3-12. Multiprocessor Operations

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

Table 3-13. MMU Operation Format

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

3.1.13 Floating-Point Arithmetic Instructions

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

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

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

Table 3-15. Dyadic Floating-Point Operations

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

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

Table 3-16. Monadic Floating-Point Operation Format

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

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

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. Prog system control instructions also use certain combinations of these bits to control and system flow. The condition codes meet consistency criteria across instructio and instances. They also meet the criteria of meaningful results, where no chang unless it provides useful information. Refer to **Section 1 Introduction** for details co the CCR.

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

ADD, ADDI, ADDQ						C = Sm ∧ Dm ∨ $\overline{Rm}$ ∧ Dm ∨ Sm ∧ $\overline{R}$
ADDX	*	*	?	?	?	V = Sm ∧ Dm ∧ $\overline{Rm}$ ∨ $\overline{Sm}$ ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ Dm ∨ $\overline{Rm}$ ∧ Dm ∨ Sm ∧ $\overline{Rm}$ Z = Z ∧ $\overline{Rm}$ ∧ …∧ $\overline{R0}$
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) ∨ (R = UB) C = (LB ≤ UB) ∧ (IR < LB) ∨ (R > UB) ∨ (UB < LB) ∧ (R > UB) ∧ (R < LB)
SUB, SUBI, SUBQ	*	*	*	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm
SUBX	*	*	?	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm Z = Z ∧ $\overline{Rm}$ ∧ …∧ $\overline{R0}$
CAS, CAS2, CMP, CMPA, CMPI, CMPM	—	*	*	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm
DIVS, DUVU	—	*	*	?	0	V = Division Overflow
MULS, MULU	—	*	*	?	0	V = Multiplication Overflow
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow Z = Z ∧ $\overline{Rm}$ ∧ …∧ $\overline{R0}$
NEG	*	*	*	?	?	V = Dm ∧ Rm C = Dm ∨ Rm
NEGX	*	*	?	?	?	V = Dm ∧ Rm C = Dm ∨ Rm Z = Z ∧ $\overline{Rm}$ ∧ …∧ $\overline{R0}$
BTST, BCHG, BSET, BCLR	—	—	?	—	—	Z = $\overline{Dn}$
BFTST, BFCHG, BFSET, BFCLR	—	?	?	0	0	N = Dm Z = $\overline{Dn}$ ∧ $\overline{Dm–1}$ ∧ …∧ $\overline{D0}$
BFEXTS, BFEXTU, BFFFO	—	?	?	0	0	N = Sm Z = Sm ∧ $\overline{Sm–1}$ ∧…∧ $\overline{S0}$
BFINS	—	?	?	0	0	N = Dm Z = $\overline{Dm}$ ∧ $\overline{Dm–1}$ ∧…∧ $\overline{D0}$
ASL	*	*	*	?	?	V = Dm ∧ $\overline{Dm–1}$ ∨…∨ $\overline{Dm–}$ r ∨ $\overline{Dm}$ ∧ (DM –1 ∨ …+ Dm – r) C = $\overline{Dm–}$ $\overline{r+1}$
ASL (r = 0)	—	*	*	0	0	
LSL, ROXL	*	*	*	0	?	C = Dm – r + 1

ROL (r = 0)	—	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	C = Dr – 1
ASR, LSR (r = 0)	—	*	*	0	0	
ROXR (r = 0)	—	*	*	0	?	X = C
ROR	—	*	*	0	?	C = Dr – 1
ROR (r = 0)	—	*	*	0	0	

? = Other—See Special Definition
N = Result Operand (MSB)
Z = $\overline{Rm} \wedge \ldots \wedge \overline{R0}$
Sm = Source Operand (MSB)
Dm = Destination Operand (MSB)
Rm = Result Operand (MSB)
$\overline{Rm}$ = 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	$C \vee Z$
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.

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

3.3.2 Using the Moves Instruction

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

3.3.3 Nested Subroutine Calls

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

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

3.3.4 Bit Field Instructions

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

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

setting the condition codes accordingly. The test bit field (BFTST) instruction tests in the field, setting the condition codes appropriately without altering the bit field first one in bit field (BFFFO) instruction scans a bit field from bit 0 to the right until bit set to one and loads the bit offset of the first set bit into the specified data regi bits in the field are set, the field offset and the field width is loaded into the registe

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

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

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

3.3.5 Pipeline Synchronization with the Nop Instruction

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

3.4 FLOATING-POINT INSTRUCTION DETAILS

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

operand type along the top, and the destination operand type along the side. numbers are normalized, denormalized, unnormalized real numbers, or integers converted to normalized or denormalized extended-precision numbers upon ente FPU.

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

DESTINATION		SOURCE[1]		
		+ In Range –	+ Zero –	+ Infinity –
In Range	+	ADD	ADD	+inf –inf
	–			
Zero	+	ADD	+ 0.0 0.0^2	+inf –inf
	–		0.0^2 –0.0	
Infinity	+	+inf	+inf	+inf NAN[3]
	–	–inf	–inf	NAN[3] –inf

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

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

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

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

that the result obtained by any conforming device can be predicted exactly for a
precision and rounding mode. The error bound defined by the IEEE 754 standard is
unit in the last place of the destination data format in the RN mode, and one unit in
in the other rounding modes. The operation's data format must have the same inp
rounding mode, and precision. The standard also specifies the maximum allowa
that can be introduced during a calculation and the manner in which rounding of
is performed.

The single- and double-precision formats provide emulation for devices that onl
those precisions. The execution speed of all instructions is the same whether usi
or double-precision rounding. When using these two data formats, the FPU prod
same results as any other device that conforms to the IEEE standard but does no
extended precision. The results are the same when performing the same ope
extended precision and storing the results in single- or double-precision format.

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

Additionally if the external operand is a denormalized number, the number is no
before an operation is performed. However, an external denormalized number mov
floating-point data register is stored as a denormalized number. The numbe
normalized and then denormalized before it is stored in the designated floating-p
register. This method simplifies the handling of all other data formats and types.

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

Figure 3-1 illustrates the intermediate result format. The intermediate result's expo
some dyadic operations (i.e., multiply and divide) can easily overflow or underflow
bit exponent of the designation floating-point register. To simplify the overflow and u
detection, intermediate results in the FPU maintain a 16-bit (17 bits for the MC68
MC68882), twos complement, integer exponent. Detection of an overflow or u
intermediate result always converts the 16-bit exponent into a 15-bit biased exponen
being stored in a floating-point data register. The FPU internally maintains th
mantissa for rounding purposes. The mantissa is always rounded to 64 bits
depending on the selected rounding precision) before it is stored in a floating-po
register.

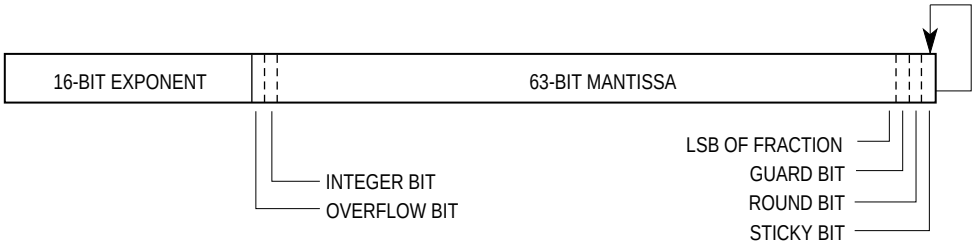


Figure 3-1. Intermediate Result Format

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

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

accomplished through the intermediate result. Single-precision results are rounde
bit boundary; double-precision results are rounded to a 53-bit boundary; and e
precision results are rounded to a 64-bit boundary. Table 3-21 lists the encoding
FPCR that denote the rounding and precision modes.

Table 3-21. FPCR Encodings

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

Rounding the intermediate result's mantissa to the specified precision and checkin
bit intermediate exponent to ensure that it is within the representable range of the
rounding precision accomplishes range control. Range control is a method used
correct emulation of a device that only supports single- or double- precision arithm
intermediate result's exponent exceeds the range of the selected precision, the
value appropriate for an underflow or overflow is stored as the result in the 16-bit e
precision format exponent. For example, if the data format and rounding mode
precision RM and the result of an arithmetic operation overflows the magnitude of t
precision format, the largest normalized single-precision value is stored as an e
precision number in the destination floating-point data register (i.e., an unbias
exponent of $00FF and a mantissa of $FFFFFF0000000000). If an infinity is the ap
result for an underflow or overflow, the infinity value for the destination data format
as the result (i.e., an exponent with the maximum value and a mantissa of zero).

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

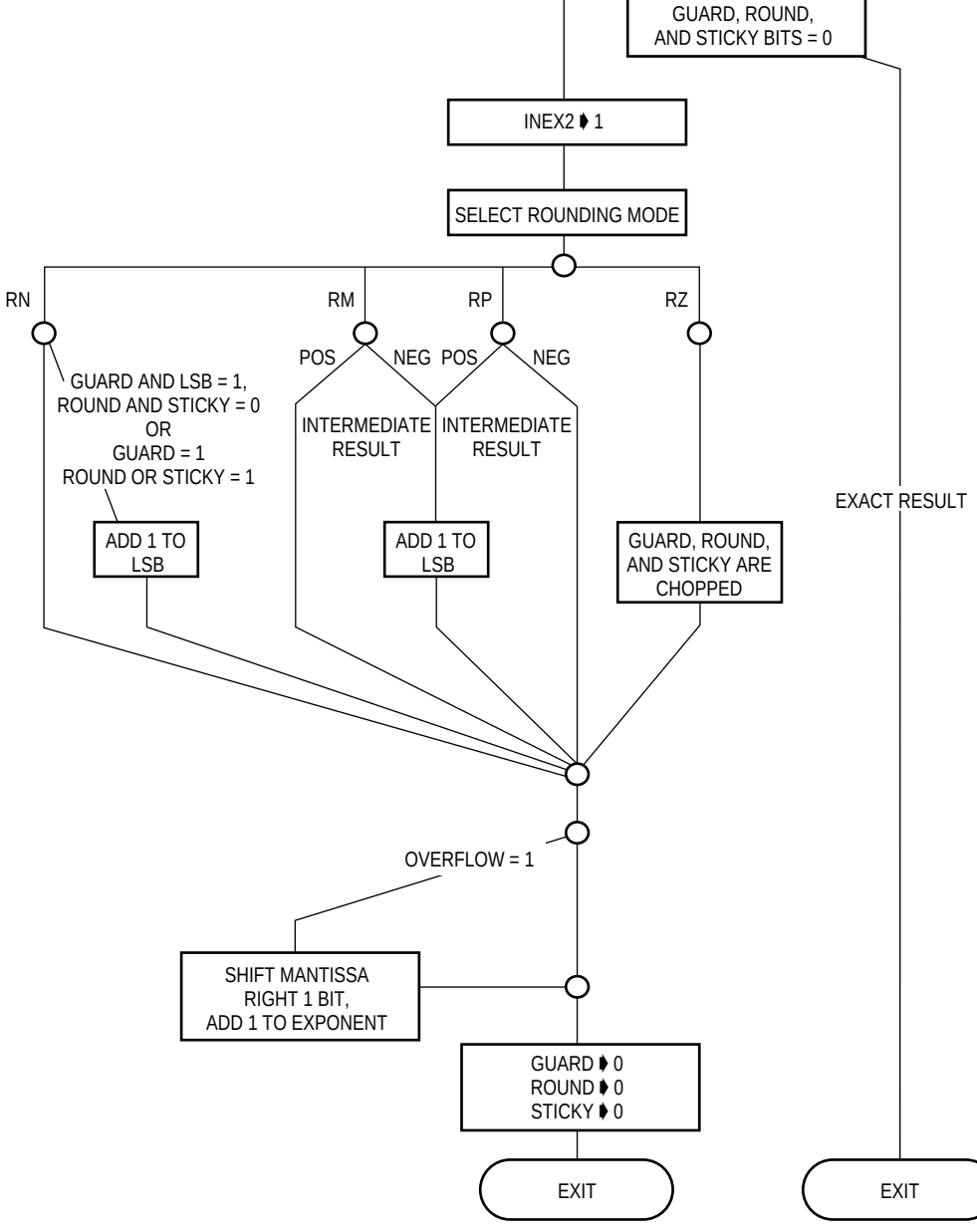


Figure 3-2. Rounding Algorithm Flowchart

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

Rounded-to-Nearest	x	xxx…x00	0	0	0

The LSB of the rounded result does not increment though the guard bit is s
intermediate result. The IEEE 754 standard specifies that tie cases should be hand
manner. If the destination data format is extended and there is a difference bet
infinitely precise intermediate result and the round-to-nearest result, the relative e
is 2 – 64 (the value of the guard bit). This error is equal to half of the least signif
value and is the worst case error that can be introduced when using the RN mode.
term one-half unit in the last place correctly identifies the error bound for this opera
error specification is the relative error present in the result; the absolute error boun
to 2exponent x 2 – 64. The following example shows the error bound for the other
modes:

Result	Integer	63-Bit Fraction	Guard	Round	Stic
Intermediate	x	xxx…x00	1	1	1
Rounded-to-Nearest	x	xxx…x00	0	0	0

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

3.6 FLOATING-POINT POSTPROCESSING

Most operations end with a postprocessing step. The FPU provides two
postprocessing. First, the condition code bits in the FPSR are set or cleared at t
each arithmetic operation or move operation to a single floating-point data reg
condition code bits are consistently set based on the result of the operation. Se
FPU supports 32 conditional tests that allow floating-point conditional instructio
floating-point conditions in exactly the same way as the integer conditional instruc
the integer condition code The combination of consistently set condition code bit
simple programming of conditional instructions gives the processor a very flexi
performance method of altering program flow based on floating-point results. Whil
the summary for each instruction, it should be assumed that an instruction
postprocessing unless the summary specifically states that the instruction does n
The following paragraphs describe postprocessing in detail.

precision. Also, the operation can generate a larger exponent or more bits of precis
can be represented in the chosen rounding precision. For these reasons, every ar
instruction ends by rounding the result and checking for overflow and underflow.

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

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

3.6.2 Conditional Testing

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

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

+ 0	0	1	0	0
– 0	1	1	0	0
+ Infinity	0	0	1	0
– Infinity	1	0	1	0
+ NAN	0	0	0	1
– NAN	1	0	0	1

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

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

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

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

conditions. Since the ordered or unordered attribute is explicitly included in the co
test, the BSUN bit is not set in the FPSR EXC byte when the unordered condition
Table 3-23 summarizes the conditional mnemonics, definitions, equations, predica
whether the BSUN bit is set in the FPSR EXC byte for the 32 floating-point condition
The equation column lists the combination of FPCC bits for each test in the for
equation. All condition codes with an overbar indicate cleared bits; all other bits are

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

amalgamation of the various parts that make up an instruction description. Ins
descriptions for the integer unit differ slightly from those for the floating-point unit; i
are no operation tables included for integer unit instruction descriptions.

The size attribute line specifies the size of the operands of an instruction. W
instruction uses operands of more than one size, the mnemonic of the instruction
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 co
words, and any other words that are always part of the instruction. The effective
extensions are not explicitly illustrated. The extension words, if any, follow immediat
the illustrated portions of the instructions.

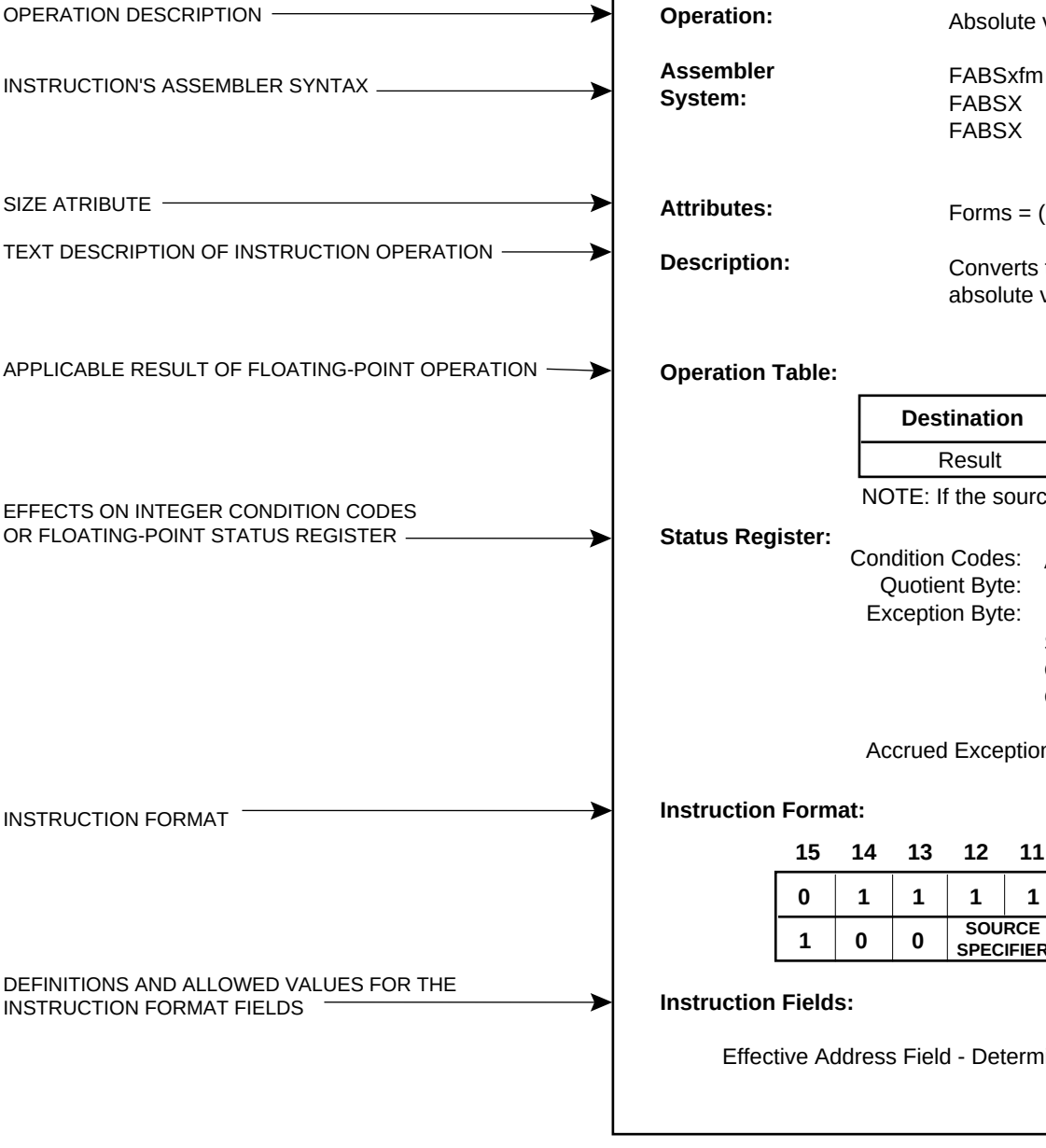


Figure 3-3. Instruction Description Format

SECTION 4
INTEGER INSTRUCTIONS

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

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

Test Bit Field and Change
(MC68030, MC68040)

The MC68HC000 is identical to the MC68000 except for power dissipation; ther instructions that apply to the MC68000 also apply to the MC68HC000. All referenc MC68000, MC68020, and MC68030 include references to the corresponding e controllers, MC68EC000, MC68EC020, and MC68EC030. All references to the I include the MC68LC040 and MC68EC040. This referencing applies throughout th unless otherwise specified.

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

MC68020, MC68030, and MC68040 only

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

**Can be used with CPU32 processor

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

Syntax: ABCD – (Ay), – (Ax)

Attributes: Size = (Byte)

Description: Adds the source operand to the destination operand along with the ex
and stores the result in the destination location. The addition is performed usin
coded decimal arithmetic. The operands, which are packed binary-coded
numbers, can be addressed in two different ways:

1. Data Register to Data Register: The operands are contained in the dat
ters specified in the instruction.
2. Memory to Memory: The operands are addressed with the predecreme
dressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Condition Codes:

X	N	Z	V	C
*	U	*	U	*

X — Set the same as the carry bit.
N — Undefined.
Z — Cleared if the result is nonzero; unchanged otherwise.
V — Undefined.
C — Set if a decimal carry was generated; cleared otherwise.

NOTE

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

1	1	0	0	REGISTER Rx	1	0	0	0	0	R/M	REGIS

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

R/M field—Specifies the operand addressing mode.
0 — The operation is data register to data register.
1 — The operation is memory to memory.

Register Ry field—Specifies the source register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing

Syntax: ADD Dn, < ea >

Attributes: Size = (Byte, Word, Long)

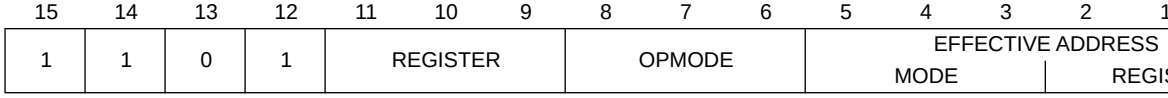
Description: Adds the source operand to the destination operand using binary addi
stores the result in the destination location. The size of the operation may be s
as byte, word, or long. The mode of the instruction indicates which operan
source and which is the destination, as well as the operand size.

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:



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
as listed in the following tables:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	000	reg. number:Dn	(xxx).W	111	
An*	001	reg. number:An	(xxx).L	111	
(An)	010	reg. number:An	#<data>	111	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	

MC68020, MC68030, and MC68040 only

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

*Word and long only
**Can be used with CPU32.

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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<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.

Syntax: ADDA < ea > , An

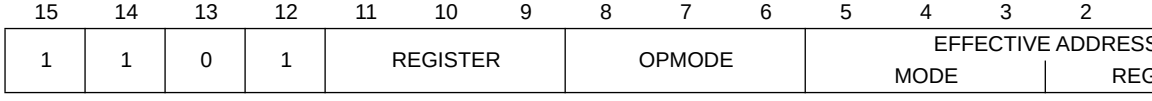
Attributes: Size = (Word, Long)

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

Condition Codes:

Not affected.

Instruction Format:



Instruction Fields:

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

Opmode field—Specifies the size of the operation.
011— Word operation; the source operand is sign-extended to a long ope
the operation is performed on the address register using all 32 bits.
111— Long operation.

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An	001	reg. number:An	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	111	1
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	0
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	0

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	111	0
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	0
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	0

*Can be used with CPU32

Syntax: ADDI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

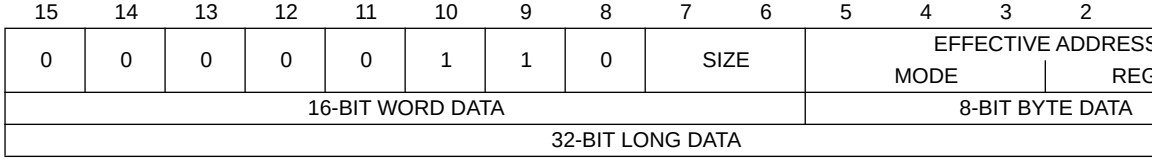
Description: Adds the immediate data to the destination operand and stores the
the destination location. The size of the operation may be specified as byte
long. The size of the immediate data matches the operation size.

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:



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

Effective Address field—Specifies the destination operand. Only data a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<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.

Syntax: ADDQ # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

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

Instruction Format:

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

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 ad
modes can be used as listed in the following tables:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An	001	reg. number:An	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	—	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

(bd,An,Xn**	110	reg. number:An	(bd,PC,Xn)†	—	
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	

*Word and long only.
**Can be used with CPU32.

Syntax: ADDX – (Ay), – (Ax)

Attributes: Size = (Byte, Word, Long)

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

1. Data register to data register—The data registers specified in the inst
contain the operands.
2. Memory to memory—The address registers specified in the instruction
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.

1	1	0	1	REGISTER Rx	1	SIZE	0	0	R/M	REGIST

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 r

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 r

Syntax: AND Dn, < ea >

Attributes: Size = (Byte, Word, Long)

Description: Performs an AND operation of the source operand with the de
operand and stores the result in the destination location. The size of the oper
be specified as byte, word, or long. The contents of an address register ma
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	1	0	0	REGISTER	OPMODE	EFFECTIVE ADDRESS MODE | REG

Instruction Fields:

Register field—Specifies any of the eight data registers.

Opmode field

Byte	Word	Long	Operation
000	001	010	< ea > Λ Dn $\rightarrow$ Dn
100	101	110	Dn Λ < ea > $\rightarrow$ < ea >

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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	—	—	(xxx).W	111	
An	—	—	(xxx).L	111	
(An)	010	reg. number:An	#<data>	—	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	—	
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	

*Can be used with CPU32.

NOTE

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.

Syntax: ANDI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Performs an AND operation of the immediate data with the de operand and stores the result in the destination location. The size of the opera be specified as byte, word, or long. The size of the immediate data mat 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	1	0	SIZE		EFFECTIVE ADDRESS				
										MODE			REGI	
16-BIT WORD DATA										8-BIT BYTE DATA				
32-BIT LONG DATA														

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
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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	—	
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	

*Can be used with CPU32

Immediate field—Data immediately following the instruction.
If size = 00, the data is the low-order byte of the immediate word.
If size = 01, the data is the entire immediate word.
If size = 10, the data is the next two immediate words.

Assembler Syntax: ANDI # < data > ,CCR

Attributes: Size = (Byte)

Description: Performs an AND operation of the immediate operand with the c 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	1	0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	8-BIT BYTE DATA						

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

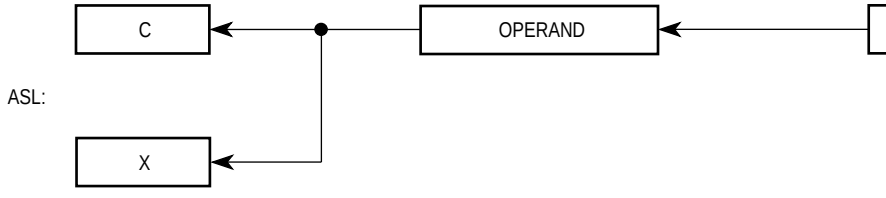
Attributes: Size = (Byte, Word, Long)

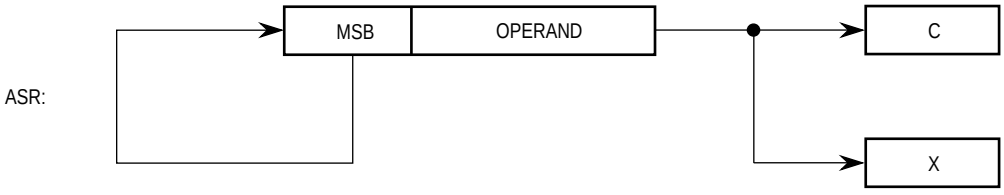
Description: Arithmetically shifts the bits of the operand in the direction (L or R)
The carry bit receives the last bit shifted out of the operand. The shift cou
shifting of a register may be specified in two different ways:

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

The size of the operation can be specified as byte, word, or long. An operand
ory 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 sh
Bits shifted out of the high-order bit go to both the carry and the extend bits;
shifted into the low-order bit. The overflow bit indicates if any sign changes o
ing the shift.





Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set according to the last bit shifted out of the operand; unaffected fo
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 op
cleared otherwise.
C — Set according to the last bit shifted out of the operand; cleared for a sh
of zero.

Instruction Format:

REGISTER SHIFTS

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

Instruction Fields:

Count/Register field—Specifies shift count or register that contains the shift co
If i/r = 0, this field contains the shift count. The values 1 – 7 represent count
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 (mod

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	1	1	0	0	0	0	dr	1	1	EFFECTIVE ADDRESS			
										MODE			REG

Instruction Fields:

dr field—Specifies the direction of the shift.
0 — Shift right
1 — Shift left

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	—	—	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(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.

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 loca
+ displacement. The program counter contains the address of the instructio
the Bcc instruction plus two. The displacement is a twos-complement int
represents the relative distance in bytes from the current program count
destination program counter. If the 8-bit displacement field in the instructio
zero, a 16-bit displacement (the word immediately following the instruction) i
the 8-bit displacement field in the instruction word is all ones ($FF), t
displacement (long word immediately following the instruction) is used. Condi
cc specifies one of the following conditional tests (refer to Table 3-19
information on these conditional tests):

Mnemonic	Condition
CC(HI)	Carry Clear
CS(LO)	Carry Set
EQ	Equal
GE	Greater or Equal
GT	Greater Than
HI	High
LE	Less or Equal

Mnemonic	Condition
LS	Low or Same
LT	Less Than
MI	Minus
NE	Not Equal
PL	Plus
VC	Overflow Clear
VS	Overflow Set

Condition Codes:

Not affected.

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—Twos complement integer specifying the number between the branch instruction and the next instruction to be execute condition is met.

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

32-Bit Displacement field—Used for the displacement when the 8-bit displ 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).

Assembler Syntax: BCHG Dn, < ea >
BCHG # < data > , < ea >

Attributes: Size = (Byte, Long)

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

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

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

X — Not affected.
N — Not affected.
Z — Set if the bit tested is zero; cleared otherwise.
V — Not affected.
C — Not affected.

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

Instruction Fields:

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

Effective Address field—Specifies the destination location. Only data a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

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

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

*Long only; all others are byte only.
**Can be used with CPU32.

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

Instruction Fields:

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

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<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,
([bd,PC,Xn],od)	—	([bd,A
([bd,PC],Xn,od)	—	([bd,A

*Long only; all others are byte only.
**Can be used with CPU32.

Bit Number field—Specifies the bit number.

Assembler Syntax: BCLR Dn, < ea >
BCLR # < data > , < ea >

Attributes: Size = (Byte, Long)

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

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

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

X — Not affected.
N — Not affected.
Z — Set if the bit tested is zero; cleared otherwise.
V — Not affected.
C — Not affected.

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

Instruction Fields:

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

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

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

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

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

*Long only; all others are byte only.
**Can be used with CPU32.

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

Instruction Fields:

Effective Address field—Specifies the destination location. Only data a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

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

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

*Long only; all others are byte only.
**Can be used with CPU32.

Bit Number field—Specifies the bit number.

Syntax: BFCHG < ea > {offset:width}

Attributes: Unsized

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

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

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the field is set; cleared otherwise.
Z — Set if all bits of the field are zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

NOTE

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

direct or control alterable addressing modes can be used as listed in the f
table:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn],od)	—	
([bd,PC],Xn,od)	—	

Do field—Determines how the field offset is specified.
0 — The offset field contains the bit field offset.
1 — Bits 8 – 6 of the extension word specify a data register that contains th
bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.
If Do = 0, the offset field is an immediate operand; the operand value is in th
0 – 31.
If Do = 1, the offset field specifies a data register that contains the offset. Th
is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.
0 — The width field contains the bit field width.
1 — Bits 2 – 0 of the extension word specify a data register that contains th
bits 3 – 4 are zero.

Width field—Specifies the field width, depending on Dw.
If Dw = 0, the width field is an immediate operand; an operand value in the
– 31 specifies a field width of 1 – 31, and a value of zero specifies a widt
If Dw = 1, the width field specifies a data register that contains the width. Th
is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value
specifies a width of 32.

Syntax: BFCLR < ea > {offset:width}

Attributes: Unsized

Description: Sets condition codes according to the value in a bit field at the effective address and clears the field.

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

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the field is set; cleared otherwise.
Z — Set if all bits of the field are zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

direct or control alterable addressing modes can be used as listed in the f
table:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn],od)	—	
([bd,PC],Xn,od)	—	

Do field—Determines how the field offset is specified.
0 — The offset field contains the bit field offset.
1 — Bits 8 – 6 of the extension word specify a data register that contains th
bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.
If Do = 0, the offset field is an immediate operand; the operand value is in th
of 0 – 31.
If Do = 1, the offset field specifies a data register that contains the offset. Th
is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.
0 — The width field contains the bit field width.
1 — Bits 2 – 0 of the extension word specify a data register that contains th
bits 3 – 4 are zero.

Width field—Specifies the field width, depending on Dw.
If Dw = 0, the width field is an immediate operand; operand values in the ra
– 31 specify a field width of 1 – 31, and a value of zero specifies a width
If Dw = 1, the width field specifies a data register that contains the width. Th
is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value
specifies a width of 32.

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

Attributes: Unsized

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

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the field is set; cleared otherwise.
Z — Set if all bits of the field are zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

direct or control addressing modes can be used as listed in the following

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0
(bd,PC,Xn)	111	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

Register field—Specifies the destination register.

Do field—Determines how the field offset is specified.
0 — The offset field contains the bit field offset.
1 — Bits 8 – 6 of the extension word specify a data register that contains th
bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.
If Do = 0, the offset field is an immediate operand; the operand value is in th
of 0 – 31.
If Do = 1, the offset field specifies a data register that contains the offset. Th
is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.
0 — The width field contains the bit field width.
1 — Bits 2 – 0 of the extension word specify a data register that contains th
bits 4 – 3 are zero.

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

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

Attributes: Unsized

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

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the source field is set; cleared otherwis
Z — Set if all bits of the field are zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

direct or control addressing modes can be used as listed in the followin

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

Register field—Specifies the destination data register.

Do field—Determines how the field offset is specified.
0 — The offset field contains the bit field offset.
1 — Bits 8 – 6 of the extension word specify a data register that contains t
bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.
If Do = 0, the offset field is an immediate operand; the operand value is in t
of 0 – 31.
If Do = 1, the offset field specifies a data register that contains the offset. T
is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.
0 — The width field contains the bit field width.
1 — Bits 2 – 0 of the extension word specify a data register that contains
bits 4 – 3 are zero.

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

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

Attributes: Unsized

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

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the field is set; cleared otherwise.
Z — Set if all bits of the field are zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

direct or control addressing modes can be used as listed in the following

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0
(bd,PC,Xn)	111	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

Register field—Specifies the destination data register operand.

Do field—Determines how the field offset is specified.
0 — The offset field contains the bit field offset.
1 — Bits 8 – 6 of the extension word specify a data register that contains th
bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.
If Do = 0, the offset field is an immediate operand; the operand value is in th
of 0 – 31.
If Do = 1, the offset field specifies a data register that contains the offset. Th
is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.
0 — The width field contains the bit field width.
1 — Bits 2 – 0 of the extension word specify a data register that contains th
bits 4 – 3 are zero.

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

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

Attributes: Unsized

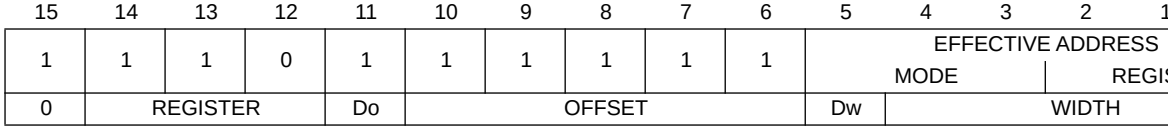
Description: Inserts a bit field taken from the low-order bits of the specified data into a bit field at the effective address location. The instruction sets the conditio according to the inserted value. The field offset and field width select the field. T offset specifies the starting bit of the field. The field width determines the numb in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the field is set; cleared otherwise.
Z — Set if all bits of the field are zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:



direct or control alterable addressing modes can be used as listed in the table:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn],od)	—	
([bd,PC],Xn,od)	—	

Register field—Specifies the source data register operand.

Do field—Determines how the field offset is specified.
0 — The offset field contains the bit field offset.
1 — Bits 8 – 6 of the extension word specify a data register that contains t bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.
If Do = 0, the offset field is an immediate operand; the operand value is in t of 0 – 31.
If Do = 1, the offset field specifies a data register that contains the offset. T is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.
0 — The width field contains the bit field width.
1 — Bits 2 – 0 of the extension word specify a data register that contains bits 4 – 3 are zero.

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

Syntax: BFSET < ea > {offset:width}

Attributes: Unsized

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

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

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the field is set; cleared otherwise.
Z — Set if all bits of the field are zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

direct or control alterable addressing modes can be used as listed in the f
table:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn],od)	—	
([bd,PC],Xn,od)	—	

Do field—Determines how the field offset is specified.
0 — The offset field contains the bit field offset.
1 — Bits 8 – 6 of the extension word specify a data register that contains th
bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.
If Do = 0, the offset field is an immediate operand; the operand value is in th
of 0 – 31.
If Do = 1, the offset field specifies a data register that contains the offset. Th
is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.
0 — The width field contains the bit field width.
1 — Bits 2 – 0 of the extension word specify a data register that contains th
bits 4 – 3 are zero.

Width field—Specifies the field width, depending on Dw.
If Dw = 0, the width field is an immediate operand; operand values in the ra
– 31 specify a field width of 1 – 31, and a value of zero specifies a width
If Dw = 1, the width field specifies a data register that contains the width. Th
is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value
specifies a width of 32.

Syntax: BFTST < ea > {offset:width}

Attributes: Unsized

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

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the field is set; cleared otherwise.
Z — Set if all bits of the field are zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

direct or control addressing modes can be used as listed in the following

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0
(bd,PC,Xn)	111	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

Do field—Determines how the field offset is specified.
0 — The offset field contains the bit field offset.
1 — Bits 8 – 6 of the extension word specify a data register that contains th
bits 10 – 9 are zero.

Offset field—Specifies the field offset, depending on Do.
If Do = 0, the offset field is an immediate operand; the operand value is in th
of 0 – 31.
If Do = 1, the offset field specifies a data register that contains the offset. Th
is in the range of -2^{31} to $2^{31} - 1$.

Dw field—Determines how the field width is specified.
0 — The width field contains the bit field width.
1 — Bits 2 – 0 of the extension word specify a data register that contains th
bits 4 – 3 are zero.

Width field—Specifies the field width, depending on Dw.
If Dw = 0, the width field is an immediate operand, operand values in the ra
– 31 specify a field width of 1 – 31, and a value of zero specifies a width
If Dw = 1, the width field specifies a data register that contains the width. Th
is modulo 32; values of 1 – 31 specify field widths of 1 – 31, and a value
specifies a width of 32.

Syntax: BKPT # < data >

Attributes: Unsized

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

For the MC68020, MC68030, and CPU32, a breakpoint acknowledge bus cyc cuted with the immediate data (value 0 – 7) on bits 2 – 4 of the address bus a on bits 0 and 1 of the address bus. The breakpoint acknowledge bus cycle the CPU space, addressing type 0, and provides the breakpoint number sp the instruction on address lines A2 – A4. If the external hardware terminates with $\overline{\text{DSACK}}$x or $\overline{\text{STERM}}$, the data on the bus (an instruction word) is inserte instruction pipe and is executed after the breakpoint instruction. The breakpoi tion requires a word to be transferred so, if the first bus cycle accesses an 8 a second bus cycle is required. If the external logic terminates the breakpoint edge bus cycle with $\overline{\text{BERR}}$ (i.e., no instruction word available), the processor illegal instruction exception.

For the MC68040, this instruction executes a breakpoint acknowledge b Regardless of the cycle termination, the MC68040 takes an illegal instructio tion.

For more information on the breakpoint instruction refer to the appropriate us ual on bus operation.

This instruction supports breakpoints for debug monitors and real- time hardw lators.

Not affected.

Instruction Format:

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

Instruction Field:

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

Syntax: BRA < label >

Attributes: Size = (Byte, Word, Long*)

*(MC68020, MC68030, MC68040 only)

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

Condition Codes:

Not affected.

Instruction Format:

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

Instruction Fields:

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

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

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

NOTE

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

Assembler Syntax: BSET Dn, < ea >
BSET # < data > , < ea >

Attributes: Size = (Byte, Long)

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

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

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

X — Not affected.
N — Not affected.
Z — Set if the bit tested is zero; cleared otherwise.
V — Not affected.
C — Not affected.

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

Instruction Fields:

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

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

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

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

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

*Long only; all others are byte only.
**Can be used with CPU32.

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

Instruction Fields:

Effective Address field—Specifies the destination location. Only data a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

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

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

*Long only; all others are byte only.
**Can be used with CPU32.

Bit Number field—Specifies the bit number.

Syntax: BSR < label >

Attributes: Size = (Byte, Word, Long*)

*(MC68020, MC68030, MC68040 only)

Description: Pushes the long-word address of the instruction immediately follo
BSR instruction onto the system stack. The program counter contains the a
the instruction word plus two. Program execution then continues at locatio
displacement. The displacement is a twos complement integer that repres
relative distance in bytes from the current program counter to the destination
counter. If the 8-bit displacement field in the instruction word is zero,
displacement (the word immediately following the instruction) is used. If
displacement field in the instruction word is all ones ($FF), the 32-bit disp
(long word immediately following the instruction) is used.

Condition Codes:

Not affected.

Instruction Format:

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

between the branch instruction and the next instruction to be executed.

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

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

NOTE

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

Syntax: BTST # < data > , < ea >

Attributes: Size = (Byte, Long)

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

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

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

X — Not affected.
N — Not affected.
Z — Set if the bit tested is zero; cleared otherwise.
V — Not affected.
C — Not affected.

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

Instruction Fields:

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

Effective Address field—Specifies the destination location. Only data ad 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*Long only; all others are byte only.
**Can be used with CPU32.

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

Instruction Fields:

Effective Address field—Specifies the destination location. Only data ad
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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	

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	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Can be used with CPU32.

Bit Number field—Specifies the bit number.

Assembler Syntax: CALLM # < data > , < ea >

Attributes: Unsized

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

Condition Codes:

Not affected.

Instruction Format:

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

addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

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

If Z, Update Operand → Destination
Else Destination → Compare Operand
CAS2 Destination 1 – Compare 1 → cc;
If Z, Destination 2 – Compare 2 → cc
If Z, Update 1 → Destination 1; Update 2 → Destination 2
Else Destination 1 → Compare 1; Destination 2 → Compa

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

Attributes: Size = (Byte*, Word, Long)

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

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

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

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

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

*. CAS2 cannot use byte operands.

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

Instruction Fields:

Size field—Specifies the size of the operation.
- 01 — Byte operation
- 10 — Word operation
- 11 — Long operation

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

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

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn],od)	—	
([bd,PC],Xn,od)	—	

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

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

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

Instruction Fields:

Size field—Specifies the size of the operation.
- 10 — Word operation
- 11 — Long operation

D/A1, D/A2 fields—Specify whether Rn1 and Rn2 reference data or address r respectively.
- 0 — The corresponding register is a data register.
- 1 — The corresponding register is an address register.

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

Du1, Du2 fields—Specify the data registers that contain the update values to b to the first and second memory operand locations if the comparison is suc

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

NOTE

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

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

Assembler Syntax: CHK < ea > ,Dn

Attributes: Size = (Word, Long*)

*(MC68020, MC68030, MC68040 only)

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

Condition Codes:

X	N	Z	V	C
—	*	U	U	U

X — Not affected.
N — Set if Dn < 0; cleared if Dn > effective address operand; undefined oth
Z — Undefined.
V — Undefined.
C — Undefined.

Instruction Format:

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

Size field—Specifies the size of the operation.
11— Word operation
10— Long operation

Effective Address field—Specifies the upper bound operand. Only data ad
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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Assembler Syntax: CHK2 < ea > ,Rn

Attributes: Size = (Byte, Word, Long)

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

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

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

Condition Codes:

X	N	Z	V	C
—	U	*	U	*

X — Not affected.
N — Undefined.
Z — Set if Rn is equal to either bound; cleared otherwise.
V — Undefined.
C — Set if Rn is out of bounds; cleared otherwise.

0	0	0	0	0	SIZE		0	1	1	EFFECTIVE ADDRESS				
										MODE			REGIS	
D/A	REGISTER			1	0	0	0	0	0	0	0	0	0	0

Instruction Fields:

Size field—Specifies the size of the operation.
- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

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

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0
(bd,PC,Xn)	111	0

MC68020, MC68030, and MC68040 only

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

([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

D/A field—Specifies whether an address register or data register is to be chec
- 0 — Data register
- 1 — Address register

Register field—Specifies the address or data register that contains the valu checked.

Syntax: CLR < ea >

Attributes: Size = (Byte, Word, Long)

Description: Clears the destination operand to zero. The size of the operation specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	0	1	0	0

X — Not affected.
N — Always cleared.
Z — Always set.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

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

Effective Address field—Specifies the destination location. Only data a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

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

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

*Can be used with CPU32.

NOTE

In the MC68000 and MC68008 a memory location is read before it is cleared.

Syntax: CMP < ea > , Dn

Attributes: Size = (Byte, Word, Long)

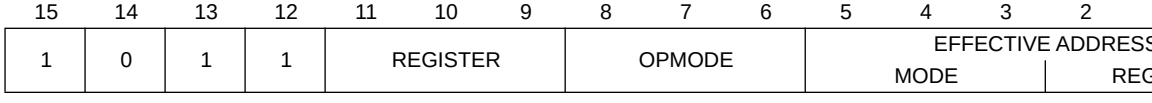
Description: Subtracts the source operand from the destination data register and
condition codes according to the result; the data register is not changed. Th
the operation can be byte, word, or long.

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:



Instruction Fields:

Register field—Specifies the destination data register.

Opmode field

Byte	Word	Long	Operation
000	001	010	Dn – < ea >

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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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.

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.

Syntax: CMPA < ea > , An

Attributes: Size = (Word, Long)

Description: Subtracts the source operand from the destination address register
the condition codes according to the result; the address register is not char
size of the operation can be specified as word or long. Word length source
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	1	REGISTER	OPMODE	EFFECTIVE ADDRESS: MODE	REG

Opmode field—Specifies the size of the operation.

011— Word operation; the source operand is sign-extended to a long opera
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
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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Syntax: CMPI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

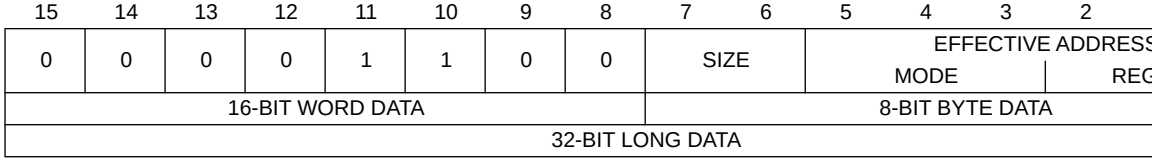
Description: Subtracts the immediate data from the destination operand and
condition codes according to the result; the destination location is not chan
size of the operation may be specified as byte, word, or long. The size of the i
data matches the operation size.

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:



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

Effective Address field—Specifies the destination operand. Only data ad
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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)*	111	0
(d_8,PC,Xn)*	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*PC relative addressing modes do not apply to MC68000, MC680008, 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.

Syntax: CMPM (Ay) + ,(Ax) +

Attributes: Size = (Byte, Word, Long)

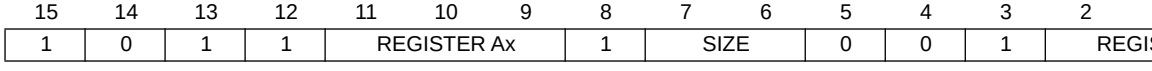
Description: Subtracts the source operand from the destination operand and condition codes according to the results; the destination location is not char operands are always addressed with the postincrement addressing mode, address registers specified in the instruction. The size of the operation specified 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:



Instruction Fields:

Register Ax field—(always the destination) Specifies an address regist 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 registe postincrement addressing mode.

Syntax: CMP2 < ea > ,Rn

Attributes: Size = (Byte, Word, Long)

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

The size of the data and the bounds can be specified as byte, word, or long. I
data register and the operation size is byte or word, only the appropriate low-o
of Rn is checked. If Rn is an address register and the operation size is byte
the bounds operands are sign-extended to 32 bits, and the resultant opera
compared to the full 32 bits of An.

If the upper bound equals the lower bound, the valid range is a single value.

NOTE

This instruction is identical to CHK2 except that it sets condition codes rather than taking an exception when the value in Rn is out of bounds.

Condition Codes:

X	N	Z	V	C
—	U	*	U	*

X — Not affected.
N — Undefined.
Z — Set if Rn is equal to either bound; cleared otherwise.
V — Undefined.
C — Set if Rn is out of bounds; cleared otherwise.

0	0	0	0	0	SIZE		0	1	1	EFFECTIVE ADDRESS				
										MODE			REG	
D/A	REGISTER			0	0	0	0	0	0	0	0	0	0	

Instruction Fields:

Size field—Specifies the size of the operation.
- 00 — Byte operation
- 01 — Word operation
- 10 — Long operation

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

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	

MC68020, MC68030, and MC68040 only

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

([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

D/A field—Specifies whether an address register or data register is compare
- 0 — Data register
- 1 — Address register

Register field—Specifies the address or data register that contains the va checked.

Assembler Syntax: cpBcc < label >

Attributes: Size = (Word, Long)

Description: If the specified coprocessor condition is true, program execution cont
location scan PC + displacement. The value of the scan PC is the address of
displacement word. The displacement is a twos complement integer that rep
the relative distance in bytes from the scan PC to the destination program cour
displacement can be either 16 or 32 bits. The coprocessor determines the
condition from the condition field in the operation word.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1
1	1	1	1	COPROCESSOR ID	0	1	SIZE	COPROCESSOR CONDITIO
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS								
WORD OR								
LONG-WORD DISPLACEMENT								

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation. Coprocess
000 results in an F-line exception for the MC68030.

Size field—Specifies the size of the displacement.
0 — The displacement is 16 bits.
1 — The displacement is 32 bits.

Coprocessor Condition field—Specifies the coprocessor condition to be test
field is passed to the coprocessor, which provides directives to th
processor for processing this instruction.

16-Bit Displacement field—The displacement value occupies 16 bits.

32-Bit Displacement field—The displacement value occupies 32 bits.

Assembler Syntax: cpDBcc Dn, < label >

Attributes: Size = (Word)

Description: If the specified coprocessor condition is true, execution continues next instruction. Otherwise, the low-order word in the specified data r decremented by one. If the result is equal to – 1, execution continues with instruction. If the result is not equal to – 1, execution continues at the location by the value of the scan PC plus the sign-extended 16-bit displacement. The the scan PC is the address of the displacement word. The displacement complement integer that represents the relative distance in bytes from the so the destination program counter. The coprocessor determines the specific from the condition word that follows the operation word.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	1	0	0	1	REG
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDIT			
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS													
16-BIT DISPLACEMENT													

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation; coproces 000 results in an F-line exception for the MC68030.

Register field—Specifies the data register used as the counter.

Coprocessor Condition field—Specifies the coprocessor condition to be tes field is passed to the coprocessor, which provides directives to processor for processing this instruction.

Displacement field—Specifies the distance of the branch (in bytes).

Syntax: cpGEN < parameters as defined by coprocessor >

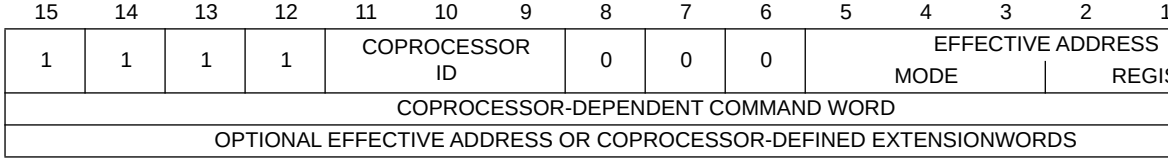
Attributes: Unsized

Description: Transfers the command word that follows the operation word to the s
coprocessor. The coprocessor determines the specific operation from the co
word. Usually a coprocessor defines specific instances of this instruction to pr
instruction set.

Condition Codes:

May be modified by coprocessor; unchanged otherwise.

Instruction Format:



Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation; no
coprocessor ID of 000 is reserved for MMU instructions for the MC6803

Effective Address field—Specifies the location of any operand not reside
coprocessor. The allowable addressing modes are determined by the o
to be performed.

Coprocessor Command field—Specifies the coprocessor operation to be pe
This word is passed to the coprocessor, which in turn provides directive
main processor for processing this instruction.

Assembler Syntax: cpScc < ea >

Attributes: Size = (Byte)

Description: Tests the specified coprocessor condition code. If the condition is
byte specified by the effective address is set to TRUE (all ones); otherwise, th
set to FALSE (all zeros). The coprocessor determines the specific condition
condition word that follows the operation word.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	1	EFFECTIVE ADDRESS			
										MODE			REG
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDIT			
OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR-DEFINED EXTENSIONWORDS													

000 results in an F-line exception for the MC68030.

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

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Coprocessor Condition field—Specifies the coprocessor condition to be test field is passed to the coprocessor, which in turn provides directives to t processor for processing this instruction.

Assembler Syntax: cpTRAPcc
cpTRAPcc # < data >

Attributes: Unsized or Size = (Word, Long)

Description: Tests the specified coprocessor condition code; if the selected cop
condition is true, the processor initiates a cpTRAPcc exception, vector numb
program counter value placed on the stack is the address of the next instruct
selected condition is not true, no operation is performed, and execution conti
the next instruction. The coprocessor determines the specific condition
condition word that follows the operation word. Following the condition word
defined data operand specified as immediate data to be used by the trap ha

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	1	1	1	1	OP
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDIT			
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS													
OPTIONAL WORD													
OR LONG-WORD OPERAND													

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation; coproces
000 results in an F-line exception for the MC68030.

Opmode field—Selects the instruction form.
010—Instruction is followed by one operand word.
011—Instruction is followed by two operand words.
100—Instruction has no following operand words.

Coprocessor Condition field—Specifies the coprocessor condition to be tes
field is passed to the coprocessor, which provides directives to
processor for processing this instruction.

Assembler Syntax: DBcc Dn, < label >

Attributes: Size = (Word)

Description: Controls a loop of instructions. The parameters are a condition code register (counter), and a displacement value. The instruction first tests the cond termination; if it is true, no operation is performed. If the termination conditio true, the low-order 16 bits of the counter data register decrement by one. If th is – 1, execution continues with the next instruction. If the result is not equa execution continues at the location indicated by the current value of the counter plus the sign-extended 16-bit displacement. The value in the program is the address of the instruction word of the DBcc instruction plus tw displacement is a twos complement integer that represents the relative dis bytes from the current program counter to the destination program counter. C code cc specifies one of the following conditional tests (refer to Table 3-19 f information on these conditional tests):

Mnemonic	Condition
CC(HI)	Carry Clear
CS(LO)	Carry Set
EQ	Equal
F	False
GE	Greater or Equal
GT	Greater Than
HI	High
LE	Less or Equal

Mnemonic	Condition
LS	Low or Same
LT	Less Than
MI	Minus
NE	Not Equal
PL	Plus
T	True
VC	Overflow Clear
VS	Overflow Set

Condition Codes:

Not affected.

0	1	0	1	CONDITION	1	1	0	0	1	REG
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.

Syntax:

*DIVS.L < ea > ,Dq	32/32 → 32q
*DIVS.L < ea > ,Dr:Dq	64/32 → 32r – 32q
*DIVSL.L < ea > ,Dr:Dq	32/32 → 32r – 32q

*Applies to MC68020, MC68030, MC68040, CPU32 only

Attributes: Size = (Word, Long)

Description: Divides the signed destination operand by the signed source opera
stores the signed result in the destination. The instruction uses one of four for
word form of the instruction divides a long word by a word. The result is a qu
the lower word (least significant 16 bits) and a remainder in the upper wor
significant 16 bits). The sign of the remainder is the same as the sign of the di

The first long form divides a long word by a long word. The result is a long quot
remainder is discarded.

The second long form divides a quad word (in any two data registers) by a lon
The result is a long-word quotient and a long-word remainder.

The third long form divides a long word by a long word. The result is a long-w
tient and a long-word remainder.

Two special conditions may arise during the operation:

1. Division by zero causes a trap.
2. Overflow may be detected and set before the instruction completes. If t
struction detects an overflow, it sets the overflow condition code, and th
ands are unaffected.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

X—Not affected.
N — Set if the quotient is negative; cleared otherwise; undefined if overflow
by zero occurs.
Z — Set if the quotient is zero; cleared otherwise; undefined if overflow or d
zero occurs.
V — Set if division overflow occurs; undefined if divide by zero occurs; clea
erwise.
C — Always cleared.

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

Instruction Fields:

Register field—Specifies any of the eight data registers. This field always spe
destination operand.

Effective Address field—Specifies the source operand. Only data alterable ad
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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	

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	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Can be used with CPU32.

NOTE

Overflow occurs if the quotient is larger than a 16-bit signed integer.

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

Instruction Fields:

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

MC68020, MC68030, and MC68040 only

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	111	1
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	0
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	0
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	0

MC68020, MC68030, and MC68040 only

([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	0
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	0

Register Dq field—Specifies a data register for the destination operand. The lo 32 bits of the dividend comes from this register, and the 32-bit quotient is into this register.

Size field—Selects a 32- or 64-bit division operation.
0 — 32-bit dividend is in register Dq.
1 — 64-bit dividend is in Dr – Dq.

dividend.

NOTE

Overflow occurs if the quotient is larger than a 32-bit signed integer.

Syntax:

*DIVU.L < ea > ,Dq	32/32 → 32q
*DIVU.L < ea > ,Dr:Dq	64/32 → 32r – 32q
*DIVUL.L < ea > ,Dr:Dq	32/32 → 32r – 32q

*Applies to MC68020, MC68030, MC68040, CPU32 only.

Attributes: Size = (Word, Long)

Description: Divides the unsigned destination operand by the unsigned operand and stores the unsigned result in the destination. The instruction use four forms. The word form of the instruction divides a long word by a word. Th is a quotient in the lower word (least significant 16 bits) and a remainder in th word (most significant 16 bits).

The first long form divides a long word by a long word. The result is a long quot remainder is discarded.

The second long form divides a quad word (in any two data registers) by a lon The result is a long-word quotient and a long-word remainder.

The third long form divides a long word by a long word. The result is a long-w tient and a long-word remainder.

Two special conditions may arise during the operation:

1. Division by zero causes a trap.
2. Overflow may be detected and set before the instruction completes. If t struction detects an overflow, it sets the overflow condition code, and th ands are unaffected.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

X — Not affected.
N — Set if the quotient is negative; cleared otherwise; undefined if overflow by zero occurs.
Z — Set if the quotient is zero; cleared otherwise; undefined if overflow or d zero occurs.
V — Set if division overflow occurs; cleared otherwise; undefined if divide occurs.
C — Always cleared.

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

Instruction Fields:

Register field—Specifies any of the eight data registers; this field always spe
destination operand.

Effective Address field—Specifies the source operand. Only data addressin
can be used as listed in the following tables:

MC68020, MC68030, and MC68040 only

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	000	reg. number:Dn	(xxx).W	111	
An	—	—	(xxx).L	111	
(An)	010	reg. number:An	#<data>	111	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	

MC68020, MC68030, and MC68040 only

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

**Can be used with CPU32.

NOTE

Overflow occurs if the quotient is larger than a 16-bit signed integer.

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

Instruction Fields:

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

MC68020, MC68030, and MC68040 only

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	111	1
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	0
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	0
(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	111	0

MC68020, MC68030, and MC68040 only

([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	0
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	0

Register Dq field—Specifies a data register for the destination operand. The lo 32 bits of the dividend comes from this register, and the 32-bit quotient is into this register.

Size field—Selects a 32- or 64-bit division operation.
0 — 32-bit dividend is in register Dq.
1 — 64-bit dividend is in Dr – Dq.

dividend.

NOTE

Overflow occurs if the quotient is larger than a 32-bit unsigned integer.

Syntax: EOR Dn, < ea >

Attributes: Size = (Byte, Word, Long)

Description: Performs an exclusive-OR operation on the destination operand u source operand and stores the result in the destination location. The siz operation may be specified to be byte, word, or long. The source operand m data register. The destination operand is specified in the effective address fiel

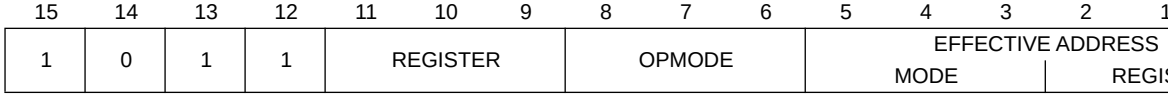
Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the result is set; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

WORD



Instruction Fields:

Register field—Specifies any of the eight data registers.

Opmode field

Byte	Word	Long	Operation
100	101	110	< ea > ⊕ Dn → < ea >

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	000	reg. number:Dn	(xxx).W	111	
An	—	—	(xxx).L	111	
(An)	010	reg. number:An	#<data>	—	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	—	
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	

*Can be used with CPU32.

NOTE

Memory-to-data-register operations are not allowed. Most assemblers use EORI when the source is immediate data.

Syntax: EORI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

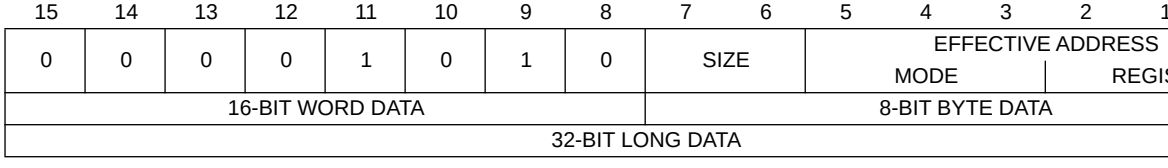
Description: Performs an exclusive-OR operation on the destination operand u
immediate data and the destination operand and stores the result in the de
location. The size of the operation may be specified as byte, word, or long. Th
the immediate data matches the operation size.

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:



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
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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

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

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

*Can be used with CPU32.

Immediate field—Data immediately following the instruction.
If size = 00, the data is the low-order byte of the immediate word.
If size = 01, the data is the entire immediate word.
If size = 10, the data is next two immediate words.

Operation: Source ⊕ CCR → CCR

Assembler Syntax: EORI # < data > ,CCR

Attributes: Size = (Byte)

Description: Performs an exclusive-OR operation on the condition code register u immediate operand and stores the result in the condition code register (low-or of the status register). All implemented bits of the condition code register are a

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Changed if bit 4 of immediate operand is one; unchanged otherwise.
N — Changed if bit 3 of immediate operand is one; unchanged otherwise.
Z — Changed if bit 2 of immediate operand is one; unchanged otherwise.
V — Changed if bit 1 of immediate operand is one; unchanged otherwise.
C — Changed if bit 0 of immediate operand is one; unchanged otherwise.

Instruction Format:

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

Syntax: EXG Ax,Ay EXG Dx,Ay

Attributes: Size = (Long)

Description: Exchanges the contents of two 32-bit registers. The instruction perfo
types of exchanges.

1. Exchange data registers.
2. Exchange address registers.
3. Exchange a data register and an address register.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	0	0	REGISTER Rx			1	OPMODE					REGI

Instruction Fields:

Register Rx field—Specifies either a data register or an address register depe
the mode. If the exchange is between data and address registers, this fie
specifies the data register.

Opmode field—Specifies the type of exchange.
01000—Data registers
01001—Address registers
10001—Data register and address register

Register Ry field—Specifies either a data register or an address register depe
the mode. If the exchange is between data and address registers, this fie
specifies the address register.

Syntax: EXT.L Dnextend word to long word
EXTB.L Dnextend byte to long word (MC68020, MC68030 MC68040, CPU32)

Attributes: Size = (Word, Long)

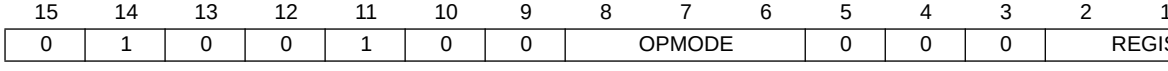
Description: Extends a byte in a data register to a word or a long word, or a word i register to a long word, by replicating the sign bit to the left. If the operation ex byte to a word, bit 7 of the designated data register is copied to bits 15 – 8 of t register. If the operation extends a word to a long word, bit 15 of the designa register is copied to bits 31 – 16 of the data register. The EXTB form copies bit designated register to bits 31 – 8 of the data register.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:



Instruction Fields:

Opmode field—Specifies the size of the sign-extension operation.
010— Sign-extend low-order byte of data register to word.
011— Sign-extend low-order word of data register to long.
111— Sign-extend low-order byte of data register to long.

Register field—Specifies the data register is to be sign-extended.

Illegal Instruction Vector Address → PC

*The MC68000 and MC68008 cannot write the vector offset and format code to the system stack.

Assembler Syntax: ILLEGAL

Attributes: Unsized

Description: Forces an illegal instruction exception, vector number 4. All oth instruction bit patterns are reserved for future extension of the instructior should not be used to force an exception.

Condition Codes:

Not affected.

Instruction Format:

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

Syntax: JMP < ea >

Attributes: Unsized

Description: Program execution continues at the effective address specified instruction. The addressing mode for the effective address must be a addressing mode.

Condition Codes:

Not affected.

Instruction Format:

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

Instruction Field:

Effective Address field—Specifies the address of the next instruction. Only 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Syntax: JSR < ea >

Attributes: Unsized

Description: Pushes the long-word address of the instruction immediately follo
JSR instruction onto the system stack. Program execution then continue
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
0	1	0	0	1	1	1	0	1	0	EFFECTIVE ADDRESS			
										MODE			REG

Instruction Field:

Effective Address field—Specifies the address of the next instruction. On
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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	

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	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Can be used with CPU32.

Syntax: LEA < ea > ,An

Attributes: Size = (Long)

Description: Loads the effective address into the specified address register. All 3
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	1	0	0	REGISTER			1	1	1	EFFECTIVE ADDRESS				
										MODE			REGI	

Instruction Fields:

Register field—Specifies the address register to be updated with the effective a

Effective Address field—Specifies the address to be loaded into the address
Only control addressing modes can be used as listed in the following tab

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	—	—	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	—	
(An) +	—	—			
– (An)	—	—			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	0
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	0

MC68020, MC68030, and MC68040 only

(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)*	111	0
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	0
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Syntax: LINK An, # < displacement >

Attributes: Size = (Word, Long*)

*MC68020, MC68030, MC68040 and CPU32 only.

Description: Pushes the contents of the specified address register onto the sta
loads the updated stack pointer into the address register. Finally, a
displacement value to the stack pointer. For word-size operation, the displac
the sign-extended word following the operation word. For long size opera
displacement is the long word following the operation word. The address
occupies one long word on the stack. The user should specify a negative disp
in order to allocate stack area.

Condition Codes:

Not affected.

Instruction Format:

WORD

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	1	1	1	0	0	1	0	1	0	REG
WORD DISPLACEMENT													

Instruction Format:

LONG

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	1	0	0	0	0	0	0	0	1	REG
HIGH-ORDER DISPLACEMENT													
LOW-ORDER DISPLACEMENT													

Displacement field—Specifies the twos complement integer to be added to t
pointer.

NOTE

LINK and UNLK can be used to maintain a linked list of local data and parameter areas on the stack for nested subroutine calls.

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

Attributes: Size = (Byte, Word, Long)

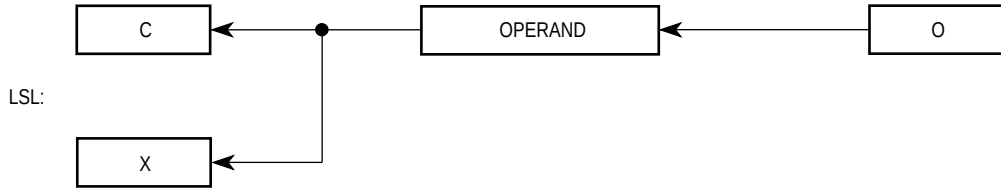
Description: Shifts the bits of the operand in the direction specified (L or R). The receives the last bit shifted out of the operand. The shift count for the shi register is specified in two different ways:

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 struction modulo 64.

The size of the operation for register destinations may be specified as byte long. The contents of memory, < ea > , can be shifted one bit only, and the size is restricted to a word.

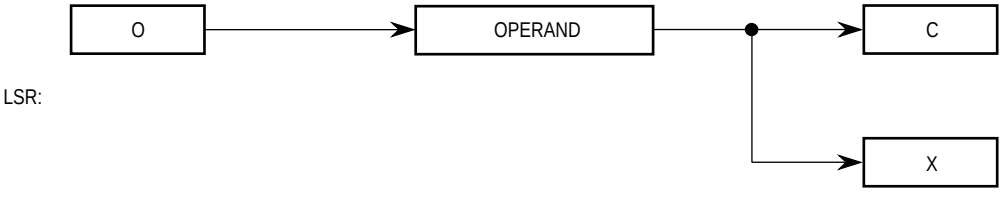
The LSL instruction shifts the operand to the left the number of positions sp the shift count. Bits shifted out of the high-order bit go to both the carry and th bits; zeros are shifted into the low-order bit.

.



The LSR instruction shifts the operand to the right the number of positions sp the shift count. Bits shifted out of the low-order bit go to both the carry and th bits; zeros are shifted into the high-order bit.

.



*	*	*	0	*

X — Set according to the last bit shifted out of the operand; unaffected fo
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 sh
of zero.

Instruction Format:

REGISTER SHIFTS

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

Instruction Fields:

Count/Register field

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

If i/r = 1, the data register specified in this field contains the shift count (mod

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.

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

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 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	Re
(xxx).W	111	
(xxx).L	111	
#<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.

Syntax: MOVE < ea > , < ea >

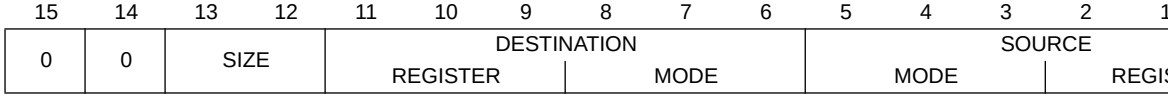
Attributes: Size = (Byte, Word, Long)

Description: Moves the data at the source to the destination location and sets the c
codes according to the data. The size of the operation may be specified as byt
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:



Instruction Fields:

Size field—Specifies the size of the operand to be moved.
01 — Byte operation
11 — Word operation
10 — Long operation

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	000	reg. number:Dn	(xxx).W	111	
An	—	—	(xxx).L	111	
(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.

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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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.

Syntax: MOVEA < ea > ,An

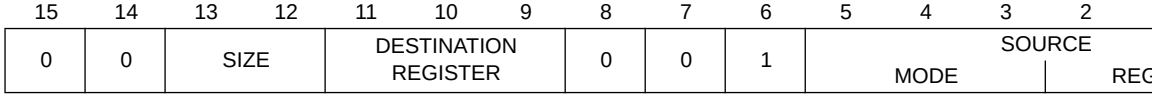
Attributes: Size = (Word, Long)

Description: Moves the contents of the source to the destination address register of the operation is specified as word or long. Word-size source operands extended to 32-bit quantities.

Condition Codes:

Not affected.

Instruction Format:



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 ope all 32 bits are loaded into the address register.

10 — Long operation.

Destination Register field—Specifies the destination address register.

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An	001	reg. number:An	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	111	1
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	0
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	0

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	111	0
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	0
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Operation: CCR → Destination

Assembler Syntax: MOVE CCR, < ea >

Attributes: Size = (Word)

Description: Moves the condition code bits (zero-extended to word size) to the d
location. The operand size is a word. Unimplemented bits are read as zeros.

Condition Codes:

Not affected.

Instruction Format:

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

Instruction Field:

Effective Address field—Specifies the destination location. Only data a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

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

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

*Can be used with CPU32.

NOTE

MOVE from CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

Assembler Syntax: MOVE < ea > ,CCR

Attributes: Size = (Word)

Description: Moves the low-order byte of the source operand to the condition code
The upper byte of the source operand is ignored; the upper byte of the statu
is not altered.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set to the value of bit 4 of the source operand.
N — Set to the value of bit 3 of the source operand.
Z — Set to the value of bit 2 of the source operand.
V — Set to the value of bit 1 of the source operand.
C — Set to the value of bit 0 of the source operand.

Instruction Format:

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

Effective Address field—Specifies the location of the source operand. O
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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

NOTE

MOVE to CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

Assembler Syntax: MOVE SR, < ea >

Attributes: Size = (Word)

Description: Moves the data in the status register to the destination loca destination is word length. Unimplemented bits are read as zeros.

Condition Codes:

Not affected.

Instruction Format:

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

Instruction Fields:

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

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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

NOTE

Use the MOVE from CCR instruction to access only the condition codes. Memory destination is read before it is written to.

Syntax: MOVE16 (xxx).L,(An)
MOVE16 (xxx).L,(An) +
MOVE16 (An),(xxx).L
MOVE16 (An) + ,(xxx).L

Attributes: Size = (Line)

Description: Moves the source line to the destination line. The lines are aligned to boundaries. Applications for this instruction include coprocessor commun memory initialization, and fast block copy operations.

MOVE16 has two formats. The postincrement format uses the postincrement a ing mode for both source and destination; whereas, the absolute format spe absolute long address for either the source or destination.

Line transfers are performed using burst reads and writes, which begin with word pointed to by the effective address of the source and destination, respect address register used in the postincrement addressing mode is incremente after the transfer.

Example: MOVE16 (A0) + $FE802 A0 = $1400F

The line at address $14000 is read into a temporary holding register by a bu transfer starting with long-word $14000. Address values in A0 of $14000 – cause the same line to be read, starting at different long words. The line is the to the line at address $FE800 beginning with long-word $FE800 after the instru contains $1401F.

Source line at $14000:

$14000	$14004	$14008	$1400C
LONG WORD 0	LONG WORD 1	LONG WORD 2	LONG WORD 3

Destination line at $FE8000:

$FE800	$FE804	$FE808	$FE80C
LONG WORD 0	LONG WORD 1	LONG WORD 2	LONG WORD 3

Instruction Format:

POSTINCREMENT SOURCE AND DESTINATION

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	1	0	0	0	1	0	0	REGI
1	REGISTER Ay			0	0	0	0	0	0	0	0	0	0

Instruction Fields:

Register Ax—Specifies a source address register for the postincrement ad
mode.

Register Ay—Specifies a destination address register for the posti
addressing mode.

Instruction Format:

Absolute Long Address Source or Destination

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	1	0	0	0	0	OPMODE		REGI
HIGH-ORDER ADDRESS													
LOW-ORDER ADDRESS													

Instruction Fields:

Opmode field—Specifies the addressing modes used for source and destina

Opmode	Source	Destinati on	Assembler Syntax
0 0	(Ay) +	(xxx).L	MOVE16 (Ay) + ,(xxx).L
0 1	(xxx).L	(Ay) +	MOVE16 (xxx).L,(Ay) +
1 0	(Ay)	(xxx).L	MOVE16 (Ay),(xxx).L
1 1	(xxx).L	(Ay)	MOVE16 (xxx).L,(Ay)

Register Ay—Specifies an address register for the indirect and posti
addressing mode used as a source or destination.

32-Bit Address field—Specifies the absolute address used as a source or de

Syntax: MOVEM < ea > , < list >

Attributes: Size = (Word, Long)

Description: Moves the contents of selected registers to or from consecutive locations starting at the location specified by the effective address. A re selected if the bit in the mask field corresponding to that register is set. The ins size determines whether 16 or 32 bits of each register are transferred. In the c word transfer to either address or data registers, each word is sign-extended to and the resulting long word is loaded into the associated register.

Selecting the addressing mode also selects the mode of operation of the instruction, and only the control modes, the predecrement mode, and the po ment mode are valid. If the effective address is specified by one of the control the registers are transferred starting at the specified address, and the address mented by the operand length (2 or 4) following each transfer. The order of th ters is from D0 to D7, then from A0 to A7.

If the effective address is specified by the predecrement mode, only a register- ory operation is allowed. The registers are stored starting at the specified minus the operand length (2 or 4), and the address is decremented by the length following each transfer. The order of storing is from A7 to A0, then fro D0. When the instruction has completed, the decremented address register the address of the last operand stored. For the MC68020, MC68030, MC680 CPU32, if the addressing register is also moved to memory, the value written i tial register value decremented by the size of the operation. The MC68(MC68010 write the initial register value (not decremented).

If the effective address is specified by the postincrement mode, only a memory ister operation is allowed. The registers are loaded starting at the specified a the address is incremented by the operand length (2 or 4) following each trans order of loading is the same as that of control mode addressing. When the ins has completed, the incremented address register contains the address of the la and loaded plus the operand length. If the addressing register is also load memory, the memory value is ignored and the register is written with the po mented effective address.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	1	dr	0	0	1	SIZE	EFFECTIVE ADDRESS			
										MODE			REG
REGISTER LIST MASK													

Instruction Fields:

dr field—Specifies the direction of the transfer.
0 — Register to memory.
1 — Memory to register.

Size field—Specifies the size of the registers being transferred.
0 — Word transfer
1 — Long transfer

Effective Address field—Specifies the memory address for the operation. Fo
to-memory transfers, only control alterable addressing modes
predecrement addressing mode can be used as listed in the following t

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Re
(xxx).W	111	
(xxx).L	111	
#<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.

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	—	—	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	—	
(An) +	011	reg. number:An			
– (An)	—	—			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	0
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	0

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	111	0
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	0
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Register List Mask field—Specifies the registers to be transferred. The low-
corresponds to the first register to be transferred; the high-order bit corr
to the last register to be transferred. Thus, for both control mod
postincrement mode addresses, the mask correspondence is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D

For the predecrement mode addresses, the mask correspondence is reverse

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A

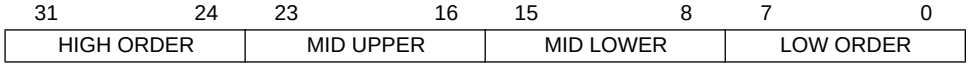
Syntax: MOVEP (d16,Ay),Dx

Attributes: Size = (Word, Long)

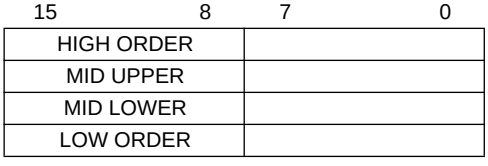
Description: Moves data between a data register and alternate bytes within the
space starting at the location specified and incrementing by two. The high-o
of the data register is transferred first, and the low-order byte is transferred
memory address is specified in the address register indirect plus 16-bit disp
addressing mode. This instruction was originally designed for interfac
peripherals on a 16-bit data bus, such as the MC68000 bus. Although support
MC68020, MC68030, and MC68040, this instruction is not useful for those pr
with an external 32-bit bus.

Example: Long transfer to/from an even address.

Byte Organization in Register



Byte Organization in 16-Bit Memory (Low Address at Top)



HIGH ORDER		MID UPPER	
MID LOWER		LOW ORDER	

or

31 24	23 16	15 8	7 0
		HIGH ORDER	
MID UPPER		MID LOWER	
LOW ORDER			

Example:Word transfer to/from (odd address).

Byte Organization in Register

31 24	23 16	15 8	7 0
		HIGH ORDER	LOW ORDER

Byte Organization in 16-Bit Memory (Low Address at Top)

15 8	7 0
	HIGH ORDER
	LOW ORDER

Byte Organization in 32-Bit Memory

31 24	23 16	15 8	7 0
			HIGH ORDER
	LOW ORDER		

or

31 24	23 16	15 8	7 0
	HIGH ORDER		LOW ORDER

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	0	0	0	DATA REGISTER			OPMODE			0	0	1	ADDRESS
16-BIT DISPLACEMENT													

Instruction Fields:

Data Register field—Specifies the data register for the instruction.

Opmode field—Specifies the direction and size of the operation.
100— Transfer word from memory to register.
101— Transfer long from memory to register.
110— Transfer word from register to memory.
111— Transfer long from register to memory.

Address Register field—Specifies the address register which is used in the register indirect plus displacement addressing mode.

Displacement field—Specifies the displacement used in the operand addres

Syntax: MOVEQ # < data > ,Dn

Attributes: Size = (Long)

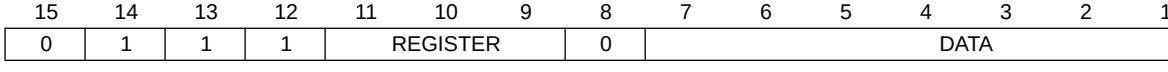
Description: Moves a byte of immediate data to a 32-bit data register. The data in field within the operation word is sign- extended to a long operand in the data as it is transferred.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:



Instruction Fields:

Register field—Specifies the data register to be loaded.

Data field—Eight bits of data, which are sign-extended to a long operand.

Syntax: *MULS.L < ea > ,Dl 32 x 32 → 32
*MULS.L < ea > ,Dh – Dl 32 x 32 → 64

*Applies to MC68020, MC68030, MC68040, CPU32

Attributes: Size = (Word, Long)

Description: Multiplies two signed operands yielding a signed result. This instruct word operand form and a long operand form.

In the word form, the multiplier and multiplicand are both word operands, and is a long-word operand. A register operand is the low-order word; the upper w register is ignored. All 32 bits of the product are saved in the destination data

In the long form, the multiplier and multiplicand are both long- word operands result is either a long word or a quad word. The long-word result is the low-ord of the quad- word result; the high-order 32 bits of the product are discarded.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

X — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if overflow; cleared otherwise.
C — Always cleared.

NOTE

Overflow (V = 1) can occur only when multiplying 32-bit operands to yield a 32-bit result. Overflow occurs if the high-order 32 bits of the quad-word product are not the sign extension of the low- order 32 bits.

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

Instruction Fields:

Register field—Specifies a data register as the destination.

Effective Address field—Specifies the source operand. Only data alterable ad
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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

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

Instruction Fields:

Effective Address field—Specifies the source operand. Only data addressin
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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	

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	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Can be used with CPU32.

Register Dl field—Specifies a data register for the destination operand. T
multiplicand comes from this register, and the low-order 32 bits of the pr
loaded into this register.

Size field—Selects a 32- or 64-bit product.
0 — 32-bit product to be returned to register Dl.
1 — 64-bit product to be returned to Dh – Dl.

Register Dh field—If size is one, specifies the data register into which the h
32 bits of the product are loaded. If Dh = Dl and size is one, the resu
operation are undefined. Otherwise, this field is unused.

Syntax: *MULU.L < ea > ,Dl 32 x 32 → 32
*MULU.L < ea > ,Dh – Dl 32 x 32 → 64

*Applies to MC68020, MC68030, MC68040, CPU32 only

Attributes: Size = (Word, Long)

Description: Multiplies two unsigned operands yielding an unsigned result. This ins has a word operand form and a long operand form.

In the word form, the multiplier and multiplicand are both word operands, and t is a long-word operand. A register operand is the low-order word; the upper wo register is ignored. All 32 bits of the product are saved in the destination data r

In the long form, the multiplier and multiplicand are both long- word operands, result is either a long word or a quad word. The long-word result is the low-orde of the quad- word result; the high-order 32 bits of the product are discarded.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

X — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if overflow; cleared otherwise.
C — Always cleared.

NOTE

Overflow (V = 1) can occur only when multiplying 32-bit operands to yield a 32-bit result. Overflow occurs if any of the high-order 32 bits of the quad-word product are not equal to zero.

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

Instruction Fields:

Register field—Specifies a data register as the destination.

Effective Address field—Specifies the source operand. Only data addressin
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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	

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	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Can be used with CPU32.

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

Instruction Fields:

Effective Address field—Specifies the source operand. Only data addressing 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<data>	111	1
(d_{16},PC)	111	0
(d_8,PC,Xn)	111	0

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	0
([bd,PC,Xn],od)	111	0
([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Register Dl field—Specifies a data register for the destination operand. Th multiplicand comes from this register, and the low-order 32 bits of the pro loaded into this register.

Size field—Selects a 32- or 64-bit product.
0 — 32-bit product to be returned to register Dl.
1 — 64-bit product to be returned to Dh – Dl.

Register Dh field—If size is one, specifies the data register into which the hi 32 bits of the product are loaded. If Dh = Dl and size is one, the result operation are undefined. Otherwise, this field is unused.

Syntax: NBCD < ea >

Attributes: Size = (Byte)

Description: Subtracts the destination operand and the extend bit from zero. The is performed using binary-coded decimal arithmetic. The packed binary-code result is saved in the destination location. This instruction produces complement of the destination if the extend bit is zero or the nines complem extend bit is one. This is a byte operation only.

Condition Codes:

X	N	Z	V	C
*	U	*	U	*

X — Set the same as the carry bit.
N — Undefined.
Z — Cleared if the result is nonzero; unchanged otherwise.
V — Undefined.
C — Set if a decimal borrow occurs; cleared otherwise.

NOTE

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

0	1	0	0	1	0	0	0	0	0	EFFECTIVE ADDRESS MODE	REGIS

Instruction Fields:

Effective Address field—Specifies the destination operand. Only data a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<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.

Syntax: NEG < ea >

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the destination operand from zero and stores the res
destination location. The size of the operation is specified as byte, word, or l

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set the same as the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow occurs; cleared otherwise.
C — Cleared if the result is zero; set otherwise.

Instruction Format:

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

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

Effective Address field—Specifies the destination operand. Only data a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<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.

Syntax: NEGX < ea >

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the destination operand and the extend bit from zero. S
result in the destination location. The size of the operation is specified as by
or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set the same as the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Cleared if the result is nonzero; unchanged otherwise.
V — Set if an overflow occurs; cleared otherwise.
C — Set if a borrow occurs; cleared otherwise.

NOTE

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

0	1	0	0	0	0	0	0	SIZE	EFFECTIVE ADDRESS	
									MODE	REGIS

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 a 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	Reg
(xxx).W	111	0
(xxx).L	111	0
#<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.

Syntax: NOP

Attributes: Unsized

Description: Performs no operation. The processor state, other than the program
is unaffected. Execution continues with the instruction following the NOP in
The NOP instruction does not begin execution until all pending bus cyc
completed. This synchronizes the pipeline and prevents instruction overlap.

Condition Codes:

Not affected.

Instruction Format:

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

Syntax: NOT < ea >

Attributes: Size = (Byte, Word, Long)

Description: Calculates the ones complement of the destination operand and st result in the destination location. The size of the operation is specified as byt 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	1	0	0	0	1	1	0	SIZE		EFFECTIVE ADDRESS				
										MODE			REGIS	

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
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	Re
(xxx).W	111	
(xxx).L	111	
#<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.

Syntax: OR Dn, < ea >

Attributes: Size = (Byte, Word, Long)

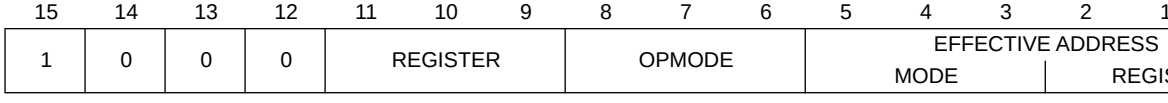
Description: Performs an inclusive-OR operation on the source operand a destination operand and stores the result in the destination location. The siz operation is specified as byte, word, or long. The contents of an address regis not be used as an operand.

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:



Instruction Fields:

Register field—Specifies any of the eight data registers.

Opmode field

Byte	Word	Long	Operation
000	001	010	< ea > V Dn → Dn
100	101	110	Dn V < ea > → < ea >

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	000	reg. number:Dn	(xxx).W	111	
An	—	—	(xxx).L	111	
(An)	010	reg. number:An	#<data>	111	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	

MC68020, MC68030, and MC68040 only

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

*Can be used with CPU32.

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	—	—	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	—	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	—	
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	

*Can be used with CPU32.

NOTE

If the destination is a data register, it must be specified using the destination Dn mode, not the destination < ea > mode.

Most assemblers use ORI when the source is immediate data.

Syntax: ORI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

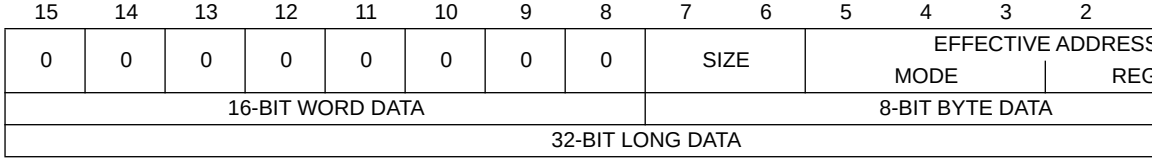
Description: Performs an inclusive-OR operation on the immediate data
destination operand and stores the result in the destination location. The s
operation is specified as byte, word, or long. The size of the immediate data
the operation size.

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:



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

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

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	—	-
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	-
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	-

MC68020, MC68030, and MC68040 only

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

*Can be used with CPU32.

Immediate field—Data immediately following the instruction.
If size = 00, the data is the low-order byte of the immediate word.
If size = 01, the data is the entire immediate word.
If size = 10, the data is the next two immediate words.

Operation: Source V CCR → CCR

Assembler Syntax: ORI # < data > ,CCR

Attributes: Size = (Byte)

Description: Performs an inclusive-OR operation on the immediate operand condition codes and stores the result in the condition code register (low-ord the status register). All implemented bits of the condition code register are af

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set if bit 4 of immediate operand is one; unchanged otherwise.
N — Set if bit 3 of immediate operand is one; unchanged otherwise.
Z — Set if bit 2 of immediate operand is one; unchanged otherwise.
V — Set if bit 1 of immediate operand is one; unchanged otherwise.
C — Set if bit 0 of immediate operand is one; unchanged otherwise.

Instruction Format:

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

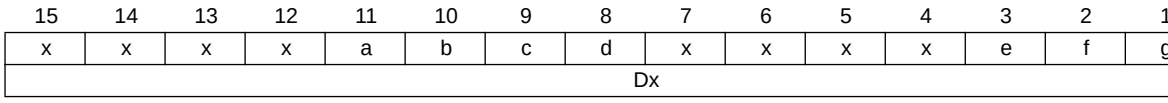
Syntax: PACK Dx,Dy,# < adjustment >

Attributes: Unsized

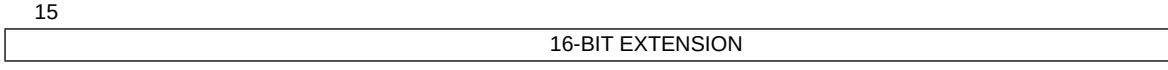
Description: Adjusts and packs the lower four bits of each of two bytes into a sing

When both operands are data registers, the adjustment is added to the value c in the source register. Bits 11 – 8 and 3 – 0 of the intermediate result are conca and placed in bits 7 – 0 of the destination register. The remainder of the de register is unaffected.

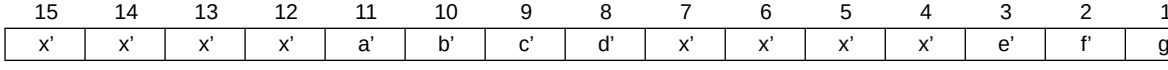
Source:



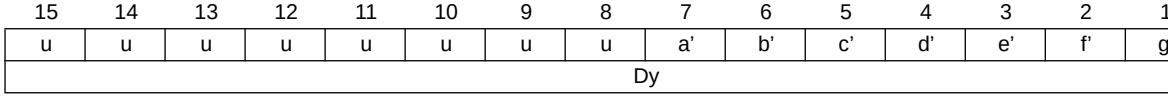
Add Adjustment Word:



Resulting in:



Destination:



When the predecrement addressing mode is specified, two bytes from the so fetched and concatenated. The adjustment word is added to the concatenate Bits 3 – 0 of each byte are extracted. These eight bits are concatenated to for byte which is then written to the destination.

x	x	x	x	a	b	c	d
x	x	x	x	e	f	g	h
Ax							

Concatenated Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
x	x	x	x	a	b	c	d	x	x	x	x	e	f	

Add Adjustment Word:

15
16-BIT EXTENSION

Destination:

7	6	5	4	3	2	1	0
a’	b’	c’	d’	e’	f’	g’	h’
Ay							

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	0	0	0	REGISTER Dy/Ay			1	0	1	0	0	R/M	REGIST
16-BIT ADJUSTMENT EXTENSION:													

If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register in the predecrement addressing m

R/M field—Specifies the operand addressing mode.
0 — The operation is data register to data register.
1 — The operation is memory to memory.

Register Dx/Ax field—Specifies the source register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register in the predecrement addressing m

Adjustment field—Immediate data word that is added to the source operand. T
is zero to pack ASCII or EBCDIC codes. Other values can be used f
codes.

Syntax: PEA < ea >

Attributes: Size = (Long)

Description: Computes the effective address and pushes it onto the stack. The address is a long address.

Condition Codes:

Not affected.

Instruction Format:

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

Instruction Field:

Effective Address field—Specifies the address to be pushed onto the sta 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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	

MC68020, MC68030, and MC68040 only

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

*Can be used with CPU32.

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

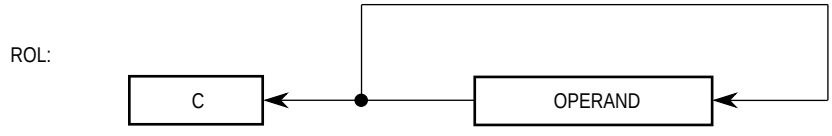
Attributes: Size = (Byte, Word, Long)

Description: Rotates the bits of the operand in the direction specified (L or R). The bit is not included in the rotation. The rotate count for the rotation of a re specified in either of two ways:

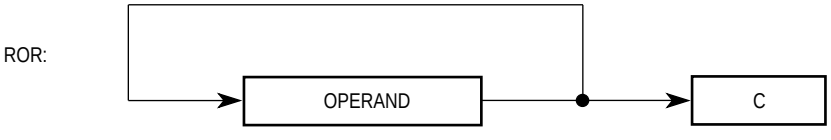
1. Immediate—The rotate count (1 – 8) is specified in the instruction.
2. Register—The rotate count is the value in the data register specified in struction, modulo 64.

The size of the operation for register destinations is specified as byte, word, The contents of memory, (ROd < ea >), can be rotated one bit only, and oper is restricted to a word.

The ROL instruction rotates the bits of the operand to the left; the rotate cou mines the number of bit positions rotated. Bits rotated out of the high-order bit carry bit and also back into the low-order bit.



The ROR instruction rotates the bits of the operand to the right; the rotate cou mines the number of bit positions rotated. Bits rotated out of the low-order bit carry bit and also back into the high-order bit.



—	*	*	0	*

X — Not affected.
N — Set if the most significant bit of the result is set; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Set according to the last bit rotated out of the operand; cleared when
count is zero.

Instruction Format:

REGISTER ROTATE

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

Instruction Fields:

Count/Register field:
If i/r = 0, this field contains the rotate count. The values 1 – 7 represent co
– 7, and zero specifies a count of eight.
If i/r = 1, this field specifies a data register that contains the rotate count (mo

dr field—Specifies the direction of the rotate.
0 — Rotate right
1 — Rotate left

Size field—Specifies the size of the operation.
00 — Byte operation
01 — Word operation
10 — Long operation

i/r field—Specifies the rotate count location.
If i/r = 0, immediate rotate count.
If i/r = 1, register rotate count.

Register field—Specifies a data register to be rotated.

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

Instruction Fields:

dr field—Specifies the direction of the rotate.
0 — Rotate right
1 — Rotate left

Effective Address field—Specifies the operand to be rotated. Only memory a 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	Reg
(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,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.

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

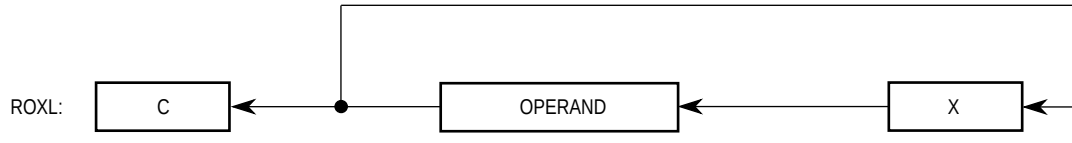
Attributes: Size = (Byte, Word, Long)

Description: Rotates the bits of the operand in the direction specified (L or R). Th
bit is included in the rotation. The rotate count for the rotation of a register is
in either of two ways:

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

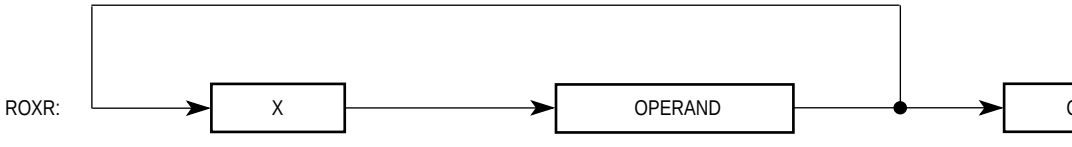
The size of the operation for register destinations is specified as byte, word
The contents of memory, < ea > , can be rotated one bit only, and operan
restricted to a word. The ROXL instruction rotates the bits of the operand to th
rotate count determines the number of bit positions rotated. Bits rotated out of
order bit go to the carry bit and the extend bit; the previous value of the e
rotates into the low-order bit.

.



The ROXR instruction rotates the bits of the operand to the right; the rotate co
mines the number of bit positions rotated. Bits rotated out of the low-order bit
carry bit and the extend bit; the previous value of the extend bit rotates into
order bit.

.



*	*	*	0	*

X — Set to the value of the last bit rotated out of the operand; unaffected w
rotate count is zero.
N — Set if the most significant bit of the result is set; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Set according to the last bit rotated out of the operand; when the rotate
zero, set to the value of the extend bit.

Instruction Format:

REGISTER ROTATE

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

Instruction Fields:

Count/Register field:
If i/r = 0, this field contains the rotate count. The values 1 – 7 represent cou
– 7, and zero specifies a count of eight.
If i/r = 1, this field specifies a data register that contains the rotate count (mod

dr field—Specifies the direction of the rotate.
0 — Rotate right
1 — Rotate left

10 — Long operation

i/r field—Specifies the rotate count location.
If i/r = 0, immediate rotate count.
If i/r = 1, register rotate count.

Register field—Specifies a data register to be rotated.

Instruction Format:

MEMORY ROTATE

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

Instruction Fields:

dr field—Specifies the direction of the rotate.
0 — Rotate right
1 — Rotate left

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

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	—	—	(xxx).W	111	
An	—	—	(xxx).L	111	
(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.

Syntax: RTD # < displacement >

Attributes: Unsized

Description: Pulls the program counter value from the stack and adds the sign-e
16-bit displacement value to the stack pointer. The previous program counter
lost.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0
16-BIT DISPLACEMENT														

Instruction Field:

Displacement field—Specifies the twos complement integer to be sign-exten
added to the stack pointer.

Syntax: RTM Rn

Attributes: Unsized

Description: A previously saved module state is reloaded from the top of stack. module state is retrieved from the top of the stack, the caller's stack p incremented by the argument count value in the module state.

Condition Codes:

Set according to the content of the word on the stack.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	0	0	0	0	1	1	0	1	1	0	0	D/A	REG

Instruction Fields:

D/A field—Specifies whether the module data pointer is in a data or an address
0 — the register is a data register
1 — the register is an address register

Register field—Specifies the register number for the module data area poi restored from the saved module state. If the register specified is A7 updated value of the register reflects the stack pointer operations, and t module data area pointer is lost.

Syntax: RTR

Attributes: Unsized

Description: Pulls the condition code and program counter values from the sta
previous condition code and program counter values are lost. The supervisor
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	1	0	0	1	1	1	0	0	1	1	1	0	1	1

Syntax: RTS

Attributes: Unsized

Description: Pulls the program counter value from the stack. The previous progra
value is lost.

Condition Codes:

Not affected.

Instruction Format:

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

Syntax: SBCD – (Ax), – (Ay)

Attributes: Size = (Byte)

Description: Subtracts the source operand and the extend bit from the destination and stores the result in the destination location. The subtraction is performe binary-coded decimal arithmetic; the operands are packed binary-coded numbers. The instruction has two modes:

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

This operation is a byte operation only.

Condition Codes:

X	N	Z	V	C
*	U	*	U	*

X — Set the same as the carry bit.
N — Undefined.
Z — Cleared if the result is nonzero; unchanged otherwise.
V — Undefined.
C — Set if a borrow (decimal) is generated; cleared otherwise.

NOTE

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

1	0	0	0	REGISTER Dy/Ay	1	0	0	0	R/M	REGIST

Instruction Fields:

Register Dy/Ay field—Specifies the destination register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing

R/M field—Specifies the operand addressing mode.
0 — The operation is data register to data register.
1 — The operation is memory to memory.

Register Dx/Ax field—Specifies the source register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing

Assembler Syntax: Scc < ea >

Attributes: Size = (Byte)

Description: Tests the specified condition code; if the condition is true, sets t specified by the effective address to TRUE (all ones). Otherwise, sets that FALSE (all zeros). Condition code cc specifies one of the following condition (refer to Table 3-19 for more information on these conditional tests):

Mnemonic	Condition
CC(HI)	Carry Clear
CS(LO)	Carry Set
EQ	Equal
F	False
GE	Greater or Equal
GT	Greater Than
HI	High
LE	Less or Equal

Mnemonic	Condition
LS	Low or Same
LT	Less Than
MI	Minus
NE	Not Equal
PL	Plus
T	True
VC	Overflow Clear
VS	Overflow Set

Condition Codes:

Not affected.

0	1	0	1	CONDITION	1	1	EFFECTIVE ADDRESS	
							MODE	REG

Instruction Fields:

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

Effective Address field—Specifies the location in which the TRUE/FALSE byt stored. Only data alterable addressing modes can be used as liste 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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	—	
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	

*Can be used with CPU32.

NOTE

A subsequent NEG.B instruction with the same effective address can be used to change the Scc result from TRUE or FALSE to the equivalent arithmetic value (TRUE = 1, FALSE = 0). In the MC68000 and MC68008, a memory destination is read before it is written.

Syntax: SUB Dn, < ea >

Attributes: Size = (Byte, Word, Long)

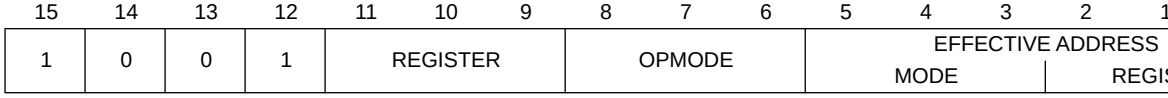
Description: Subtracts the source operand from the destination operand and st
result in the destination. The size of the operation is specified as byte, word,
The mode of the instruction indicates which operand is the source, whic
destination, and which is the operand size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set to the value of the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Instruction Format:



Opmode field

Byte	Word	Long	Operation
000	001	010	Dn – < ea > → Dn
100	101	110	< ea > – Dn → < ea >

Effective Address field—Determines the addressing mode. If the location spe
source operand, all addressing modes can be used as listed in the
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	Re
(xxx).W	111	
(xxx).L	111	
#<data>	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	

MC68020, MC68030, and MC68040 only

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

*For byte-sized operation, address register direct is not allowed.
**Can be used with CPU32.

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	—	—	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	—	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	—	
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	

*Can be used with CPU32.

NOTE

If the destination is a data register, it must be specified as a destination Dn address, not as a destination < ea > address.

Most assemblers use SUBA when the destination is an address register and SUBI or SUBQ when the source is immediate data.

Syntax: SUBA < ea > ,An

Attributes: Size = (Word, Long)

Description: Subtracts the source operand from the destination address register a
the result in the address register. The size of the operation is specified as wor
Word-sized source operands are sign-extended to 32-bit quantities pri
subtraction.

Condition Codes:

Not affected.

Instruction Format:

Instruction Fields:

Register field—Specifies the destination, any of the eight address registers.

Opmode field—Specifies the size of the operation.
011— Word operation. The source operand is sign-extended to a long ope
the operation is performed on the address register using all 32 bits.
111— Long operation.

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An	001	reg. number:An	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	111	1
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	0
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	0

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	111	0
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	0
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	0

*Can be used with CPU32.

Syntax: SUBI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the immediate data from the destination operand and s
result in the destination location. The size of the operation is specified as by
or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set to the value of the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow occurs; cleared otherwise.
C — Set if a borrow occurs; cleared otherwise.

Instruction Format:

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

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

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

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An	—	—	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	—	-
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	-
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	-

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Immediate field—Data immediately following the instruction.
If size = 00, the data is the low-order byte of the immediate word.
If size = 01, the data is the entire immediate word.
If size = 10, the data is the next two immediate words.

Syntax: SUBQ # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

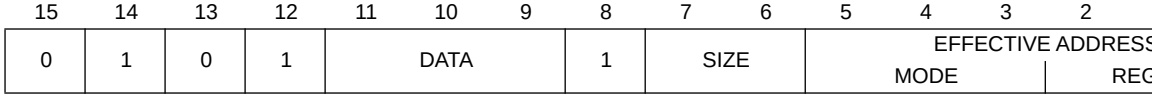
Description: Subtracts the immediate data (1 – 8) from the destination operand. of the operation is specified as byte, word, or long. Only word and long opera be used with address registers, and the condition codes are not affecte subtracting from address registers, the entire destination address register despite the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set to the value of the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow occurs; cleared otherwise.
C — Set if a borrow occurs; cleared otherwise.

Instruction Format:



and zero represents eight.

Size field—Specifies the size of the operation.
00 — Byte operation
01 — Word operation
10 — Long operation

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

Addressing Mode	Mode	Register	Addressing Mode	Mode	Reg
Dn	000	reg. number:Dn	(xxx).W	111	0
An*	001	reg. number:An	(xxx).L	111	0
(An)	010	reg. number:An	#<data>	—	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	

MC68020, MC68030, and MC68040 only

(bd,An,Xn)**	110	reg. number:An	(bd,PC,Xn)**	—	
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	

*Word and long only.
**Can be used with CPU32.

Syntax: SUBX – (Ax), – (Ay)

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand and the extend bit from the destinatio
and stores the result in the destination

location. The instruction has two modes:

1. Data register to data register—the data registers specified in the instru
tain the operands.
2. Memory to memory—the address registers specified in the instructio
the operands from memory using the predecrement addressing mod

The size of the operand is specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set to the value of the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Cleared if the result is nonzero; unchanged otherwise.
V — Set if an overflow occurs; cleared otherwise.
C — Set if a borrow occurs; cleared otherwise.

NOTE

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

1	0	0	1	REGISTER Dy/Ay	1	SIZE		0	0	R/M	REGISTE

Instruction Fields:

Register Dy/Ay field—Specifies the destination register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing r

Size field—Specifies the size of the operation.
00 — Byte operation
01 — Word operation
10 — Long operation

R/M field—Specifies the operand addressing mode.
0 — The operation is data register to data register.
1 — The operation is memory to memory.

Register Dx/Ax field—Specifies the source register:
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing r

Syntax: SWAP Dn

Attributes: Size = (Word)

Description: Exchange the 16-bit words (halves) of a data register.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the 32-bit result is set; cleared otherwi
Z — Set if the 32-bit result is zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

Instruction Field:

Register field—Specifies the data register to swap.

Syntax: TAS < ea >

Attributes: Size = (Byte)

Description: Tests and sets the byte operand addressed by the effective address fi
instruction tests the current value of the operand and sets the N and Z cond
appropriately. TAS also sets the high-order bit of the operand. The operation
locked or read-modify-write transfer sequence. This instruction supports use
or semaphore to coordinate several processors.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the most significant bit of the operand is currently set; cleared oth
Z — Set if the operand was zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:

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

alterable addressing modes can be used as listed in the following table

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	000	reg. number:Dn	(xxx).W	111	
An	—	—	(xxx).L	111	
(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.

SR → (SSP); Vector Address → PC

*The MC68000 and MC68008 do not write vector offset or format code to the system stack.

Assembler Syntax: TRAP # < vector >

Attributes: Unsized

Description: Causes a TRAP # < vector > exception. The instruction adds the im operand (vector) of the instruction to 32 to obtain the vector number. The r vector values is 0 – 15, which provides 16 vectors.

Condition Codes:

Not affected.

Instruction Format:

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

Instruction Fields:

Vector field—Specifies the trap vector to be taken.

Assembler Syntax: TRAPcc
TRAPcc.W # < data >
TRAPcc.L # < data >

Attributes: Unsized or Size = (Word, Long)

Description: If the specified condition is true, causes a TRAPcc exception with number 7. The processor pushes the address of the next instruction word (cu the program counter) onto the stack. If the condition is not true, the processor no operation, and execution continues with the next instruction. The immed operand should be placed in the next word(s) following the operation wo available to the trap handler. Condition code cc specifies one of the conditional tests (refer to Table 3-19 for more information on these conditiona

Mnemonic	Condition
CC(HI)	Carry Clear
CS(LO)	Carry Set
EQ	Equal
F	False
GE	Greater or Equal
GT	Greater Than
HI	High
LE	Less or Equal

Mnemonic	Condition
LS	Low or Same
LT	Less Than
MI	Minus
NE	Not Equal
PL	Plus
T	True
VC	Overflow Clear
VS	Overflow Set

Condition Codes:

Not affected.

0	1	0	1	CONDITION	1	1	1	1	1	OPM
OPTIONAL WORD										
OR LONG WORD										

Instruction Fields:

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

Opmode field—Selects the instruction form.
010—Instruction is followed by word-sized operand.
011—Instruction is followed by long-word-sized operand.
100—Instruction has no operand.

Assembler Syntax: TRAPV

Attributes: Unsized

Description: If the overflow condition is set, causes a TRAPV exception with number 7. If the overflow condition is not set, the processor performs no oper execution continues with the next instruction.

Condition Codes:

Not affected.

Instruction Format:

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

Syntax: TST < ea >

Attributes: Size = (Byte, Word, Long)

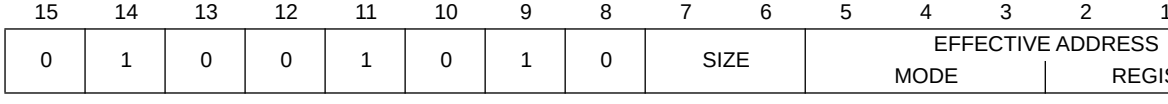
Description: Compares the operand with zero and sets the condition codes acco
the results of the test. The size of the operation is specified as byte, word, or l

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.
N — Set if the operand is negative; cleared otherwise.
Z — Set if the operand is zero; cleared otherwise.
V — Always cleared.
C — Always cleared.

Instruction Format:



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

Effective Address field—Specifies the addressing mode for the destination op
listed in the following tables:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Re
Dn	000	reg. number:Dn	(xxx).W	111	
An*	001	reg. number:An	(xxx).L	111	
(An)	010	reg. number:An	#<data>*	111	
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)**	111	
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)**	111	

MC68020, MC68030, and MC68040 only

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

*MC68020, MC68030, MC68040, and CPU32. Address register direct allowed only for word and long.
**PC relative addressing modes do not apply to MC68000, MC680008, or MC68010.
***Can be used with CPU32.

Syntax: UNLK An

Attributes: Unsized

Description: Loads the stack pointer from the specified address register, then lo
address register with the long word pulled from the top of the stack.

Condition Codes:

Not affected.

Instruction Format:

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

Instruction Field:

Register field—Specifies the address register for the instruction.

Syntax: UNPK Dx,Dy,# < adjustment >

Attributes: Unsized

Description: Places the two binary-coded decimal digits in the source operand byt
lower four bits of two bytes and places zero bits in the upper four bits of bo
Adds the adjustment value to this unpacked value. Condition codes are not a

When both operands are data registers, the instruction unpacks the source
contents, adds the extension word, and places the result in the destination
The high word of the destination register is unaffected.

Source:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
u	u	u	u	u	u	u	u	a	b	c	d	e	f	
Dx														

Intermediate Expansion:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
0	0	0	0	a	b	c	d	0	0	0	0	e	f	

Add Adjustment Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
16-BIT EXTENSION													

Destination:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
v	v	v	v	a’	b’	c’	d’	w	w	w	w	e’	f’	
Dy														

tion address. Source:

7	6	5	4	3	2	1	0
a	b	c	d	e	f	g	h
Ax							

Intermediate Expansion:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	0	a	b	c	d	0	0	0	0	e	f	g

Add Adjustment Word:

15
16-BIT EXTENSION

Destination:

7	6	5	4	3	2	1	0
v	v	v	v	a’	b’	c’	d’
w	w	w	w	e’	f’	g’	h’
Ay							

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	0	REGISTER Dy/Ay			1	1	0	0	0	R/M	REGISTE	
16-BIT EXTENSION: ADJUSTMENT														

If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register in the predecrement addressing m

R/M field—Specifies the operand addressing mode.
0 — The operation is data register to data register.
1 — The operation is memory to memory.

Register Dx/Ax field—Specifies the data register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register in the predecrement addressing m

Adjustment field—Immediate data word that is added to the source
Appropriate constants can be used as the adjustment to translate fro
coded decimal to the desired code. The constant used for ASCII is $
EBCDIC, $F0F0.

SECTION 5
FLOATING POINT INSTRUCTIONS

This section contains information about the floating-point instructions for the M
MC68882, and MC68040. In this section, all references to the MC68040 do not in
MC68LC040 and MC68EC040. Each instruction is described in detail, and the in
descriptions are arranged in alphabetical order by instruction mnemonic.

All floating-point instructions apply to the MC68881 and MC68882 process
MC68040 directly supports part of the floating-point instructions through har
indirectly supports the remainder by providing special traps and/or stack frame
unimplemented instructions and data types. The following identification is noted
instruction title for the MC68040:

Directly Supported—(MC6888X, MC68040)

Software Supported—(MC6888X, MC68040FPSW)

For all MC68040 floating-point instructions, the coprocessor ID field must be 001.

Table 5-1 lists the floating-point instructions directly supported by the MC68040, a
5-2 lists the floating-point instructions indirectly supported.

FBcc	Floating-Point Branch Conditionally
FCMP	Floating-Point Compare
FDBcc	Floating-Point Test Condition, Decrement, and Branch
FDIV	Floating-Point Divide
FMOVE	Move Floating-Point Data Register
FMOVE	Move Floating-Point System Control Register
FMOVEM	Move Multiple Floating-Point System Data Register
FMOVEM	Move Multiple Floating-Point Control Data Register
FMUL	Floating-Point Multiply
FNEG	Floating-Point Negate
FNOP	No Operation
FRESTORE*	Restore Internal Floating-Point State*
FSAVE*	Save Internal Floating-Point State*
FScc	Set According to Floating-Point Condition
FSORT	Floating-Point Square Root
FSUB	Floating-Point Subtract
FSGLDIV	Floating-Point Single-Precision Divide
FSFLMUL	Floating-Point Single-Precision Multiply
FTRAPcc	Trap on Floating-Point Condition
FTST	Test Floating-Point Operand

*These are privileged instructions; refer to **Section 6 Supervisor (Privaleged) Instructio** detailed information.

FATAN	Floating-Point Arc Tangent
FATANH	Floating-Point Hyperbolic Arc Tangent
FCOS	Floating-Point Cosine
FCOSH	Floating-Point Hyperbolic Cosine
FETOX	Floating-Point e^x
FETOXM1	Floating-Point $e^x - 1$
FGETEXP	Floating-Point Get Exponent
FGETMAN	Floating-Point Get Mantissa
FINT	Floating-Point Integer Part
FINTRZ	Floating-Point Integer Part, Round-to- Zero
FLOG10	Floating-Point Log10
FLOG2	Floating-Point Log2
FLOGN	Floating-Point Loge
FLOGNP1	Floating-Point $\log_e (x + 1)$
FMOD	Floating-Point Modulo Remainder
FMOVECR	Floating-Point Move Constant ROM
FREM	Floating-Point IEEE Remainder
FSCALE	Floating-Point Scale Exponent
FSIN	Floating-Point Sine
FSINCOS	Floating-Point Simultaneous Sine and Cosine
FSINH	Floating-Point Hyperbolic Sine
FTAN	Floating-Point Tangent
FTANH	Floating-Point Hyperbolic Tangent
FTENTOX	Floating-Point 10^x
FTWOTOX	Floating-Point 2^x

Syntax:

FABS. < fmt >	< ea > ,FPn
FABS.X	FPm,FPn
FABS.X	FPn
*FrABS. < fmt >	< ea > ,FPn
*FrABS.X	FPm,FPn
*FrABS.X	Pn

where r is rounding precision, S or D

*Supported by MC68040 only.

Attributes: Format = (Byte, Word, Long, Single, Quad, Extended, Packed

Description: Converts the source operand to extended precision (if necessary) an the absolute value of that number in the destination floating-point data registe

FABS will round the result to the precision selected in the floating-point control FSABS and FDABS will round the result to single or double precision, resp regardless of the rounding precision selected in the floating-point control regis

Operation Table:

DESTINATION	SOURCE		
	+ In Range –	+ Zero –	+ Infinity
Result	Absolute Value	Absolute Value	Absolute Value

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**
	OPERR	Cleared
	OVFL	Cleared
	UNFL	If the source is an extended-denormalized number, refer to processing in the appropriate user's cleared otherwise.
	DZ	Cleared
	INEX2	Cleared
	INEX1	If < fmt > is packed, refer to processing in the appropriate user's cleared otherwise.
Accrued Exception Byte:	Affected as described in exception processing; re appropriate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			OPMODE			

If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field specifies the location of the source operand. Only data addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

Destination Register field—Specifies the destination floating- point data regis

Opmode field—Specifies the instruction and rounding precision.

0011000	FABS	Rounding precision specified by the floating-point con register.
1011000	FSABS	Single-precision rounding specified.
1011100	FDABS	Double-precision rounding specified.

Syntax: FACOS.X FPm,FPn
FACOS.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the arc cosine of that number. Stores the result in the destination point data register. This function is not defined for source operands outside of th [– 1... + 1]; if the source is not in the correct range, a NAN is returned as the re the OPERR bit is set in the floating- point status register. If the source is in the range, the result is in the range of [0...π].

Operation Table:

DESTINATION	SOURCE		
	+ In Range –	+ Zero –	+ Infinity
Result	Arc Cosine	+ π/2	NAN

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if the source is infinity, > + 1 c cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REG
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	1	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe
instruction. Motorola assemblers default to ID = 1 for the floa
coprocessor.

Effective Address field—Determines the addressing mode for external opera
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as
the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

If R/M = 1, specifies the source data format:

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is the
into the same register. If the single register syntax is used, Motorola ass
set the source and destination fields to the same value.

Syntax: FADD.X FPm,FPn
*FrADD. < fmt > < ea > ,FPn
*FrADD.X FPm,FPn
where r is rounding precision, S or D

*Supported by MC68040 only.

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necessary) that number to the number contained in the destination floating-point data Stores the result in the destination floating-point data register.

FADD will round the result to the precision selected in the floating-point contro FSADD and FDADD will round the result to single or double-precision, res regardless of the rounding precision selected in the floating-point control reg

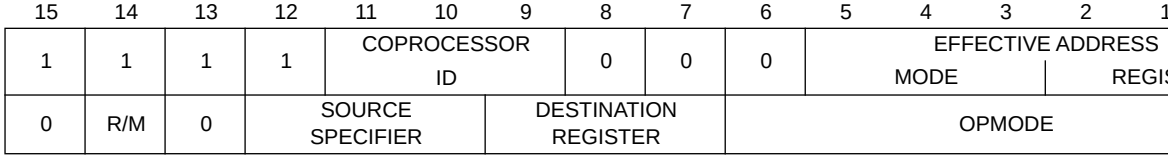
Operation Table:

DESTINATION		SOURCE[1]		
		+ In Range –	+ Zero –	+ Infinity
In Range	+	Add	Add	+ inf
	–			
Zero	+	Add	+ 0.0 0.0^2	0.0^2 – 0.0	+ inf
	–			
Infinity	+	+ inf	+ inf	+ inf
	–	– inf	– inf	NAN‡

1. If either operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Returns + 0.0 in rounding modes RN, RZ, and RP; returns – 0.0 in RM.
3. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers.**
	OPERR	Set if the source and the destina opposite-signed infinities; cleared oth
	OVFL	Refer to exception processing appropriate user's manual.
	UNFL	Refer to exception processing appropriate user's manual.
	DZ	Cleared
	INEX2	Refer to exception processing appropriate user's manual.
	INEX1	If < fmt > is packed, refer to e processing in the appropriate user's cleared otherwise.
Accrued Exception Byte:	Affected as described in exception processing in th priate user's manual.	

Instruction Format:



Instruction Fields:

Effective Address field—Determines the addressing mode for external operan
If R/M = 0, this field is unused and should be all zeros.

Addressing Mode	Mode	Register	Addressing Mode	Mode
Dn	000	reg. number:Dn	(xxx).W	111
An	—	—	(xxx).L	111
(An)	010	reg. number:An	# < data >	111
(An) +	011	reg. number:An		
– (An)	100	reg. number:An		
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)
*This encoding will cause an unimplemented data type exception to allow emulation in software.

Destination Register field—Specifies the destination floating- point data regis

Opmode field—Specifies the instruction and rounding precision.

0100010	FADD	Rounding precision specified by the floating-point con register.
1100010	FSADD	Single-precision rounding specified.
1100110	FDADD	Double-precision rounding specified.

Syntax: FASIN.X FPm,FPn
FASIN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the arc sine of the number. Stores the result in the destination floati data register. This function is not defined for source operands outside of the r 1... + 1]; if the source is not in the correct range, a NAN is returned as the re the OPERR bit is set in the floating- point status register. If the source is in the range, the result is in the range of [– π/2... + π/2].

Operation Table:

DESTINATION	SOURCE[1]					
	+	In Range	–	+	Zero	–	+	Infinity
Result	Arc Sine			+ 0.0		– 0.0	NAN[2]	

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if the source is infinity, > + 1 cleared otherwise
	OVFL	Cleared
	UNFL	Can be set for an underflow conditi
	DZ	Cleared
	INEX2	Refer to inexact result in the ap user's manual.
	INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.
Accrued Exception Byte:	Affected as described in IEEE exception and trap bility in the appropriate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER		0	0	0	1	1	

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

If R/M = 1, this field is encoded with an M68000 family addressing mode as
the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is the
into the same register. If the single register syntax is used, Motorola ass
set the source and destination fields to the same value.

Syntax: FATAN.X FPm,FPn
FATAN.X FPm,FPnz

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess calculates the arc tangent of that number. Stores the result in the destinatio point data register. The result is in the range of [– π/2... + π/2].

Operation Table:

DESTINATION	SOURCE		
	+ In Range –	+ Zero –	+ Infinity
Result	Arc Tangent	+ 0.0 – 0.0	+ π/2

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Cleared
UNFL	Refer to underflow in the appropria manual.
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	0	1	1	0

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

Effective Address field—Determines the addressing mode for external operan
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

If R/M = 1, specifies the source data format:

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, then the input o
taken from the specified floating-point data register, and the result is the
into the same register. If the single register syntax is used, Motorola as
set the source and destination fields to the same value.

Syntax: FATANH.X FPm,FPn
FATANH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the hyperbolic arc tangent of that value. Stores the result in the de floating-point data register. This function is not defined for source operands ou the range (– 1... + 1); and the result is equal to – infinity or + infinity if the s equal to + 1 or – 1, respectively. If the source is outside of the range [– 1... + 1 is returned as the result, and the OPERR bit is set in the floating-point status r

Operation Table:

DESTINATION	SOURCE[1]		
	+ In Range –	+ Zero –	+ Infinity
Result	Hyperbolic Arc Tangent	+ 0.0 – 0.0	NAN[2]

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

OPERR	Set if the source is > + 1 or < – 1 otherwise.
OVFL	Cleared
UNFL	Refer to underflow in the appropria manual.
DZ	Set if the source is equal to + 1 or – 1 otherwise.
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	0	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

If R/M = 1, this field is encoded with an M68000 family addressing mode as
the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is the
into the same register. If the single register syntax is used, Motorola ass
set the source and destination fields to the same value.

Assembler:
Syntax: FBcc. < size > , < label >

Attributes: Size = (Word, Long)

Description: If the specified floating-point condition is met, program execution co
the location (PC) + displacement. The displacement is a twos-complement in
counts the relative distance in bytes. The value of the program counte
calculate the destination address is the address of the branch instruction pl
the displacement size is word, then a 16- bit displacement is stored in
immediately following the instruction operation word. If the displacement siz
word, then a 32-bit displacement is stored in the two words immediately follo
instruction operation word. The conditional specifier cc selects any one
floating- point conditional tests as described in **3.6.2 Conditional Testing**.

Floating-Point Status Register:

Condition Codes:	Not affected.	
Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Set if the NAN condition code is se condition selected is an IEEE nonav
	SNAN	Not Affected.
	OPERR	Not Affected.
	OVF	Not Affected.
	UNFL	Not Affected.
	DZ	Not Affected.
	INEX2	Not Affected.
	INEX1	Not Affected.
Accrued Exception Byte:	The IOP bit is set if the BSUN bit is set in the byte. No other bit is affected.	

1	1	1	1	COPROCESSOR ID	0	1	SIZE	CONDITIONAL PREDICAT
16-BIT DISPLACEMENT OR MOST SIGNIFICANT WORD OF 32-BITDISPLACEMENT								
LEAST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT (IF NEEDED)								

Instruction Fields:

Size field—Specifies the size of the signed displacement.
If Format = 0, then the displacement is 16 bits and is sign- extended before
If Format = 1, then the displacement is 32 bits.

Conditional Predicate field—Specifies one of 32 conditional tests as defined
3-23 Floating-Point Conditional Tests.

NOTE

When a BSUN exception occurs, the main processor takes a preinstruction exception. If the exception handler returns without modifying the image of the program counter on the stack frame (to point to the instruction following the FBcc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap), or the exception will occur again immediately upon return to the routine that caused the exception.

Syntax: FCMP.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess subtracts the operand from the destination floating- point data register. The re subtraction is not retained, but it is used to set the floating-point condition described in **3.6.2 Conditional Testing**.

Operation Table: The entries in this operation table differ from those of t describing most of the floating-point instructions. For each combination operand types, the condition code bits that may be set are indicated. If the r condition code bit is given and is not enclosed in brackets, then it is always name of a condition code bit is enclosed in brackets, then that bit is eith cleared, as appropriate. If the name of a condition code bit is not given, then always cleared by the operation. The infinity bit is always cleared by th instruction since it is not used by any of the conditional predicate equations. the NAN bit is not shown since NANs are always handled in the same ma described in **1.6.5 Not-A-Numbers**).

DESTINATION		SOURCE					
		+	In Range	–	+	Zero	–	+	Infinity
In Range	+	{NZ}		none	none		none	N	
	–	N		{NZ}	N		N	N	
Zero	+	N		none	Z		Z	N	
	–	N		none	NZ		NZ	N	
Infinity	+	none		none	none		none	Z	
	–	N		N	N		N	N	

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Cleared
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Cleared
	INEX1	If < fmt > is packed, refer to e processing in the appropriate user's cleared otherwise.
Accrued Exception Byte:	Affected as described in exception processing in th priate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	1	0

If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, specifies the location of the source operand location. Only data addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)
*This encoding in the MC68040 will cause an unimplemented data type exception to allow emulation in software.

Destination Register field—Specifies the destination floating- point data regis

Syntax: FCOS.X FPm,FPn FCOS.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the cosine of that number. Stores the result in the destination floati data register. This function is not defined for source operands of ± infinity. If th operand is not in the range of [– 2π... + 2π], then the argument is reduced to w range before the cosine is calculated. However, large arguments may lose a during reduction, and very large arguments (greater than approximately 10^{20} accuracy. The result is in the range of [– 1... + 1].

Operation Table:

DESTINATION	SOURCE[1]					
	+	In Range	–	+	Zero	–	+	Infinity
Result	Cosine			+ 1.0			NAN[2]	

NOTES:

1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if the source operand is ± infinity otherwise.
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to inexact result in the ap user's manual.
	INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.
Accrued Exception Byte:	Affected as described in IEEE exception and trap bility in the appropriate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE		REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	1	1	1	

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is wri
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Syntax: FCOSH.X FPm,FPn FCOSH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess calculates the hyperbolic cosine of that number. Stores the result in the d floating-point data register.

Operation Table:

DESTINATION	SOURCE					
	+	In Range	–	+	Zero	–	+	Infinity
Result	Hyperbolic Cosine			+ 1.0			+ inf	

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Refer to overflow in the appropria manual.
UNFL	Cleared
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	1	1	1	0

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floating-point copro

Effective Address field—Determines the addressing mode for external operan
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, then the input o
taken from the specified floating-point data register, and the result is w
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Else Dn – 1 → Dn
If Dn ≠ – 1
Then PC + d_n → PC
Else Execute Next Instruction

Assembler Syntax: FDBcc Dn, < label >

Attributes: Unsized

Description: This instruction is a looping primitive of three parameters: a floati condition, a counter (data register), and a 16-bit displacement. The instruction f the condition to determine if the termination condition for the loop has been m so, execution continues with the next instruction in the instruction stream termination condition is not true, the low-order 16 bits of the counter regi decremented by one. If the result is – 1, the count is exhausted, and e continues with the next instruction. If the result is not equal to – 1, execution c at the location specified by the current value of the program counter plus t extended 16-bit displacement. The value of the program counter used in the address calculation is the address of the displacement word.

The conditional specifier cc selects any one of the 32 floating- point condition as described in **3.6.2 Conditional Testing**.

Floating-Point Status Register:

Condition Codes:	Not affected.	
Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Set if the NAN condition code is set condition selected is an IEEE nonaw
	SNAN	Not Affected.
	OPERR	Not Affected.
	OVFL	Not Affected.
	UNFL	Not Affected.
	DZ	Not Affected.
	NEX2	Not Affected.
	INEX1	Not Affected.
Accrued Exception Byte:	The IOP bit is set if the BSUN bit is set in the e byte. No other bit is affected.	

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	1	0	0	1	CO REG
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICA			
16-BIT DISPLACEMENT													

Instruction Fields:

Count Register field—Specifies data register that is used as the counter.

Conditional Predicate field—Specifies one of the 32 floating-point conditiona described in **3.6.2 Conditional Testing**.

Displacement field—Specifies the branch distance (from the address of the i plus two) to the destination in bytes.

NOTE

The terminating condition is like that defined by the UNTIL loop constructs of high-level languages. For example: FDBOLT can be stated as "decrement and branch until ordered less than".

There are two basic ways of entering a loop: at the beginning or by branching to the trailing FDBcc instruction. If a loop structure terminated with FDBcc is entered at the beginning, the control counter must be one less than the number of loop executions desired. This count is useful for indexed addressing modes and dynamically specified bit operations. However, when entering a loop by branching directly to the trailing FDBcc instruction, the count should equal the loop execution count. In this case, if the counter is zero when the loop is entered, the FDBcc instruction does not branch, causing a complete bypass of the main loop.

When a BSUN exception occurs, a preinstruction exception is taken by the main processor. If the exception handler returns without modifying the image of the program counter on the stack frame (to point to the instruction following the FDBcc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap), or the exception will occur again immediately upon return to the routine that caused the exception.

Syntax: FDIV.X FPm,FPn
*FrDIV. < fmt > < ea > ,FPn
*FrDIV.X FPm,FPn
where r is rounding precision, S or D

*Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessary) and that number into the number in the destination floating-point data register. St result in the destination floating-point data register.

FDIV will round the result to the precision selected in the floating-point control FSDIV and FDDIV will round the result to single or double precision, resp regardless of the rounding precision selected in the floating-point control regis

Operation Table:

DESTINATION		SOURCE[1]							
		+	In Range	–	+	Zero	–	+	Infinity
In Range	+	Divide			+ inf[2]		– inf[2]	+ 0.0	
	–				– inf[2]		+ inf[2]	– 0.0	
Zero	+	+ 0.0		+ 0.0	NAN[3]			+ 0.0	
	–	– 0.0		+ 0.0				– 0.0	
Infinity	+	+ inf		– inf	+ inf		– inf	NAN‡	
	–	– inf		+ inf	– inf		+ inf		

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the DZ bit in the floating-point status register exception byte.
3. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set for 0 ÷ 0 or infinity ÷ infinity otherwise.
	OVFL	Refer to exception processing appropriate user's manual.
	UNFL	Refer to exception processing appropriate user's manual.
	DZ	Set if the source is zero and the des in range; cleared otherwise.
	INEX2	Refer to exception processing appropriate user's manual.
	INEX1	If < fmt > is packed, refer to processing in the appropriate user's cleared otherwise.
Accrued Exception Byte:	Affected as described in exception processing in t priate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			OPMODE			

If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, specifies the location of the source operand location. Only data addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)
*This encoding in the MC68040 will cause an unimplemented data type exception to allow emulation in software.

0100000	FDIV	Rounding precision specified by the floating- point control register.
1100000	FSDIV	Single-precision rounding specified.
1100100	FDDIV	Double-precision rounding specified.

Syntax: FETOX.X FPm,FPn
Syntax: FETOX.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates e to the power of that number. Stores the result in the destination point data register.

Operation Table:

DESTINATION	SOURCE								
	+	In Range	–	+	Zero	–	+	Infinity	
Result	e^x			+ 1.0			+ inf		+

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Refer to overflow in the appropriat manual.
UNFL	Refer to underflow in the appropriat manual.
DZ	Cleared
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REG
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	0	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

Effective Address field—Determines the addressing mode for external opera
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

Source Specifier Field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is wri
the same register. If the single register syntax is used, Motorola assemb
the source and destination fields to the same value.

Syntax: FETOXM1.X FPm,FPn
FETOXM1.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess calculates e to the power of that number. Subtracts one from the value and s result in the destination floating-point data register.

Operation Table:

DESTINATION	SOURCE		
	+ In Range –	+ Zero –	+ Infinity
Result	$e^X - 1$	+ 0.0 – 0.0	+ inf

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Refer to overflow in the appropria manual.
UNFL	Refer to underflow in the appropria manual.
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	1	0

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

Effective Address field—Determines the addressing mode for external operan
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

Source Specifier Field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, then the input o
taken from the specified floating-point data register, and the result is w
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Syntax: FGETEXP.X FPm,FPn
FGETEXP.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa extracts the binary exponent. Removes the exponent bias, converts the expon extended-precision floating- point number, and stores the result in the de floating- point data register.

Operation Table:

DESTINATION	SOURCE[1]		
	+ In Range –	+ Zero –	+ Infinity
Result	Exponent	+ 0.0 – 0.0	NAN[2]

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if the source is ± infinity; otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Cleared
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REG
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	1	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe
instruction. Motorola assemblers default to ID = 1 for the floa
coprocessor.

Effective Address field—Determines the addressing mode for external opera
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as
the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is wri
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Syntax: FGETMAN.X FPm,FPn
FGETMAN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess extracts the mantissa. Converts the mantissa to an extended-precision v stores the result in the destination floating-point data register. The result is in [1.0...2.0] with the sign of the source mantissa, zero, or a NAN.

Operation Table:

DESTINATION	SOURCE[1]								
	+	In Range	–	+	Zero	–	+	Infinity	
Result	Mantissa			+ 0.0		– 0.0	NAN[2]		

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if the source is ± infinity; otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Cleared
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	1	1	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

Effective Address field—Determines the addressing mode for external operan
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, then the input o
taken from the specified floating-point data register, and the result is w
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Syntax: FINT.X FPm,FPn
FINT.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessary),
the integer part, and converts it to an extended-precision floating-point numbe
the result in the destination floating-point data register. The integer part is extr
rounding the extended-precision number to an integer using the current roundi
selected in the floating-point control register mode control byte. Thus, the inte
returned is the number that is to the left of the radix point when the exponent
after rounding. For example, the integer part of 137.57 is 137.0 for the round
and round-to- negative infinity modes and 138.0 for the round-to-nearest and r
positive infinity modes. Note that the result of this operation is a floating-point

Operation Table:

DESTINATION	SOURCE					
	+	In Range	–	+	Zero	–	+	Infinity
Result		Integer		+ 0.0		– 0.0		+ inf – inf

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	0	0	0

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is wri
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Syntax: FINTRZ.X FPm,FPn
FINTRZ.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess extracts the integer part and converts it to an extended-precision floating-poin Stores the result in the destination floating-point data register. The intege extracted by rounding the extended-precision number to an integer using the zero mode, regardless of the rounding mode selected in the floating-poi register mode control byte (making it useful for FORTRAN assignments). integer part returned is the number that is to the left of the radix point exponent is zero. For example, the integer part of 137.57 is 137.0; the integ 0.1245 x 102 is 12.0. Note that the result of this operation is a floating-point

Operation Table:

DESTINATION	SOURCE		
	+ In Range –	+ Zero –	+ Infinity
Result	Integer, Forced Round-to- Zero	+ 0.0 – 0.0	+ inf

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap c bility in the appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	0	0	0	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If RM = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, then the input o
taken from the specified floating-point data register, and the result is w
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Syntax: FLOG10.X FPm,FPn
FLOG10.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Convert the source operand to extended precision (if necessa calculates the logarithm of that number using base 10 arithmetic. Stores the the destination floating-point data register. This function is not defined for inpu less than zero.

Operation Table:

DESTINATION	SOURCE[1]						
	+	In Range	–	+	Zero	–	+ Infinity
Result	Log_{10}		NAN[2]		– inf[3]		+ inf N

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.
3. Sets the DZ bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if the source operand is < 0; otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Set if the source is ± 0; cleared other
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REG
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	1	0	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

Effective Address field—Determines the addressing mode for external opera
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is wri
the same register. If the single register syntax is used, Motorola assemb
the source and destination fields to the same value.

Syntax: FLOG2.X FPm,FPn
FLOG2.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess calculates the logarithm of that number using base two arithmetic. Stores the the destination floating- point data register. This function is not defined for inp less than zero.

Operation Table:

DESTINATION	SOURCE[1]					
	+	In Range	–	+	Zero	–	+	Infinity
Result	Log_2	NAN[2]		– inf[3]		+ inf	

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.
3. Sets the DZ bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if the source is < 0; cleared othe
OVFL	Cleared
UNFL	Cleared
DZ	Set if the source is ± 0; cleared othe
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

Effective Address field—Determines the addressing mode for external operan
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, then the input o
taken from the specified floating-point data register, and the result is w
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Syntax: FLOGN.X FPm,FPn
FLOGN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the natural logarithm of that number. Stores the result in the de floating-point data register. This function is not defined for input values less tha

Operation Table:

DESTINATION	SOURCE[1]							
	+	In Range	–	+	Zero	–	+	Infinity
Result	ln(x)		NAN[2]		– inf[3]		+ inf	N

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.
3. Sets the DZ bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if the source operand is < 0; otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Set if the source is ± 0; cleared othe
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REG
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	1	0	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

Effective Address field—Determines the addressing mode for external opera
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is wri
the same register. If the single register syntax is used, Motorola assemb
the source and destination fields to the same value.

Syntax: FLOGNP1.X FPm,FPn
FLOGNP1.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necessary),
to that value, and calculates the natural logarithm of that intermediate result. S
result in the destination floating-point data register. This function is not define
values less than – 1.

Operation Table:

DESTINATION	SOURCE[1]							
	+	In Range	–	+	Zero	–	+	Infinity
Result	ln(x + 1)		$\ln(x+1)^2$	+ 0.0		– 0.0	+ inf	

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. If the source is – 1, sets the DZ bit in the floating-point status register exception byte and returns a NAN. If the source is < – 1, sets the OPERR bit in the floating-point status register exception byte and returns a NAN.
3. Sets the OPERR bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

OPERR	Set if the source operand is < – 1; otherwise.
OVFL	Cleared
UNFL	Refer to underflow in the appropriat manual.
DZ	Set if the source operand is – 1; otherwise
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap c bility in the appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, then the input o
taken from the specified floating-point data register, and the result is w
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Syntax: FMOD.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the modulo remainder of the number in the destination floating-po register, using the source operand as the modulus. Stores the result in the de floating-point data register and stores the sign and seven least significant bi quotient in the floating-point status register quotient byte (the quotient is the FPn ÷ Source). The modulo remainder function is defined as:

$$FPn - (Source \times N)$$

where N = INT(FPn ÷ Source) in the round-to-zero mode.

The FMOD function is not defined for a source operand equal to zero or for a de operand equal to infinity. Note that this function is not the same as the FREM ins which uses the round-to-nearest mode and thus returns the remainder that is req the IEEE *Specification for Binary Floating-Point Arithmetic*.

Operation Table:

DESTINATION		SOURCE[1]		
		+ In Range –	+ Zero# –	+ Infinity
In Range	+ –	Modulo Remainder	NAN[2]	FPn[3]
Zero	+ –	+ 0.0 – 0.0	NAN[2]	+ 0.0 – 0.0
Infinity	+ –	NAN[2]	NAN[2]	NAN[2]

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.
3. Returns the value of FPn before the operation. However, the result is processed by the normal instruction termination procedure to round it as required. Thus, an overflow and/or inexact result may occur if the rounding precision has been changed to a smaller size since the FPn value was loaded

Quotient Byte: Loaded with the sign and least significant seven b
quotient (FPn ÷ Source). The sign of the quotie
exclusive-OR of the sign bits of the source and de
operands.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if the source is zero or the dest infinity; cleared otherwise.
OVFL	Cleared
UNFL	Refer to underflow in the appropria manual.
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, in the appropria manual for inexact result on decim cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap
bility in the appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	0	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe
instruction. Motorola assemblers default to ID = 1 for the floa
coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regist

Syntax: FMOVE. < fmt > FPm, < ea >
FMOVE.P FPm, < ea > {Dn}
FMOVE.P FPm, < ea > {k}
*FrMOVE. < fmt > < ea > ,FPn
where r is rounding precision, S or D

*Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Moves the contents of the source operand to the destination
Although the primary function of this instruction is data movement, it is also c
an arithmetic instruction since conversions from the source operand form
destination operand format are performed implicitly during the move operat
the source operand is rounded according to the selected rounding precision a

Unlike the MOVE instruction, the FMOVE instruction does not support a m
memory format. For such transfers, it is much faster to utilize the MOVE inst
transfer the floating- point data than to use the FMOVE instruction. The
instruction only supports memory-to-register, register-to- register, and re
memory operations (in this context, memory may refer to an integer data regi
data format is byte, word, long, or single). The memory-to-register and registe
ister operation uses a command word encoding distinctly different from tha
the register-to-memory operation; these two operation classes are describ
rately.

Memory-to-Register and Register-to-Register Operation: Converts the source
to an extended-precision floating-point number (if necessary) and stores
destination floating-point data register. MOVE will round the result to the
selected in the floating-point control register. FSMOVE and FDMOVE will r
result to single or double precision, respectively, regardless of the rounding
selected in the floating-point control register. Depending on the source data fo
the rounding precision, some operations may produce an inexact result. In the
table, combinations that can produce an inexact result are marked with a dot
other combinations produce an exact result.

Single			•		•	•	•
Double						•	•
Extended							•

Floating-Point Status Register (< ea > to Register):

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers.**
OPERR	Cleared
OVFL	Cleared
UNFL	Refer to exception processing appropriate user's manual if the sou extended-precision denormalized cleared otherwise.
DZ	Cleared
INEX2	Refer to exception processing appropriate user's manual if < fmt > i X; cleared otherwise.
INEX1	Refer to exception processing appropriate user's manual if < fmt cleared otherwise.

Accrued Exception Byte: Affected as described in exception processing in th priate user's manual.

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			OPMODE			

Instruction Fields:

Effective Address field—Determines the addressing mode for external opera
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, specifies the location of the source operand. Only data addressin
can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

*This encoding in the MC68040 will cause an unimplemented data type exception to allow emulation in software.

Destination Register field—Specifies the destination floating- point data regist

Opmode field—Specifies the instruction and rounding precision.

0000000	FMOVE	Rounding precision specified by the floating-point control register.
1000000	FSMOVE	Single-precision rounding specified.
1000100	FDMOVE	Double-precision rounding specified.

Register-to-Memory Operation: Rounds the source operand to the size of the s destination format and stores it at the destination effective address. If the form destination is packed decimal, a third operand is required to specify the form resultant string. This operand, called the k-factor, is a 7-bit signed integ complement) and may be specified as an immediate value or in an integ register. If a data register contains the k-factor, only the least significant seven used, and the rest of the register is ignored.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
< fmt > is B, W, or L	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if the source operand is infinity destination size is exceeded after c and rounding; cleared otherwise.
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to exception processing appropriate user's manual.
	INEX1	Cleared
< fmt > is S, D, or X	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**
	OVFL	Refer to exception processing appropriate user's manual.
	UNFL	Refer to exception processing appropriate user's manual.
	DZ	Cleared
	INEX2	Refer to exception processing appropriate user's manual.
	INEX1	Cleared
< fmt > is P	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if the k-factor > + 17 or the mag the decimal exponent exceeds thr cleared otherwise.
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to exception processing appropriate user's manual.
	INEX1	Cleared
Accrued Exception Byte:	Affected as described in exception processing in t priate user's manual.	

15	14	13	12 11 10	9	8	7	6	5 4 3	2 1
1	1	1	COPROCESSOR ID	1	0	0	0	EFFECTIVE ADDRESS MODE	REGIS
0	1	1	DESTINATION FORMAT	SOURCE REGISTER				K-FACTOR (IF REQUIRED)	

Instruction Fields:

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

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn],od)	—	
([bd,PC],Xn,od)	—	

*Only if < fmt > is byte, word, long, or single.

Destination Format field—Specifies the data format of the destination operand

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real with Static k-Factor (P{#k})*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)
111 — Packed-Decimal Real with Dynamic k-Factor (P{Dn})*

*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

of the decimal string. For any other destination format, this field should
all zeros. For a static k-factor, this field is encoded with a twos-co
integer where the value defines the format as follows:

– 64 to 0—Indicates the number of significant digits to the right of the
point (FORTRAN "F" format).
+ 1 to + 17—Indicates the number of significant digits in the mantis
TRAN "E" format).
+ 18 to + 63—Sets the OPERR bit in the floating-point status register
byte and treated as + 17.

The format of this field for a dynamic k-factor is:

r r r 0 0 0 0

where "rrr" is the number of the main processor data register that contains th
value.

The following table gives several examples of how the k-factor value affects t
of the decimal string that is produced by the floating-point coprocessor. The
the string that is generated is independent of the source of the k-factor
dynamic).

k- Factor	Source Operand Value	Destination String
– 5	+ 12345.678765	+ 1.234567877E + 4
– 3	+ 12345.678765	+ 1.2345679E + 4
– 1	+ 12345.678765	+ 1.23457E + 4
0	+ 12345.678765	+ 1.2346E + 4
+ 1	+ 12345.678765	+ 1.E + 4
+ 3	+ 12345.678765	+ 1.23E + 4
+ 5	+ 12345.678765	+ 1.2346E + 4

Assembler Syntax: FMOVE.L < ea > ,FPCR
FMOVE.L FPCR, < ea >

Attributes: Size = (Long)

Description: Moves the contents of a floating-point system control register (floati
control register, floating-point status register, or floating-point instruction
register) to or from an effective address. A 32-bit transfer is always performe
though the system control register may not have 32 implemented bits. Unimple
bits of a control register are read as zeros and are ignored during writes (must
for compatibility with future devices). For the MC68881, this instruction does n
pending exceptions (other than protocol violations) to be reported. Furthermore
to the floating-point control register exception enable byte or the floating-poi
register exception status byte cannot generate a new exception, regardles
value written.

Floating-Point Status Register: Changed only if the destination is the floating-poi
register, in which case all bits are modified to reflect the value of the source o

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
1	0	dr	REGISTER SELECT			0	0	0	0	0	0	0	0	0

Effective Address field—(Memory-to-Register) All addressing modes can be listed in the following table:

Addressing Mode	Mode	Register	Addressing Mode	Mode
Dn	000	reg. number:Dn	(xxx).W	111
An*	001	reg. number:An	(xxx).L	111
(An)	010	reg. number:An	# < data >	111
(An) +	011	reg. number:An		
– (An)	100	reg. number:An		
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111

*Only if the source register is the floating-point instruction address register.

Effective Address field—(Register-to-Memory) Only alterable addressing m be used as listed in the following table:

Addressing Mode	Mode	Register	Addressing Mode	Mode
Dn	000	reg. number:Dn	(xxx).W	111
An*	001	reg. number:An	(xxx).L	111
(An)	010	reg. number:An	# < data >	—
(An) +	011	reg. number:An		
– (An)	100	reg. number:An		
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—

*Only if the source register is the floating-point instruction address register.

1 — From the specified system control register to < ea > .

Register Select field—Specifies the system control register to be moved:
100 Floating-Point Control Register
010 Floating-Point Status Register
001 Floating-Point Instruction Address Register

Syntax: FMOVECR.X # < ccc > ,FPn

Attributes: Format = (Extended)

Description: Fetches an extended-precision constant from the floating- point co on-chip ROM, rounds the mantissa to the precision specified in the floa control register mode control byte, and stores it in the destination floating-p register. The constant is specified by a predefined offset into the constant R values of the constants contained in the ROM are shown in the offset table a of this description.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Cleared
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	Cleared

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	0	0	0	0	
0	1	0	1	1	1	DESTINATION REGISTER			ROM OFFSET					

instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

Destination Register field—Specifies the destination floating- point data regist

ROM Offset field—Specifies the offset into the floating-point coprocessor constant ROM where the desired constant is located. The offsets for the a constants are as follows:

Offset	Constant
$00	π
$0B	$Log_{10}(2)$
$0C	e
$0D	$Log_2(e)$
$0E	$Log_{10}(e)$
$0F	0.0
$30	1n(2)
$31	1n(10)
$32	100
$33	10^1
$34	10^2
$35	10^4
$36	10^8
$37	10^{16}
$38	10^{32}
$39	10^{64}
$3A	10^{128}
$3B	10^{256}
$3C	10^{512}
$3D	10^{1024}
$3E	10^{2048}
$3F	10^{4096}

The on-chip ROM contains other constants useful only to the on- chip microc tines. The values contained at offsets other than those defined above are rese the use of Motorola and may be different on various mask sets of the floati coprocessor. These undefined values yield the value 0.0 in the M68040FPSP

Assembler Syntax:
FMOVEM.X < list > , < ea >
FMOVEM.X Dn, < ea >
FMOVEM.X < ea > , < list > FMOVEM.X < ea > ,Dn

Attributes: Format = (Extended)

Description: Moves one or more extended-precision numbers to or from a list o
point data registers. No conversion or rounding is performed during this opera
the floating-point status register is not affected by the instruction. For the MC6
instruction does not cause pending exceptions (other than protocol violatio
reported. Furthermore, a write to the floating- point control register exceptio
byte or the floating-point status register exception status byte connot genera
exception, despite the value written.

Any combination of the eight floating-point data registers can be transferred
selected registers specified by a user- supplied mask. This mask is an 8-bi
where each bit corresponds to one register; if a bit is set in the mask, that r
moved. The register select mask may be specified as a static value contain
instruction or a dynamic value in the least significant eight bits of an integer
ister (the remaining bits of the register are ignored).

FMOVEM allows three types of addressing modes: the control modes, the
ment mode, or the postincrement mode. If the effective address is one of th
addressing modes, the registers are transferred between the processor and
starting at the specified address and up through higher addresses. The or
transfer is from FP0 – FP7.

address register and down through lower addresses. Before each register is st
address register is decremented by 12 (the size of an extended-precision nu
memory) and the floating-point data register is then stored at the resultant a
When the operation is complete, the address register points to the image of
floating- point data register stored. The order of the transfer is from FP7 – FP

If the effective address is the postincrement mode, only a memory- to-registe
tion is allowed. The registers are loaded starting at the specified address
through higher addresses. After each register is stored, the address register
mented by 12 (the size of an extended-precision number in memory). When t
ation is complete, the address register points to the byte immediately follo
image of the last floating-point data register loaded. The order of the transf
same as for the control addressing modes: FP0 – FP7.

Floating-Point Status Register: Not Affected. Note that the FMOVEM instruction
the only mechanism for moving a floating- point data item between the floati
unit and memory without performing any data conversions or affecting the c
code and exception status bits.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
1	1	dr	MODE		0	0	0	REGISTER LIST						

Effective Address field—(Memory-to-Register) Only control addressing mod
postincrement addressing mode can be used as listed in the following t

Addressing Mode	Mode	Register	Addressing Mode	Mode
Dn	—	—	(xxx).W	111
An	—	—	(xxx).L	111
(An)	010	reg. number:An	# < data >	—
(An) +	011	reg. number:An		
– (An)	—	—		
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111

Effective Address field—(Register-to-Memory) Only control alterable ad
modes or the predecrement addressing mode can be used as liste
following table:

Addressing Mode	Mode	Register	Addressing Mode	Mode
Dn	—	—	(xxx).W	111
An	—	—	(xxx).L	111
(An)	010	reg. number:An	# < data >	—
(An) +	—	—		
– (An)	100	reg. number:An		
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—

1 — Move the listed registers from the floating-point unit to memory.

Mode field—Specifies the type of the register list and addressing mode.
00 — Static register list, predecrement addressing mode.
01 — Dynamic register list, predecrement addressing mode.
10 — Static register list, postincrement or control addressing mode.
11 — Dynamic register list, postincrement or control addressing mode.

Register List field:

Static list—contains the register select mask. If a register is to be moved, th
sponding bit in the mask is set as shown below; otherwise it is clear.

Dynamic list—contains the integer data register number, rrr, as listed in the f
table:

List Type	Register List Format							
Static, – (An)	FP7	FP6	FP5	FP4	FP3	FP2	FP1	FP0
Static, (An) + , or Control	FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7
Dynamic	0	r	r	r	0	0	0	0

The format of the dynamic list mask is the same as for the static list and is co
in the least significant eight bits of the specified main processor data register.

Programming Note: This instruction provides a very useful feature, dynamic reg
specification, that can significantly enhance system performance. If the
conventions used for procedure calls utilize the dynamic register list feat
number of floating-point data registers saved and restored can be reduced.

To utilize the dynamic register specification feature of the FMOVEM instructi
the calling and the called procedures must be written to communicate info
about register usage. When one procedure calls another, a register mask
passed to the called procedure to indicate which registers must not be alter
return to the calling procedure. The called procedure then saves only those r
that are modified and are already in use. Several techniques can be used to ut
mechanism, and an example follows.

dure. Bits 15 – 8 identify the registers in the order FP0 – FP7, and bits 7 – 0 id
registers in the order FP7 – FP0 (the two masks are required due to the differ
fer order used by the predecrement and postincrement addressing modes).
used by the calling procedure consists of simply moving the mask (which is g
at compile time) for the floating-point data registers currently in use into D7:

Calling procedure...

```
MOVE.W    #ACTIVE,D7      Load the list of FP registers that a
                          in use.
BSR       PROC_2
```

The entry code for all other procedures computes two masks. The first mask
the registers in use by the calling procedure that are used by the called proce
therefore saved and restored by the called procedure). The second mask ide
registers in use by the calling procedure that are used by the called proce
therefore not saved on entry). The appropriate registers are then stored alon
two masks:

Called procedure...

```
MOVE.W    D7,D6           Copy the list of active registers.
AND.W     #WILL_USE,D7    Generate the list of doubly-used
                          registers.
FMOVEM    D7, – (A7)      Save those registers.
MOVE.W    D7, – (A7)      Save the register list.
EOR.W     D7,D6           Generate the list of not saved act
                          registers.
MOVE.W    D6, – (A7)      Save it for later use.
```

If the second procedure calls a third procedure, a register mask is passed to
procedure that indicates which registers must not be altered by the third p
This mask identifies any registers in the list from the first procedure that were
by the second procedure, plus any registers used by the second procedure
not be altered by the third procedure.

Nested calling sequence...

```
MOVE.W    UNSAVED (A7),D7  Load the list of active registers not
                           saved at entry.
OR.W      #WILL_USE,D7     Combine with those active at this t
BSR       PROC_3
```

Upon return from a procedure, the restoration of the necessary registers foll
same convention, and the register mask generated during the save operation
is used to restore the required floating-point data registers:

Return to caller...

```
ADDQ.L    #2,A7            Discard the list of registers not sav
MOVE.B    (A7) + ,D7       Get the saved register list (pop wo
                           use byte).
FMOVEM    (A7) + ,D7       Restore the registers.
          *
          *
          *
RTS                        Return to the calling routine.
```

Assembler Syntax: FMOVEM.L < list > , < ea >
FMOVEM.L < ea > , < list >

Attributes: Size = (Long)

Description: Moves one or more 32-bit values into or out of the specified syste registers. Any combination of the three system control registers may be spec registers are always moved in the same order, regardless of the addressi used; the floating-point control register is moved first, followed by the floa status register, and the floating-point instruction address register is moved register is not selected for the transfer, the relative order of the transfer of registers is the same. The first register is transferred between the floating-poin the specified address, with successive registers located up through higher ad

For the MC68881, this instruction does not cause pending exceptions (other tocol violations) to be reported. Furthermore, a write to the floating-point con ter exception enable byte or the floating-point status register exception st connot generate a new exception, despite the value written.

When more than one register is moved, the memory or memory- alterable ad modes can be used as shown in the addressing mode tables. If the address is predecrement, the address register is first decremented by the total size o ister images to be moved (i.e., four times the number of registers), and then ters are transferred starting at the resultant address. For the postincrement ad mode, the selected registers are transferred to or from the specified address, the address register is incremented by the total size of the register images tra If a single system control register is selected, the data register direct address may be used; if the only register selected is the floating-point instruction add ister, then the address register direct addressing mode is allowed. Note that register is selected, the opcode generated is the same as for the FMOVE sing control register instruction.

register image.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
1	0	dr	REGISTER LIST			0	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field—Determines the addressing mode for the operation.

Memory-to-Register—Only control addressing modes or the postin addressing mode can be used as listed in the following table:

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

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if a single floating-point instruction address register, floating-point status register, o floating-point control register is selected.

**Only if the floating-point instruction address register is the single register selected.

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

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn],od)	—
([bd,PC],Xn,od)	—

*Only if a single floating-point control register is selected.
**Only if the floating-point instruction address register is the single register selected.

dr field—Specifies the direction of the transfer.
0 — Move the listed registers from memory to the floating-point unit.
1 — Move the listed registers from the floating-point unit to memory.

Register List field—Contains the register select mask. If a register is to be m
corresponding bit in the list is set; otherwise, it is clear. At least one regi
be specified.

Bit Number	Register
12	Floating-Point Control Register
11	Floating-Point Status Register
10	Floating-Point Instruction Address Register

Syntax: FMUL.X FPm,FPn
*FrMUL < fmt > < ea > ,FPn
*FrMUL.X FPm,FPn
where r is rounding precision, S or D
*Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa multiplies that number by the number in the destination floating-point data Stores the result in the destination floating-point data register.

FMUL will round the result to the precision selected in the floating-point control FSMUL and FDMUL will round the result to single or double precision, resp regardless of the rounding precision selected in the floating-point control regis

Operation Table:

DESTINATION		SOURCE[1]					
		+	In Range	–	+	Zero	–	+	Infinity
In Range	+	Multiply			+ 0.0 – 0.0		– 0.0 + 0.0	+ inf – inf	
	–								
Zero	+	+ 0.0 – 0.0		– 0.0 + 0.0	+ 0.0 – 0.0		– 0.0 + 0.0	NAN[2]	
	–								
Infinity	+	+ inf – inf		– inf + inf	NAN[2]			+ inf – inf	
	–								

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set for 0 x infinity; cleared otherwis
	OVFL	Refer to exception processing appropriate user's manual.
	UNFL	Refer to exception processing appropriate user's manual.
	DZ	Cleared
	INEX2	Refer to exception processing appropriate user's manual.
	INEX1	If < fmt > is packed, refer to processing in the appropriate user's cleared otherwise.
Accrued Exception Byte:	Affected as described in exception processing in t priate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			OPMODE			

If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, specifies the location of the source operand location. Only data addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)
*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

0100011	FMUL	Rounding precision specified by the floating-point control register.
1100011	FSMUL	Single-precision rounding specified.
1100111	FDMUL	Double-precision rounding specified.

Syntax: FNEG.X FPm,FPn
FNEG.X FPn
*FrNEG. < fmt > < ea > ,FPn
*FrNEG.X FPm,FPn
*FrNEG.X FPn
where r is rounding precision, S or D
*Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessary) an the sign of the mantissa. Stores the result in the destination floating-point data

FNEG will round the result to the precision selected in the floating-point control FSNEG and FDNEG will round the result to single or double precision, resp regardless of the rounding precision selected in the floating-point control regis

Operation Table:

DESTINATION	SOURCE								
	+	In Range	–	+	Zero	–	+	Infinity	
Result	Negate			– 0.0		+ 0.0	– inf		

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Cleared
	OVFL	Cleared
	UNFL	If source is an extended-denormalized number, refer to processing in the appropriate user's cleared otherwise.
	DZ	Cleared
	INEX2	Cleared
	INEX1	If < fmt > is packed, refer to processing in the appropriate user's cleared otherwise.
Accrued Exception Byte:	Affected as described in exception processing in t priate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE				

If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, specifies the location of the source operand. Only data addressing can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)
*This encoding will cause an unimplemented data type exception to allow emulation in software.

the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Opmode field—Specifies the instruction and rounding precision.

0011010	FNEG	Rounding precision specified by the floating-point control register.
1011010	FSNEG	Single-precision rounding specified.
1011110	FDNEG	Double-precision rounding specified.

Syntax: FNOP

Attributes: Unsized

Description: This instruction does not perform any explicit operation. However, it
to force synchronization of the floating- point unit with an integer unit or
processing of pending exceptions. For most floating-point instructions, the inte
is allowed to continue with the execution of the next instruction once the floati
unit has any operands needed for an operation, thus supporting concurrent e
of floating-point and integer instructions. The FNOP instruction synchron
floating-point unit and the integer unit by causing the integer unit to wait
previous floating-point instructions have completed. Execution of FNOP als
any exceptions pending from the execution of a previous floating-point instruct
processed as a preinstruction exception.

The MC68882 may not wait to begin execution of another floating- point instruc
it has completed execution of the current instruction. The FNOP instruction s
nizes the coprocessor and microprocessor unit by causing the microprocesso
wait until the current instruction (or both instructions) have completed.

The FNOP instruction also forces the processing of exceptions pending from
cution of previous instructions. This is also inherent in the way that the floati
coprocessor utilizes the M68000 family coprocessor interface. Once the floati
coprocessor has received the input operand for an arithmetic instruction, i
releases the main processor to execute the next instruction (regardless of wh
not concurrent execution is prevented for the instruction due to tracing) withou
ing the exception during the execution of that instruction. Then, when the main
sor attempts to initiate the execution of the next floating-point coprocessor ins
a preinstruction exception may be reported to initiate exception processin
exception that occurred during a previous instruction. By using the FNOP ins
the user can force any pending exceptions to be processed without perform
other operations.

Floating-Point Status Register: Not Affected.

1	1	1	1	COPROCESSOR ID			0	1	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe
instruction. Motorola assemblers default to ID = 1 for the floa
coprocessor.

NOTE

FNOP uses the same opcode as the FBcc.W < label > instruction, with cc = F (nontrapping false) and < label > = + 2 (which results in a displacement of 0).

Syntax: FREM.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the modulo remainder of the number in the destination floating-po register, using the source operand as the modulus. Stores the result in the de floating-point data register and stores the sign and seven least significant bi quotient in the floating-point status register quotient byte (the quotient is the FPn ÷ Source). The IEEE remainder function is defined as:

$$FPn - (Source \times N)$$

where N = INT (FPn ÷ Source) in the round-to-nearest mode.

The FREM function is not defined for a source operand equal to zero or for a tion operand equal to infinity. Note that this function is not the same as the instruction, which uses the round-to-zero mode and thus returns a remainder th ferent from the remainder required by the IEEE *Specification for Binary Floati Arithmetic*.

Operation Table:

DESTINATION		SOURCE[1]		
		+ In Range –	+ Zero# –	+ Infinity
In Range	+ –	IEEE Remainder	NAN[2]	FPn[2]
Zero	+	+ 0.0	NAN[2]	+ 0.0
	–	– 0.0		– 0.0
Infinity	+ –	NAN[2]	NAN[2]	NAN†[2]

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.
3. Returns the value of FPn before the operation. However, the result is processed by the normal instruction termination procedure to round it as required. Thus, an overflow and/or inexact result may occur if the rounding precision has been changed to a smaller size since the FPn value was loaded.

Quotient Byte:	Loaded with the sign and least significant seven b qotient (FPn ÷ Source). The sign of the quotie exclusive-OR of the sign bits of the source and de operands.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if the source is zero or the dest infinity; cleared otherwise.
	OVFL	Cleared
	UNFL	Refer to underflow in the appropria manual.
	DZ	Cleared
	INEX2	Cleared
	INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.
Accrued Exception Byte:	Affected as described in IEEE exception and trap bility in the appropriate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	0	0	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regist

Syntax: FSCALE.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to an integer (if necessary) and adds th to the destination exponent. Stores the result in the destination floating-p register. This function has the effect of multiplying the destination by 2^{Sou} much faster than a multiply operation when the source is an integer value.

The floating-point coprocessor assumes that the scale factor is an integer val the operation is executed. If not, the value is chopped (i.e., rounded using th to-zero mode) to an integer before it is added to the exponent. When the abso of the source operand is $\geq 2^{14}$, an overflow or underflow always results.

Operation Table:

DESTINATION	SOURCE[1]					
	+	In Range	–	+	Zero	–	+	Infinity
In Range + –	Scale Exponent			FPn[2]			NAN[3]	
Zero + –	+ 0.0		– 0.0	+ 0.0		– 0.0	NAN[3]	
Infinity + –	+ inf		– inf	+ inf		– inf	NAN[3]	

NOTES:

1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Returns the value of FPn before the operation. However, the result is processed by the normal instruction termination procedure to round it as required. Thus, an overflow and/or inexact result may occur if the rounding precision has been changed to a smaller size since the FPn value was loaded.
3. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if the source operand is ± infinity; otherwise.
	OVFL	Refer to overflow in the appropriat manual.
	UNFL	Refer to underflow in the appropriat manual.
	DZ	Cleared
	INEX2	Cleared
	INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.
Accrued Exception Byte:	Affected as described in IEEE exception and trap bility in the appropriate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

the following table:

Addressing Mode	Mode	Register	Addressing Mode	Mode
Dn*	000	reg. number:Dn	(xxx).W	111
An	—	—	(xxx).L	111
(An)	010	reg. number:An	# < data >	111
(An) +	011	reg. number:An		
– (An)	100	reg. number:An		
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis

Assembler Syntax: FScc. < size > < ea >

Attributes: Size = (Byte)

Description: If the specified floating-point condition is true, sets the byte integer op
the destination to TRUE (all ones); otherwise, sets the byte to FALSE (all zer
conditional specifier cc may select any one of the 32 floating-point conditional
described in **Table 3-23 Floating-Point Conditional Tests**.

Floating-Point Status Register:

Condition Codes:	Not affected.	
Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Set if the NAN condition code is set condition selected is an IEEE nonaw
	SNAN	Not Affected.
	OPERR	Not Affected.
	OVFL	Not Affected.
	UNFL	Not Affected.
	DZ	Not Affected.
	INEX2	Not Affected.
	INEX1	Not Affected.
Accrued Exception Byte:	The IOP bit is set if the BSUN bit is set in the e byte. No other bit is affected.	

1	1	1	1	COPROCESSOR ID			0	0	1	EFFECTIVE ADDRESS MODE | REG
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICA

Instruction Fields:

Effective Address field—Specifies the addressing mode for the byte integer
Only data alterable addressing modes can be used as listed in the follow

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn],od)	—
([bd,PC],Xn,od)	—

Conditional Predicate field—Specifies one of 32 conditional tests as define
Conditional Testing.

NOTE

When a BSUN exception occurs, a preinstruction exception is taken. If the exception handler returns without modifying the image of the program counter on the stack frame (to point to the instruction following the FScc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception occurs again immediately upon return to the routine that caused the exception.

Syntax: FSGLDIV.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessary) and that number into the number in the destination floating-point data register. St result in the destination floating-point data register, rounded to single precision the current rounding precision). This function is undefined for 0 ÷ 0 and infinity ÷

Both the source and destination operands are assumed to be representable in gle-precision format. If either operand requires more than 24 bits of mantis accurately represented, the extraneous mantissa bits are trancated prior to sion, hence the accuracy of the result is not guaranteed. Furthermore, the resu nent may exceed the range of single precision, regardless of the rounding p selected in the floating-point control register mode control byte. Refer to **3.6.1 flow, Round, Overflow** for more information.

The accuracy of the result is not affected by the number of mantissa bits req represent each input operand since the input operands just change to extende sion. The result mantissa is rounded to single precision, and the result exp rounded to extended precision, despite the rounding precision selected in the point control register.

Operation Table:

DESTINATION		SOURCE[3,1]					
		+	In Range	–	+	Zero	–	+	Infinity	
In Range	+	Divide (Single Precision)			+ inf[2]		– inf[2]	+ 0.0		–
	–				– inf[2]		+ inf[2]	– 0.0		+
Zero	+	+ 0.0		– 0.0	NAN[3]			+ 0.0		–
	–	– 0.0		+ 0.0				– 0.0		+
Infinity	+	+ inf		– inf	+ inf		– inf	NAN[3]		
	–	– inf		+ inf	– inf		+ inf			

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the DZ bit in the floating-point status register exception byte.
3. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set for 0 ÷ 0 or infinity ÷ infinity.
	OVFL	Refer to overflow in the appropria manual.
	UNFL	Refer to underflow in the appropria manual.
	DZ	Set if the source is zero and the des in range; cleared otherwise.
	INEX2	Refer to inexact result in the ap user's manual.
	INEX1	If < fmt > is packed, refer to the ap user's manual for inexact result or input; cleared otherwise.
Accrued Exception Byte:	Affected as described in IEEE exception and trap bility in the appropriate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REG	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regist

Syntax: FSGLMUL.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess multiplies that number by the number in the destination floating-point data Stores the result in the destination floating-point data register, rounded precision (regardless of the current rounding precision).

Both the source and destination operands are assumed to be representable i gle-precision format. If either operand requires more than 24 bits of manti accurately represented, the extraneous mantissa bits are truncated prior to pliction; hence, the accuracy of the result is not guaranteed. Furthermore, exponent may exceed the range of single precision, regardless of the round sion selected in the floating-point control register mode control byte. Refe **Underflow, Round, Overflow** for more information.

Operation Table:

DESTINATION		SOURCE[1]					
		+	In Range	–	+	Zero	–	+	Infinity
In Range	+	Multiply (Single Precision)			+ 0.0	– 0.0	+ inf		
	–				– 0.0	+ 0.0	– inf		
Zero	+	+ 0.0		– 0.0	+ 0.0		– 0.0	NAN[2]	
	–	– 0.0		+ 0.0	– 0.0		+ 0.0		
Infinity	+	+ inf		– inf	NAN			+ inf	
	–	– inf		+ inf				– inf	

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

NOTE

The input operand mantissas truncate to single precision before the multiply operation. The result mantissa rounds to single precision despite the rounding precision selected in the floating-point control register.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if one operand is zero and the infinity; cleared otherwise.
	OVFL	Refer to overflow in the appropriat manual.
	UNFL	Refer to underflow in the appropriat manual.
	DZ	Cleared
	INEX2	Refer to inexact result in the app user's manual.
	INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriate manual; cleared otherwise.
Accrued Exception Byte:	Affected as described in IEEE exception and trap bility in the appropriate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	0	0	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

the following table:

Addressing Mode	Mode	Register	Addressing Mode	Mode
Dn*	000	reg. number:Dn	(xxx).W	111
An	—	—	(xxx).L	111
(An)	010	reg. number:An	# < data >	111
(An) +	011	reg. number:An		
– (An)	100	reg. number:An		
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis

Syntax: FSIN.X FPm,FPn
FSIN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the sine of that number. Stores the result in the destination floati data register. This function is not defined for source operands of ± infinity. If th operand is not in the range of [– 2π... + 2π], the argument is reduced to wi range before the sine is calculated. However, large arguments may lose a during reduction, and very large arguments (greater than approximately 10^{20} accuracy. The result is in the range of [– 1... + 1].

Operation Table:

DESTINATION	SOURCE[1]					
	+	In Range	–	+	Zero	–	+	Infinity
Result	Sine			+ 0.0		– 0.0	NAN[2]	

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

OPERR	Set if the source is ± infinity, otherwise.
OVFL	Cleared
UNFL	Refer to underflow in the appropria manual.
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REG	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	1	

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, then the input op
taken from the specified floating-point data register, and the result is wri
the same register. If the single register syntax is used, Motorola assem
the source and destination fields to the same value.

Assembler Syntax: FSINCOS. < fmt > < ea > ,FPc,FPs
FSINCOS.X FPm,FPc,FPs

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess calculates both the sine and the cosine of that number. Calculates both simultaneously; thus, this instruction is significantly faster than performing FSIN and FCOS instructions. Loads the sine and cosine results into the d floating-point data register. Sets the condition code bits according to the sine FPs and FPc are specified to be the same register, the cosine result is first lo the register and then is overwritten with the sine result. This function is not d source operands of ± infinity.

If the source operand is not in the range of [– 2π... + 2π], the argument is re within that range before the sine and cosine are calculated. However, large a may lose accuracy during reduction, and very large arguments (greater than mately 10^{20}) lose all accuracy. The results are in the range of [– 1... + 1].

Operation Table:

DESTINATION	SOURCE[1]		
	+ In Range –	+ Zero –	+ Infinity
FPs	Sine	+ 0.0 – 0.0	NAN[2]
FPc	Cosine	+ 1.0	NAN[2]

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

sine result).

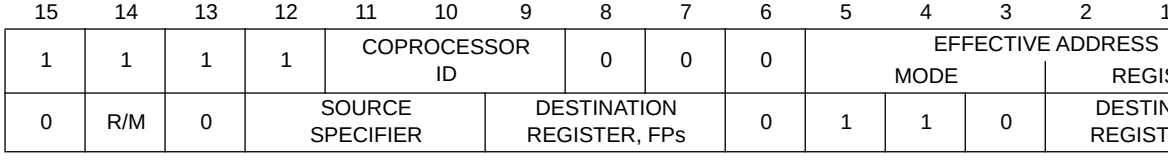
Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if the source is ± infinity; otherwise.
OVFL	Cleared
UNFL	Set if a sine underflow occurs, in wh the cosine result is 1. Cosine underflow. Refer to underflow appropriate user's manual.
DZ	Cleared
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap c bility in the appropriate user's manual.

Instruction Format:



Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

the following table:

Addressing Mode	Mode	Register	Addressing Mode	Mode
Dn*	000	reg. number:Dn	(xxx).W	111
An	—	—	(xxx).L	111
(An)	010	reg. number:An	# < data >	111
(An) +	011	reg. number:An		
– (An)	100	reg. number:An		
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register, FPc field—Specifies the destination floating- point data
FPc. The cosine result is stored in this register.

If R/M = 0 and the source register field is equal to either of the destination
fields, the input operand is taken from the specified floating-point data register
appropriate result is written into the same register.

Syntax: FSINH.X FPm,FPn
FSINH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess calculates the hyperbolic sine of that number. Stores the result in the d floating-point data register.

Operation Table:

DESTINATION	SOURCE		
	+ In Range –	+ Zero –	+ Infinity
Result	Hyperbolic Sine	+ 0.0 – 0.0	+ inf

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Refer to overflow in the appropria manual.
UNFL	Refer to underflow in the appropria manual.
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	1

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

Effective Address field—Determines the addressing mode for external operan
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, the input operan
from the specified floating-point data register, and the result is writte
same register. If the single register syntax is used, Motorola assemble
source and destination fields to the same value.

Syntax: FSQRT.X FPm,FPn
FSQRT.X FPn
*FrSQRT. < fmt > < ea > ,FPn
*FrSQRT FPm,FPn
*FrSQRT FPn
where r is rounding precision, S or D

*Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the square root of that number. Stores the result in the destination point data register. This function is not defined for negative operands.

FSQRT will round the result to the precision selected in the floating-point cor ister. FSFSQRT and FDFSQRT will round the result to single or double p respectively, regardless of the rounding precision selected in the floating-poin register.Operation Table:

DESTINATION	SOURCE[1]					
	+	In Range	–	+	Zero	–	+	Infinity
Result	$\sqrt{x}$		NAN[2]	+ 0.0		– 0.0	+ inf	N

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if the source operand is not ze negative; cleared otherwise.
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to exception processing appropriate user's manual.
	INEX1	If < fmt > is packed, refer to processing in the appropriate user's cleared otherwise.
Accrued Exception Byte:	Affected as described in exception processing in t priate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			OPMODE			

If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, specifies the location of the source operand. Only data addressing can be used as listed in the following table:

Addressing Mode	Mode	Register		Addressing Mode	Mode
Dn*	000	reg. number:Dn		(xxx).W	111
An	—	—		(xxx).L	111
(An)	010	reg. number:An		# < data >	111
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An		(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An		(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An		(bd,PC,Xn)	111
([bd,An,Xn],od)	110	reg. number:An		([bd,PC,Xn],od)	111
([bd,An],Xn,od)	110	reg. number:An		([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)
*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

same register. If the single register syntax is used, Motorola assemble source and destination fields to the same value.

Opmode field—Specifies the instruction and rounding precision.

0000100	FSQRT	Rounding precision specified by the floating-point control register.
1000001	FSSQRT	Single-precision rounding specified.
1000101	FDSQRT	Double-precision rounding specified.

Syntax: FSUB. < fmt > < ea > ,FPn
FSUB.X FPm,FPn
*FrSUB. < fmt > < ea > ,FPn
*FrSUB.X FPm,FPn
where r is rounding precision, S or D

*Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa subtracts that number from the number in the destination floating-point data Stores the result in the destination floating-point data register.

Operation Table:

DESTINATION		SOURCE[1] + In Range	–	+ Zero	–	+ Infinity
In Range	+	Subtract		Subtract		– inf
	–					
Zero	+	Subtract		+ 0.0[2]	+ 0.0	– inf
	–			+ 0.0	+ 0.0[2]	
Infinity	+	+ inf		+ inf		NAN[2]
	–	– inf		– inf		– inf N

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Returns + 0.0 in rounding modes RN, RZ, and RP; returns – 0.0 in RM.
3. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Set if both the source and destin like-signed infinities; cleared otherw
OVFL	Refer to exception processing appropriate user's manual.
UNFL	Refer to exception processing appropriate user's manual.
DZ	Cleared
INEX2	Refer to exception processing appropriate user's manual.
INEX1	If < fmt > is packed, refer to processing in the appropriate user's cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			OPMODE			

If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, specifies the location of the source operand. Only data addressing can be used as listed in the following table:

Addressing Mode	Mode	Register		Addressing Mode	Mode
Dn*	000	reg. number:Dn		(xxx).W	111
An	—	—		(xxx).L	111
(An)	010	reg. number:An		# < data >	111
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An		(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An		(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An		(bd,PC,Xn)	111
([bd,An,Xn],od)	110	reg. number:An		([bd,PC,Xn],od)	111
([bd,An],Xn,od)	110	reg. number:An		([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)
*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

0101000	FSUB	Rounding precision specified by the floating- point control register.
1101000	FSSUB	Single-precision rounding specified.
1101100	FDSUB	Double-precision rounding specified.

Syntax: FTAN.X FPm,FPn
FTAN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates the tangent of that number. Stores the result in the destination floati data register. This function is not defined for source operands of ± infinity. If th operand is not in the range of [– π/2... + π/2], the argument is reduced to wi range before the tangent is calculated. However, large arguments may lose a during reduction, and very large arguments (greater than approximately 10^{20} accuracy.

Operation Table:

DESTINATION	SOURCE[1]					
	+	In Range	–	+	Zero	–	+	Infinity
Result	Tangent			+ 0.0		– 0.0	NAN[2]	

NOTES:
1. If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.
2. Sets the OPERR bit in the floating-point status register exception byte.

Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to **1.6.5 Not-A-Numbers**.
	OPERR	Set if the source is ± infinity; otherwise.
	OVFL	Refer to overflow in the appropria manual.
	UNFL	Refer to underflow in the appropria manual.
	DZ	Cleared
	INEX2	Refer to inexact result in the ap user's manual.
	INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.
Accrued Exception Byte:	Affected as described in IEEE exception and trap bility in the appropriate user's manual.	

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REG	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	1	

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:
000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, the input operand
from the specified floating-point data register, and the result is written
same register. If the single register syntax is used, Motorola assemblers
source and destination fields to the same value.

Syntax: FTANH.X FPm,FPn
FTANH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necess calculates the hyperbolic tangent of that number. Stores the result in the d floating-point data register.

Operation Table:

DESTINATION	SOURCE		
	+ In Range –	+ Zero –	+ Infinity
Result	Hyperbolic Tangent	+ 0.0 – 0.0	+ 1.0

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Cleared
UNFL	Refer to underflow in the appropria manual.
DZ	Cleared
INEX2	Refer to inexact result in the ap user's manual.
INEX1	If < fmt > is packed, refer to inexact decimal input in the appropriat manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	0	1	0	0

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exec instruction. Motorola assemblers default to ID = 1 for the floati coprocessor.

Effective Address field—Determines the addressing mode for external operan
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

R/M field—Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is < ea > to register.

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regi
M = 0 and the source and destination fields are equal, the input operan
from the specified floating-point data register, and the result is writte
same register. If the single register syntax is used, Motorola assemble
source and destination fields to the same value.

Syntax: FTENTOX.X FPm,FPn
FTENTOX.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates 10 to the power of that number. Stores the result in the destination point data register.

Operation Table:

DESTINATION	SOURCE		
	+ In Range –	+ Zero –	+ Infinity
Result	10^X	+ 1.0	+ inf +

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Refer to overflow in the appropriat manual.
UNFL	Refer to underflow in the appropriat manual.
DZ	Cleared
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to the app user's manual inexact result on input; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE		REG
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	0

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

Effective Address field—Determines the addressing mode for external opera
If R/M = 0, this field is unused and should be all zeros.
If R/M = 1, this field is encoded with an M68000 family addressing mode as the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, the input operand
from the specified floating-point data register, and the result is written
same register. If the single register syntax is used, Motorola assemblers
source and destination fields to the same value.

Assembler Syntax: FTRAPcc
FTRAPcc.W # < data >
FTRAPcc.L # < data >

Attributes: Size = (Word, Long)

Description: If the selected condition is true, the processor initiates exception pr
A vector number is generated to reference the TRAPcc exception vector. Th
program counter points to the next instruction. If the selected condition is not t
is no operation performed and execution continues with the next instr
sequence. The immediate data operand is placed in the word(s) follo
conditional predicate word and is available for user definition for use within
handler.

The conditional specifier cc selects one of the 32 conditional tests define
Conditional Testing.

Floating-Point Status Register:

Condition Codes:	Not affected.	
Quotient Byte:	Not affected.	
Exception Byte:	BSUN	Set if the NAN condition code is se condition selected is an IEEE nonav
	SNAN	Not Affected.
	OPERR	Not Affected.
	OVFL	Not Affected.
	UNFL	Not Affected.
	DZ	Not Affected.
	INEX2	Not Affected.
	INEX1	Not Affected.
Accrued Exception Byte:	The IOP bit is set if the BSUN bit is set in the byte; no other bit is affected.	

1	1	1	1	COPROCESSOR ID			0	0	1	1	1	1	MO
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICAT		
16-BIT OPERAND OR MOST SIGNIFICANT WORD OF 32-BIT OPERAND (IFNEEDED)												
LEAST SIGNIFICANT WORD OR 32-BIT OPERAND (IF NEEDED)												

Instruction Fields:

Mode field—Specifies the form of the instruction.

010 — The instruction is followed by a word operand.
011 — The instruction is followed by a long-word operand.
100 — The instruction has no operand.

Conditional Predicate field—Specifies one of 32 conditional tests as described **Conditional Testing**.

Operand field—Contains an optional word or long-word operand that is user d

NOTE

When a BSUN exception occurs, a preinstruction exception is taken by the main processor. If the exception handler returns without modifying the image of the program counter on the stack frame (to point to the instruction following the FTRAPcc), it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap), or the exception occurs again immediately upon return to the routine that caused the exception.

Syntax: FTST.X FPm

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Pack

Description: Converts the source operand to extended precision (if necessary) an condition code bits according to the data type of the result.

Operation Table: The contents of this table differfromtheother operation tables. A an entry of this table indicates that the designated condition code bit is alwa the FTST operation. All unspecified condition code bits are cleared d operation.

DESTINATION	SOURCE							
	+	In Range	–	+	Zero	–	+	Infinity
Result	none		N	Z		NZ	I	

NOTE: If the source operand is a NAN, set the NAN condition code bit. If the source operand is an SNAN, set the SNAN bit in the floating-point status register exception byte

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testi**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Cleared
INEX1	If < fmt > is packed, refer to processing in the appropriate user's cleared otherwise.

Accrued Exception Byte: Affected as described in exception processing in t priate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	1	1	1	0	1

Instruction Fields:

Effective Address field—Determines the addressing mode for external operan

If R/M = 0, this field is unused and should be all zeros.

If R/M = 1, specifies the location of the source operand. Only data addressing can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn],od)	111	
([bd,PC],Xn,od)	111	

*Only if < fmt > is byte, word, long, or single.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)*
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

Destination Register field—Since the floating-point unit uses a common word format for all of the arithmetic instructions (including FTST), th treated in the same manner for FTST as for the other arithmetic instructi though the destination register is not modified. This field should be set maintain compatibility with future devices; however, the floating-point not signal an illegal instruction trap if it is not zero.

Syntax: FTWOTOX.X FPm,FPn
FTWOTOX.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packe

Description: Converts the source operand to extended precision (if necessa calculates two to the power of that number. Stores the result in the destination point data register.

Operation Table:

DESTINATION	SOURCE								
	+	In Range	–	+	Zero	–	+	Infinity	
Result	2^x			+ 1.0			+ inf		+

NOTE: If the source operand is a NAN, refer to **1.6.5 Not-A-Numbers** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **3.6.2 Conditional Testin**

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **1.6.5 Not-A-Numbers**.
OPERR	Cleared
OVFL	Refer to overflow in the appropriat manual.
UNFL	Refer to underflow in the appropriat manual.
DZ	Cleared
INEX2	Refer to inexact result in the app user's manual.
INEX1	If < fmt > is packed, refer to inexact r decimal input in the appropriate manual; cleared otherwise.

Accrued Exception Byte: Affected as described in IEEE exception and trap bility in the appropriate user's manual.

1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE			REG
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	1	0	0

Instruction Fields:

Coprocessor ID field—Specifies which coprocessor in the system is to exe instruction. Motorola assemblers default to ID = 1 for the floa coprocessor.

Effective Address field—Determines the addressing mode for external opera

If R/M = 0, this field is unused and should be all zeros.

If R/M = 1, this field is encoded with an M68000 family addressing mode a the following table:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111
(bd,PC,Xn)	111
([bd,PC,Xn],od)	111
([bd,PC],Xn,od)	111

*Only if < fmt > is byte, word, long, or single.

Source Specifier field—Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register.
If R/M = 1, specifies the source data format:

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

Destination Register field—Specifies the destination floating- point data regis
M = 0 and the source and destination fields are equal, the input operand
from the specified floating-point data register, and the result is written
same register. If the single register syntax is used, Motorola assemblers
source and destination fields to the same value.

SECTION 6
SUPERVISOR (PRIVILEGED) INSTRUCTIONS

This section contains information about the supervisor privileged instructions M68000 family. Each instruction is described in detail, and the instruction descrip arranged in alphabetical order by instruction mnemonic.

Any differences within the M68000 family of instructions are identified in the instruc instruction only applies to a certain processor or processors, the processor(s) instruction pertains to is identified under the title of the instruction. For example:

Invalidate Cache Lines
(MC68040)

All references to the MC68000, MC68020, and MC68030 include reference corresponding embedded controllers, MC68EC000, MC68EC020, and MC68E references to the MC68040 include the MC68LC040 and MC68EC040. This throughout this section unless otherwise specified.

If the instruction applies to all the M68000 family but a processor or processors n different instruction field, instruction format, etc., the differences will be identified paragraph. For example:

MC68020, MC68030 and MC68040 only

(bd,An,Xn)*	110	reg. number: An	(bd,PC,Xn)*	—	—

*Can be used with CPU32 processo

The following instructions are listed separately for each processor due to t differences involved within the instruction:

PFLUSH	Flush ATC Entries
PMOVE	Move PMMU Register
PTEST	Test Logical Address

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

Then Source L SR → SR
ELSE TRAP

Assembler Syntax: ANDI # < data > ,SR

Attributes: size = (word)

Description: Performs an AND operation of the immediate operand with the conten status register and stores the result in the status register. All implemented bi status register are affected.

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:

5	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	0	0	0	1	0	0	1	1	1	1	1	0
16-BIT WORD DATA														

Assembler Syntax:

CINVL < caches > ,(An)
CINVP < caches > ,(An)
CINVA < caches >

Where < caches > specifies the instruction cache, data cache, both caches, or neither cache.

Attributes: Unsized

Description: Invalidates selected cache lines. The data cache, instruction ca caches, or neither cache can be specified. Any dirty data in data cache invalidate are lost; the CPUSH instruction must be used when dirty data contained in the data cache.

Specific cache lines can be selected in three ways:

1. CINVL invalidates the cache line (if any) matching the physical addres specified address register.
2. CINVP invalidates the cache lines (if any) matching the physical mem in the specified address register. For example, if 4K-byte page sizes a ed and An contains $12345000, all cache lines matching page $1234 validate.
3. CINVA invalidates all cache entries.

Condition Codes:

Not affected.

1	1	1	1	0	1	0	0	CACHE	0	SCOPE	REGIS

Instruction Fields:

Cache field—Specifies the Cache.
- 00—No Operation
- 01—Data Cache
- 10—Instruction Cache
- 11—Data and Instruction Caches

Scope field—Specifies the Scope of the Operation.
- 00—Illegal (causes illegal instruction trap)
- 01—Line
- 10—Page
- 11—All

Register field—Specifies the address register for line and page operations. operations, the low-order bits 3–0 of the address are don‘t cares. Bits 11–0 of the address are don‘t care for 4K-byte or 8K-byte page ope respectively.

ELSE TRAP

Assembler Syntax: cpRESTORE < ea >

Attributes: Unsized

Description: Restores the internal state of a coprocessor usually after it has been a preceding cpSAVE instruction.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3	2
1	1	1	1	COPROCESSOR ID	1	0	1	EFFECTIVE ADDRESS MODE	REG

Coprocessor ID field—Identifies the coprocessor that is to be restored. Coproc
of 000 results in an F-line exception for the MC68030.

Effective Address field—Specifies the location where the internal state
coprocessor is located. Only postincrement or control addressing modes
used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
— (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn], od)	110	reg. number:An
([bd,An],Xn,od)	110	reg.number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn], od)	111	
([bd,PC],Xn, od	111	

NOTE

If the format word returned by the coprocessor indicates “come again”, pending interrupts are not serviced.

Assembler Syntax: cpSAVE < ea >

Attributes: Unsized

Description: Saves the internal state of a coprocessor in a format that can be re
a cpRESTORE instruction.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3 2
1	1	1	1	COPROCESSOR ID	1	0	0	EFFECTIVE ADDRESS: MODE | REG

Instruction Fields:

Coprocessor ID field—Identifies the coprocessor for this operation. Coproces
000 results in an F-line exception for the MC68030.

Effective Address field—Specifies the location where the internal stat
coprocessor is to be saved. Only predecrement or control alterable ad
modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
— (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn], od)	110	reg. number:An
([bd,An],Xn, od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn],od)	—
([bd,PC],Xn, od)	—

Invalidate Selected Cache Lines
ELSE TRAP

Assembler Syntax:
CPUSHL < caches > ,(An)
CPUSHP < caches > ,(An)
CPUSHA < caches >

Where < caches > specifies the instruction cache, data cache, both caches, or neither cache.

Attributes: Unsized

Description: Pushes and then invalidates selected cache lines. The DATA instruction cache, both caches, or neither cache can be specified. When the da is specified, the selected data cache lines are first pushed to memory (if they dirty DATA) and then invalidated. Selected instruction cache lines are invalida

Specific cache lines can be selected in three ways:

1. CPUSHL pushes and invalidates the cache line (if any) matching the p address in the specified address register.
2. CPUSHP pushes and invalidates the cache lines (if any) matching the p memory page in the specified address register. For example, if 4K-byte sizes are selected and An contains $12345000, all cache lines matchin $12345000 are selected.
3. CPUSHA pushes and invalidates all cache entries.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7 6	5	4 3	2 1
1	1	1	1	0	1	0	0	CACHE	1	SCOPE	REGI

00—No Operation
01—Data Cache
10—Instruction Cache
11—Data and Instruction Caches

Scope field—Specifies the Scope of the Operation.
00—Illegal (causes illegal instruction trap)
01—Line
10—Page
11—All

Register field—Specifies the address register for line and page operations
operations, the low-order bits 3–0 of the address are don‘t care. Bits 11
0 of the address are don‘t care for 4K-byte or 8K-byte page op
respectively.

Then Source ⊕ SR → SR
ELSE TRAP

Assembler Syntax: EORI # < data > ,SR

Attributes: Size = (Word)

Description: Performs an exclusive-OR operation on the contents of the status using the immediate operand and stores the result in the status regi implemented bits of the status register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X—Changed if bit 4 of immediate operand is one; unchanged otherwise.

N—Changed if bit 3 of immediate operand is one; unchanged otherwise.

Z—Changed if bit 2 of immediate operand is one; unchanged otherwise.

V—Changed if bit 1 of immediate operand is one; unchanged otherwise.

C—Changed if bit 0 of immediate operand is one; unchanged otherwise.

Instruction Format:

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

ELSE TRAP

Assembler Syntax: FRESTORE < ea >

Attributes: Unsized

Description: Aborts the execution of any floating-point operation in progress an new floating-point unit internal state from the state frame located at the address. The first word at the specified address is the format word of the sta It specifies the size of the frame and the revision number of the floating-poin created it. A format word is invalid if it does not recognize the size of the fra revision number does not match the revision of the floating-point unit. If the for is invalid, FRESTORE aborts, and a format exception is generated. If the for is valid, the appropriate state frame is loaded, starting at the specified loc proceeding through higher addresses.

The FRESTORE instruction does not normally affect the programmer's model of the floating-point coprocessor, except for the NULL state size, as describe It is only for restoring the user invisible portion of the machine. The FR instruction is used with the FMOVEM instruction to perform a full context rest the floating-point unit, including the floating- point data registers and syste registers. To accomplish a complete restoration, the FMOVEM instructions executed to load the programmer's model, followed by the FRESTORE inst load the internal state and continue any previously suspended operation.

STORE operation with this size state frame is equivalent to a hardwa
of the floating-point unit. The programmer's model is set to the res
with nonsignaling NANs in the floating-point data registers and zer
floating-point control register, floating-point status register, and
point instruction address register. (Thus, it is unnecessary to load
grammer's model before this operation.)

IDLE: This state frame is 4 bytes long in the MC68040, 28 ($1C) bytes lo
MC68881, and 60 ($3C) bytes long in the MC68882. An FRESTOR
ation with this state frame causes the floating-point unit to be restor
idle state, waiting for the initiation of the next instruction, with no ex
pending. The programmer's model is not affected by loading this
state frame.

UNIMP: This state frame is generated only by the MC68040. It is 48 ($30) by
An FSAVE that generates this size frame indicates either an unimple
floating-point instruction or only an E1 exception is pending. This
never generated when an unsupported data type exception is pendi
E3 exception is pending. If both E1 and E3 exceptions are pending,
frame is generated.

BUSY: This state frame is 96 ($60) bytes long in the MC68040, 184 ($B8) by
in the MC68881, and 216 ($D8) bytes long in the MC68882. An FRE
operation with this size state frame causes the floating-point un
restored to the busy state, executing the instructions that were sus
by a previous FSAVE operation. The programmer's model is not aff
loading this type of state frame; however, the completion of the sus
instructions after the restore is executed may modify the progra
model.

Floating-Point Status Register: Cleared if the state size is NULL; otherwise, not a

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			1	0	1	EFFECTIVE ADDRESS MODE			REG

Instruction Field:

Effective Address field—Determines the addressing mode for the state fra
postincrement or control addressing modes can be used as listed in the
table:

Addressing Mode	Mode	Register		Addressing Mode	Mode
Dn	—	—		(xxx).W	111
An	—	—		(xxx).L	111
(An)	010	reg. number:An		# < data >	—
(An) +	011	reg. number:An			
—(An)	—	—			
(d_{16},An)	101	reg. number:An		(d_{16},PC)	111
(d_8,An,Xn)	110	reg. number:An		(d_8,PC,Xn)	111
(bd,An,Xn)	110	reg. number:An		(bd,PC,Xn)	111
([bd,An,Xn], od)	110	reg. number:An		([bd,PC,Xn], od)	111
([bd,An],Xn, od)	110	reg. number:An		([bd,PC],Xn, od)	111

Assembler Syntax: FSAVE < ea >

Attributes: Unsized

Description: FSAVE allows the completion of any floating-point operation in prog the MC68040. It saves the internal state of the floating-point unit in a stat located at the effective address. After the save operation, the floating-point unit idle state, waiting for the execution of the next instruction. The first word writte state frame is the format word specifying the size of the frame and the revision of the floating-point unit.

Any floating-point operations in progress when an FSAVE instruction is enco can be completed before the FSAVE executes, saving an IDLE state frame. E of instructions already in the floating-point unit pipeline continues until complet instructions in the pipeline or generation of an exception by one of the instruct IDLE state frame is created by the FSAVE if no exceptions occurred; othe BUSY or an UNIMP stack frame is created.

FSAVE suspends the execution of any operation in progress and saves the state in a state frame located at the effective address for the MC68881/MC688 the save operation, the floating-point coprocessor is in the idle state, waiting execution of the next instruction. The first word written to the state frame is th word, specifying the size of the frame and the revision number of the floati coprocessor. The microprocessor unit initiates the FSAVE instruction by rea floating-point coprocessor save CIR. The floating-point coprocessor save encoded with a format word that indicates the appropriate action to be take main processor. The current implementation of the floating-point coprocesso returns one of five responses in the save CIR:

Value	Definition
$0018	Save NULL state frame
$0118	Not ready, come again
$0218	Illegal, take format exception
$XX18	Save IDLE state frame
$XXB4	Save BUSY state frame

NOTE: XX is the floating-point coprocessor version number.

word to cause the main processor to wait while an internal operation complet
sible, to allow an IDLE frame rather than a BUSY frame to be saved. The illeg
word aborts an FSAVE instruction that is attempted while the floating-point c
sor executes a previous FSAVE instruction. All other format words cause the
cessor unit to save the indicated state frame at the specified address. For st
details see state frames in the appropriate user's manual.

The following state frames apply to both the MC68040 and the MC68881/MC

NULL: This state frame is 4 bytes long. An FSAVE instruction that gene
state frame indicates that the floating-point unit state has not been
since the last hardware reset or FRESTORE instruction with a NU
frame. This indicates that the programmer's model is in the reset s
nonsignaling NANs in the floating-point data registers and zeros in
ing- point control register, floating-point status register, and floa
instruction address register. (Thus, it is not necessary to save the
mer's model.)

IDLE: This state frame is 4 bytes long in the MC68040, 28 ($1C) bytes lo
MC68881, and 60 ($3C) bytes long in the MC68882. An FSAVE in
that generates this state frame indicates that the floating-point uni
in an idle condition and is without any pending exceptions waiting f
tiation of the next instruction.

UNIMP: This state frame is generated only by the MC68040. It is 48 ($30) by
An FSAVE that generates this size frame indicates either an unimp
floating-point instruction or that only an E1 exception is pending. T
is never generated when an unsupported data type exception
exception is pending. If both E1 and E3 exceptions are pending,
frame is generated.

BUSY: This state frame is 96 ($60) bytes long in the MC68040, 184 ($B8) b
in the MC68881, and 216 ($D8) bytes long in the MC68882. An
instruction that generates this size state frame indicates that the
point unit encountered an exception while attempting to complete th
tion of the previous floating-point instructions.

unit that includes the floating-point data registers and system control regis
accomplish a complete context save, first execute an FSAVE instruction to s
the current operation and save the internal state, then execute the app
FMOVEM instructions to store the programmer's model.

Floating-Point Status Register: Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1
1	1	1	1	COPROCESSOR ID	1	0	0	EFFECTIVE ADDRESS MODE	REGIS

Instruction Field:

Effective Address field—Determines the addressing mode for the state fram
predecrement or control alterable addressing modes can be used as liste
following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn] ,od)	—	
([bd,PC],Xn ,od)	—	

Operation: If Supervisor State
Then SR → Destination
Else TRAP

Assembler Syntax: MOVE SR, < ea >

Attributes: Size = (Word)

Description: Moves the data in the status register to the destination loca
destination is word length. Unimplemented bits are read as zeros.

Condition Codes:

Not affected.

Instruction Format:

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

Instruction Field:

Effective Address field—Specifies the destination location. Only data a 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
(xxx).W	111
(xxx).L	111
# < 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)	—	

*Available for the CPU32.

NOTE

Use the MOVE from CCR instruction to access only the condition codes.

Then Source → SR
Else TRAP

Assembler Syntax: MOVE < ea > ,SR

Attributes: Size = (Word)

Description: Moves the data in the source operand to the status register. Th operand is a word, and all implemented bits of the status register are affecte

Condition Codes:

Set according to the source operand.

Instruction Format:

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

Effective Address field—Specifies the location of the source operand. O
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
(xxx).W	111
(xxx).L	111
# < data >	111
(d_{16},PC)	111
(d_8,PC,Xn)	111

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
([bd,PC,Xn] ,od)	111
([bd,PC],Xn ,od)	111

*Available for the CPU32.

Then USP → An or An → USP
Else TRAP

Assembler Syntax: MOVE USP,An
MOVE An,USP

Attributes: Size = (Long)

Description: Moves the contents of the user stack pointer to or from the specified register.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	1	1	1	0	0	1	1	0	dr	REG

Instruction Fields:

dr field—Specifies the direction of transfer.
0—Transfer the address register to the user stack pointer.
1—Transfer the user stack pointer to the address register.

Register field—Specifies the address register for the operation.

Assembler Syntax: MOVEC Rc,Rn
MOVEC Rn,Rc

Attributes: Size = (Long)

Description: Moves the contents of the specified control register (Rc) to the s general register (Rn) or copies the contents of the specified general registe specified control register. This is always a 32-bit transfer, even though the register may be implemented with fewer bits. Unimplemented bits are read as

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1
A/D	REGISTER			CONTROL REGISTER										

Instruction Fields:

dr field—Specifies the direction of the transfer.
0—Control register to general register.
1—General register to control register.

A/D field—Specifies the type of general register.
0—Data Register
1—Address Rregister

Hex[1]	Control Register
MC68010/MC68020/MC68030/MC68040/CPU32	
000	Source Function Code (SFC)
001	Destination Function Code (DFC)
800	User Stack Pointer (USP)
801	Vector Base Register (VBR)
MC68020/MC68030/MC68040	
002	Cache Control Register (CACR)
802	Cache Address Register (CAAR)[2]
803	Master Stack Pointer (MSP)
804	Interrupt Stack Pointer (ISP)
MC68040/MC68LC040	
003	MMU Translation Control Register (TC)
004	Instruction Transparent Translation Register 0 (ITT0)
005	Instruction Transparent Translation Register 1 (ITT1)
006	Data Transparent Translation Register 0 (DTT0)
007	Data Transparent Translation Register 1 (DTT1)
805	MMU Status Register (MMUSR)
806	User Root Pointer (URP)
807	Supervisor Root Pointer (SRP)
MC68EC040 only	
004	Instruction Access Control Register 0 (IACR0)
005	Instruction Access Control Register 1 (IACR1)
006	Data Access Control Register 0 (DACR1)
007	Data Access Control Register 1 (DACR1)

NOTES:
1. Any other code causes an illegal instruction exception
2. For the MC68020 and MC68030 only.

Assembler Syntax: MOVES Rn, < ea >
MOVES < ea > ,Rn

Attributes: Size = (Byte, Word, Long)

Description: This instruction moves the byte, word, or long operand from the s general register to a location within the address space specified by the de function code (DFC) register, or it moves the byte, word, or long operand location within the address space specified by the source function code (SFC) to the specified general register. If the destination is a data register, the source replaces the corresponding low-order bits of that data register, depending on of the operation. If the destination is an address register, the source operand extended to 32 bits and then loaded into that address register.

Condition Codes:

Not affected.

Instruction Format:

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

00—Byte Operation
01—Word Operation
10—Long Operation

Effective Address field—Specifies the source or destination location within the address space. Only memory alterable addressing modes can be used a 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
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(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)	—

*Available for the CPU32.

A/D field—Specifies the type of general register.
0—Data Register
1—Address Register

Register field—Specifies the register number.

dr field—Specifies the direction of the transfer.
0—From < ea > to general register.
1—From general register to < ea > .

examples with the same address register as both source and destination.

```
MOVES.x An,(An) +
MOVES.x An,D(An)
```

The current implementations of the MC68010, MC68020, MC68030, and MC68040 store the incremented or decremented value of An. Check the following code sequence to determine what value is stored for each case.

```
MOVEA.L #$1000,A0
MOVES.L A0,(A0) +
MOVES.L A0,D(A0)
```

Because the MC68040 implements a merged instruction and data space, the MC68040's integer unit into data references (SFC/DFC = 5 or 1) translates MOVES accesses to the OinstructionO address spaces (SFC/DFC = 6 or 2). The data memory unit handles these translated accesses as normal data accesses. If the access fails due to an ATC fault or a physical bus error, the resulting access error stack frame contains the converted function code in the TM field for the faulted access. To maintain cache coherency, MOVES accesses to write the OinstructionO address space must be preceded by invalidation of the instruction cache line containing the referenced location.

Then Source V SR → SR
Else TRAP

Assembler Syntax: ORI # < data > ,SR

Attributes: Size = (Word)

Description: Performs an inclusive-OR operation of the immediate operand and t register's contents and stores the result in the status register. All implement the status register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X—Set if bit 4 of immediate operand is one; unchanged otherwise.

N—Set if bit 3 of immediate operand is one; unchanged otherwise.

Z—Set if bit 2 of immediate operand is one; unchanged otherwise.

V—Set if bit 1 of immediate operand is one; unchanged otherwise.

C—Set if bit 0 of immediate operand is one; unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
0	0	0	0	0	0	0	0	0	1	1	1	1	1	
16—BIT WORD DATA														

Else TRAP

Assembler Syntax: PBcc. < size > < label >

Attributes: Size = (Word, Long)

Description: If the specified paged memory management unit condition is met, e
continues at location (PC) + displacement. The displacement is a twos com
integer that counts the relative distance in bytes. The value in the program c
the address of the displacement word(s). The displacement may be either 16 o

The condition specifier cc indicates the following conditions:

Specifier	Description	Condition Field
BS	B set	000000
LS	L set	000010
SS	S set	000100
AS	A set	000110
WS	W set	001000
IS	I set	001010
GS	G set	001100
CS	C set	001110

Specifier	Description	Condition Field
BC	B clear	000001
LC	L clear	000011
SC	S clear	000101
AC	A clear	000111
WC	W clear	001001
IC	I clear	001011
GC	G clear	001101
CC	C clear	001111

PMMU Status Register: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	1	SIZE	MC68851 CONDITION				
16-BIT DISPLACEMENT OR MOST SIGNIFICANT WORD OF 32-BITDISPLACEMENT														
LEAST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT (IF NEEDED)														

0—Displacement is 16 bits.
1—Displacement is 32 bits.

MC68851 Condition field—Specifies the coprocessor condition to be tested. is passed to the MC68851, which provides directives to the main proc processing this instruction.

Word Displacement field—The shortest displacement form for MC68851 br 16 bits.

Long-Word Displacement field—Allows a displacement larger than 16 bits.

Else No Operation
Else TRAP

Assembler Syntax: PDBcc Dn, < label >

Attributes: Size = (Word)

Description: This instruction is a looping primitive of three parameters: an M condition, a counter (an MC68020 data register), and a 16-bit displaceme instruction first tests the condition to determine if the termination condition for has been met. If so, the main processor executes the next instruction in the ins stream. If the termination condition is not true, the low-order 16 bits of the register are decremented by one. If the result is not D1, execution continue location specified by the current value of the program counter plus the sign-e 16-bit displacement. The value of the program counter used in the branch calculation is the address of the PDBcc instruction plus two.

The condition specifier cc indicates the following conditions:

Specifier	Description	Condition Field
BS	B set	000000
LS	L set	000010
SS	S set	000100
AS	A set	000110
WS	W set	001000
IS	I set	001010
GS	G set	001100
CS	C set	001110

Specifier	Description	Condition Field
BC	B clear	000001
LC	L clear	000011
SC	S clear	000101
AC	A clear	000111
WC	W clear	001001
IC	I clear	001011
GC	G clear	001101
CC	C clear	001111

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	1	0	0	1	COUNT
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION			
16-BIT DISPLACEMENT													

Instruction Fields:

Register field—Specifies the data register in the main processor to be use counter.

MC68851 Condition field—Specifies the MC68851 condition to be tested. Th passed to the MC68851, which provides directives to the main proc processing this instruction.

Displacement field—Specifies the distance of the branch in bytes.

Assembler Syntax: PFLUSHA
PFLUSH FC,MASK
PFLUSH FC,MASK, < ea >

Attributes: Unsized

Description: PFLUSH invalidates address translation cache entries. The instruc three forms. The PFLUSHA instruction invalidates all entries. When the ins specifies a function code and mask, the instruction invalidates all entries for a function code(s). When the instruction also specifies an < ea > , the ins invalidates the page descriptor for that effective address entry in each selected code.

The mask operand contains three bits that correspond to the three function c Each bit in the mask that is set to one indicates that the corresponding bit o operand applies to the operation. Each bit in the mask that is zero indicates a and of the ignored function code. For example, a mask operand of 100 cau instruction to consider only the most significant bit of the FC operand. If the FC is 001, function codes 000, 001, 010, and 011 are selected.

The FC operand is specified in one of the following ways:

1. Immediate—Three bits in the command word.
2. Data Register—The three least significant bits of the data register spec the instruction.
3. Source Function Code (SFC) Register
4. Destination Function Code (DFC) Register

Condition Codes:

Not affected.

MMU Status Register:

Not affected.

1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS	
										MODE	REG
0	0	1	MODE			0	0	MASK		FC	

Instruction Fields:

Effective Address field—Specifies a control alterable address. The address tr
cache entry for this address is invalidated. Valid addressing modes a
following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn] ,od)	—
([bd,PC],Xn ,od)	—

NOTE

The address field must provide the memory management unit with the effective address to be flushed from the address translation cache, not the effective address describing where the PFLUSH operand is located. For example, to flush the address translation cache entry corresponding to a logical address that is temporarily stored on top of the system stack, the instruction PFLUSH [(SP)] must be used since PFLUSH (SP) would invalidate the address translation cache entry mapping the system stack (i.e., the effective address passed to the memory management unit is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

110—Flush by function code and effective address.

Mask field—Mask for selecting function codes. Ones in the mask corres
applicable bits; zeros are bits to be ignored. When mode is 001, mask
000.

FC field—Function code of entries to be flushed. If the mode field is 001, FC fi
be 00000; otherwise:

10XXX — Function code is specified as bits XXX.
01DDD — Function code is specified as bits 2–0 of data register DDD.
00000 — Function code is specified as SFC register.
00001 — Function code is specified as DFC register.

Else TRAP

Assembler Syntax: PFLUSH (An)
Syntax: PFLUSHN (An)
Syntax: PFLUSHA
Syntax: PFLUSHAN

Attributes: Unsized

Description: Invalidates address translation cache entries in both the instruction address translation caches. The instruction has two forms. The PFLUSHA i invalidates all entries. The PFLUSH (An) instruction invalidates the entry address translation cache which matches the logical address in An and the function code.

The function code for PFLUSH is specified in the destination function code Destination function code values of 1 or 2 will result in flushing of user addre lation cache entries in both address translation caches; whereas, values of 5 result in flushing of supervisor address translation cache entries. PFLUSH is u for destination function code values of 0, 3, 4, and 7 and may cause flush unexpected entry.

The PFLUSHN and PFLUSHAN instructions have a global option specified a date only nonglobal entries. For example, if only page descriptors for operatin code have the global bit set, these two PFLUSH variants can be used to flush address translation cache entries during task swaps.

Condition Codes:

Not affected.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	1	0	1	0	0	0	OPMODE		REGI	

Instruction Fields:

Opmode field—Specifies the flush operation.

Opcode	Operation	Assembler Syntax
00	Flush page entry if not global	PFLUSHN (An)
01	Flush page entry	PFLUSH (An)
10	Flush all except global entries	PFLUSHAN
11	Flush all entries	PFLUSHA

Register field—Specifies the address register containing the effective addre
flushed when flushing a page entry.

Assembler Syntax: PFLUSH (An)
PFLUSHN (An)

Attributes: Unsized

Description: This instruction should not be executed when using an MC68EC
PFLUSH encoding suspends operation of the MC68EC040 for an indefinite
time and subsequently continues with no adverse effects.

Condition Codes:

Not affected.

Instruction Format:

Postincrement Source and Destination

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	0	1	0	0	0	OPMODE		REG

Instruction Fields:

Opmode field—Specifies the flush operation.

Opcode	Operation	Assembler Syntax
00	Flush page entry if not global	PFLUSHN (An)
01	Flush page entry	PFLUSH (An)
10	Flush all except global entries	PFLUSHAN
11	Flush all entries	PFLUSHA

Register field—Specifies the address register containing the effective addr
flushed when flushing a page entry.

Operation:
If Supervisor State
Then Address Translation Cache Entries For Destination A
Are Invalidated
Else TRAP

Assembler Syntax:
PFLUSHA
PFLUSH FC,MASK
PFLUSHS FC,MASK
PFLUSH FC,MASK, < ea >
PFLUSHS FC,MASK, < ea >

Attributes: Unsigned

Description: PFLUSHA invalidates all entries in the address translation cache.

PFLUSH invalidates a set of address translation cache entries whose functi bits satisfy the relation: (address translation cache function code bits and mas and MASK) for all entries whose task alias matches the task alias currently acti the instruction is executed. With an additional effective address argument, P invalidates a set of address translation cache entries whose function code sati relation above and whose effective address field matches the corresponding b evaluated effective address argument. In both of these cases, address tra cache entries whose SG bit is set will not be invalidated unless the PFLUSHS ified.

The function code for this operation may be specified as follows:

1. Immediate—The function code is four bits in the command word.
2. Data Register—The function code is in the lower four bits of the MC680 register specified in the instruction.
3. Source Function Code (SFC) Register—The function code is in the CP register. Since the SFC of the MC68020 has only three implemented b function codes $0D$7 can be specified in this manner.
4. Destination Function Code (DFC) Register—The function code is in the DFC register. Since the DFC of the MC68020 has only three implement only function codes $0D$7 can be specified in this manner.

PMMU Status Register: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	0	1	MODE			0	MASK					FC	

Instruction Fields:

Effective Address field—Specifies an address whose page descriptor is to b
from (invalidated) the address translation cache. Only control alterable ad
modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn] ,od)	—
([bd,PC],Xn ,od)	—

NOTE

The effective address field must provide the MC68851 with the effective address of the entry to be flushed from the address translation cache, not the effective address describing where the PFLUSH operand is located. For example, in order to flush the address translation cache entry corresponding to a logical address that is temporarily stored on the top of the system stack, the instruction PFLUSH [(SP)] must be used since PFLUSH (SP) would invalidate the address translation cache entry mapping the system stack (i.e., the effective address passed to the MC68851 is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

Mode field—Specifies how the address translation cache is to be flushed.
001—Flush all entries.
100—Flush by function code only.
101—Flush by function code including shared entries.
110—Flush by function code and effective address.
111—Flush by function code and effective address including shared entries

Mask field—Indicates which bits are significant in the function code compare indicates that the bit position is not significant; a one indicates that the bit is significant. If mode = 001 (flush all entries), mask must be 0000.

FC field—Function code of address to be flushed. If the mode field is 001 (entries), function code must be 00000; otherwise:
1DDDD — Function code is specified as four bits DDDD.
01RRR — Function code is contained in CPU data register RRR.
00000 — Function code is contained in CPU SFC register.
00001 — Function code is contained in CPU DFC register.

Else TRAP

Assembler Syntax: PFLUSHR < ea >

Attributes: Unsized

Description: The quad word pointed to by < ea > is regarded as a previously use the CPU root pointer register. The root pointer table entry matching this pointer register (if any) is flushed, and all address translation cache entries lo this value of CPU root pointer register (except for those that are globally sh invalidated. If no entry in the root pointer table matches the operand of this in no action is taken.

If the supervisor root pointer is not in use, the operating system should not PFLUSHR command to destroy a task identified by the current CPU root po ister. It should wait until the CPU root pointer register has been loaded with pointer identifying the next task until using the PFLUSHR instruction. At any t cution of the PFLUSHR instruction for the current CPU root pointer register ca current task alias to be corrupted.

Instruction Format:

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

pointer register register. Only memory addressing modes can be used as
the following table:

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

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn] ,od)	111	
([bd,PC],Xn ,od)	111	

NOTE

The effective address usage of this instruction is different than that of other PFLUSH variants.

Else TRAP

Assembler Syntax: PLOADR FC, < ea >
PLOADW FC, < ea >

Attributes: Unsized

Description: For the MC68851, PLOAD searches the translation table for a tran the specified effective address. If one is found, it is flushed from the address tr cache, and an entry is made as if a bus master had run a bus cycle. Used and bits in the table are updated as part of the table search. The MC68851 ign logical bus arbitration signals during the flush and load phases at the en instruction. This prevents the possibility of an entry temporarily disappearing address translation cache and causing a false table search.

This instruction will cause a paged memory management unit illegal operatio tion (vector $39) if the E-bit of the translation control register is clear.

The function code for this operation may be specified to be:

1. Immediate—The function code is specified as four bits in the comman
2. Data Register—The function code is contained in the lower four bits in MC68020 data register specified in the instruction.
3. Source Function Code (SFC) Register—The function code is in the C register. Since the SFC of the MC68020 has only three implemented b function codes $0D$7 can be specified in this manner.
4. Destination Function Code (DFC) Register—The function code is in th DFC register. Since the DFC of the MC68020 has only three impleme only function codes $0D$7 can be specified in this manner.

attempted to access that address. Sets the used and modified bits appropriatel
of the search. The instruction executes despite the value of the E-bit in the tra
control register or the state of the MMUDIS signal.

The < function code > operand is specified in one of the following ways:

1. Immediate—Three bits in the command word.
2. Data Register—The three least significant bits of the data register spec
   the instruction.
3. Source Function Code (SFC) Register
4. Destination Function Code (DFC) Register

The effective address field specifies the logical address whose translation
loaded.

PLOADR causes U bits in the translation tables to be updated as if a read acc
occurred. PLOADW causes U and M bits in the translation tables to be update
write access had occurred.

PMMU Status Register: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	0	1	0	0	0	R/ W	0	0	0	0	FC			

into the address translation cache. Only control alterable addressing m
allowed as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn] ,od)	—
([bd,PC],Xn ,od)	—

NOTE

The effective address field must provide the MC68851 with the effective address of the entry to be loaded into the address translation cache, not the effective address describing where the PLOAD operand is located. For example, to load an address translation cache entry to map a logical address that is temporarily stored on the system stack, the instruction PLOAD [(SP)] must be used since PLOAD (SP) would load an address translation cache entry mapping the system stack (i.e., the effective address passed to the MC68851 is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

R/W field—Specifies whether the tables should be updated for a read or a w
1—Read
0—Write

00000 — Function code is contained in CPU SFC register.
00001 — Function code is contained in CPU DFC register.

FC field (MC68030)—Function code of address corresponding to entry to be l
10XXX — Function code is specified as bits XXX.
01DDD — Function code is specified as bits 2–0 of data register DDD.
00000 — Function code is specified as SFC register.
00001 — Function code is specified as DFC register.

Assembler Syntax:
PMOVE MRn, < ea >
PMOVE < ea > ,MRn
PMOVEFD < ea > ,MRn

Attributes: Size = (Word, Long, Quad)

Description: Moves the contents of the source effective address to the specified management unit register or moves the contents of the memory manager register to the destination effective address.

The instruction is a quad-word (8 byte) operation for the CPU root pointe supervisor root pointer. It is a long-word operation for the translation contro and the transparent translation registers (TT0 and TT1). It is a word operati MMU status register.

The PMOVEFD form of this instruction sets the FD-bit to disable flushing the translation cache when a new value loads into the supervisor root pointer, pointer, TT0, TT1 or translation control register (but not the MMU status regi

Writing to the following registers has the indicated side effects:

CPU Root Pointer—When the FD-bit is zero, it flushes the address translati If the operand value is invalid for a root pointer descriptor, the instruction memory management unit configuration error exception after moving the op the CPU root pointer.

Supervisor Root Pointer—When the FD-bit is zero, it flushes the address tr cache. If the operand value is invalid as a root pointer descriptor, the instruct an memory management unit configuration error exception after moving the to the supervisor root pointer.

Translation Control Register—When the FD-bit is zero, it flushes the addres tion cache. If the E-bit = 1, consistency checks are performed on the PS and If the checks fail, the instruction takes an memory management unit con exception after moving the operand to the translation control register. If th pass, the translation control register is loaded with the operand and the E-bit is

TT0, TT1—When the FD-bit is zero, it flushes the address translation cache. or disables the transparent translation register according to the E-bit written bit = 1, the transparent translation register is enabled. If the E- bit = 0, the r disabled.

MMU Status Register:

Not affected (unless the MMU status register is specified as the destination op

Instruction Format:

SRP, CRP, and TC Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	1	0	P-REGISTER			R/ W	FD	0	0	0	0	0	0	0

Instruction Fields:

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

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn] ,od)	—	
([bd,PC],Xn ,od)	—	

011—CPU Root Pointer

R/W field—Specifies the direction of transfer.
0—Memory to memeory management unit register.
1—Memeory management unit register to memory.

FD field—Disables flushing of the address translation cache on writes to management unit registers.
0—Address translation cache is flushed.
1—Address translation cache is not flushed.

Instruction Format:

MMU Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	1	1	0	0	0	R/ W	0	0	0	0	0	0	0	

Instruction Fields:

Effective Address field—Specifies the memory location for the transfer alterable addressing modes shown for supervisor root pointer register app

R/W field—Specifies the direction of transfer.
0—Memory to MMU status register.
1—MMU status register to memory.

NOTE

The syntax of assemblers for the MC68851 use the symbol PMMU status register for the MMU status register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGI	
0	0	0	P-REGISTER			R/ W	FD	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field—Specifies the memory location for the transfer. alterable addressing modes shown for supervisor root pointer register appl

P-Register field—Specifies the transparent translation register.
010—Transparent Translation Register 0
011—Transparent Translation Register 1

R/W field—Specifies the direction of transfer.
0—Memory to MMU status register.
1—MMU status register to memory.

FD field—Disables flushing of the address translation cache.
0—Address translation cache is flushed.
1—Address translation cache does not flush.

Assembler Syntax: PMOVE MRn, < ea >
PMOVE < ea > ,MRn

Attributes: Size = (Word, Long, Quad)

Description: Moves the contents of the source effective address to an acces
register or moves the contents of an access control register to the destinatio
address.

The instruction is a long-word operation for the access control registers (
AC1). It is a word operation for the access control unit status register (ACUS

Writing to the ACx registers enables or disables the access control register a
to the E-bit written. If the E-bit = 1, the access control register is enabled. If t
0, the register is disabled

Condition Codes:

Not affected.

ACUSR:

Not affected unless the ACUSR is specified as the destination operand.

Instruction Format:

ACUSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	1	1	0	0	0	R/W	0	0	0	0	0	0	0	

Instruction Fields:

Effective Address field—Specifies the memory location for the transfer.

R/W field—Specifies the direction of transfer.
0—Memory to ACUSR
1—ACUSR to memory

status register for the ACUSR; and for the MC68030, the symbols TT0 and TT1 for AC0 and AC1.

Instruction Format:

ACx Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	0	0	P-REGISTER			R/W	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field—Specifies the memory location for the transfer.

P-Register field—Specifies the ACx register.
001—Access Control Register 0
011—Access Control Register 1

R/W field—Specifies the direction of transfer.
0—Memory to ACUSR
1—ACUSR to memory

Else TRAP

Assembler Syntax: PMOVE < PMMU Register > , < ea >
PMOVE < ea > , < PMMU Register >

Attributes: Size = (Byte, Word, Long, Double Long)

Description: The contents of the MC68851 register copies to the address specifie
> , or the data at < ea > copies into the MC68851 register.

The instruction is a quad-word operation for CPU root pointer, supervisor roo
and DMA root pointer registers. It is a long-word operation for the translatio
register and a word operation for the breakpoint acknowledge control, b
acknowledge data, access control, PMMU status, and PMMU cache status
PMOVE is a byte operation for the current access level, valid access level, a
change control registers.

The following side effects occur when data is read into certain registers:

CPU Root Pointer—Causes the internal root pointer table to be searche
new value. If there is no matching value, an entry in the root pointer table is
for replacement, and all address translation cache entries associated
replaced entry are invalidated.

Supervisor Root Pointer—Causes all entries in the address translation c
were formed with the supervisor root pointer (even globally shared entri
invalidated.

DMA Root Pointer—Causes all entries in the address translation cache
formed with the DMA root pointer (even globally shared entries) to be inv

Translation Control Register—If data written to the translation contro
attempts to set the E-bit and the E-bit is currently clear, a consistency che
formed on the IS, TIA, TIB, TIC, TID, and PS fields.

Instruction Format 1:

PMOVE to/from TC, CRP, DRP, SRP, CAL, VAL, SCC, AC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	1	0	P-REGISTER			R/ W	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field—for memory-to-register transfers, any addressing
allowed as listed in the following table:

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

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	111	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn] ,od)	111	
([bd,PC],Xn ,od)	111	

*PMOVE to CRP, SRP, and DMA root pointer not allowed with these modes

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

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn] ,od)	—
([bd,PC],Xn ,od)	—

*PMOVE to CRP, SRP, and DMA root pointer not allowed with these modes

Register field—Specifies the MC68851 register.
- 000—Translation Control Register
- 001—DMA Root Pointer
- 010—Supervisor Root Pointer
- 011—CPU Root Pointer
- 100—Current Access Level
- 101—Valid Access Level
- 110—Stack Change Control Register
- 111—Access Control Register

R/W field—Specifies the direction of transfer.
- 0—Transfer < ea > to MC68851 register.
- 1—Transfer MC68851 register to < ea > .

Instruction Format 2:

PMOVE to/from BADx, BACx

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	1	1	P-REGISTER			R/ W	0	0	0	0	NUM		

P-Register field—Specifies the type of MC68851 register.
100—Breakpoint Acknowledge Data
101—Breakpoint Acknowledge Control

R/W field—Specifies the direction of transfer.
0—Transfer < ea > to MC68851 register
1—Transfer MC68851 register to < ea >

Num field—Specifies the number of the BACx or BADx register to be used.

Instruction Format 3:

PMOVE to/from PSR, from PCSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	1	1	P-REGISTER			R/ W	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field—Same as format 1.

P Register field—Specifies the MC68851 register.
000 — PMMU Status Register
001 — PMMU Cache Status Register

R/W field—Specifies direction of transfer.
0—Transfer < ea > to MC68851 register.
1—Transfer MC68851 register to < ea > (must be one to access PMMU ca
status register using this format).

Else TRAP

Assembler Syntax: PRESTORE < ea >

Attributes: Unsized, Privileged

Description: The MC68851 aborts execution of any operation in progre programmer registers and internal states are loaded from the state frame loca effective address. The first word at the specified address is the format word o frame, specifying the size of the frame and the revision number of the MC6 created it. The MC68020 writes the first word to the MC68851 restore cop interface register, initiating the restore operation. Then it reads the coprocessor interface register to verify that the MC68851 recognizes the valid. The format is invalid if the MC68851 does not recognize the frame si revision number does not match. If the format is invalid, the MC68020 takes exception, and the MC68851 returns to the idle state with its user visible unchanged. However, if the format is valid, then the appropriate state fra starting at the specified location and proceeding up through the higher addre

The PRESTORE instruction restores the nonuser visible state of the MC6885 as the PMMU status register, CPU root pointer, supervisor root pointer, curre level, valid access level, and stack change control registers of the user prog model. In addition, if any breakpoints are enabled, all breakpoint acknowledg and breakpoint acknowledge data registers are restored. This instruction is th of the PSAVE instruction.

The current implementation of the MC68851 supports four state frame sizes

NULL: This state frame is 4 bytes long, with a format word of $0. A PREST this size state frame places the MC68851 in the idle state with no sor or module operations in progress.

IDLE: This state frame is 36 ($24) bytes long. A PRESTORE with this frame causes the MC68851 to place itself in an idle state with no sor operations in progress and no breakpoints enabled. A module may or may not be in progress. This state frame restores the mini MC68851 registers.

BREAKPOINTS ENABLED: This state frame is 76 ($4C) bytes long. A PRE
with this size state frame restores all breakpoint registers, along w
states. A coprocessor operation may or may not be in progress.

PMMU Status Register: Set according to restored data.

Instruction Format:

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

Instruction Fields:

Effective Address field—Specifies the source location. Only control or post-in
addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	
([bd,PC,Xn] ,od)	111	
([bd,PC],Xn ,od)	111	

Else TRAP

Assembler Syntax: PSAVE < ea >

Attributes: Unsized, Privileged

Description: The MC68851 suspends execution of any operation that it is perfor saves its internal state and some programmer registers in a state frame loca effective address. The following registers are copied: PMMU status, co pointer, supervisor root pointer, current access level, valid access level, a change control. If any breakpoint is enabled, all breakpoint acknowledge co breakpoint acknowledge data registers are copied. After the save opera MC68851 is in an idle state waiting for another operation to be requested. Pro registers are not changed.

The state frame format saved by the MC68851 depends on its state at the ti PSAVE operation. In the current implementation, three state frames are pos

IDLE: This state frame is 36 ($24) bytes long. A PSAVE of this size state fr cates that the MC68851 was in an idle state with no coprocessor o in progress and no breakpoints enabled. A module call operation m not have been in progress when this state frame was saved.

MID-COPROCESSOR:This state frame is 44 ($2C) bytes long. A PSAVE o frame indicates that the MC68851 was in a state with a coprocesso ule call operation in progress and no breakpoints enabled.

BREAKPOINTS ENABLED:This state frame is 76 ($4C) bytes long. A PSAV size state frame indicates that one or more breakpoints were en coprocessor or module call operation may or may not have been in p

PMMU Status Register: Not affected

1	1	1	1	0	0	0	1	0	0	EFFECTIVE ADDRESS MODE	REGIS

Instruction Fields:

Effective Address field—Specifies the destination location. Only co
predecrement addressing modes can be used as listed in the following tab

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn] ,od)	—	
([bd,PC],Xn ,od)	—	

Else 0s → Destination
Else TRAP

Assembler Syntax: PScc < ea >

Attributes: Size = (Byte)

Description: The specified MC68851 condition code is tested. If the condition is
byte specified by the effective address is set to TRUE (all ones); otherwise, th
set to FALSE (all zeros).

The condition code specifier cc may specify the following conditions:

Specifier	Description	Condition Field
BS	B set	000000
LS	L set	000010
SS	S set	000100
AS	A set	000110
WS	W set	001000
IS	I set	001010
GS	G set	001100
CS	C set	001110

Specifier	Description	Con
BC	B clear	
LC	L clear	
SC	S clear	
AC	A clear	
WC	W clear	
IC	I clear	
GC	G clear	
CC	C clear	

PMMU Status Register: Not affected

1	1	1	1	0	0	0	0	0	1	EFFECTIVE ADDRESS MODE | REGIS
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION

Instruction Fields:

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

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
—(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn] ,od)	—
([bd,PC],Xn ,od)	—

MC68851 Condition field—Specifies the coprocessor condition to be tested. T is passed to the MC68851, which provides directives to the main proce processing this instruction.

Assembler Syntax:
PTESTR FC, < ea > ,# < level >
PTESTR FC, < ea > ,# < level > ,An
PTESTW FC, < ea > ,# < level >
PTESTW FC, < ea > ,# < level > ,An

Attributes: Unsized

Description: This instruction searches the address translation cache or the tr
tables to a specified level. Searching for the translation descriptor correspond
< ea > field, it sets the bits of the MMU status register according to the sta
descriptor. Optionally, PTEST stores the physical address of the last ta
accessed during the search in the specified address register. The PTEST in
searches the address translation cache or the translation tables to obta
information, but alters neither the used or modified bits of the translation table
address translation cache. When the level operand is zero, only the tra
translation of either read or write accesses causes the operations of the PTE
PTESTW to return different results.

The < function code > operand is specified as one of the following:

1. Immediate—Three bits in the command word.
2. Data Register—The three least significant bits of the data register spe
the instruction.
3. Source Function Code (SFC) Register
4. Destination Function Code (DFC) Register

The effective address is the address to test. The < level > operand specifies
of the search. Level 0 specifies searching the addrass translation cache on
1–7 specify searching the translation tables only. The search ends at the
level. A level 0 test does not return the same MMU status register values as a
nonzero level number.

Execution of the instruction continues to the requested level or until detecti
the following conditions:

- Invalid Descriptor
- Limit Violation
- Bus Error Assertion (Physical Bus Error)

If there is a parameter specification for a translation table search, the physical
of the last descriptor successfully fetched loads into the address register. A s
fully fetched descriptor occurs only if all portions of the descriptor can be rea
MC68030 without abnormal termination of the bus cycle. If the root pointer's
indicates page descriptor, the returned address is $0. For a long descrip
address of the first long word is returned. The size of the descriptor (short or
not returned and must be determined from a knowledge of the translation tabl

Condition Codes:

Not affected.

MMUSR:

B	L	S		W	I	M			T				N
*	*	*	0	*	*	*	0	0	0	0	0	0	*

MMUSR Bit	PTEST, Level 0	PTEST, Levels 1–7
Bus Error (B)	This bit is set if the bus error bit is set in the ATC entry for the specified logical address.	This bit is set if a bus error is encounte during the table search for the PTEST struction.
Limit (L)	This bit is cleared.	This bit is set if an index exceeds a lim during the table search.
Supervis or Violatio n (S)	This bit is cleared.	This bit is set if the S-bit of a long (S) for table descriptor or long format page de scriptor encountered during the search set and if the FC2-bit of the function co specified by the PTEST instruction is n equal to one. The S-bit is undefined if t bit is set.
Write Protecte d (W)	The bit is set if the WP-bit of the ATC entry is set. It is undefined if the I-bit is set.	This bit is set if a descriptor or page de scriptor is encountered with the WP-bit during the table search. The W-bit is un fined if the I-bit is set.
Invalid (I)	This bit indicates an invalid translation. The I- bit is set if the translation for the specified logical address is not resident in the ATC or if the B-bit of the corresponding ATC entry is set.	This bit indicates an invalid translation. I-bit is set if the DT field of a table or a p descriptor encountered during the sea is set to invalid or if either the B or L bit the MMUSR are set during the table search.
Modified (M)	This bit is set if the ATC entry corresponding to the specified address has the modified bit set. It is undefined if the I-bit is set.	This bit is set if the page descriptor for specified address has the modified bit It is undefined if I-bit is set.
Transparent (T)	This bit is set if a match occurred in either (or both) of the transparent translation registers (TT0 or TT1).	This bit is set to zero.
Number of Levels (N)	This 3-bit field is set to zero.	This 3-bit field contains the actual num of tables accessed during the search.

1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS MODE	REGIS
1	0	0	LEVEL			R/ W	A	REGISTER			FC

Instruction Fields:

Effective Address field—Specifies the logical address to be tested. Only alterable addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn] ,od)	—	
([bd,PC],Xn ,od)	—	

Level field—Specifies the highest numbered level to be searched in the table this field contains 0, the A field and the register field must also be instruction takes an F-line exception when the level field is 0 and the A fie 0.

R/W field—Specifies simulating a read or write bus cycle (no difference for M MMU).
0—Write
1—Read

A field—Specifies the address register option.
0—No address register.
1—Return the address of the last descriptor searched in the address regist ified in the register field.

FC field—Function code of address to be tested.

10XXX — Function code is specified as bits XXX.

01DDD — Function code is specified as bits 2–0 of data register DDD

00000 — Function code is specified as source function code register.

00001 — Function code is specified as destination function code regi

Assembler Syntax: PTESTR FC, < ea >
PTESTW FC, < ea >

Attributes: Unsized

Description: This instruction searches the access control registers for the descriptor corresponding to the < ea > field and sets the bit of the access co status register (ACUSR) according to the status of the descriptor.

The < function code > operand is specified in one of the following ways:

1. Immediate—Three bits in the command word.
2. Data Register—The three least significant bits of the data register spec the instruction.
3. Source Function Code (SFC) Register
4. Destination Function Code (DFC) Register

The effective address is the address to test.

Condition Codes:

Not affected.

ACUSR:

x	x	x	0	x	x	x	0	0	AC	0	0	0	x	x

x = May be 0 or 1.

The AC-bit is set if a match occurs in either (or both) of the access control reg

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
1	0	0	0	0	0	R/ W	0	REGISTER				FC		

alterable addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn] ,od)	—
([bd,PC],Xn ,od)	—

R/W field—Specifies simulating a read or write bus cycle.
0—Write
1—Read

Register field—Specifies an address register for the instruction. When th
contains 0, this field must contain 0.

FC field—Function code of address to be tested.
10XXX — Function code is specified as bits XXX.
01DDD — Function code is specified as bits 2–0 of data register DDD
00000 — Function code is specified as source function code register.
00001 — Function code is specified as destination function code regi

NOTE

Assembler syntax for the MC68030 is PTESTR FC, < ea > ,#0
and PTESTW FC, < ea > ,#0.

Assembler Syntax: PTESTR (An)
PTESTW (An)

Attributes: Unsized

Description: This instruction searches the translation tables for the page de corresponding to the test address in An and sets the bits of the MMU status according to the status of the descriptors. The upper address bits of the tra physical address are also stored in the MMU status register. The PTESTR ins simulates a read access and sets the U-bit in each descriptor during table se PTESTW simulates a write access and also sets the M-bit in the descript address translation cache entry, and the MMU status register.

A matching entry in the address translation cache (data or instruction) specifie function code will be flushed by PTEST. Completion of PTEST results in the of a new address translation cache entry. The specification of the function cod test address is in the destination function code (DFC) register. A PTEST ins with a DFC value of 0, 3, 4, or 7 is undefined and will return an unknown valu MMUSR.

Execution of the instruction continues until one of the following conditions occ

- Match with one of the two transparent translation registers.
- Transfer Error Assertion (physical transfer error)
- Invalid Descriptor
- Valid Page Descriptor

Condition Codes:

Not affected.

MMU Status Register:

PHYSICAL ADDRESS	B	G	U1	U0	S	CM	M		W	T
*	*	*	*	*	*	*	*	0	*	*

Physical Address—This 20-bit field contains the upper bits of the translated address. Merging these bits with the lower bits of the logical address f actual physical address.

Bus Error (B)—Set if a transfer error is encountered during the table searc PTEST instruction. If this bit is set, all other bits are zero.

Globally Shared (G)—Set if the G-bit is set in the page descriptor.

User Page Attributes (U1, U0)—Set if corresponding bits in the page descriptc

Supervisor Protection (S)—Set if the S-bit in the page descriptor is set. This not indicate that a violation has occurred.

Cache Mode (CM)—This 2-bit field is copied from the CM-bit in the page des

Modified (M)—Set if the M-bit is set in the page descriptor associated with the

Write Protect (W)—Set if the W-bit is set in any of the descriptors encounter the table search. Setting of this bit does not indicate that a violation occ

Transparent Translation Register Hit (T)—Set if the PTEST address ma instruction or data transparent translation register and the R-bit is set; all are zero.

Resident (R)—Set if the PTEST address matches a transparent translation r if the table search completes by obtaining a valid page descriptor.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	0	1	0	1	R/ W	0	1	REG

Instruction Fields:

R/W field—Specifies simulating a read or write bus transfer.
0—Write
1—Read

Register field—Specifies the address register containing the effective addre instruction.

Assembler Syntax: PTESTR (An)
PTESTW (An)

Attributes: Unsized

Description: This instruction must not be executed on an MC68EC040. This ins
may cause extraneous bus cycles to occur and may result in unexpected e
types.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	1	0	1	0	1	R/ W	0	1	REGI	

Instruction Fields:

R/W field—Specifies simulating a read or write bus transfer.
0—Write
1—Read

Register field—Specifies the address register containing the effective addres
instruction.

Assembler Syntax: PTESTR FC, < ea > ,# < level > ,(An)
PTESTW FC, < ea > ,# < level > ,(An)

Attributes: Unsized

Description: If the E-bit of the translation control register is set, information about t address specified by FC and < ea > is placed in the PMMU status register. If of the translation control register is clear, this instruction will cause a paged management unit illegal operation exception (vector $39).

The function code for this operation may be specified as follows:

1. Immediate—The function code is four bits in the command word.
2. Data Register—The function code is in the lower four bits in the MC68 register specified in the instruction.
3. Source Function Code (SFC) Register—The function code is in the SFC in the CPU. Since the SFC of the MC68020 has only three implement only function codes $0D$7 can be specified in this manner.
4. Destination Function Code (DFC) Register—The function code is in th register in the CPU. Since the DFC of the MC68020 has only three impl bits, only function codes $0D$7 can be specified in this manner.

The effective address field specifies the logical address to be tested.

The # < level > parameter specifies the depth to which the translation table searched. A value of zero specifies a search of the address translation cache ues 1–7 cause the address translation cache to be ignored and specify the number of descriptors to fetch.

NOTE

Finding an address translation cache entry with < level > set to zero may result in a different value in the PMMU status register than forcing a table search. Only the I, W, G, M, and C bits of the PMMU status register are always the same in both cases.

are not modified by this instruction.

If there is a specified address register parameter, the physical address of the
cessfully fetched descriptor is loaded into the address register. A descriptor is s
fully fetched if all portions of the descriptor can be read by the MC68851
abnormal termination of the bus cycle. If the DT field of the root pointer used i
page descriptor, the returned address is $0.

The PTEST instruction continues searching the translation tables until reac
requested level or until a condition occurs that makes further searching impossi
a DT field set to invalid, a limit violation, or a bus error from memory). The info
in the PMMU status register reflects the accumulated values.

PMMU Status Register:

Bus Error (B)—Set if a bus error was received during a descriptor fetch, or if
= 0 and an entry was found in the address translation cache with its BERR
cleared otherwise.

Limit (L)—Set if the limit field of a long descriptor was exceeded; cleared othe

Supervisor Violation (S)—Set if a long descriptor indicated supervisor-only acc
the < fc > parameter did not have bit 2 set; cleared otherwise.

Access Level Violation (A)—If PTESTR was specified, set if the RAL field o
descriptor would deny access. If PTESTW was specified, set if a WAL or R
of a long descriptor would deny access; cleared otherwise.

Write Protection (W)—Set if the WP-bit of a descriptor was set or if a WAL f
long descriptor would deny access; cleared otherwise.

Invalid (I)—Set if a valid translation was not available; cleared otherwise.

Modified (M)—If the tested address is found in the address translation cache, s
value of the M-bit in the address translation cache. If the tested address
in the translation table, set if the M-bit of the page descriptor is set;
otherwise.

Globally Shared (C)—Set if the address is globally shared; cleared otherwise

Level Number (N)—Set to the number of levels searched. A value of zero ind
early termination of the table search in the root pointer (DT = page des
the level specification was not zero. If the level specification was zero, N
set to zero.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
1	0	0	LEVEL			R/ W	A-REGISTER			FC			

requested. Only control alterable addressing modes can be used as liste
following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn] ,od)	—	
([bd,PC],Xn ,od)	—	

NOTE

The effective address field must provide the MC68851 with the effective address of the logical address to be tested, not the effective address describing where the PTEST operand is located. For example, to test a logical address that is temporarily stored on the system stack, the instruction PTEST [(SP)] must be used since PTEST (SP) would test the mapping of the system stack (i.e., the effective address passed to the MC68851 is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

1—Read
0—Write

A-Register field—Specifies the address register in which to load the last d
address.
0xxx — Do not return the last descriptor address to an address regi
1RRR — Return the last descriptor address to address register RRR.

NOTE

When the PTEST instruction specifies a level of zero, the A-register field must be 0000. Otherwise, an F-line exception is generated.

FC field—Function code of address to test.
1DDDD — Function code is specified as four bits DDDD.
01RRR — Function code is contained in CPU data register RRR.
00000 — Function code is contained in CPU source function code re
00001 — Function code is contained in CPU destination function cod
register.

Else TRAP

Assembler Syntax: PTRAPcc
PTRAPcc.W # < data > PTRAPcc.L # < data >

Attributes: Unsized or Size = (Word, Long)

Description: If the selected MC68851 condition is true, the processor initiates e processing. The vector number is generated referencing the cpTRAPcc e vector; the stacked program counter is the address of the next instructio selected condition is not true, no operation is performed, and execution contin the next instruction. The immediate data operand is placed in the next word(s) f the MC68851 condition and is available for user definition to be used within handler. Following the condition word, there may be a user-defined data specified as immediate data, to be used by the trap handler.

The condition specifier cc may specify the following conditions:

Specifier	Description	Condition Field
BS	B set	000000
LS	L set	000010
SS	S set	000100
AS	A set	000110
WS	W set	001000
IS	I set	001010
GS	G set	001100
CS	C set	001110

Specifier	Description	Condi
BC	B clear	0
LC	L clear	0
SC	S clear	0
AC	A clear	0
WC	W clear	0
IC	I clear	0
GC	G clear	0
CC	C clear	0

PMMU Status Register: Not affected

1	1	1	1	0	0	0	0	0	1	1	1	1	OP
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION			
16-BIT OPERAND OR MOST SIGNIFICANT WORD OF 32-BIT OPERAND (IFNEEDED)													
LEAST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)													

Instruction Fields:

Opmode field—Selects the instruction form.

010 — Instruction is followed by one operand word.

011 — Instruction is followed by two operand words.

100 — Instruction has no following operand words.

MC68851 Condition field—Specifies the coprocessor condition to be tested. is passed to the MC68851, which provides directives to the main proc processing this instruction.

Assembler Syntax: PVALID VAL, < ea >
PVALID An, < ea >

Attributes: Size = (Long)

Description: The upper bits of the source, VAL or An, compare with the upper bi
destination, < ea > . The ALC field of the access control register defines the nu
bits compared. If the upper bits of the source are numerically greater th
privileged than) the destination, they cause a memory management acce
exception. Otherwise, execution continues with the next instruction. If the MC
the access control register = 0, then this instruction always causes a paged
management unit access level exception.

PMMU Status Register: Not affected.

Instruction Format 1:

VAL Contains Access Level to Test Against

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

against the valid access level register. Only control alterable addressing m
be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode
(xxx).W	111
(xxx).L	111
# < data >	—
(d_{16},PC)	—
(d_8,PC,Xn)	—
(bd,PC,Xn)	—
([bd,PC,Xn] ,od)	—
([bd,PC],Xn ,od)	—

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

Instruction Fields:

Effective Address field—Specifies the logical address to be evaluated and co against specified main processor address register. Only control alterable ad modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
—(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn] ,od)	110	reg. number:An
([bd,An],Xn ,od)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	—	
(d_8,PC,Xn)	—	
(bd,PC,Xn)	—	
([bd,PC,Xn] ,od)	—	
([bd,PC],Xn ,od)	—	

NOTE

The effective address field must provide the MC68851 with the effective address of the logical address to be validated, not the effective address describing where the PVALID operand is located. For example, to validate a logical address that is temporarily stored on the system stack, the instruction PVALID VAL,[(SP)] must be used since PVALID VAL,(SP) would validate the mapping on the system stack (i.e., the effective address passed to the MC68851 is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

Register field—Specifies the main processor address register to be used compare.

Assembler Syntax: RESET

Attributes: Unsized

Description: Asserts the $\overline{\text{RSTO}}$ signal for 512 (124 for MC68000, MC6 MC68HC000, MC68HC001, MC68008, MC68010, and MC68302) clock resetting all external devices. The processor state, other than the program c unaffected, and execution continues with the next instruction.

Condition Codes:

Not affected.

Instruction Format:

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

Else TRAP

Assembler Syntax: RTE

Attributes: Unsized

Description: Loads the processor state information stored in the exception stac
located at the top of the stack into the processor. The instruction examines t
format field in the format/offset word to determine how much information
restored.

Condition Codes:

Set according to the condition code bits in the status register value restored
stack.

Instruction Format:

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

Format/Offset Word (in Stack Frame):

MC68010, MC68020, MC68030, MC68040, CPU32

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

Format Field of Format/Offset Word:

Contains the format code, which implies the stack frame size (including the
offset word). For further information, refer to **Appendix B Exception Pro
Reference**.

Assembler Syntax: STOP # < data >

Attributes: Unsized

Description: Moves the immediate operand into the status register (both supervisor portions), advances the program counter to point to the next instru stops the fetching and executing of instructions. A trace, interrupt, or reset causes the processor to resume instruction execution. A trace exception instruction tracing is enabled (T0 = 1, T1 = 0) when the STOP instructio execution. If an interrupt request is asserted with a priority higher than the pri set by the new status register value, an interrupt exception occurs; other interrupt request is ignored. External reset always initiates reset exception pro

Condition Codes:

Set according to the immediate operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
0	1	0	0	1	1	1	0	0	1	1	1	0	0	
IMMEDIATE DATA														

Instruction Fields:

Immediate field—Specifies the data to be loaded into the status register.

SECTION 7
CPU32 INSTRUCTIONS

This section describes the instructions provided for the CPU32. The CPU32 can object code from an MC68000 and MC68010 and many of the instructions of the M

There are three new instructions provided for the CPU32: enter background mode low-power stop (LPSTOP), and table lookup and interpolate (TBLS, TBLSN, TB TBLUN). Table 7-1 lists the MC68020 instructions not supported by the CPU32.

Table 7-1. MC68020 Instructions Not Supported

Mnemonic	Description
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
CALLM	CALL Module
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
cpBcc	Branch on Coprocessor Condition
cpDBcc	Test Coprocessor Condition Decrement and Branch
cpGEN	Coprocessor General Function
cpRESTORE	Coprocessor Restore Function
cpSAVE	Coprocessor Save Function
cpScc	Set on Coprocessor Condition
cpTRAPcc	Trap on Coprocessor Condition
RTM	Return from Module
PACK	Pack BCD
UNPK	Unpack BCD

for the CPU32.

Table 7-2. M68000 Family Addressing Modes

Addressing Mode	Syntax	MC68000 MC68010	CPU32	M
Register Indirect	Rn	X	X	
Address Register Indirect	(An)	X	X	
Address Register Indirect with Postincrement	(An) +	X	X	
Address Register Indirect with Postdecrement	– (An)	X	X	
Address Register Indirect with Displacement	(d_{16},An)	X	X	
Address Register Indirect with Index (8-Bit Displacement)	(d_8,An,Xn)	X	X	
Address Register Indirect with Index (Base Displacement)	(d_8,An,Xn∗SCALE)		X	
Memory Indirect with Postincrement	([bd,An],Xn, od)			
Memory Indirect with Preincrement	([bd,An],Xn, od)			
Absolute Short	(xxx).W	X	X	
Absolute Long	(xxx).L	X	X	
Program Counter Indirect with Displacement	(d_{16},PC)	X	X	
Program Counter Indirect with Index (8-Bit Displacement)	(d_8,PC,Xn)	X	X	
Program Counter Indirect with Index (Base Displacement)	(d_8,PC,Xn*SC ALE)		X	
Immediate	# < data >	X	X	
PC Memory Indirect with Postincrement	([bd,PC],Xn, od)			
PC Memory Indirect with Predecrement	([bd,PC],Xn, od)			

NOTE: Xn,SIZE*SCALE—Denotes index register n (data or address), the index size (W for word, L for long word and scale factor (1, 2, 4, or 8 for no-word, long-word, or 8 for quad- word scaling, respectively).
X—Supported

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
BGND	Enter Background Mode
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bound
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
DBcc	Test Condition, Decrement, and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	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, LSR	Sign-Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LPSTOP	Low Power Stop
LSL, LSR	Logical Shift Left and Right

CCR	
Move from SR	Move from Status Register
MOVE to SR	Move to Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Sp
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Exten
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement
PEA	Push Effective Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and
RTD	Return and Deallocate
RTE	Return from Exception
RTR	Return and Restore Codes
RTS	Return from Subroutine
SBCD	Subtract Decimal with Exte
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
TBLS, TBLSN	Signed/Unsigned Table Loo Interpolate
TBLU, TBLUN	Signed/Unsigned Table Loo Interpolate
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap an Overflow
TST	Test Operand
UNLK	Unlink

PC → – (SSP)
SR → – (SSP)
(Vector) → PC

Assembler Syntax: BGND

Attributes: Size = (Unsized)

Description: The processor suspends instruction execution and enters backgrou if background mode is enabled. The freeze output is asserted to acknowledge e into background mode. Upon exiting background mode, instruction e continues with the instruction pointed to by the current program counter. If bac mode is not enabled, the processor initiates illegal instruction exception pro The vector number is generated to reference the illegal instruction exceptio Refer to the appropriate user's manual for detailed information on backgroun

Condition Codes:

X	N	Z	V	C
—	—	—	—	—

X — Not affected.
N — Not affected.
Z — Not affected.
V — Not affected.
C — Not affected.

Instruction Format:

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

STOP
Else TRAP

Assembler Syntax: LPSTOP # < data >

Attributes: Size = (Word) Privileged

Description: The immediate operand moves into the entire status register, the counter advances to point to the next instruction, and the processor stops fet executing instructions. A CPU LPSTOP broadcast cycle is executed to CPU to copy the updated interrupt mask to the external bus interface (EBI). Th clocks are stopped.

Instruction execution resumes when a trace, interrupt, or reset exception trace exception will occur if the trace state is on when the LPSTOP inst executed. If an interrupt request is asserted with a higher priority that th priority level set by the new status register value, an interrupt exceptio otherwise, the interrupt request is ignored. If the bit of the immedi corresponding to the S-bit is off, execution of the instruction will cause a violation. An external reset always initiates reset exception processing.

Condition Codes:

Set according to the immediate operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	1	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	1	1	1	0	0	0	0	
IMMEDIATE DATA														

Instruction Fields:

Immediate field—Specifies the data to be loaded into the status register.

ENTRY(n) + {(ENTRY(n + 1) – ENTRY(n)) x Dx 7 – 0} ÷ 25

Unrounded:

ENTRY(n) x 256 + {(ENTRY(n + 1) – ENTRY(n)) x Dx 7 – 0

Where ENTRY(n) and ENTRY(n + 1) are either:

1. Consecutive entries in the table pointed to by the < e indexed by Dx 15 – 8 π SIZE or;
2. The registers Dym, Dyn respectively.

Assembler Syntax:	TBLS. < size > < ea > ,Dx	Result rounded
	TBLSN. < size > < ea > ,Dx	Result not rounded
	TBLS. < size > Dym:Dyn, Dx	Result rounded
	TBLSN. < size > Dym:Dyn, Dx	Result not rounded

Attributes: Size = (Byte, Word, Long)

Description: The TBLS and TBLSN instructions allow the efficient use of piecewis compressed data tables to model complex functions. The TBLS instruction modes of operation: table lookup and interpolate mode and data register int mode.

For table lookup and interpolate mode, data register Dx 15 – 0 conta independent variable X. The effective address points to the start of a signed by or long-word table containing a linearized representation of the dependent var as a function of X. In general, the independent variable, located in the low-ord of Dx, consists of an 8-bit integer part and an 8-bit fractional part. An assum point is located between bits 7 and 8. The integer part, Dx 15 – 8, is scale operand size and is used as an offset into the table. The selected entry in the subtracted from the next consecutive entry. A fractional portion of this diffe taken by multiplying by the interpolation fraction, Dx 7 – 0 .The adjusted diffe then added to the selected table entry. The result is returned in the destinat register, Dx.

in place of the two table entries. For this mode, only the fractional portion, Dx used in the interpolation, and the integer portion, Dx 15 – 8, is ignored. The interpolation mode may be used with several table lookup and interpolations multidimensional functions.

Signed table entries range from -2^{n-1} to $2^{n-1}-1$; whereas, unsigned tab range from 0 to 2^{n-1} where n is 8, 16, or 32 for byte, word, and long-wo respectively.

Rounding of the result is optionally selected via the "R" instruction field. (TABLE), the fractional portion is rounded according to the round-to-nearest a The following table summerizes the rounding procedure:

Adjusted Difference Fraction	Rounding Adjustment
≤ – 1/2	– 1
> – 1/2 and < 1/2	+ 0
≥ 1/2	+ 1

The adjusted difference is then added to the selected table entry. The round is returned in the destination data register, Dx. Only the portion of the corresponding to the selected size is affected.

	31 24	23 16	15 8	7
BYTE	UNAFFECTED	UNAFFECTED	UNAFFECTED	RESULT
WORD	UNAFFECTED	UNAFFECTED	RESULT	RESULT
LONG	RESULT	RESULT	RESULT	RESULT

byte, the integer portion of the result is returned in Dx 15 – 8; the integer por
word result is stored in Dx 23 – 8; the least significant 24 bits of a long result ar
in Dx 31 – 8. Byte and word results are sign-extended to fill the entire 32-bit re

	31 24	23 16	15 8	7
BYTE	SIGN-EXTENDED	SIGN-EXTENDED	RESULT	FRACTION
WORD	SIGN-EXTENDED	RESULT	RESULT	FRACTION
LONG	RESULT	RESULT	RESULT	FRACTION

NOTE

The long-word result contains only the least significant 24 bits of integer precision.

For all sizes, the 8-bit fractional portion of the result is returned to the low by
data register, Dx 7 – 0. User software can make use of the fractional data to
cumulative errors in lengthy calculations or implement rounding algorithms
from that provided by other forms of TBLS. The previously described assum
point places two restrictions on the programmer:

1. Tables are limited to 257 entries in length.
2. Interpolation resolution is limited to 1/256, the distance between consec
ble entries. The assumed radix point should not, however, be construe
programmer as a requirement that the independent variable be calcula
fractional number in the range $0 < \pi < 255$. On the contrary, X should be
ered an integer in the range $0 < \pi < 65535$, realizing that the table is ac
compressed representation of a linearized function in which only every
value is actually stored in memory.

X	N	Z	V	C
—	*	*	*	0

X — Not affected.
N — Set if the most significant bit of the result is set; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the integer portion of an unrounded long result is not in the rang
≤ Result ≤ (2^{23}) – 1; cleared otherwise.
C — Always cleared.

Instruction Format:

TABLE LOOKUP AND INTERPOLATE

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

DATA REGISTER INTERPOLATE

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	1	0	0	0	0	0	0	0	0	REGIS
0	REGISTER Dx			1	R	0	1	SIZE		0	0	0	REGIS

Effective address field (table lookup and interpolate mode only)—Speci
destination location. Only control alterable addressing modes are allowed
in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	—	—
(An) +	—	—
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	

Size Field—Specifies the size of operation.
00 — Byte Operation
01 — Word Operation
10 — Long Operation

Register field—Specifies the destination data register, Dx. On entry, the
contains the interpolation fraction and entry number.

Dym, Dyn field—If the effective address mode field is nonzero, this operand re
unused and should be zero. If the effective address mode field is zero, the
interpolation variant of this instruction is implied, and Dyn specifies one o
source operands.

Rounding mode field—The R-bit controls the rounding of the final result. Whe
the result is rounded according to the round-to-nearest algorithm. When R
result is returned unrounded.

ENTRY(n) + {(ENTRY(n + 1) – ENTRY(n)) x Dx 7 – 0} ÷ 2

Unrounded:

ENTRY(n) x 256 + {(ENTRY(n + 1) – ENTRY(n)) x Dx 7 –

Where ENTRY(n) and ENTRY(n + 1) are either:

1. Consecutive entries in the table pointed to by the <
   indexed by Dx 15 – 8 π SIZE or;
2. The registers Dym, Dyn respectively

Assembler Syntax:

TBLU. < size > < ea > ,Dx	Result rounded
TBLUN. < size > < ea > ,Dx	Result not rounded
TBLU. < size > Dym:Dyn, Dx	Result rounded
TBLUN. < size > Dym:Dyn, Dx	Result not rounded

Attributes: Size = (Byte, Word, Long)

Description: The TBLU and TBLUN instructions allow the efficient use of piecew compressed data tables to model complex functions. The TBLU instructior modes of operation: table lookup and interpolate mode and data register in mode.

For table lookup and interpolate mode, data register Dx 15 – 0 con independent variable X. The effective address points to the start of a unsig word, or long-word table containing a linearized representation of the d variable, Y, as a function of X. In general, the independent variable, located i order word of Dx, consists of an 8-bit integer part and an 8-bit fractional assumed radix point is located between bits 7 and 8. The integer part, Dx scaled by the operand size and is used as an offset into the table. The selec in the table is subtracted from the next consecutive entry. A fractional porti difference is taken by multiplying by the interpolation fraction, Dx 7 – 0. The difference is then added to the selected table entry. The result is return destination data register, Dx.

in place of the two table entries. For this mode, only the fractional portion, Dx used in the interpolation and the integer portion, Dx 15 – 8, is ignored. The interpolation mode may be used with several table lookup and interpolations t multidimensional functions.

Signed table entries range from -2^{n-1} to $2^{n-1}-1$; whereas, unsigned table range from 0 to 2^{n-1} where n is 8, 16, or 32 for byte, word, and long-word respectively. The unsigned and unrounded table results will be zero-extended of sign-extended.

Rounding of the result is optionally selected via the "R" instruction field. I (TABLE), the fractional portion is rounded according to the round-to-nearest al The rounding procedure can be summarized by the following table:

Adjusted Difference Fraction	Rounding Adjustment
≥ 1/2	+ 1
< 1/2	+ 0

The adjusted difference is then added to the selected table entry. The rounde is returned in the destination data register, Dx. Only the portion of the corresponding to the selected size is affected.

	31 24	23 16	15 8	7
BYTE	UNAFFECTED	UNAFFECTED	UNAFFECTED	RESULT
WORD	UNAFFECTED	UNAFFECTED	RESULT	RESULT
LONG	RESULT	RESULT	RESULT	RESULT

If R = 1 (TBLUN), the result is returned in register Dx without rounding. If th byte, the integer portion of the result is returned in Dx 15 – 8; the integer por word result is stored in Dx 23 – 8; the least significant 24 bits of a long result ar in Dx 31 – 8. Byte and word results are sign-extended to fill the entire 32-bit re

	31 24	23 16	15 8	7
BYTE	SIGN-EXTENDED	SIGN-EXTENDED	RESULT	FRACTION
WORD	SIGN-EXTENDED	RESULT	RESULT	FRACTION
LONG	RESULT	RESULT	RESULT	FRACTION

NOTE

The long-word result contains only the least significant 24 bits of integer precision.

For all sizes, the 8-bit fractional portion of the result is returned in the low b data register, Dx 7 – 0. User software can make use of the fractional data cumulative errors in lengthy calculations or implement rounding algorithms from that provided by other forms of TBLU. The previously described assur point places two restrictions on the programmer:

1. Tables are limited to 257 entries in length.
2. Interpolation resolution is limited to 1/256, the distance between conse ble entries. The assumed radix point should not, however, be construe programmer as a requirement that the independent variable be calcula fractional number in the range $0 \leq X \leq 255$. On the contrary, X should b ered to be an integer in the range $0 \leq X \leq 65535$, realizing that the tab ally a compressed representation of a linearized function in which onl 256th value is actually stored in memory.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

X — Not affected.
N — Set if the most significant bit of the result is set; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the integer portion of an unrounded long result is not in the rang ≤ Result ≤ $(2^{23}) – 1$; cleared otherwise.
C — Always cleared.

TABLE LOOKUP AND INTERPOLATE

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

DATA REGISTER INTERPOLATE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	1	0	0	0	0	0	0	0	0	REGIST	
0	REGISTER Dx			0	R	0	0	SIZE		0	0	0	REGIST	

Instruction Fields:

Effective address field (table lookup and interpolate mode only)—Speci
destination location. Only control alterable addressing modes are allowed
in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An

Addressing Mode	Mode	
(xxx).W	111	
(xxx).L	111	
# < data >	—	
(d_{16},PC)	111	
(d_8,PC,Xn)	111	
(bd,PC,Xn)	111	

Size field—Specifies the size of operation.
00 — Byte Operation
01 — Word Operation
10 — Long Operation

contains the interpolation fraction and entry number.

Dym, Dyn field—If the effective address mode field is nonzero, this operand r
unused and should be zero. If the effective address mode field is zero, th
interpolation variant of this instruction is implied, and Dyn specifies one
source operands.

Rounding mode field—The R-bit controls the rounding of the final result. Whe
the result is rounded according to the round-to-nearest algorithm. Whe
the result is returned unrounded.

SECTION 8
INSTRUCTION FORMAT SUMMARY

This section contains a listing of the M68000 family instructions in binary format. in opcode order for the M68000 family instruction set.

8.1 INSTRUCTION FORMAT

The following paragraphs present a summary of the binary encoding fields.

8.1.1 Coprocessor ID Field

This field specifies which coprocessor in a system is to perform the operation. Wh directly supported floating-point instructions for the MC68040, this field must be se

8.1.2 Effective Address Field

This field specifies which addressing mode is to be used. For some operations, hardware-enforced restrictions on the available addressing modes allowed.

8.1.3 Register/Memory Field

This field is common to all arithmetic instructions. A zero in this field indicates a re register operation, and a one indicates an < ea > -to-register operation.

8.1.4 Source Specifier Field

This field is common to all artihmetic instructions. The value of the register/mem field affects this field,s definition. If R/M = 0, specifies the source floating-point dat (FPDR). If R/M = 1, specifies the source operand data format.

000 — Long-Word Integer (L)
001 — Single-Precision Real (S)
010 — Extended-Precision Real (X)
011 — Packed-Decimal Real (P)
100 — Word Integer (W)
101 — Double-Precision Real (D)
110 — Byte Integer (B)

This field is common to all conditional instructions and specifies the conditional tes
to be evaluated. Table 8-1 shows the binary encodings for the conditional tests.

8.1.7 Shift and Rotate Instructions

The following paragraphs define the fields used with the shift and rotate instruction

8.1.7.1 Count Register Field. If i/r = 0, this field contains the rotate (shift) count of
zero specifies 8). If i/r = 1, this field specifies a data register that contains the rota
count. The following shift and rotate fields are encoded as follows:

dr field
0 — Rotate (shift) Right
1 — Rotate (shift) Left

i/r field
0 — Immediate Rotate (shift) Count
1 — Register Rotate (shift) Count

8.1.7.2 Register Field. This field specifies a data register to be rotated (shifted).

000010	OGT	Ordered Greater Than
000011	OGE	Ordered Greater Than or Equal
000100	OLT	Ordered Less Than
000101	OLE	Ordered Less Than or Equal
000110	OGL	Ordered Greater Than or Less Than
000111	OR	Ordered
001000	UN	Unordered
001001	UEQ	Unordered or Equal
001010	UGT	Unordered or Greater Than
001011	UGE	Unordered or Greater Than or Equal
001100	ULT	Unordered or Less Than
001101	ULE	Unordered or Less Than or Equal
001110	NE	Not Equal
001111	T	True
010000	SF	Signaling False
010001	SEQ	Signaling Equal
010010	GT	Greater Than
010011	GE	Greater Than or Equal
010100	LT	Less Than
010101	LE	Less Than or Equal
010110	GL	Greater Than or Less Than
010111	GLE	Greater Than or Less Than or Equal
011000	NGLE	Not (Greater Than or Less Than or Equal)
011001	NGL	Not (Greater Than or Less Than)
011010	NLE	Not (Less Than or Equal)
011011	NLT	Not (Less Than)
011100	NGE	Not (Greater Than or Equal)
011101	NGT	Not (Greater Than)
011110	SNE	Signaling Not Equal
011111	ST	Signaling True

10 — Long Operation

8.1.9 Opmode Field

Refer to the applicable instruction descriptions for the encoding of this field in **Se Integer Instructions**, **Section 5 Floating Point Instructions**, **Section 6 Sup (Privaleged) Instructions**, and **Section 7 CPU32 Instructions**.

8.1.10 Address/Data Field

This field specifies the type of general register. The encoding is:

0 — Data Register
1 — Address Register

8.2 OPERATION CODE MAP

Table 8-2 lists the encoding for bits 15 – 12 and the operation performed.

Table 8-2. Operation Code Map

Bits 15 – 12	Operation
0000	Bit Manipulation/MOVEP/Immed iate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/Scc/DBcc/TRAPc c
0110	Bcc/BSR/BRA
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned, Reserved)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate/Bit Field
1111	Coprocessor Interface/MC68040 and CPU32 Extensions

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

ORI

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

ANDI to CCR

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

ANDI to SR

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

ANDI

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

RTM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	0	0	1	1	0	1	1	0	0	D/A	REGIS	

CALLM

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

ADDI

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

CMP2

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

CHK2

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

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

EORI

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

CMPI

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

BTST

BIT NUMBER STATIC, SPECIFIED AS IMMEDIATE DATA

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

BCHG

BIT NUMBER STATIC, SPECIFIED AS IMMEDIATE DATA

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

BSET

BIT NUMBER STATIC, SPECIFIED AS IMMEDIATE DATA

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

MOVES

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

CAS2

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

CAS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	0	1	SIZE		0	1	1	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	0	0	0	0	0	0	Du			0	0	0	D	

BTST

BIT NUMBER DYNAMIC, SPECIFIED IN A REGISTER

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

BCLR

BIT NUMBER DYNAMIC, SPECIFIED IN A REGISTER

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

BSET

BIT NUMBER DYNAMIC, SPECIFIED IN A REGISTER

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

MOVEP

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	0	0	0	DATA REGISTER			OPMODE			0	0	1	ADD REG
16-BIT DISPLACEMENT													

MOVEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	0	SIZE		DESTINATION REGISTER			0	0	1	SOURCE			
										MODE		REG	

MOVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	0	SIZE		DESTINATION						SOURCE			
				REGISTER			MODE			MODE		REG	

MOVE from SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	0	0	0	0	1	1	SOURCE			
										MODE		REG	

0	1	0	0	0	0	0	0	SIZE1	EFFECTIVE ADDRESS MODE	REGI

CLR

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

MOVE to CCR

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

NEG

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

NOT

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

MOVE to SR

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

LONG

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	1	0	0	0	0	0	0	0	1	REG
HIGH-ORDER DISPLACEMENT													
LOW-ORDER DISPLACEMENT													

NBCD

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

SWAP

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

BKPT

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

PEA

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

BGND

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

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

TST

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

MULU

LONG

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

MULS

LONG

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

DIVU, DIVUL

LONG

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

TRAP

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

LINK

WORD

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	1	1	1	0	0	1	0	1	0	REG
WORD DISPLACEMENT													

UNLK

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

MOVE USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	1	1	1	0	0	1	1	0	dr	REG

RESET

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

NOP

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

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

RTD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0
16-BIT DISPLACEMENT														

RTS

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

TRAPV

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

RTR

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

MOVEC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1
A/D	REGISTER			CONTROL REGISTER										

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

MOVEM

15	14	13	12	11	10	9	8	7	6	5	4	3	2
0	1	0	0	1	dr	0	0	1	SIZE	EFFECTIVE ADDRESS			
										MODE			REG
REGISTER LIST MASK													

LEA

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

CHK

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

ADDQ

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

SUBQ

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	1	CONDITION				1	1	1	1	1	OPM	
OPTIONAL WORD														
OR LONG WORD														

Scc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	1	CONDITION				1	1	EFFECTIVE ADDRESS				
										MODE			REGI	

BRA

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

BSR

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

Bcc

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

WORD

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

SBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	0	0	0	REGISTER Dy/Ay			1	0	0	0	0	R/M	REGIST

PACK

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	0	0	0	REGISTER Dy/Ay			1	0	1	0	0	R/M	REGIST
16-BIT EXTENSION: ADJUSTMENT													

UNPK

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	0	0	0	REGISTER Dy/Ay			1	1	0	0	0	R/M	REGIST
16-BIT EXTENSION: ADJUSTMENT													

DIVS, DIVSL

WORD

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

OR

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

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

SUBA

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

CMPM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	1	REGISTER Ax			1	SIZE		0	0	1	REGIST	

CMP

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

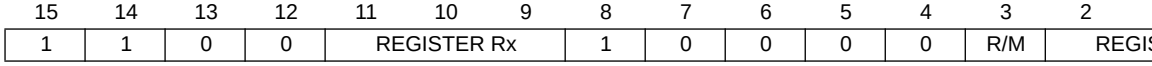
CMPA

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

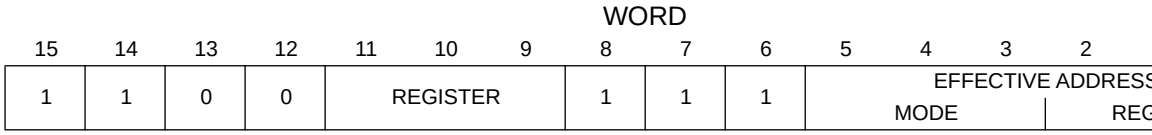
EOR

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

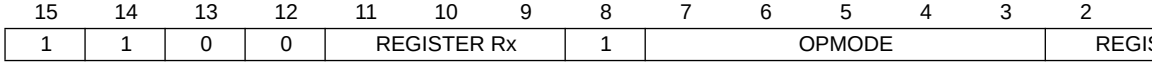
ABCD



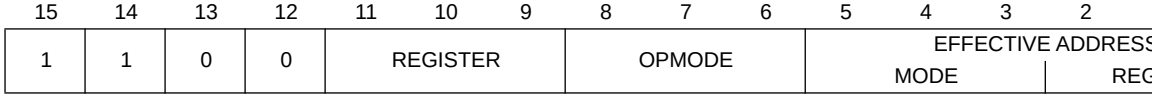
MULS



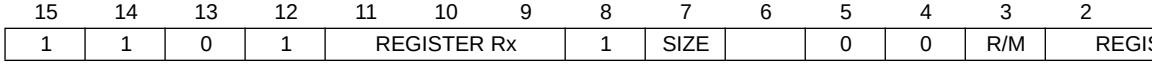
EXG



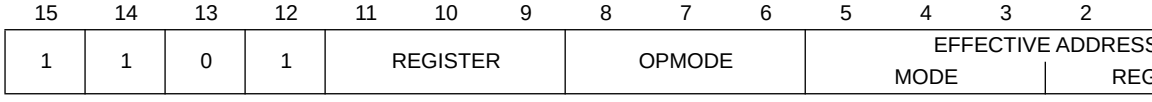
AND



ADDX



ADDA



MEMORY SHIFT

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

LSL, LSR

MEMORY SHIFT

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

ROXL, ROXR

MEMORY ROTATE

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

ROL, ROR

MEMORY ROTATE

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

BFTST

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

BFCHG

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

BFEXTS

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

BFCLR

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

BFFFO

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

BFSET

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

ASL, ASR

REGISTER SHIFT

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

LSL, LSR

REGISTER SHIFT

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

ROXL, ROXR

REGISTER ROTATE

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

ROL, ROR

REGISTER ROTATE

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

PMOVE

MC68EC030, ACX REGISTERS

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS MODE	REGI
0	0	0	P REGISTER			R/W	0	0	0	0 0 0	0 0

PLOAD

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	0	1	0	0	0	R/W	0	0	0	0	FC		

PVALID

VAL CONTAINS ACCESS LEVEL TO TEST AGAINST

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

PVALID

MAIN PROCESSOR REGISTER CONTAINS ACCESS LEVEL TO TEST AGAINST

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

PFLUSH

MC68030 ONLY

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	0	1	MODE			0	0	MASK			FC		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGI	
0	0	1	MODE			0	MASK			FC				

PMOVE

MC68851, TO/FROM TC, CRP, DRP, SRP, CAL, VAL, SCC, AND AC REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGI	
0	1	0	P REGISTER			R/W	0	0	0	0	0	0	0	0

PMOVE

MC68030 ONLY, SRP, CRP, AND TC REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGI	
0	1	0	P REGISTER			R/W	FD	0	0	0	0	0	0	0

PMOVE

MC68030 ONLY, MMUSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGI	
0	1	1	0	0	0	R/W	0	0	0	0	0	0	0	0

PMOVE

MC68EC030, ACUSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS				
										MODE			REGI	
0	1	1	0	0	0	R/W	0	0	0	0	0	0	0	0

PMOVE

MC68851, TO/FROM BADx AND BACx REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
0	1	1	P REGISTER			R/W	0	0	0	0	NUM		

PTEST

MC68EC030

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
1	0	0	0	0	0	R/W	0	REGISTER		FC			

PTEST

MC68030 ONLY

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
1	0	0	LEVEL			R/W	A	REGISTER		FC			

PTEST

MC68851

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS			
										MODE			REG
1	0	0	LEVEL			R/W	A REGISTER			FC			

PScc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	1	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION				

PDBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	1	0	0	1	COUNT R	
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION				
16-BIT DISPLACEMENT														

PTRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	0	1	1	1	1	OPM	
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION				
16-BIT OPERAND OR MOST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)														
LEAST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)														

PBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	0	0	0	1	SIZE	MC68851 CONDITION				
16-BIT DISPLACEMENT OR MOST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT														
LEAST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT (IF NEEDED)														

PSAVE

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

MC68EC040, POSTINCREMENT SOURCE AND DESTINATION

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	0	1	0	0	0	OPMODE		REG

PFLUSH

MC68040/MC68LC040

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	0	1	0	0	0	OPMODE		REG

PTEST

MC68040/MC68LC040

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	0	1	0	1	R/W	0	1	REG

PTEST

MC68EC040

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	0	1	0	1	R/W	0	1	REG

CINV

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	0	0	CACHE		0	SCOPE		REG

CPUSH

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	1	0	0	CACHE		1	SCOPE		REG

MOVE16

POSTINCREMENT SOURCE AND DESTINATION

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	0	1	1	0	0	0	1	0	0	REGIST	
1	REGISTER Ay			0	0	0	0	0	0	0	0	0	0	0

TBLU, TBLUN

TABLE LOOKUP AND INTERPOLATE

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

TBLS, TBLSN

TABLE LOOKUP AND INTERPOLATE

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

TBLU, TBLUN

DATA REGISTER INTERPOLATE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	1	0	0	0	0	0	0	0	0	REGIST	
0	REGISTER Dx			0	R	0	0	SIZE		0	0	0	REGIST	

LPSTOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	1	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	1	1	1	0	0	0	0	
IMMEDIATE DATA														

FMOVECR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	0	0	0	0	
0	1	0	1	1	1	DESTINATION REGISTER			ROM OFFSET					

FINT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE		REG		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	

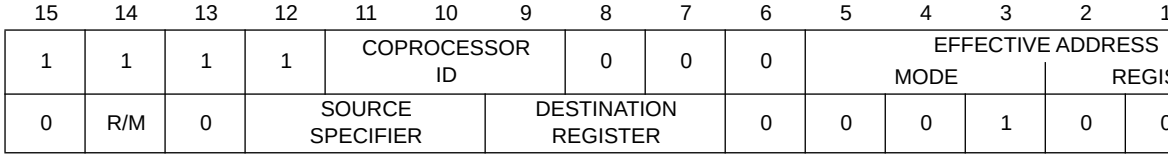
FSINH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE		REG		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	

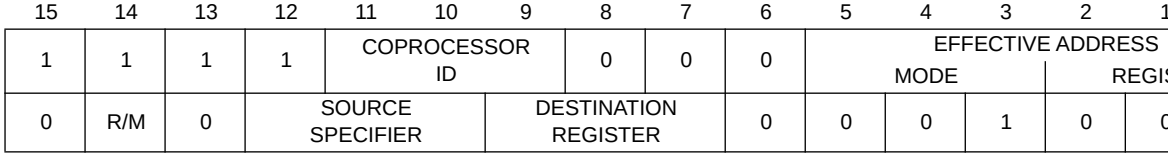
FINTRZ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE		REG		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	

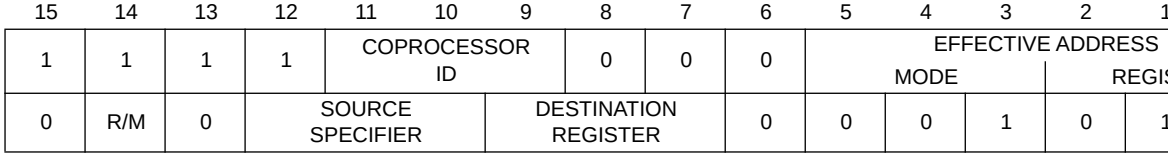
FETOXM1



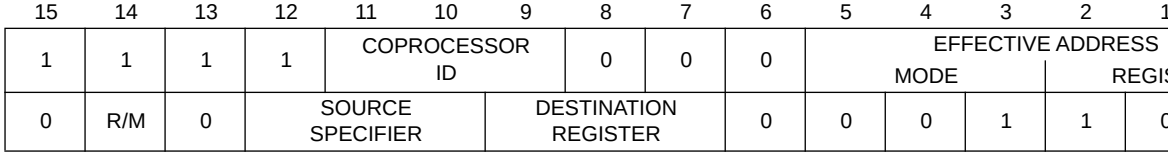
FTANH



FATAN



FASIN



FSIN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	0	1	1

FTAN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	0	1	1

FETOX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	1	0	0

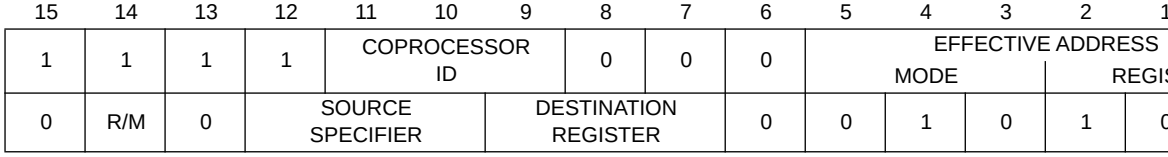
FTWOTOX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	1	0	0

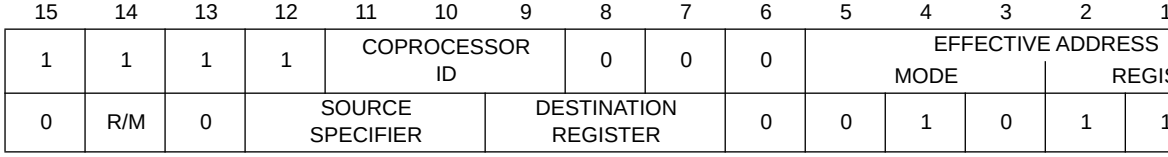
FTENTOX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	1	0	0

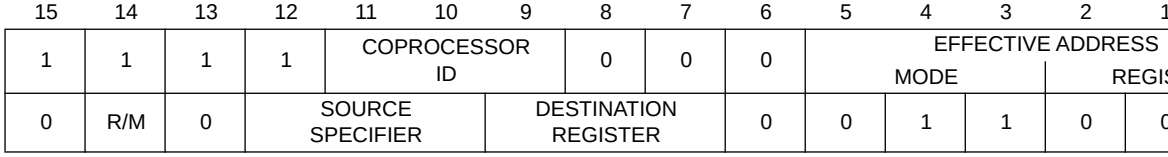
FLOG10



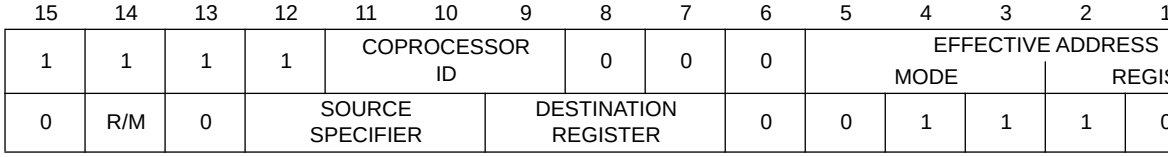
FLOG2



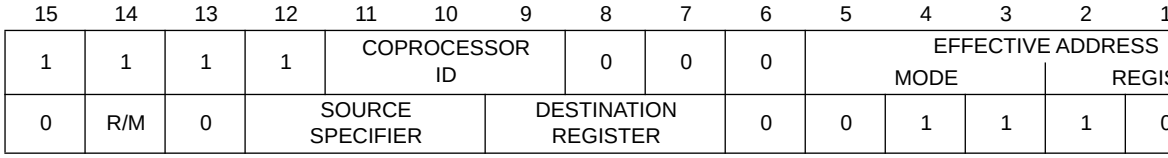
FCOSH



FACOS



FCOS



FGETMAN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	0	1	1	1

FMOD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	0	0	0

FSGLDIV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	0	0	1

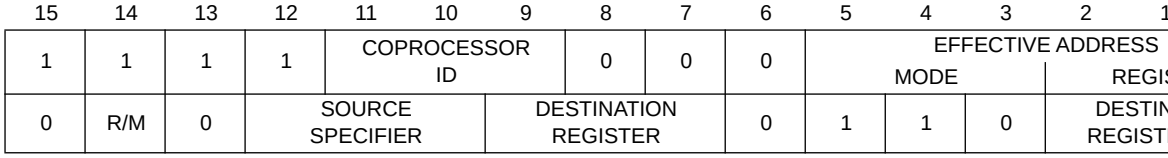
FREM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	0	0	1

FSCALE

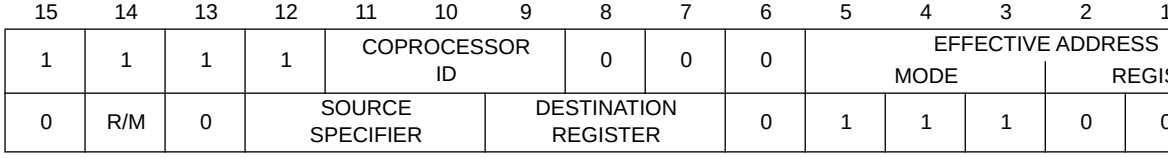
15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REG	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	0	0	1

FSINCOS



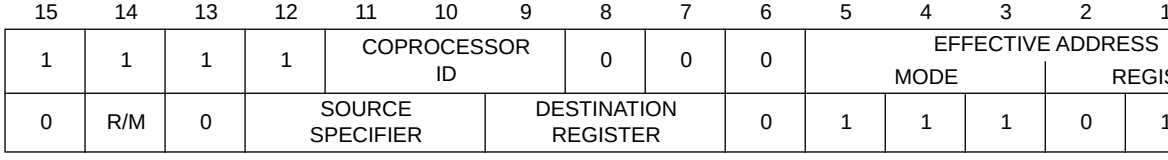
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	1	0	DESTIN
														REGIST

FCMP



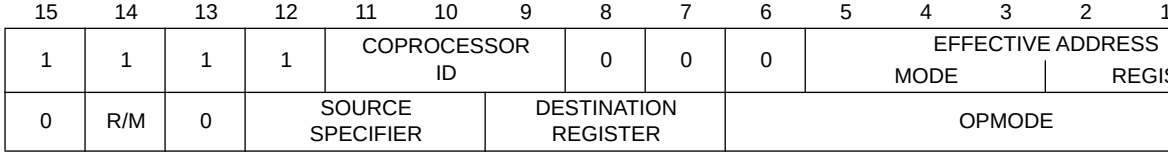
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	1	1	0	0

FTST



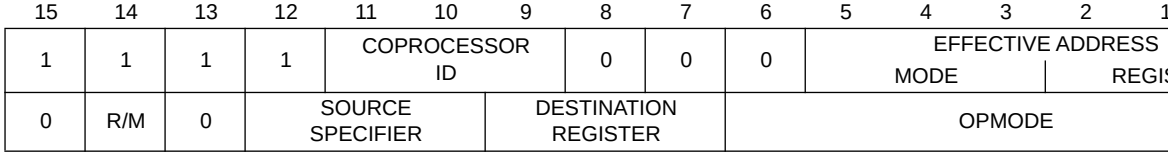
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			0	1	1	1	0	1

FABS



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			OPMODE				

FADD



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS				
										MODE			REGIS	
0	R/M	0	SOURCE SPECIFIER				DESTINATION REGISTER			OPMODE				

FMOVE

DATA REGISTER, EFFECTIVE ADDRESS TO REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE		REG	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE				

FMUL

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE		REG	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE				

FNEG

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE		REG	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE				

FSQRT

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE		REG	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE				

FSUB

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS			
										MODE		REG	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE				

			SPECIFIER		REGISTER				(IF REQUIRED)					

FMOVE

SYSTEM CONTROL REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
1	0	dr	REGISTER SELECT			0	0	0	0	0	0	0	0	0

FMOVEM

CONTROL REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
1	0	dr	REGISTER SELECT			0	0	0	0	0	0	0	0	0

FMOVEM

DATA REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
1	1	dr	MODE		0	0	0	REGISTER LIST						

cpGEN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGIS	
COPROCESSOR ID-DEPENDENT COMMAND WORD														
OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR ID-DEFINED EXTENSION WORDS														

cpScc

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	0	1	EFFECTIVE ADDRESS			
										MODE		REG	
0	0	0	0	0	0	0	0	0	0	COPROCESSOR ID CONDI			
OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR ID-DEFINED EXTENSION WORDS													

FBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	1	SIZE	CONDITIONAL PREDICATE			
16-BIT DISPLACEMENT OR MOST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT													
LEAST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT (IF NEEDED)													

cpBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			0	1	SIZE	COPROCESSOR ID CONDI			
OPTIONAL COPROCESSOR ID-DEFINED EXTENSION WORDS													
WORD OR													
LONG-WORD DISPLACEMENT													

cpSAVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			1	0	0	EFFECTIVE ADDRESS			
										MODE		REG	

FSAVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	COPROCESSOR ID			1	0	0	EFFECTIVE ADDRESS			
										MODE		REG	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			1	0	1	EFFECTIVE ADDRESS MODE			REGIS	

FDBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	1	0	0	1	COU REGIS	
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICAT				
16-BIT DISPLACEMENT														

cpDBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	1	0	0	1	REGIS	
0	0	0	0	0	0	0	0	0	0	COPROCESSOR ID CONDIT				
OPTIONAL COPROCESSOR ID-DEFINED EXTENSION WORDS														
16-BIT DISPLACEMENT														

FTRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	COPROCESSOR ID			0	0	1	1	1	1	MO	
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICAT				
16-BIT OPERAND OR MOST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)														
LEAST SIGNIFICANT WORD OR 32-BIT OPERAND (IF NEEDED)														

OPTIONAL WORD
OR LONG-WORD OPERAND

FNOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	
1	1	1	1	COPROCESSOR ID			0	1	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	

APPENDIX A
PROCESSOR INSTRUCTION SUMMARY

This appendix provides a quick reference of the M68000 family instructions. The
tion of this section is by processors and their addressing modes. All referenc
MC68000, MC68020, and MC68030 include references to the corresponding e
controllers, MC68EC000, MC68EC020, and MC68EC030. All references to the I
include the MC68LC040 and MC68EC040. This referencing applies throughout th
unless otherwise specified. Table A-1 lists the M68000 family instructions by mnem
indicates which processors they apply to.

Table A-1. M68000 Family Instruction Set And Processor Cross-Reference

Mnemonic	68000	68008	68010	68020	68030	68040	68881/ 68882	68851
ABCD	X	X	X	X	X	X		
ADD	X	X	X	X	X	X		
ADDA	X	X	X	X	X	X		
ADDI	X	X	X	X	X	X		
ADDQ	X	X	X	X	X	X		
ADDX	X	X	X	X	X	X		
AND	X	X	X	X	X	X		
ANDI	X	X	X	X	X	X		
ANDI to CCR	X	X	X	X	X	X		
ANDI to SR[1]	X	X	X	X	X	X		
ASL, ASR	X	X	X	X	X	X		
Bcc	X	X	X	X	X	X		
BCHG	X	X	X	X	X	X		
BCLR	X	X	X	X	X	X		
BFCHG				X	X	X		
BFCLR				X	X	X		
BFEXTS				X	X	X		
BFEXTU				X	X	X		
BFFFO				X	X	X		

BFSET				X	X	X			
BFTST				X	X	X			
BGND									
BKPT			X	X	X	X			
BRA	X	X	X	X	X	X			
BSET	X	X	X	X	X	X			
BSR	X	X	X	X	X	X			
BTST	X	X	X	X	X	X			
CALLM				X					
CAS, CAS2				X	X	X			
CHK	X	X	X	X	X	X			
CHK2				X	X	X			
CINV[1]						X			
CLR	X	X	X	X	X	X			
CMP	X	X	X	X	X	X			
CMPA	X	X	X	X	X	X			
CMPI	X	X	X	X	X	X			
CMPM	X	X	X	X	X	X			
CMP2				X	X	X			
cpBcc				X	X				
cpDBcc				X	X				
cpGEN				X	X				
cpRESTORE[1]				X	X				
cpSAVE[1]				X	X				
cpScc				X	X				
cpTRAPcc				X	X				
CPUSH[1]						X			
DBcc	X	X	X	X	X	X			
DIVS	X	X	X	X	X	X			
DIVSL				X	X	X			
DIVU	X	X	X	X	X	X			
DIVUL				X	X	X			

EORI	X	X	X	X	X	X		
EORI to CCR	X	X	X	X	X	X		
EORI to SR[1]	X	X	X	X	X	X		
EXG	X	X	X	X	X	X		
EXT	X	X	X	X	X	X		
EXTB				X	X	X		
FABS						X[2]	X	
FSABS, FDABS						X[2]		
FACOS						2,3	X	
FADD						X[2]	X	
FSADD, FDADD						X[2]		
FASIN						2,3	X	
FATAN						2,3	X	
FATANH						2,3	X	
FBcc						X[2]	X	
FCMP						X[2]	X	
FCOS						2,3	X	
FCOSH						2,3	X	
FDBcc						X[2]	X	
FDIV						X[2]	X	
FSDIV, FDDIV						X[2]		
FETOX						2,3	X	
FETOXM1						2,3	X	
FGETEXP						2,3	X	
FGETMAN						2,3	X	
FINT						2,3	X	
FINTRZ						2,3	X	
FLOG10						2,3	X	
FLOG2						2,3	X	
FLOGN						2,3	X	

FMOD						2,3	X		
FMOVE						X[2]	X		
FSMOVE, FDMOVE						X[2]			
FMOVECR						2,3	X		
FMOVEM						X[2]	X		
FMUL						X[2]	X		
FSMUL, FDMUL						X[2]			
FNEG						X[2]	X		
FSNEG, FDNEG						X[2]			
FNOP						X[2]	X		
FREM						2,3	X		
FRESTORE[1]						X[2]	X		
FSAVE*						X[2]	X		
FSCALE						2,3	X		
FScc						X[2]	X		
FSGLDIV						2,3	X		
FSGLMUL						2,3	X		
FSIN						2,3	X		
FSINCOS						2,3	X		
FSINH						2,3	X		
FSQRT						X[2]	X		
FSSQRT, FDSQRT						X[2]			
FSUB						X[2]	X		
FSSUB, FDSUB						X[2]			
FTAN						2,3	X		
FTANH						2,3	X		
FTENTOX						2,3	X		
FTRAPcc						X[2]	X		
FTST						X[2]	X		

ILLEGAL	X	X	X	X	X	X		
JMP	X	X	X	X	X	X		
JSR	X	X	X	X	X	X		
LEA	X	X	X	X	X	X		
LINK	X	X	X	X	X	X		
LPSTOP								
LSL,LSR	X	X	X	X	X	X		
MOVE	X	X	X	X	X	X		
MOVEA	X	X	X	X	X	X		
MOVE from CCR			X	X	X	X		
MOVE to CCR	X	X	X	X	X	X		
MOVE from SR[1]	4	4	X	X	X	X		
MOVE to SR[1]	X	X	X	X	X	X		
MOVE USP[1]	X	X	X	X	X	X		
MOVE16						X		
MOVEC[1]			X	X	X	X		
MOVEM	X	X	X	X	X	X		
MOVEP	X	X	X	X	X	X		
MOVEQ	X	X	X	X	X	X		
MOVES[1]			X	X	X	X		
MULS	X	X	X	X	X	X		
MULU	X	X	X	X	X	X		
NBCD	X	X	X	X	X	X		
NEG	X	X	X	X	X	X		
NEGX	X	X	X	X	X	X		
NOP	X	X	X	X	X	X		
NOT	X	X	X	X	X	X		
OR	X	X	X	X	X	X		

ORI to CCR	X	X	X	X	X	X			
ORI to SR[1]	X	X	X	X	X	X			
PACK				X	X	X			
PBcc[1]								X	
PDBcc[1]								X	
PEA	X	X	X	X	X	X			
PFLUSH[1]					X[5]	X		X	
PFLUSHA[1]					X[5]			X	
PFLUSHR[1]								X	
PFLUSHS[1]								X	
PLOAD[1]					X[5]			X	
PMOVE[1]					X			X	
PRESTORE[1]								X	
PSAVE[1]								X	
PScc[1]								X	
PTEST[1]					X	X		X	
PTRAPcc[1]								X	
PVALID								X	
RESET[1]	X	X	X	X	X	X			
ROL,ROR	X	X	X	X	X	X			
ROXL, ROXR	X	X	X	X	X	X			
RTD			X	X	X	X			
RTE[1]	X	X	X	X	X	X			
RTM				X					
RTR	X	X	X	X	X	X			
RTS	X	X	X	X	X	X			
SBCD	X	X	X	X	X	X			
Scc	X	X	X	X	X	X			
STOP[1]	X	X	X	X	X	X			
SUB	X	X	X	X	X	X			
SUBA	X	X	X	X	X	X			
SUBI	X	X	X	X	X	X			
SUBQ	X	X	X	X	X	X			
SUBX	X	X	X	X	X	X			

TAS	X	X	X	X	X	X		
TBLS, TBLSN								
TBLU, TBLUN								
TRAP	X	X	X	X	X	X		
TRAPcc				X	X	X		
TRAPV	X	X	X	X	X	X		
TST	X	X	X	X	X	X		
UNLK	X	X	X	X	X	X		
UNPK				X	X	X		

NOTES:

1. Privileged (Supervisor) Instruction.
2. Not applicable to MC68EC040 and MC68LC040
3. These instructions are software supported on the MC68040.
4. This instruction is not privileged for the MC68000 and MC68008.
5. Not applicable to MC68EC030.

ABCD	Add Decimal with Extend
ADD	Add
ADDA	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
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BGND	Enter Background Mode
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CALLM	CALL Module
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CINV	Invalidate Cache Entries
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
cpBcc	Branch on Coprocessor Condition
cpDBcc	Test Coprocessor Condition Decrement and Branch
cpGEN	Coprocessor General Function
cpRESTORE	Coprocessor Restore Function

CPUSH	Push then Invalidate Cache Entries
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	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, EXTB	Sign Extend
FABS	Floating-Point Absolute Value
FSFABS, FDFABS	Floating-Point Absolute Value (Single/Double Precision)
FACOS	Floating-Point Arc Cosine
FADD	Floating-Point Add
FSADD, FDADD	Floating-Point Add (Single/Double Precision)
FASIN	Floating-Point Arc Sine
FATAN	Floating-Point Arc Tangent
FATANH	Floating-Point Hyperbolic Arc Tangent
FBcc	Floating-Point Branch
FCMP	Floating-Point Compare
FCOS	Floating-Point Cosine
FCOSH	Floating-Point Hyperbolic Cosine
FDBcc	Floating-Point Decrement and Branch
FDIV	Floating-Point Divide
FSDIV, FDDIV	Floating-Point Divide (Single/Double Precision)
FETOX	Floating-Point ex
FETOXM1	Floating-Point ex - 1
FGETEXP	Floating-Point Get Exponent
FGETMAN	Floating-Point Get Mantissa
FINT	Floating-Point Integer Part
FINTRZ	Floating-Point Integer Part, Round-to-Zero
FLOG10	Floating-Point Log10
FLOG2	Floating-Point Log2
FLOGN	Floating-Point Loge
FLOGNP1	Floating-Point Loge (x + 1)
FMOD	Floating-Point Modulo Remainder
FMOVE	Move Floating-Point Register
FSMOVE,FDMOVE	Move Floating-Point Register (Single/Double Precision)
FMOVECR	Move Constant ROM
FMOVEM	Move Multiple Floating-Point Registers
FMUL	Floating-Point Multiply
FSMUL,FDMUL	Floating-Point Multiply (Single/Double Precision)
FNEG	Floating-Point Negate
FSNEG,FDNEG	Floating-Point Negate (Single/Double Precision)
FNOP	Floating-Point No Operation

FSCALE	Floating-Point Scale Exponent
FScc	Floating-Point Set According to Condition
FSGLDIV	Single-Precision Divide
FSGLMUL	Single-Precision Multiply
FSIN	Sine
FSINCOS	Simultaneous Sine and Cosine
FSINH	Hyperbolic Sine
FSQRT	Floating-Point Square Root
FSSQRT,FDSQRT	Floating-Point Square Root (Single/Double Precision)
FSUB	Floating-Point Subtract
FSSUB,FDSUB	Floating-Point Subtract (Single/Double Precision)
FTAN	Tangent
FTANH	Hyperbolic Tangent
FTENTOX	Floating-Point 10x
FTRAPcc	Floating-Point Trap On Condition
FTST	Floating-Point Test
FTWOTOX	Floating-Point 2x
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LPSTOP	Low-Power Stop
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE from CCR	Move from Condition Code Register
MOVE from SR	Move from Status Register
MOVE to CCR	Move to Condition Code Register
MOVE to SR	Move to Status Register
MOVE USP	Move User Stack Pointer
MOVE16	16-Byte Block Move
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement

ORI to SR	Inclusive-OR Immediate to Status Register
PACK	Pack BCD
PBcc	Branch on PMMU Condition
PDBcc	Test, Decrement, and Branch on PMMU Condition
PEA	Push Effective Address
PFLUSH	Flush Entry(ies) in the ATCs
PFLUSHA	Flush Entry(ies) in the ATCs
PFLUSHR	Flush Entry(ies) in the ATCs and RPT Entries
PFLUSHS	Flush Entry(ies) in the ATCs
PLOAD	Load an Entry into the ATC
PMOVE	Move PMMU Register
PRESTORE	PMMU Restore Function
PSAVE	PMMU Save Function
PScc	Set on PMMU Condition
PTEST	Test a Logical Address
PTRAPcc	Trap on PMMU Condition
PVALID	Validate a Pointer
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTM	Return from Module
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
TBLS, TBLSN	Signed Table Lookup with Interpolate
TBLU, TBLUN	Unsigned Table Lookup with Interpolate
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

Table A-3 lists the instructions used with the MC68000 and MC68008 processors, a
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

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

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
BKPT	Breakpoint
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

MOVE	Move
MOVEA	Move Address
MOVE from CCR	Move from Condition Code Register
MOVE from SR	Move from Status Register
MOVE to CCR	Move to Condition Code Register
MOVE to SR	Move to Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Address Space
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
RTD	Return and Deallocate
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

Data Addressing Modes

Mode	Generation
Register Direct Addressing	
Data Register Direct	<ea> = Dn
Address Register Direct	<ea> = An
Absolute Data Addressing	
Absolute Short	<ea> = (Next Word)
Absolute Long	<ea> = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	<ea> = (PC) + d_{16}
Relative with Index and Offset	<ea> = (PC) + d_8
Register Indirect Addressing	
Register Indirect	<ea> = (An)
Postincrement Register Indirect	<ea> = (An), An ⁻An + N
Predecrement Register Indirect	An ⁻An–N, <ea> = (An)
Register Indirect with Offset	<ea> = (An) + d_{16}
Indexed Register Indirect with Offset	<ea> = (An) + (Xn) + d_8
Immediate Data Addressing	
Immediate	DATA = Next Word(s)
Quick Immediate	Inherent Data
Implied Addressing	<ea> = SR, USP, SSP, PC, VBR,
Implied Register	SFC, DFC

N = 1 for byte, 2 for word, and 4 for long word. If An is the stack pointer and the operand size is byte, N = 2 to keep the stack pointer on a word boundary.

Table A-6 lists the instructions used with the MC68020 processors.

Table A-6. MC68020 Instruction Set Summary

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
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CALLM	CALL Module
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CLR	Clear
CMP	Compare
CMP2	Compare Register Against Upper and Lower Bounds
CMPA	Compare Address
CMPI	Compare Immediate

cpDBcc	Test Coprocessor Condition, Decrement and Branch
cpGEN	Coprocessor General Function
cpRESTORE	Coprocessor Restore Function
cpSAVE	Coprocessor Save Function
cpScc	Set on Coprocessor Condition
cpTRACPcc	Trap on Coprocessor Condition
DBcc	Test Condition, Decrement, and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	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, EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE from CCR	Move from Condition Code Register
MOVE from SR	Move from Status Register
MOVE to CCR	Move to Condition Code Register
MOVE to SR	Move to Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement

ORI to CCR	Inclusive-OR Immediate to Condition Code Register
ORI to SR	Inclusive-OR Immediate to Status Register
PACK	Pack BCD
PEA	Push Effective Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTM	Return from Module
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
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

Addressing Modes	Syntax
Register Direct	
Address Register Direct	Dn
Address Register Direct	An
Register Indirect	
Address Register Indirect	(An)
Address Register Indirect with Postincrement	(An)+
Address Register Indirect with Predecrement	–(An)
Address Register Indirect with Displacement	(d_{16},An)
Register Indirect with Index	
Address Register Indirect with Index (8-Bit Displacement)	(d_8,An,Xn)
Address Register Indirect with Index (Base Displacement)	(bd,An,Xn)
Memory Indirect	
Memory Indirect Postindexed	([bd,An],Xn,od)
Memory Indirect Preindexed	([bd,An,Xn],od)
Program Counter Indirect with Displacement	(d_{16},PC)
Program Counter Indirect with Index	
PC Indirect with Index (8-Bit Displacement)	(d_8,PC,Xn)
PC Indirect with Index (Base Displacement)	(bd,PC,Xn)
Program Counter Memory Indirect	
PC Memory Indirect Postindexed	([bd,PC],Xn,od)
PC Memory Indirect Preindexed	([bd,PC,Xn],od)
Absolute	
Absolute Short	(xxx).W
Absolute Long	(xxx).L
Immediate	#<data>

Table A-8 lists the instructions used with the MC68030 processors.

Table A-8. MC68030 Instruction Set Summary

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
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory

cpDBcc	Test Coprocessor Condition, Decrement and Branch
cpGEN	Coprocessor General Function
cpRESTORE	Coprocessor Restore Function
cpSAVE	Coprocessor Save Function
cpScc	Set on Coprocessor Condition
cpTRAPcc	Trap on Coprocessor Condition
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	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, EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE from CCR	Move from Condition Code Register
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
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement

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
PACK	Pack BCD
PEA	Push Effective Address
PFLUSH*	Invalidate Entries in the ATC
PFLUSHA*	Invalidate all Entries in the ATC
PLOAD*	Load an Entry into the ATC
PMOVE	Move PMMU Register
PTEST	Get Information about Logical Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
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
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

*Not applicable to the MC68EC030

Addressing Modes	Syntax
Register Direct	
Data Register Direct	Dn
Address Register Direct	An
Register Indirect	
Address Register Indirect	(An)
Address Register Indirect with Postincrement	(An)+
Address Register Indirect with Predecrement	–(An)
Address Register Indirect with Displacement	(d_{16},An)
Register Indirect with Index	
Address Register Indirect with Index (8-Bit Displacement)	(d_8,An,Xn)
Address Register Indirect with Index (Base Displacement)	(bd,An,Xn)
Memory Indirect	
Memory Indirect Postindexed	([bd,An],Xn,od)
Memory Indirect Preindexed	([bd,An,Xn],od)
Program Counter Indirect with Displacement	(d_{16},PC)
Program Counter Indirect with Index	
PC Indirect with Index (8-Bit Displacement)	(d_8,PC,Xn)
PC Indirect with Index (Base Displacement)	(bd,PC,Xn)
Program Counter Memory Indirect	
PC Memory Indirect Postindexed	([bd,PC],Xn,od)
PC Memory Indirect Preindexed	([bd,PC,Xn],od)
Absolute	
Absolute Short	(xxx).W
Absolute Long	(xxx).L
Immediate	#<data>

Table A-10 lists the instructions used with the MC68040 processor.

Table A-10. MC68040 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
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CINV	Invalidate Cache Entries
CLR	Clear
CMP	Compare
CMPA	Compare Address

CPUSH	Push then Invalidate Cache Entries
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	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, EXTB	Sign Extend
FABS[1]	Floating-Point Absolute Value
FSABS, FDABS[1]	Floating-Point Absolute Value (Single/Double Precision)
FACOS[1,2]	Floating-Point Arc Cosine
FADD[1]	Floating-Point Add
FSADD, FDADD[1]	Floating-Point Add (Single/Double Precision)
FASIN[1,2]	Floating-Point Arc Sine
FATAN[1,2]	Floating-Point Arc Tangent
FATANH[1,2]	Floating-Point Hyperbolic Arc Tangent
FBcc[1]	Floating-Point Branch
FCMP[1]	Floating-Point Compare
FCOS[1,2]	Floating-Point Cosine
FCOSH[1,2]	Floating-Point Hyperbolic Cosine
FDBcc[1]	Floating-Point Decrement and Branch
FDIV[1]	Floating-Point Divide
FSDIV, FDDIV[1]	Floating-Point Divide (Single/Double Precision)
FETOX[1,2]	Floating-Point e^x
FETOXM1[1,2]	Floating-Point $e^x - 1$
FGETEXP[1,2]	Floating-Point Get Exponent
FGETMAN[1,2]	Floating-Point Get Mantissa
FINT[1,2]	Floating-Point Integer Part
FINTRZ[1,2]	Floating-Point Integer Part, Round-to-Zero
FLOG10[1,2]	Floating-Point $\log_{10}$
FLOG2[1,2]	Floating-Point $\log_2$
FLOGN[1,2]	Floating-Point $\log_e$
FLOGNP1[1,2]	Floating-Point $\log_e (x + 1)$
FMOD[1,2]	Floating-Point Modulo Remainder
FMOVE[1]	Move Floating-Point Register
FSMOVE, FDMOVE[1]	Move Floating-Point Register (Single/Double Precision)
FMOVECR[1]	Move Constant ROM
FMOVEM[1]	Move Multiple Floating-Point Registers
FMUL[1]	Floating-Point Multiply
FSMUL, FDMUL[1]	Floating-Point Multiply (Single/Double Precision)

FREM[1,2]	IEEE Remainder
FRESTORE[1]	Restore Floating-Point Internal State
FSAVE[1]	Save Floating-Point Internal State
FSCALE[1,2]	Floating-Point Scale Exponent
FScc[1]	Floating-Point Set According to Condition
FSGLDIV[1,2]	Single-Precision Divide
FSGLMUL[1,2]	Single-Precision Multiply
FSIN[1,2]	Sine
FSINCOS[1,2]	Simultaneous Sine and Cosine
FSINH[1,2]	Hyperbolic Sine
FSQRT[1]	Floating-Point Square Root
FSSQRT, FDSQRT[1]	Floating-Point Square Root (Single/Double Precision)
FSUB[1]	Floating-Point Subtract
FSSUB, FDSUB[1]	Floating-Point Subtract (Single/Double Precision)
FTAN[1,2]	Tangent
FTANH[1,2]	Hyperbolic Tangent
FTENTOX[1,2]	Floating-Point 10^x
FTRAPcc[1,2]	Floating-Point Trap On Condition
FTST[1]	Floating-Point Test
FTWOTOX[1,2]	Floating-Point 2^x
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE from CCR	Move from Condition Code Register
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
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MOVE16	16-Byte Block Move
MULS	Signed Multiply
MULU	Unsigned Multiply

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
PACK	Pack BCD
PEA	Push Effective Address
PFLUSH	Flush Entry(ies) in the ATCs
PFLUSHA	Flush all Entry(ies) in the ATCs
PTEST	Test a Logical Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
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
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

NOTES:
1. Not applicable to the MC68EC040 and MC68LC040.
2. These instructions are software supported.

Addressing Modes	Syntax
Register Direct Data Register Direct Address Register Direct	 Dn An
Register Indirect Address Register Indirect Address Register Indirect with Postincrement Address Register Indirect with Predecrement Address Register Indirect with Displacement	 (An) (An) + –(An) (d_{16},An)
Register Indirect with Index Address Register Indirect with Index (8-Bit Displacement) Address Register Indirect with Index (Base Displacement)	 (d_8,An,Xn) (bd,An,Xn)
Memory Indirect Memory Indirect Postindexed Memory Indirect Preindexed	 ([bd,An],Xn,od) ([bd,An,Xn],od)
Program Counter Indirect with Displacement	(d_{16},PC)
Program Counter Indirect with Index PC Indirect with Index (8-Bit Displacement) PC Indirect with Index (Base Displacement)	 (d_8,PC,Xn) (bd,PC,Xn)
Program Counter Memory Indirect PC Memory Indirect Postindexed PC Memory Indirect Preindexed	 ([bd,PC],Xn,od) ([bd,PC,Xn],od)
Absolute Absolute Short Absolute Long	 xxx.W xxx.L
Immediate	# < data >

Table A-12 lists the instructions used with the MC68881/MC68882 coprocessors.

Table A-12. MC68881/MC68882 Instruction Set

Mnemonic	Description
FABS	Floating-Point Absolute Value
FACOS	Floating-Point Arc Cosine
FADD	Floating-Point Add
FASIN	Floating-Point Arc Sine
FATAN	Floating-Point Arc Tangent
FATANH	Floating-Point Hyperbolic Arc Tangent
FBcc	Floating-Point Branch
FCMP	Floating-Point Compare
FCOS	Floating-Point Cosine
FCOSH	Floating-Point Hyperbolic Cosine
FDBcc	Floating-Point Decrement and Branch
FDIV	Floating-Point Divide
FETOX	Floating-Point ex
FETOXM1	Floating-Point ex - 1
FGETEXP	Floating-Point Get Exponent
FGETMAN	Floating-Point Get Mantissa
FINT	Floating-Point Integer Part
FINTRZ	Floating-Point Integer Part, Round-to-Zero
FLOG10	Floating-Point Log10
FLOG2	Floating-Point Log2
FLOGN	Floating-Point Loge
FLOGNP1	Floating-Point Loge (x + 1)
FMOD	Floating-Point Modulo Remainder
FMOVE	Move Floating-Point Register
FMOVECR	Move Constant ROM
FMOVEM	Move Multiple Floating-Point Registers
FMUL	Floating-Point Multiply
FNEG	Floating-Point Negate
FNOP	Floating-Point No Operation
FREM	IEEE Remainder
FRESTORE	Restore Floating-Point Internal State
FSAVE	Save Floating-Point Internal State
FSCALE	Floating-Point Scale Exponent
FScc	Floating-Point Set According to Condition
FSGLDIV	Single-Precision Divide
FSGLMUL	Single-Precision Multiply
FSIN	Sine
FSINCOS	Simultaneous Sine and Cosine
FSINH	Hyperbolic Sine

FTANH	Hyperbolic Tangent
FTENTOX	Floating-Point 10x
FTRAPcc	Floating-Point Trap On Condition
FTST	Floating-Point Test
FTWOTOX	Floating-Point 2x

A.5.2 MC68881/MC68882 Addressing Modes

The MC68881/MC68882 does not perform address calculations. When the floa coprocessor instructs the processor to transfer an operand via the coprocessor the processor performs the addressing mode calculation requested in the instruct

A.6 MC68851 COPROCESSORS

The following paragraphs provide information on the MC68851 instruction set and ing modes.

A.6.1 MC68851 Instruction Set

Table A-13 lists the instructions used with the MC68851 coprocessor.

Table A-13. MC68851 Instruction Set

Mnemonic	Description
PBcc	Branch on PMMU Condition
PDBcc	Test, Decrement, and Branch on PMMU Condition
PFLUSH	Flush Entry(ies) in the ATCs
PFLUSHA	Flush Entry(ies) in the ATCs
PFLUSHR	Flush Entry(ies) in the ATCs and RPT Entries
PFLUSHS	Flush Entry(ies) in the ATCs
PLOAD	Load an Entry into the ATC
PMOVE	Move PMMU Register
PRESTORE	PMMU Restore Function
PSAVE	PMMU Save Function
PScc	Set on PMMU Condition
PTEST	Test a Logical Address
PTRAPcc	Trap on PMMU Condition
PVALID	Validate a Pointer

A.6.2 MC68851 Addressing Modes

The MC68851 supports the same addressing modes as the MC68020 (see Table

APPENDIX B
EXCEPTION PROCESSING REFERENCE

This appendix provides a quick reference for system programmers who are alread
with the stack frames. For more detail, please refer to the appropriate userOs ma

B.1 EXCEPTION VECTOR ASSIGNMENTS FOR THE M68000 FAM

Table B-1 lists all vector assignments up to and including the MC68040 and its de
Many of these vector assignments are processor specific. For instance, vecto
coprocessor protocol violation vector, only applies to the MC68020, MC6
MC68030, and MC68EC030. Refer to the appropriate user's manual to determi
exception type is applicable to a specific processor.

2	008	Access Fault
3	00C	Address Error
4	010	Illegal Instruction
5	014	Integer Divide by Zero
6	018	CHK, CHK2 Instruction
7	01C	FTRAPcc, TRAPcc, TRAPV Instructions
8	020	Privilege Violation
9	024	Trace
10	028	Line 1010 Emulator (Unimplemented A- Line Opcode)
11	02C	Line 1111 Emulator (Unimplemented F-Line Opcode)
12	030	(Unassigned, Reserved)
13	034	Coprocessor Protocol Violation
14	038	Format Error
15	03C	Uninitialized Interrupt
16–23	040–05C	(Unassigned, Reserved)
24	060	Spurious Interrupt
25	064	Level 1 Interrupt Autovector
26	068	Level 2 Interrupt Autovector
27	06C	Level 3 Interrupt Autovector
28	070	Level 4 Interrupt Autovector
29	074	Level 5 Interrupt Autovector
30	078	Level 6 Interrupt Autovector
31	07C	Level 7 Interrupt Autovector
32–47	080–0BC	TRAP #0 D 15 Instruction Vectors
48	0C0	FP Branch or Set on Unordered Condition
49	0C4	FP Inexact Result
50	0C8	FP Divide by Zero
51	0CC	FP Underflow
52	0D0	FP Operand Error
53	0D4	FP Overflow
54	0D8	FP Signaling NAN
55	0DC	FP Unimplemented Data Type (Defined for MC68040)
56	0E0	MMU Configuration Error
57	0E4	MMU Illegal Operation Error
58	0E8	MMU Access Level Violation Error
59–63	0ECD0FC	(Unassigned, Reserved)
64–255	100D3FC	User Defined Vectors (192)

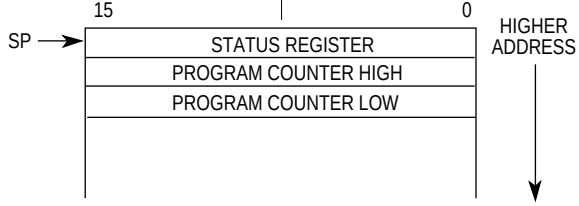


Figure B-1. MC68000 Group 1 and 2 Exception Stack Frame

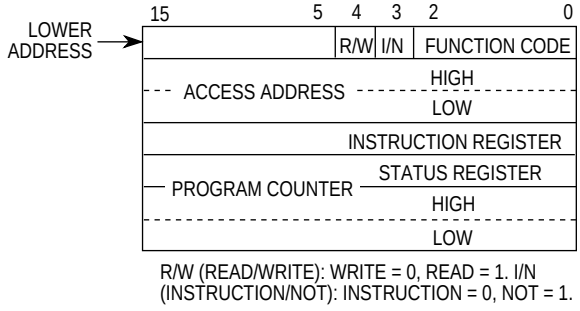


Figure B-2. MC68000 Bus or Address Error Exception Stack Frame

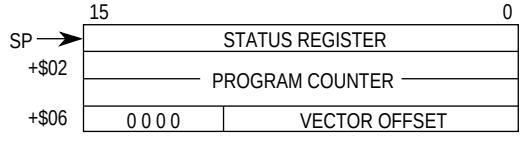


Figure B-3. Four-Word Stack Frame, Format $0

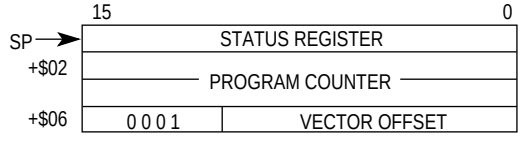


Figure B-4. Throwaway Four-Word Stack Frame, Format $1

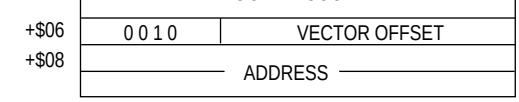


Figure B-5. Six-Word Stack Frame, Format $2

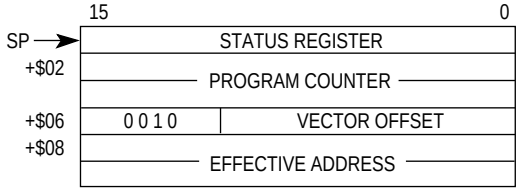


Figure B-6. MC68040 Floating-Point Post-Instruction Stack Frame, Forma

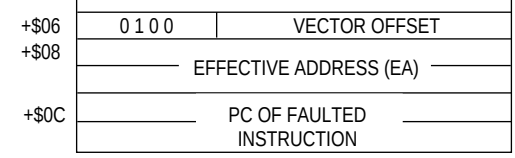


Figure B-7. MC68EC040 and MC68LC040 Floating-Point Unimplemented Frame, Format $4

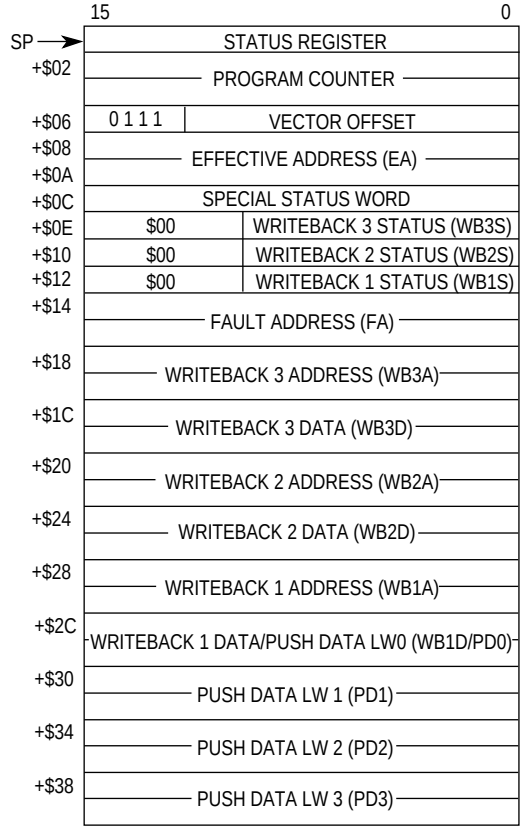
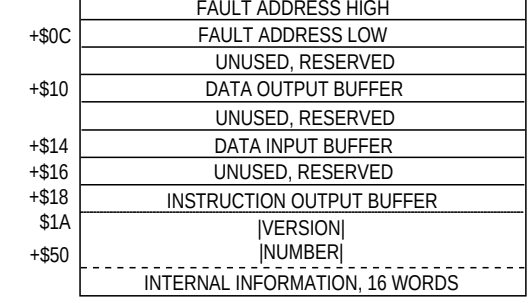


Figure B-8. MC68040 Access Error Stack Frame, Format $7



NOTE: The stack pointer decrements by 29 words, although only 26 words of information actually write to memory. Motorola reserves the three additional words for future use.

Figure B-9. MC68010 Bus and Address Error Stack Frame, Format $8

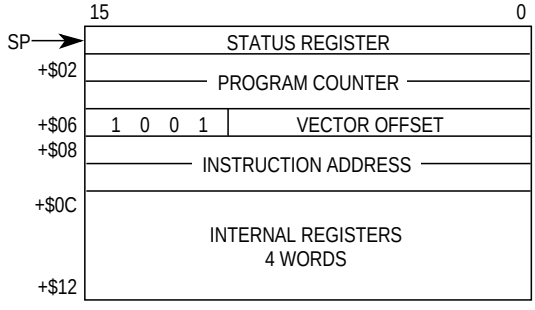


Figure B-10. MC68020 Bus and MC68030 Coprocessor Mid-Instruction Stack Format $9

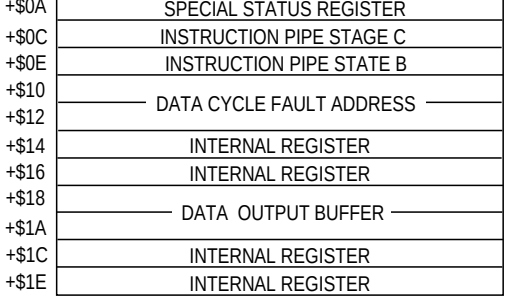


Figure B-11. MC68020 and MC68030 Short Bus Cycle Stack Frame, Form

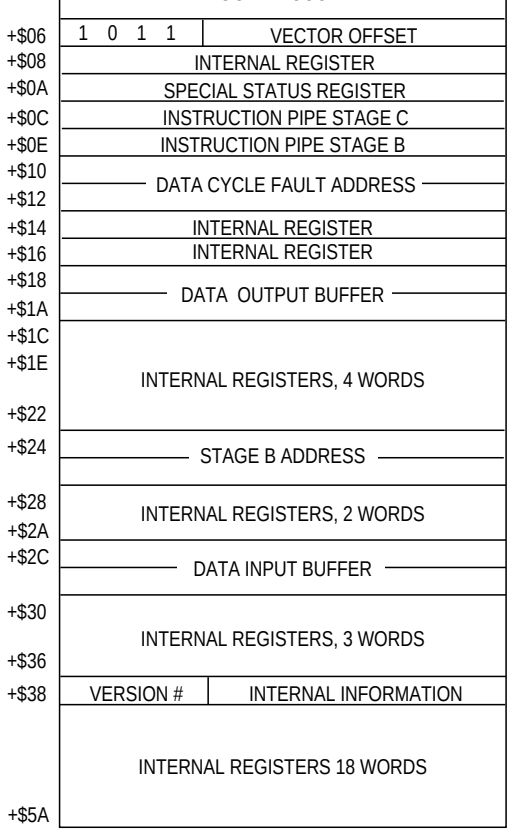


Figure B-12. MC68020 and MC68030 Long Bus Cycle Stack Frame, Forma

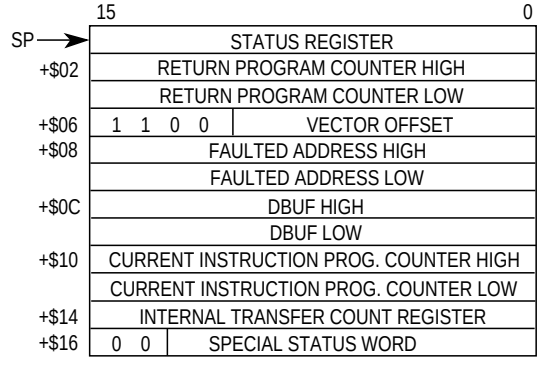


Figure B-13. CPU32 Bus Error for Prefetches and Operands Stack Frame, Fo

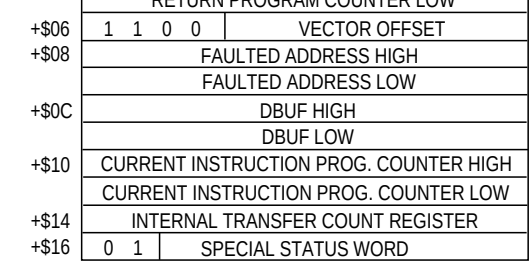


Figure B-14. CPU32 Bus Error on MOVEM Operand Stack Frame, Forma

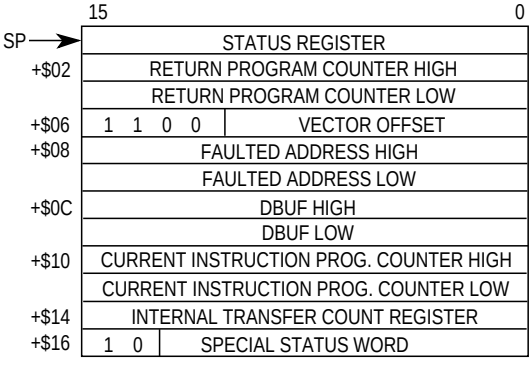


Figure B-15. CPU32 Four- and Six-Word Bus Error Stack Frame, Forma

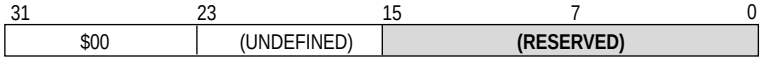


Figure B-16. MC68881/MC68882 and MC68040 Null Stack Frame

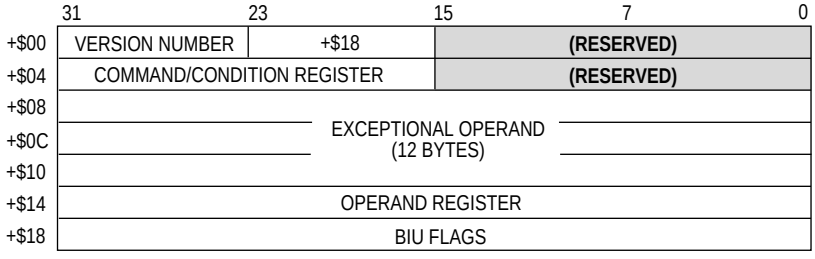


Figure B-17. MC68881 Idle Stack Frame

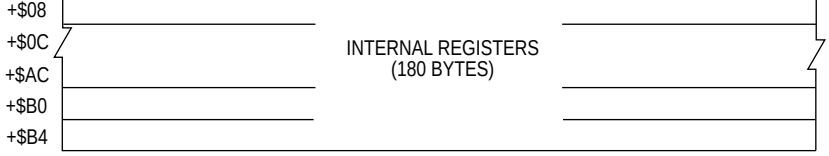


Figure B-18. MC68881 Busy Stack Frame

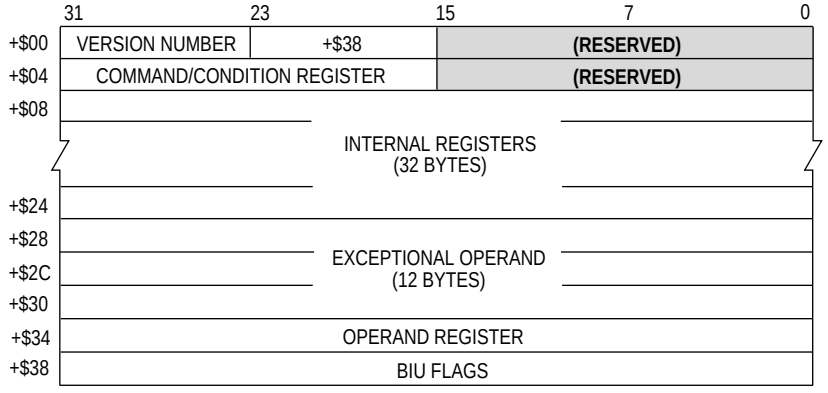


Figure B-19. MC68882 Idle Stack Frame

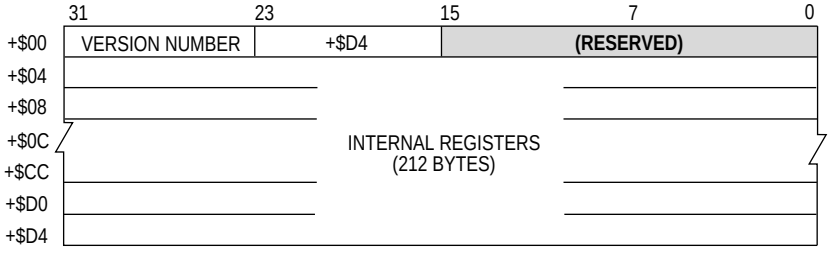


Figure B-20. MC68882 Busy Stack Frame

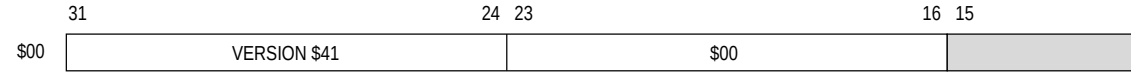


Figure B-21. MC68040 Idle Stack Fram

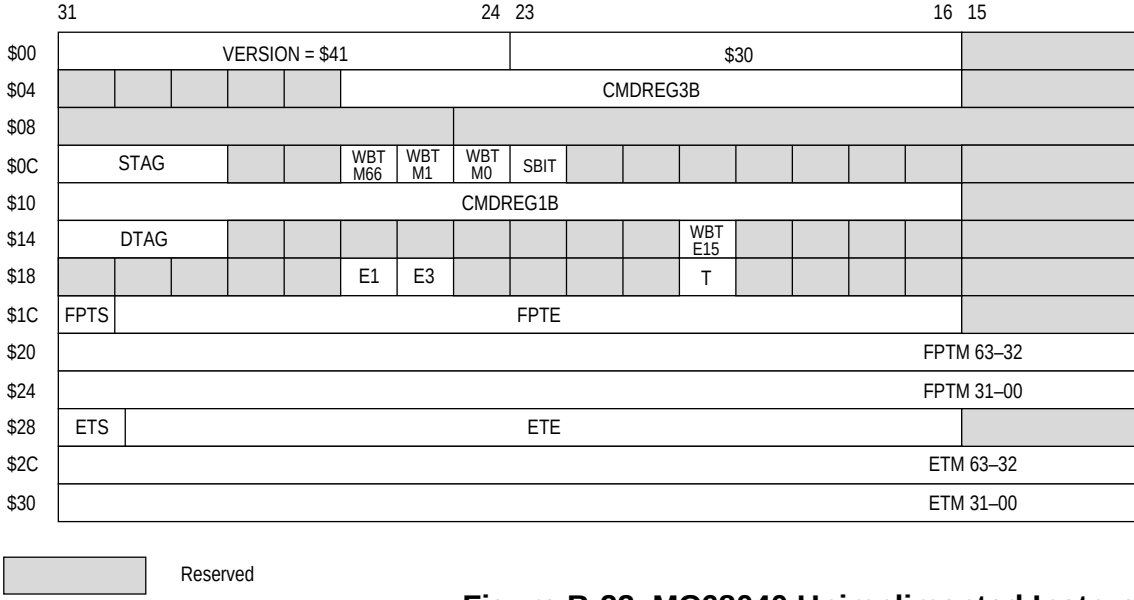


Figure B-22. MC68040 Unimplimented Instruc

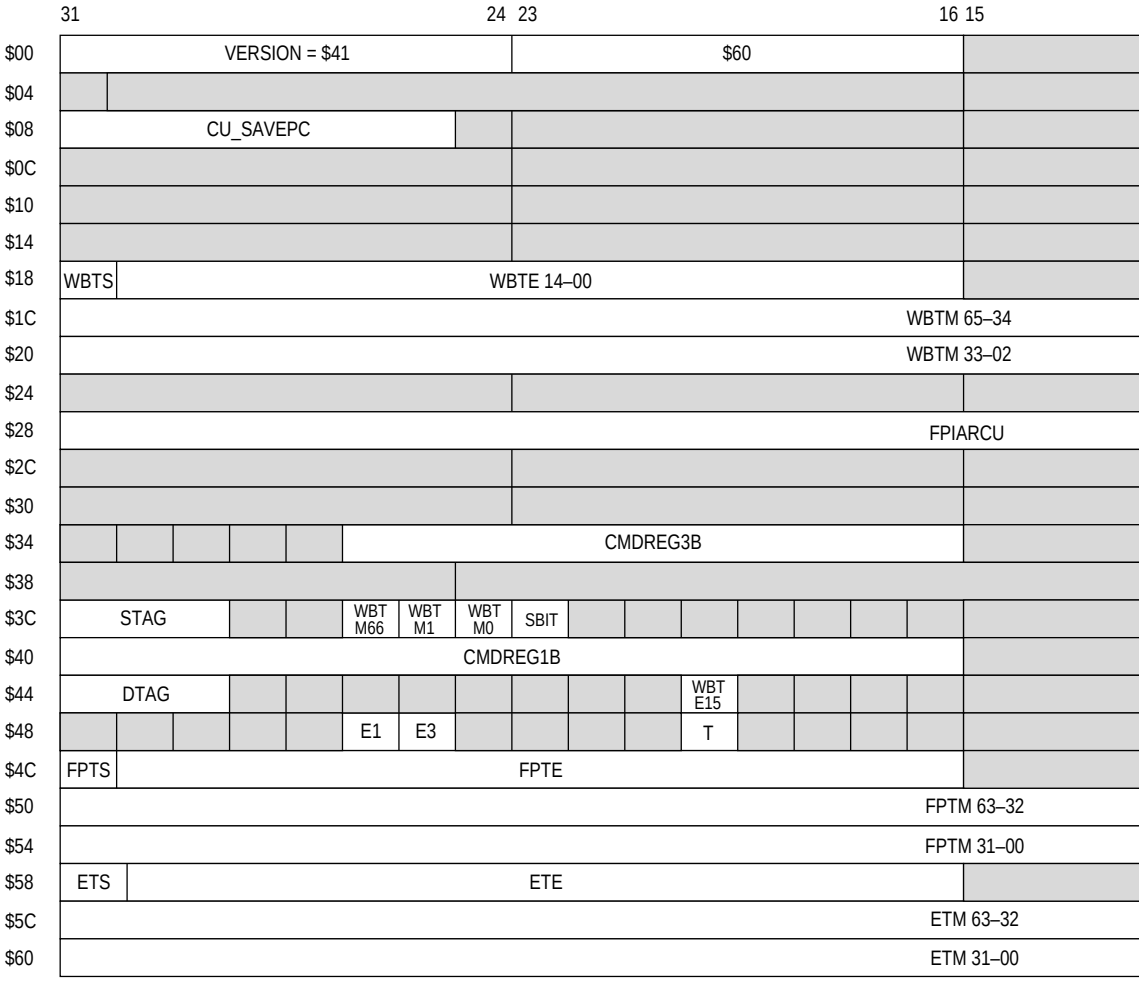


Figure B-23. MC68040 Busy Stack

APPENDIX C
S-RECORD OUTPUT FORMAT

The S-record format for output modules is for encoding programs or data files in a format for transportation between computer systems. The transportation proces visually monitored, and the S-records can be easily edited.

C.1 S-RECORD CONTENT

Visually, S-records are essentially character strings made of several fields that id record type, record length, memory address, code/data, and checksum. Each byte data encodes as a two- character hexadecimal number: the first character repre high- order four bits, and the second character represents the low-order four bits of Figure C-1 illustrates the five fields that comprise an S-record. Table C-1 lists the tion of each S- record field.

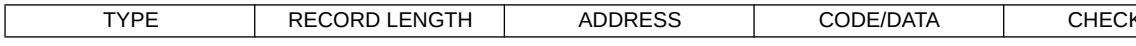


Figure C-1. Five Fields of an S-Record

Table C-1. Field Composition of an S-Record

Field	Printable Characters	Contents
Type	2	S-record type—S0, S1, etc.
Record Length	2	The count of the character pairs in the record, excluding the ty and record length.
Address	4, 6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be load into memory.
Code/Data	0–2n	From 0 to n bytes of executable code, memory loadable data, descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 printable characters in the S-record).
Checksum	2	The least significant byte of the one's complement of the sum the values represented by the pairs of characters making up th record length, address, and the code/data fields.

There are eight types of S-records to accommodate the encoding, transportati
decoding functions. The various Motorola record transportation control progra
upload, download, etc.), cross assemblers, linkers, and other file creating or debugg
grams, only utilize S-records serving the programOs purpose. For more information
port of specific S-records, refer to the userOs manual for that program.

An S-record format module may contain S-records of the following types:

S0 — The header record for each block of S-records. The code/data field m
tain any descriptive information identifying the following block of S-
Under VERSAdos, the resident linkerOs IDENT command can be used
ignate module name, version number, revision number, and descripti
mation that will make up the header record. The address field is normall

S1 — A record containing code/data and the 2-byte address at which the co
is to reside.

S2 — A record containing code/data and the 3-byte address at which the co
is to reside.

S3 — A record containing code/data and the 4-byte address at which the co
is to reside.

S5 — A record containing the number of S1, S2, and S3 records transmitted
ticular block. This count appears in the address field. There is no co
field.

S7 — A termination record for a block of S3 records. The address field may o
contain the 4-byte address of the instruction to which control is to be
There is no code/data field.

S8 — A termination record for a block of S2 records. The address field may o
contain the 3-byte address of the instruction to which control is to be
There is no code/data field.

S9 — A termination record for a block of S1 records. The address field may o
contain the 2-byte address of the instruction to which control is to be
Under VERSAdos, the resident linkerOs ENTRY command can be
specify this address. If this address is not specified, the first entry poi
fication encountered in the object module input will be used. There is
data field.

Each block of S-records uses only one termination record. S7 and S8 records are on
when control is to be passed to a 3- or 4- byte address; otherwise, an S9 is used f
nation. Normally, there is only one header record, although it is possible for multiple
records to occur.

Programs are available for downloading or uploading a file in S- record format fro
system to an 8- or 16-bit microprocessor- based system. A typical S-record-forma
is printed or displayed as follows:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The module has an S0 record, four S1 records, and an S9 record. The following
pairs comprise the S-record-format module.

S0 Record:

S0 — S-record type S0, indicating that it is a header record.
06 — Hexadecimal 06 (decimal 6), indicating that six character pairs (or AS
follow.
0000—A 4-character, 2-byte address field; zeros in this example.
48 — ASCII H
44 — ASCII D
52 — ASCII R
1B — The checksum.

First S1 Record:

S1 — S-record type S1, indicating that it is a code/data record to be loaded/
a 2-byte address.
13 — Hexadecimal 13 (decimal 19), indicating that 19 character pairs, rep
19 bytes of binary data, follow.
0000—A 4-character, 2-byte address field (hexadecimal address 0000)
where the data that follows is to be loaded.

285F	MOVE.L	(A7) +, A4
245F	MOVE.L	(A7) +, A2
2212	MOVE.L	(A2), D1
226A0004	MOVE.L	4(A2), A1
24290008	MOVE.L	FUNCTION(A1), D2
237C	MOVE.L	#FORCEFUNC, FUNCTION(A1)

The rest of this code continues in the remaining S1 recordOs code/data fields and s
memory location 0010, etc.

2A — The checksum of the first S1 record.

The second and third S1 records also contain hexadecimal 13 (decimal 19) charac
and end with checksums 13 and 52, respectively. The fourth S1 record contains 07
ter pairs and has a checksum of 92.

S9 Record:

S9 — S-record type S9, indicating that it is a termination record.
03 — Hexadecimal 03, indicating that three character pairs (3 bytes) follow.
0000—The address field, zeros.
FC — The checksum of the S9 record.

Each printable character in an S-record encodes in hexadecimal (ASCII in this e
representation of the binary bits that transmit. Figure C-2 illustrates the sending of
S1 record. Table C-2 lists the ASCII code for S-records.

.

TYPE				RECORD LENGTH				ADDRESS								CODE/DATA																	
S		1		1		3		0		0		0		0		2		8		5		F		****									
5	3	3	1	3	1	3	3	3	0	3	0	3	0	3	0	3	2	3	8	3	5	4	6	****									
0101	0011	0011	0001	0011	0001	0011	0011	0011	0000	0011	0000	0011	0000	0011	0000	0011	0010	0011	1000	0011	0101	0100	0110	****	0								

Figure C-2. Transmission of an S1 Record

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

MOTOROLA

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217. 303–675–2140 or 1–800–441–2447

Customer Focus Center: 1–800–521–6274

Mfax™: RMFAX0@email.sps.mot.com – TOUCHTONE 602–244–6609
Motorola Fax Back System – US & Canada ONLY 1–800–774–1848
– http://sps.motorola.com/mfax/

HOME PAGE: http://motorola.com/sps/

JAPAN: Nippon Motorola Ltd.; SPD, Strategic Planning Office, 141, 4–32–1, Nishi–Gotanda, Shagawa–ku, Tokyo, Japan. 03–5487–8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852–26629298

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for CPU - Central Processing Units ***category:***

Click to view products by Freescale ***manufacturer:***

Other Similar products are found below :

AT80612003090AAS LBWJ B4860NSE7QUMD NPIXP2855AB-S-LA88 IVPX7225-RTM-1 96MPI5-2.9-6M10T 2SC4646D-AN SMBV1061LT1G 2SC4646E-AN 2SJ268-DL-E CP80617003981AHS LBTQ MATXM-CORE-411-HTSNK TEC0193BPF TND516SS-TL-E BGSF 1717MN26 E6327 D8086-2 LF80538GF0282M-S-L8VY CM8063401293902S R1A4 CM8063501521302S R1B8 CM8066201919901 SR2L0 CM8066201928505 SR2HT CPH5855-TL-E CM8063501293200S R1A0 AV8063801129600S R10F EMS36-02-2H-MDT EMM04-MDT NG80386DX33 NHIXP432AC NK80530MZ866256S-L7XH P1021NXE2HFB R0K5ML001SS00BR LC87F2608A LC87FBK08A PRIXP425BC PRIXP423BB CM8066201921712S R2LF CM8064601467102S R152 CM8063501375800S R1AX CM8063501293506S R1A2 CM8063401293802S R1A3 CM8062107185405S R0KM LC87F0G08A CM8067702867061S R374 PB-8SMB COMX-300-HSP RTM-ATCA-7360 96MPI7-3.4-8M11T CM8066002023801S R2J1 96MPP-2.3-3M10T 96MPI7-3.4-8M11T1 96MPXE-2.0-15M20T