Tektronix®
COMMITTED TO EXCELLENCE

This manual supports the following TEKTRONIX products:

8550 Options	Products	8560 Options	Products
1V	8300B26	1L	8560B17

This manual supports a software module that is compatible with the following software modules:

TEKTRONIX B Series Assembler Version 1 (8550)
TEKTRONIX B Series Linker Version 1 (8550)
TEKTRONIX B Series LibGen Version 1 (8550)
TEKTRONIX B Series Assembler Version 1 (8560)
TEKTRONIX B Series Linker Version 1 (8560)
TEKTRONIX B Series LibGen Version 1 (8560)

These modules are compatible with:

DOS/50 Version 2 (8550)
TNIX Version 1 (8560)

PLEASE CHECK FOR CHANGE INFORMATION AT THE REAR OF THIS MANUAL.

8500
MODULAR MDL SERIES
68000
ASSEMBLER SPECIFICS
USERS MANUAL
for B Series Assembler

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

070-3855-00
Product Group 61

Serial Number ____________________

First Printing FEB 1982

LIMITED RIGHTS LEGEND

Software License No.________________________

Contractor: Tektronix, Inc.
Explanation of Limited Rights Data Identification Method
Used: Entire document subject to limited rights.

Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Printed in U.S.A. Specification and price change privileges are reserved.

Section 9C

8500 SERIES B 68000 ASSEMBLER SPECIFICS

Page

TABLES

ILLUSTRATIONS

68000 INSTRUCTION SUMMARY

Page

Section 9C

8500 SERIES B 68000 ASSEMBLER SPECIFICS

INTRODUCTION

This section is designed to be inserted into Section 9 of the 8500 Series B Assembler Users Manual and describes the TEKTRONIX 8500 Series B 68000 Assembler. In this section you will find summaries of the registers, control and status bits, addressing modes, and the supported instruction set, along with the statement syntax accepted by the assembler. (Differences between the Tektronix assembler and the Motorola assembler are discussed under the heading "Irregularities", later in this section.) For more information about the processor and its specifications, refer to the manufacturer's literature.

68000 ARCHITECTURE

The 68000 is a 32-bit microprocessor that has seventeen 32-bit registers, a 32-bit program counter, and a 16-bit status register, as shown in Fig. 9C-1.

Registers

The eight **data registers** (D0, D1, D2, D3, D4, D5, D6, D7) are used for bit, byte (8-bit), word (16-bit), and long word (32-bit) data operations.

The seven **address registers** (A0, A1, A2, A3, A4, A5, A6) are used for word or long word address operations. Any one of the address registers may be used as a stack pointer with the register indirect postincrement/predecrement addressing mode. (See the discussion of Addressing Modes later in this section.)

The **system stack pointer** is address register A7. The system stack fills from high to low memory. The active stack pointer is either the User Stack Pointer (USP) or the Supervisor Stack Pointer (SSP), depending on the S bit in the status register. S bit low selects USP; S bit high selects SSP. Certain instructions (known as privileged instructions) are available only in the supervisor state.

Any one of the registers (data or address) may be used as an index register. (See the discussion of Addressing Modes later in this section: address register indirect with index and relative with index.)

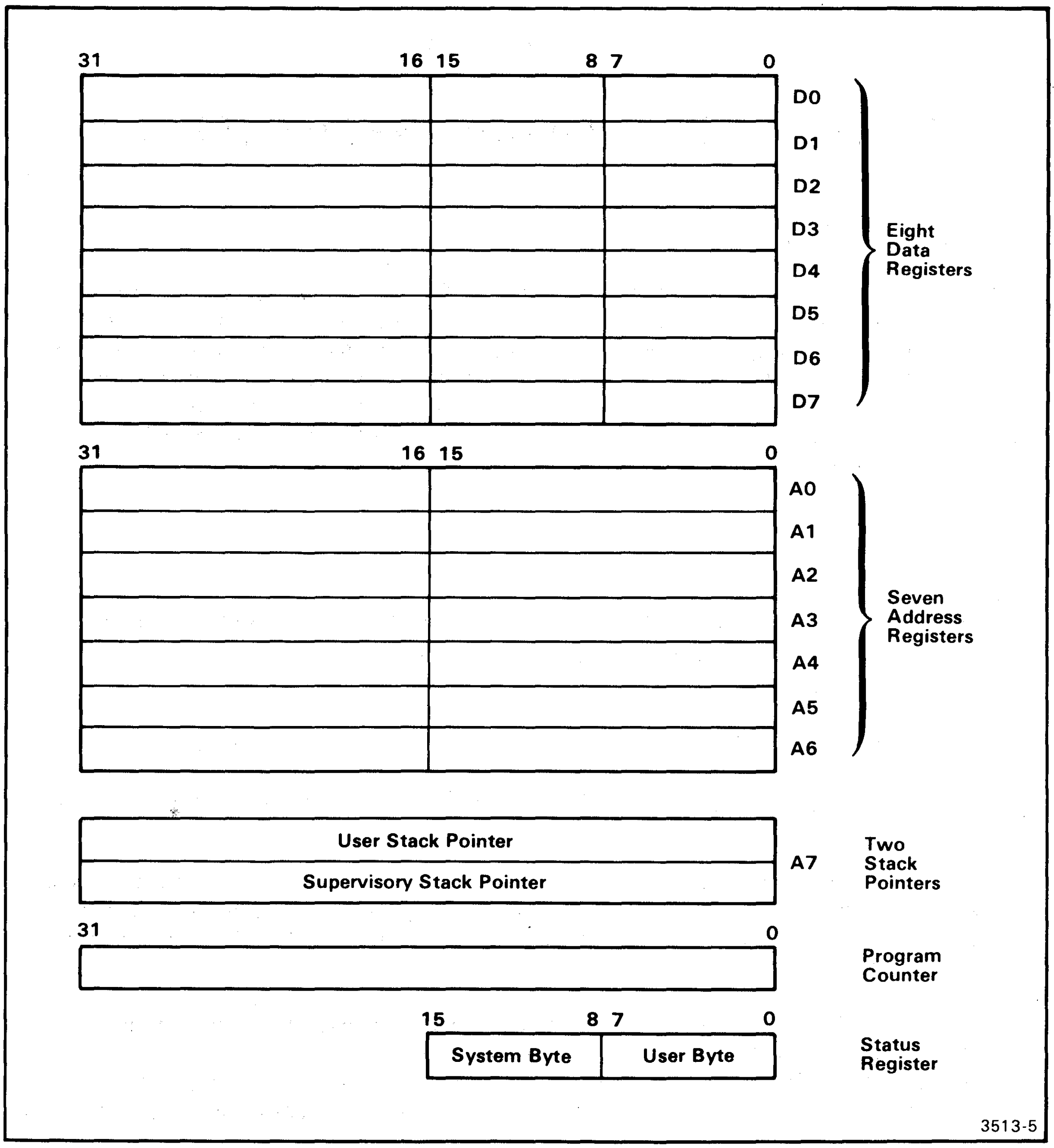


Fig. 9C-1. 68000 registers.

Status Register

The 16-bit status register consists of a system byte (most significant byte) and a user byte (least significant byte). See Fig. 9C-2. The system byte contains a trace mode indicator, a supervisor state indicator, and a 3-bit interrupt mask. A selected set of "privileged" instructions may reference the system byte.

The **user byte** contains five condition flag bits: extend (X), negative (N), zero (Z), overflow (V), and carry (C). These flag bits reflect certain processor states resulting from arithmetic and logical operations.

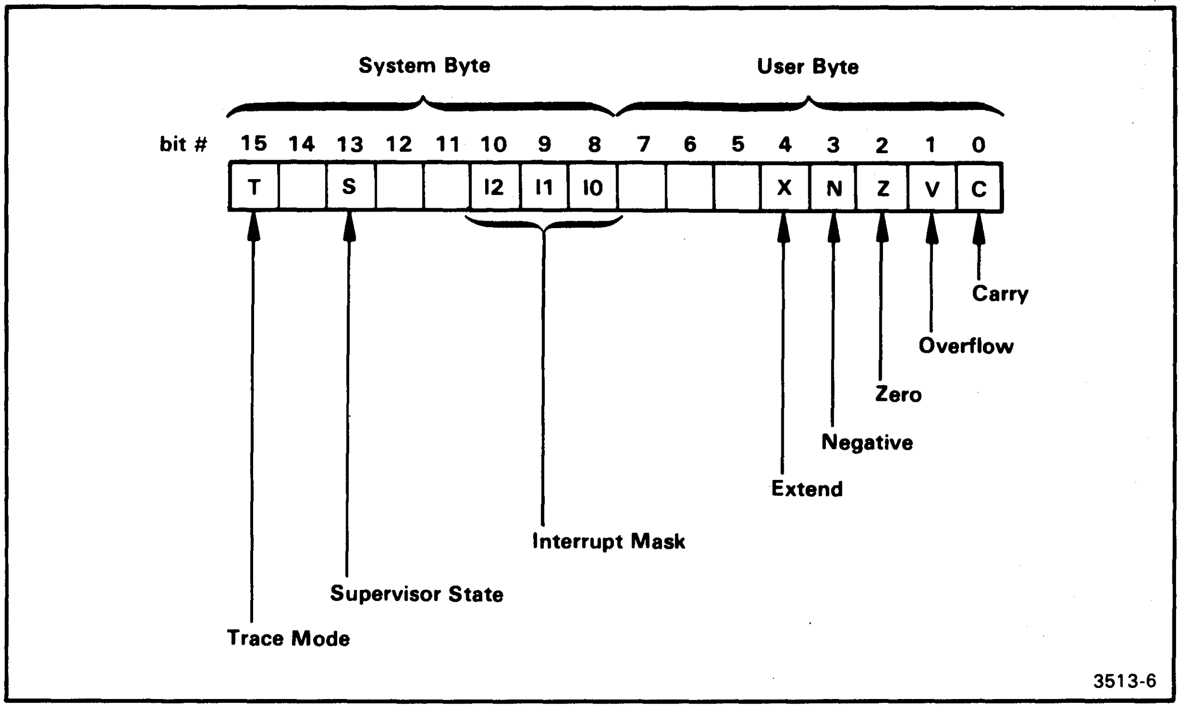


Fig. 9C-2. 68000 status register.

ADDRESSING

The 68000 assembler supports byte (8-bit), word (16-bit), and long word (32-bit) addressing. A byte is the minimal addressable element in memory. A **word** in memory spans two bytes. The most significant byte must have an even address. The least significant byte has the immediately following odd address. Words are addressed by the most significant byte. Instructions always begin on an even address (a word boundary).

A **long word** in memory spans two words (four bytes). The most significant 16 bits have a word address. The least significant 16 bits are located in the immediately following word.

Many 68000 instructions are used for byte, word, or long word operations. A suffix added to the instruction indicates the size.

Suffix	Operation Size
.B	byte
.W	word
.L	long word
none	word (default)

For example, the MOVE.B instruction moves a byte of data; the MOVE.W or MOVE instruction moves a word of data; and the MOVE.L instruction moves a long word of data.

For each instruction, the TEKTRONIX 68000 Assembler accepts only the allowed suffixes for that instruction, as listed in the Summary of Instructions (in this section).

The **data registers** support data operands of 1, 8, 16, or 32 bits. When a data register is used as an operand for an instruction, only the appropriate low-order portion of the register is used. The high-order portion is neither used nor changed.

The **address registers** do not support byte-sized operands. When an address register is used as a source operand, either the low-order word or the entire long word is used, depending on the operation size. When an address register is used as a destination operand, the entire 32 bits are used for the address.

Addressing Modes

Most instructions specify the location of an operand by using the **effective address** field. For example, a single effective address instruction has the format shown in Fig. 9C-3.

The mode field selects the addressing mode; the register field contains the number of the register. When required, an effective address extension is contained in the following word or words and is considered part of the instruction. The effective addressing modes are grouped in three categories: register direct, memory address, and special address.

Register Direct Mode

In register direct effective addressing modes, the operand is one of the 16 data or address registers.

Data Register Direct. The operand is the contents of the specified data register: D0, D1, D2, D3, D4, D5, D6, or D7.

Address Register Direct. The operand is the contents of the specified address register: A0, A1, A2, A3, A4, A5, A6, or A7.

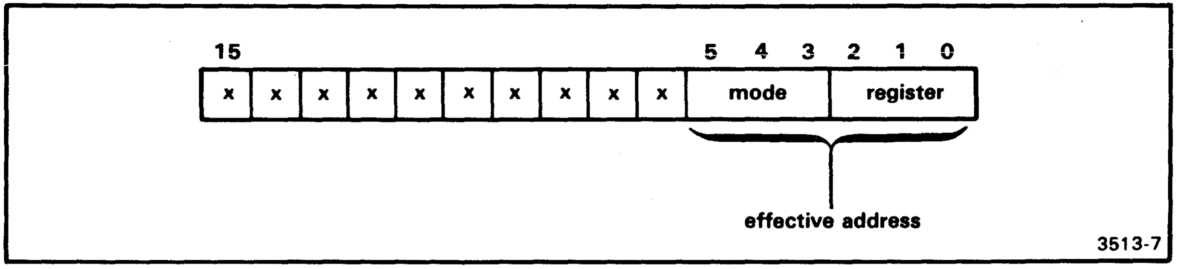


Fig. 9C-3. Single effective address instruction format.

Memory Address Mode

In memory address effective addressing modes, the operand is in a specified memory address.

Address Register Indirect. The address of the operand is in the specified address register. The address register is enclosed in parentheses to indicate this mode. For example, (A2) indicates that the operand is in the memory location addressed by the A2 register.

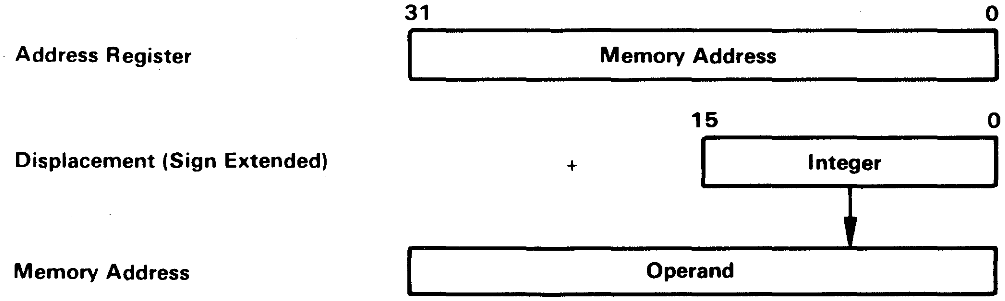
Address Register Indirect with Postincrement. The address of the operand is in the specified address register. After the operand address is used, the address register is incremented by 1, 2, or 4, depending on the operand size. If the address register is SP (the stack pointer), and the operand size is byte, the address is incremented by two to keep the stack pointer on a word boundary. The notation "(An)+" is used to specify this mode.

Address Register Indirect with Predecrement. The address of the operand is in the specified address register. Before the operand address is used, the address register is decremented by 1, 2, or 4, depending on the operand size. If the address register is SP (the stack pointer), and the operand size is byte, the address is decremented by two to keep the stack pointer on a word boundary. The notation "−(An)" is used to specify this mode.

Address Register Indirect with Displacement. The address of the operand is the sum of the address in the specified address register and the sign-extended 16-bit displacement integer. The notation "d(An)" is used to specify this mode. For example:

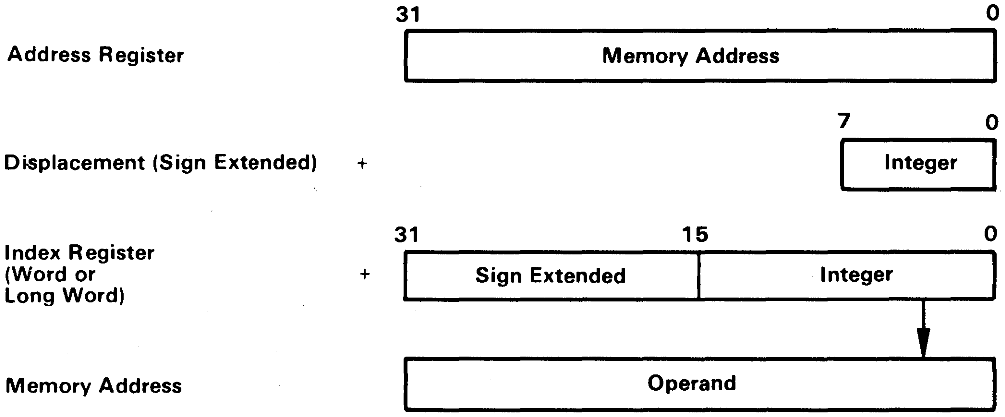
```
MOVE 4(A1),D2
```

The contents of the address register A1, plus 4, specifies the address of the data operand to be moved to the register D2.



3596-5

Address Register Indirect with Index. The address of the operand is the sum of the address in the specified address register, the sign-extended 8-bit displacement integer, and the contents of the index register. The notation "d(An,Rn)" is used to specify this mode, where Rn is any of the 17 registers: D0−D7, A0−A6, or A7 (the system stack pointer, USP or SSP). The suffix .W or .L may be added to the index register to specify word or long word. When no suffix is used, the low-order word of the index register is used.



3596-6

Special Address Modes

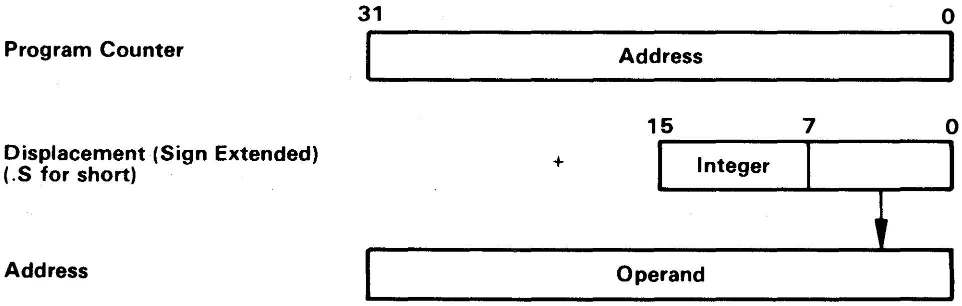
The Special Address Modes include Absolute Short Address, Relative, Relative with Index, Immediate, and Status Register addressing.

Absolute Short Address. The address of the operand is the address contained in the extension **word** of the instruction. The 16-bit address is sign extended to 32 bits.

Absolute Long Address. The address of the operand is the address contained in the two extension words following the instruction.

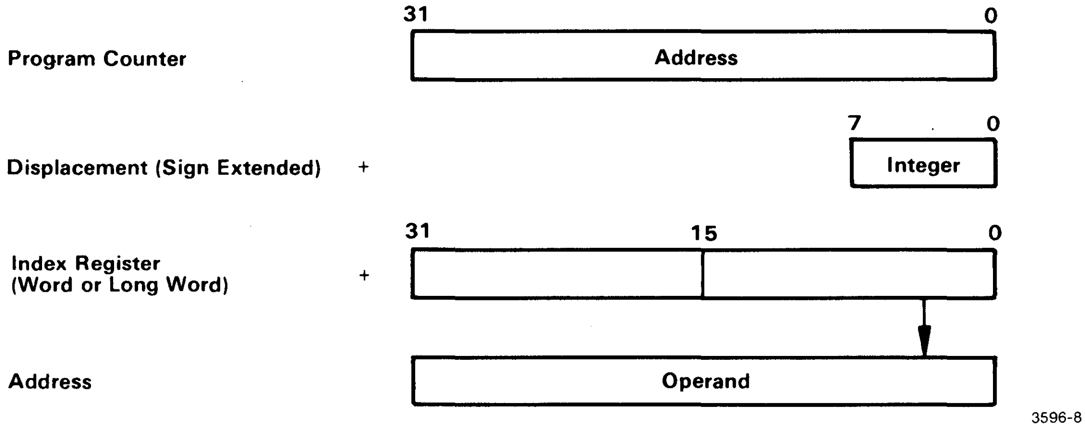
Relative. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer. This mode of addressing is used by the assembler when a label representing an address in the current section is used as an operand, or when the $ symbol is used to represent the current value of the program counter.

An 8-bit displacement integer can be used in the branch instructions by using a suffix .S to represent a short relative jump. For example: if the relative jump is within −126 or +129 bytes (−63 to +64 words) and not zero, the branch instruction BRA.S is used. The assembler automatically uses the short form for any backward branch within range or when the $ symbol is used in a forward branch within range.



3596-7

Relative With Index. The address of the operand is the sum of the address in the program counter, the sign-extended 8-bit displacement integer, and the contents of the index register. Any of the 17 registers, D0–D7, A0–A6, or the system stack pointer A7 (USP or SSP) may be used as the index register. The suffix .W or .L may be added to the index register to specify word or long word. When no suffix is used, the low-order word of the register is used.



Immediate Data. The operand is the byte, word, or long word contained in the extension of the operation. The notation "#" precedes the expression to specify immediate addressing.

Condition Codes or Status Register. Some instructions may reference the status register. CCR (Condition Code Register) is used as an operand to specify the least significant byte; or SR (Status Register) is used as an operand to specify the complete 16-bit register.

Specifying Addressing Modes

Referring to Table 9C-1, note that the syntax for absolute short, absolute long, and relative are the same. This is also true of address register indirect with displacement and relative with index.

Three special assembler directives (GEN.R, GEN.S, GEN.L) are provided to enable the user to control which form the assembler will use to encode the address. GEN.S is the default form. Table 9C-2 lists these directives and their uses.

The GEN (GEN.R, GEN.S, GEN.L) directives may be used globally or locally.

1. For global use the GEN directive is placed on a line by itself in the operation field. Used in this way it controls all subsequent coding of addresses.

Example:

```
A   CLR     DO
    ORG     100
    GEN.S
    MOVE    A, DO          A is referenced using absolute short addressing.
```

2. For local use the address expression in question is preceded by the GEN directive enclosed in parentheses. This overrides any global GEN directive.

Example:

```
A   CLR     DO
    ORG     100
    GEN.S
    MOVE    (GEN.R) A, DO   A is referenced using relative with displacement.
```

Table 9C-1
Special Assembler Directives

Directive	Address Form Used	
	Syntax: d	Syntax: d(Rn)
GEN.R	Relative with displacement	Relative with index
GEN.S (default)	Absolute short	Address register indirect with displacement
GEN.L	Absolute long unless d is a backward reference 0—32767, when absolute short is used.	Address register indirect with displacement

Address, Quick, and Immediate Forms

The assembler will automatically select the address (A), quick (Q) or immediate (I) form of the ADD, AND, CMP, EOR, MOVE, NEG, OR, and SUB instruction if it is valid. If more than one form is valid, the shortest form is chosen.

Summary of Addressing Modes

The 68000 addressing modes may be categorized into classes by the way they are used in the instructions. Table 9C-2 summarizes the addressing modes and the classes in which they may be used. In the Summary of Instructions on the following pages, the class of addressing modes is indicated when various modes are allowed for an instruction.

Table 9C-2
Classes of Effective Addressing Modes

Addressing Mode	Operand	Addressing Classes							
		1	2	3	4	5	6	7	8
REGISTER DIRECT MODE									
Data Register Direct	Dn		x	x	x	x	x		
Address Register Direct	An				x		x		
MEMORY ADDRESS MODE									
Address Register Indirect	(An)	x	x	x	x	x	x	x	x
Address Register Indirect with Postincrement	(An)+		x	x	x	x	x	x	
Address Register Indirect with Predecrement	−(An)		x	x	x	x	x	x	
Address Register Indirect with Displacement	d(An)	x	x	x	x	x	x	x	x
Address Register Indirect with Index	d(An,An)	x	x	x	x	x	x	x	x
	d(An,Dn)	x	x	x	x	x	x	x	x
	d(An,An.W)	x	x	x	x	x	x	x	x
	d(An,Dn.W)	x	x	x	x	x	x	x	x
	d(An,An.L)	x	x	x	x	x	x	x	x
	d(An,Dn.L)	x	x	x	x	x	x	x	x
SPECIAL ADDRESS MODE									
Absolute Short	d	x	x	x	x	x	x	x	x
Absolute long	d	x	x	x	x	x	x	x	x
Relative	d	x	x	x	x				
Relative with Index	d(An)	x	x	x	x				
	d(Dn)	x	x	x	x				
	d(An.W)	x	x	x	x				
	d(Dn.W)	x	x	x	x				
	d(An.L)	x	x	x	x				
	d(Dn.L)	x	x	x	x				
Immediate	#exp			x	x				

These classes may be compared with the Motorola addressing categories:

Classes of Addressing Modes	Motorola Addressing Categories
Class 1	Control (Memory operands without associated size)
Class 2	BTST instruction only
Class 3	Data Operand
Class 4	All addressing modes
Class 5	Data Alterable (writable)
Class 6	Alterable (writable)
Class 7	Memory Alterable (writable)
Class 8	Control Alterable (writable)

NOTATIONAL CONVENTIONS

The following notational conventions are used in this section:

#	Indicates immediate addressing.
( )	The contents of the referenced location.
→	The value of the expression on the left is stored in the location on the right.
(An)+	Indicates register indirect with postincrement addressing mode.
-(An)	Indicates register indirect with predecrement addressing mode.
addr	Memory address.
An	One of the eight address registers: A0, A1, A2, A3, A4, A5, A6, A7. May be used for word (16-bit) or long word (32-bit) address operations.
bound	A value at which the processor initiates exception processing.
CCR	Condition Code Register (low-order byte of the status register).
class1-8	Class of addressing modes allowed. See the Summary of Addressing Modes in this section.
count	A value to be decremented for looping instructions.
dest	An operand that indicates the destination of the result. In some instructions dest is both source and destination.
d	Displacement integer for relative addressing.
Dn	One of eight data registers: D0, D1, D2, D3, D4, D5, D6, D7. May be used for byte (8-bit), word (16-bit), or long word (32-bit) data operations.

immed — An immediate operand; 8-bit, 16-bit, or 32-bit.

immed8 — A 1-byte (8-bit) immediate operand.

immed16 — A 2-byte (16-bit) immediate operand.

immedS — An immediate operand in the range 1–8.

immedvect — A 4-bit immediate operand representing a trap vector number.

k — An expression in the range 0 to 255.

LSB — Least Significant Bit.

MSB — Most Significant Bit.

Rn — Any one of the 17 registers: D0–D7, A0–A7.

r-list — A list of one or more register selections separated by a slash (/), and/or an inclusive range of registers separated by a dash. For example:

```
A3
A1/D1/D3/D4
A0--A6
A2/D3--D6/A5
```

rel-addr — An address displacement from the current location counter.

source — The source operand that the instruction operates on.

SP — Active stack pointer; equivalent to A7 (USP in user state, SSP in supervisor state).

SR — Status Register.

SSP — The Supervisor Stack Pointer.

target — The address of the instruction to execute next.

USP — The User Stack Pointer.

vector — Exception vector—memory location that contains the address of a routine that will handle the exception.

SUMMARY OF INSTRUCTIONS

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
Data Movement Instructions				
EXG	**source,source**	**(Exchange registers)**		**none**
	Dn,Dn	2	EXG D1,D2	
	An,An	2	EXG A1,A1	
	Dn,An	2	EXG D4,A1	
	An,Dn	2	EXG A0,D7	
LEA	**addr,dest**	**(Load effective address)**		**none**
	class1,An	2/4	LEA START,A1	
LINK	**source,d**	**(link and allocate; An→(SP), SP→An, SP + d→SP)**		**none**
	An,immed	4	LINK A0,#2	
MOVE[a]				
MOVE.B				
MOVE.W				
MOVE.L	**source,dest**	**(Move data from source to destination)**		**N,Z,V=0,C=0**
	class4,class6[b]	2/4/6/8	MOVE D2,(A1)	
MOVEA				
MOVEA.W				
MOVEA.L	**source,dest**	**(Move address)**		**none**
	class4,An	2/4/6	MOVE.W ADDR,A1	
MOVEM				
MOVEM.W[c]				
MOVEM.L	**source,dest**	**(Move multiple registers)**		**none**
	r-list,class8	4/6	MOVEM A1/A2/A5,STORE	
	class1,r-list	4/6	MOVEM.L NEW,A1-A4	
	r-list,-(An)	4	MOVEM.W D0-D4,-(A0)	
	(An+,r-list)	4	MOVEM (A0)+,D0-D7	
MOVEP				
MOVEP.W				
MOVEP.L	**source,dest**	**(Move peripheral data)**		**none**
	Dn,d(An)	4	MOVEP D1,PLUS(A0)	
	d(An),Dn	4	MOVEP.L 10H(A3),D2	
MOVEQ	**source,dest**	**(Move quick; sign extended)**		**N,Z,V=0,C=0**
	immed8,Dn	2	MOVEQ #3,D2	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
PEA	**dest**	**(Push effective address onto stack)**		**none**
	class1	2/4	PEA (A6)	
SWAP	**source**	**(Swap data register halves)**		**N,Z,V=0,C=0**
	Dn	2	SWAP D2	
UNLK	**source**	**(Unlink; An→SP, (SP)→An)**		**none**
	An	2	UNLK A5	

Integer Arithmetic Instructions

Mnemonic	Operands	(Description)		Flags Affected
ADD				
ADD.B				
ADD.W				
ADD.L	**source,dest**	**(Add source to dest, store in dest)**		**X,N,Z,V,C**
	class4,Dn[b]	2/4/6	ADD.B DATA,D2	
	Dn,class7	2/4/6	ADD D4,DATA	
ADDA				
ADDA.W				
ADDA.L	**source,dest**	**(Add source to address register)**		**none**
	class4,An	2/4/6	ADDA LOCATE,A4	
ADDI				
ADDI.B				
ADDI.W				
ADDI.L	**source,dest**	**(Add immediate)**		**X,N,Z,V,C**
	immed,class5	4/6/8	ADDI.L #LONGW,DATA1	
ADDQ				
ADDQ.B				
ADDQ.W				
ADDQ.L	**source,dest**	**(Add quick)**		**X,N,Z,V,C**
	immedS,class6[b]	2/4	ADDQ.L #1,A4	
ADDX				
ADDX.B				
ADDX.W				
ADDX.L	**source,dest**	**(Add source to dest with extend bit)**		**X,N,Z,V,C**
	Dn,Dn	2	ADDX.L D0,D4	
	-(An),-(An)	2	ADDX -(A1),-(A2)	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
CLR				
CLR.B				
CLR.W				
CLR.L	dest	(Clear operand)		N=0,Z=1,V=0,C=0
	class5	2/4/6	CLR D4	
CMP[i]				
CMP.B				
CMP.W				
CMP.L	source1,source2	(Compare)		N,Z,V,C
	class4,Dn[b]	2/4/6	CMP.B #0FFH,D2	
CMPA[i]				
CMPA.W				
CMPA.L	source1,source2	(Compare address)		N,Z,V,C
	class4,An	2/4/6	CMPA.L A2,A4	
CMPI[i]				
CMPI.B				
CMPI.W				
CMPI.L	source1,source2	(Compare immediate)		N,Z,V,C
	immed,class5	4/6/8	CMPI.B #0FFH,D4	
CMPM[i]				
CMPM.B				
CMPM.W				
CMPM.L	source1,source2	(Compare memory)		N,Z,V,C
	(An)+,(An)+	2	CMPM (A1)+,(A2)+	
DIVS	source,dest	(Dest/source; signed)		N,Z,V,C=0
	class3,Dn	2/4/6	DIVS (A2),D4	
DIVU	source,dest	(Dest/source; unsigned)		N,Z,V,C=0
	class3,Dn	2/4/6	DIVU D0,D1	
EXT				
EXT.W				
EXT.L	source	(Sign extend)		N,Z,V=0,C=0
	Dn	2	EXT.W D3	
MULS	source,dest	(Signed multiply)		N,Z,V=0,C=0
	class3,Dn	2/4/6	MULS DATA,D3	
MULU	source,dest	(Unsigned multiply)		N,Z,V=0,C=0
	class3,Dn	2/4/6	MULU (A2),D2	
NEG				
NEG.B				
NEG.W				
NEG.L	dest	(Negate—subtract from zero)		X,N,Z,V,C
	class5	2/4/6	NEG D3	

Mnemonic	**Operands**	**(Description)**		**Flags Affected**
	Operand Types	Bytes	Example	
NEGX				
NEGX.B				
NEGX.W				
NEGX.L	**dest**	**(Negate with extend)**		**X,N,Z,V,C**
	class5	2/4/6	NEGX.B (A2)	
SUB				
SUB.B				
SUB.W				
SUB.L	**source,dest**	**(Subtract source from destination)**		**X,N,V,C**
	class4,Dn[b]	2/4/6	SUB DATA,D0	
	Dn,class7	2/4/6	SUB.B D2,DATA1	
SUBA				
SUBA.W				
SUBA.L	**source,dest**	**(Subtract address)**		**none**
	class4,An	2/4/6	SUBA A0,A1	
SUBI				
SUBI.B				
SUBI.W				
SUBI.L	**source,dest**	**(Subtract immediate)**		**X,N,Z,V,C**
	immed,class5	4/6/8	SUBI.W #0FFFFH,D4	
SUBQ				
SUBQ.B				
SUBQ.W				
SUBQ.L	**source,dest**	**(Subtract quick)**		**X,N,Z,V,C**
	immedS,class6[b]	2/4	SUBQ #2,A4	
SUBX				
SUBX.B				
SUBX.W				
SUBX.L	**source,dest**	**(Subtract with extend)**		**X,N,Z,V,C**
	Dn,Dn	2	SUBX.B D0,D1	
	-(An),-(An)	2	SUBX -(A3),-(A0)	
TAS	**source**	**(Compare byte with zero, set condition codes, set bit 7 in source to 1)**		**N,Z,V=0,C=0**
	class5	2/4	TAS TESTBIT	
TST				
TST.B				
TST.W				
TST.L	**source**	**(Compare source with zero, set condition codes)**		**N,Z,V=0,C=0**
	class5	2/4	TST COUNT	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	

Logical Instructions

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
AND **AND.B** **AND.W** **AND.L**	**source,dest**	**(Logical AND source with destination)**		**N,Z,V=0,C=0**
	class3,Dn	2/4/6	AND (A1),D3	
	Dn,class7	2/4	AND.B D2,VALUE	
ANDI[a] **ANDI.B** **ANDI.W** **ANDI.L**	**source,dest**	**(Logical AND immediate)**		**N,Z,V=0,C=0**
	immed,class5	4/6/8	ANDI.B #0FFH,D4	
EOR **EOR.B** **EOR.W** **EOR.L**	**source,dest**	**(Logical exclusive-OR)**		**N,Z,V=0,C=0**
	Dn,class5	2/4	EOR D4,(A2)	
EORI[a] **EORI.B** **EORI.W** **EORI.L**	**source,dest**	**(Logical exclusive-OR immediate)**		**N,Z,V=0,C=0**
	immed,class5	4/6/8	EORI #0FFFFH,SYST	
NOT **NOT.B** **NOT.W** **NOT.L**	**dest**	**(One's complement)**		**N,Z,V=0,C=0**
	class5	2/4	NOT.L D6	
OR **OR.B** **OR.W** **OR.L**	**source,dest**	**(Logical OR source with destination)**		**N,Z,V=0,C=0**
	class3,Dn	2/4/6	OR.B BYTE,D2	
	Dn,class7	2/4	OR D4,(A4)	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
ORI[a] ORI.B ORI.W ORI.L	source,dest	(Logical OR immediate)		N,Z,V=0,C=0
	immed,class5	4/6/8	ORI.W #101010H,3(A6)	

Shift and Rotate Instructions

Mnemonic	Operands	(Description)		Flags Affected
ASL ASL.B ASL.W ASL.L	count,source	(Arithmetic shift left, 0→LSB, MSB→C, X)		X,N,Z,V,C
	Dn,Dn	2	ASL.L D0,D4	
	immedS,Dn[d]	2	ASL.B #3,D2	
	class7[e]	2/4	ASL (A4)	
ASR ASR.B ASR.W ASR.L	count,source	(Arithmetic shift right, MSB→MSB, LSB→C, X)		X,N,Z,V,C
	Dn,Dn	2	ASR.W D2,D4	
	immedS,Dn[d]	2	ASR #7,D4	
	class7[e]	2/4	ASR ROUND	
LSL LSL.B LSL.W LSL.L	count,source	(Logical shift left, 0→LSB, MSB→C, X)		X,N,Z,V=0,C
	Dn,Dn	2	LSL.B D4,D2	
	immedS,Dn[d]	2	LSL.L #3,D4	
	class7[e]	2/4	LSL (A2)	
LSR LSR.B LSR.W LSR.L	count,source	(Logical shift right 0→MSB, LSB→C, X)		X,N,Z,V=0,C
	Dn,Dn	2	LSR D2,D3	
	immedS,Dn[d]	2	LSR.W #2,D2	
	class7[e]	2/4	LSR -(A2)	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
ROL				
ROL.B				
ROL.W				
ROL.L	count,source	(Rotate left without extend, MSB→LSB, C)		N,Z,V=0,C
	Dn,Dn	2	ROL.L D2,D4	
	immedS,Dn[d]	2	ROL.B #1,D3	
	class7[e]	2/4	ROL (A2)+	
ROR				
ROR.B				
ROR.W				
ROR.L	count,source	(Rotate right without extend, LSB→MSB, C)		N,Z,V=0,C
	Dn,Dn	2	ROR D2,D4	
	immedS,Dn[d]	2	ROR.W #4,D5	
	class7[e]	2/4	ROR (A2)	
ROXL				
ROXL.B				
ROXL.W				
ROXL.L	count,source	(Rotate left with extend, MSB→X, C, X→LSB)		X,N,Z,V=0,C
	Dn,Dn	2	ROXL D3,D4	
	immedS,Dn[d]	2	ROXL.L #1,D0	
	class7[e]	2/4	ROXL NUMB	
ROXR				
ROXR.B				
ROXR.W				
ROXR.L	count,source	(Rotate right with extend, LSB→X, C, X→MSB)		X,N,Z,V=0,C
	Dn,Dn	2	ROXR D0,D1	
	immedS,Dn[d]	2	ROXR.B #2,D0	
	class7[e]	2/4	ROXR (A0)	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	

Bit Manipulation Instructions

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
BCHG[f]	**k,source**	**(Test and change kth bit)**		**Z**
	Dn,class5	2/4	BCHG D4,DATA	
	immed2,class5	4/6	BCHG #0,D4	
BCLR[f]	**k,source**	**(Test and clear kth bit)**		**Z**
	Dn,class5	2/4	BCLR D0,TESTVAL	
	immed2,class5	4/6	BCLR #31,D3	
BSET[f]	**k,source**	**(Test and set kth bit)**		**Z**
	Dn,class5	2/4	BSET D2,(A2)	
	immed2,class5	4/6	BSET #15,D3	
BTST[f]	**k,source**	**(Test kth bit in source)**		**Z**
	Dn,class2	2/4	BTST D4,D2	
	immed2,class2	4/6	BTST #7,VALUE	

Binary Coded Decimal Instructions

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
ABCD	**source,dest**	**(Add BCD, source + dest + X→dest)**		**X,Z,C [N,V]**
	Dn,Dn	2	ABCD D0,D1	
	-(An),-(An)	2	ABCD -(A0),-(A1)	
NBCD	**dest**	**(Negate BCD, 0 − dest − X→dest)**		**X,Z,C [N,V]**
	class5	2/4	NBCD D4	
SBCD	**source,dest**	**(Subtract BCD, dest−source−X→dest)**		**X,Z,C [N,V]**
	Dn,Dn	2	SBCD D7,D6	
	-(An),-(An)	2	SBCD -(A6),-(A5)	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
Program Control Instructions				
Unconditional				
BRA **BRA.S**	**target**	**(Branch)**		**none**
	rel-addr	2/4	BRA.S $-4	
BSR **BSR.S**	**target**	**(Branch to subroutine PC→(SP))**		**none**
	rel-addr	2/4	BSR SUB1	
JMP	**target**	**(Jump)**		**none**
	class1	2/4	JMP STARTOVER	
JSR	**target**	**(Jump to subroutine PC→(SP))**		**none**
	class1	2/4	JSR MOD	
NOP	**no operand**	**(No operation)**		**none**
	—	2	NOP	
Conditional				
BCC **BCC.S**	**target**	**(Branch if carry clear)**		**none**
	rel-addr	2/4	BCC INRANGE	
BCS **BCS.S**	**target**	**(Branch if carry set)**		**none**
	rel-addr	2/4	BCS TOOBIG	
BEQ **BEQ.S**	**target**	**(Branch if equal, (Z=1))**		**none**
	rel-addr	2/4	BEQ.S $+4	
BGE **BGE.S**	**target**	**(Branch if greater or equal, N=V)**		**none**
	rel-addr	2/4	BGE NOW	
BGT **BGT.S**	**target**	**(Branch if greater, (Z=0 and N=V))**		**none**
	rel-addr	2/4	BGT TARGET	

Mnemonic	**Operands**	**(Description)**		**Flags Affected**
	Operand Types	Bytes	Example	
BHI **BHI.S**	**target**	**(Branch if high, (C=0 and Z=0))**		**none**
	rel-addr	2/4	BHI TOOHIGH	
BLE **BLE.S**	**target**	**(Branch if less or equal, (Z=1 or N <>V))**		**none**
	rel-addr	2/4	BLE.S LESS	
BLS **BLS.S**	**target**	**(Branch if low or same, (C=1 or Z=1))**		**none**
	rel-addr	2/4	BLS LOWVAL	
BLT **BLT.S**	**target**	**(Branch if less, (N<>V))**		**none**
	rel-addr	2/4	BLT LESS	
BMI **BMI.S**	**target**	**(Branch if minus, (N=1))**		**none**
	rel-addr	2/4	BMI MINUS	
BNE **BNE.S**	**target**	**(Branch if not equal, (Z=0))**		**none**
	rel-addr	2/4	BNE NOTNOW	
BPL **BPL.S**	**target**	**(Branch if plus, (N=0))**		**none**
	rel-addr	2/4	BPL PLUSVAL	
BVC **BVC.S**	**target**	**(Branch if no overflow, (V=0))**		**none**
	rel-addr	2/4	BVC INSIDE	
BVS **BVS.S**	**target**	**(Branch if overflow, (V=1))**		**none**
	rel-addr	2/4	BVS OVERF	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
DBCC[g]	count,target	(Unless C=0, decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBCC D1,AGAIN	
DBCS[g]	count,target	(Unless C=1, decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBCS D0,$−4	
DBEQ[g]	count,target	(Unless Z=1, decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBEQ D1,REPEAT	
DBF[g,h]	count,target	(Decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBF D3,JUMP	
DBGE[g]	count,target	(Unless N=V, decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBGE D1,OVER	
DBGT[g]	count,target	(Unless Z=0 and N=V, decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBGT D0,GREAT	
DBHI[g]	count,target	(Unless C=0 and Z=0, decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBHI D1,HIGH	
DBLE[g]	count,target	(Unless Z=1 or N<>V, decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBLE D0 $−4	
DBLS[g]	count,target	(Unless C=1 or Z=1, decrement count, branch if count <> −1)		none
	Dn,rel-addr	4	DBLS D1,AGAIN	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
DBLT[g]	**count,target**	**(Unless N<>V, decrement count, branch if count <> −1)**		**none**
	Dn,rel-addr	4	DBLT D0,TEST	
DBMI[g]	**count,target**	**(Unless N=1, decrement count, branch if count <> −1)**		**none**
	Dn,rel-addr	4	DBMI D7,MINUS	
DBNE[g]	**count,target**	**(Unless Z=0, decrement count, branch if count <> −1)**		**none**
	Dn,rel-addr	4	DBNE D0,NOW	
DBPL[g]	**count,target**	**(Unless N=0, decrement count, branch if count <> −1)**		**none**
	Dn,rel-addr	4	DBPL D3,NOTPLUS	
DBRA[g,h]	**count,target**	**(Decrement and branch if count <> −1)**		**none**
	Dn,rel-addr	4	DBRA AGAIN	
DBT[g]	**count,target**	**(No operation)**		**none**
	Dn,rel-addr	4	DBT D0,NEVER	
DBVC[g]	**count,target**	**(Unless V=0, decrement count, branch if count <> −1)**		**none**
	Dn,rel-addr	4	DBVC D1,OVER	
DBVS[g]	**count,target**	**(Unless V=1 decrement count, branch if count <> −1)**		**none**
	Dn,rel-addr	4	DBVS D2,NOVER	
SCC	**dest**	**(If C=0, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SCC TRFAL	
SCS	**dest**	**(If C=1, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SCS D2	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
SEQ	**dest**	**(If Z=1, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SEQ TEST	
SF	**dest**	**(Set dest to 1's)**		**none**
	class5	2/4	SF (A1)	
SGE	**dest**	**(If N=V, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SGE TFLAG	
SGT	**dest**	**(If Z=0 and N=V, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SGT D2	
SHI	**dest**	**If C=0 and Z=0, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SHI TEST	
SLE	**dest**	**(If Z=1 or N<>V, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SLE D4	
SLS	**dest**	**(If C=1 or Z=1, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SLS (A2)	
SLT	**dest**	**(If N<>V, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SLT D4	
SMI	**dest**	**(If N=1, set dest to 1's, otherwise set to 0)**		**none**
	class5	2/4	SMI T12	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
SNE	dest	If Z=0, set dest to 1's, otherwise set to 0)		none
	class5	2/4	SNE TEST	
SPL	dest	(If N=0, set dest to 1's, otherwise set to 0)		none
	class5	2/4	SPL TRY	
ST	dest	(Set dest to 0)		none
	class5	2/4	ST D3	
SVC	dest	(If V=0, set dest to 1's, otherwise set to 0)		none
	class5	2/4	SVC WHEN	
SVS	dest	(If V=1, set dest to 1's, otherwise set to 0)		none
	class5	2/4	SVS D1	
Returns				
RTR	no operands	(Return and restore condition codes)		X,N,Z,V,C
	—	2	RTR	
RTS	no operands	(Return from subroutine)		none
	—	2	RTS	

System Control Instructions

Privileged (Supervisor State Only)

ANDI ANDI.W	source,dest	(Logical AND status register)		N,Z,V=0,C=0
	immed,SR	2/4	ANDI #0FFFFH,SR	
EORI EORI.W	source,dest	(Logical exclusive-OR status register)		N,Z,V=0,C=0
	immed,SR	2/4	EORI #0FFH,SR	

Mnemonic	**Operands**	**(Description)**		**Flags Affected**
	Operand Types	Bytes	Example	
ORI				
ORI.W	**source,dest**	**(Logical OR status register)**		**none**
	immed,SR	2/4	ORI #700H,SR	
MOVE	**source,dest**	**(Move data to status register)**		**none**
	class3,SR	2/4	MOVE D2,SR	
MOVE	**source,dest**	**(Move data to/from user stack pointer)**		**none**
	An,USP	2	MOVE A1,USP	
	USP,An	2	MOVE USP,A2	
RESET	**no operands**	**(Reset external devices)**		**none**
		2	RESET	
RTE	**none**	**(Return from exception**		**X,N,Z,V,C**
	—	**(SP)→SR;** **(SP)→PC)**		
	—	2	RTE	
STOP	**source**	**(Load status register with data and stop)**		**X,N,Z,V,C**
	immed	4	STOP #8700H	
Trap Generating				
CHK	**bound,source**	**(Compare low-order word of source with bound; if (Dn) is greater than bound or negative an exception is processed)**		**N [Z,V,C]**
	class3,Dn	2/4	CHK EXCEP,D2	
TRAP	**vector**	**(Trap—initiate exception processing)**		**none**
	immedvect	2	TRAP #8H	
TRAPV	**none**	**(Trap on overflow)**		**none**
	—	2	TRAPV	

Mnemonic	Operands	(Description)		Flags Affected
	Operand Types	Bytes	Example	
Status Register				
ANDI				
ANDI.B	**source,dest**	**(AND source with condition codes)**		**X,N,Z,V,C**
	immed,CCR	4	ANDI #1FH,CCR	
EORI				
EORI.B	**data,dest**	**(Exclusive-OR condition code register)**		**X,N,Z,V,C**
	immed,CCR	4	EORI #10101B,CR	
ORI				
ORI.B	**data,dest**	**(Logical OR condition codes)**		**X,N,Z,V,C**
	immed,CCR	4	ORI #1111B,CCR	
MOVE	**source,dest**	**(Move byte to condition codes)**		**X,N,Z,V,C**
	class3,CCR	2/4	MOVE D2,CCR	
MOVE	**SR,dest**	**(Move status register to dest)**		**none**
	SR,class5	2/4	MOVE SR,D2	

[a] See the list of System Control instructions.

[b] The address register direct mode is not allowed for byte size.

[c] A word moved to a register will be sign extended.

[d] The immediate value must be in the range 1–8.

[e] The operand size is restricted to a word and only one bit may be shifted at a time.

[f] The value of the kth bit is copied to the Z flag. If the source to be tested is a data register k is interpreted modulo 32. If the source is a memory location k is interpreted modulo 8.

[g] This instruction only operates on the lower 16 bits of the data register.

[h] The DBF instruction is identical to the DBRA instruction.

[i] Source2 is subtracted from source1 and the condition codes are set. The results of the subtraction are not stored.

RESERVED WORDS

The following names may not be used to represent an address, data item, or variable.

68000 Mnemonics

ABCD
ADD
ADD.B
ADD.L
ADD.W
ADDA
ADDA.L
ADDA.W
ADDI
ADDI.B
ADDI.L
ADDI.W
ADDQ
ADDQ.B
ADDQ.L
ADDQ.W
ADDX
ADDX.B
ADDX.L
ADDX.W
AND
AND.B
AND.L
AND.W
ANDI
ANDI.B
ANDI.L
ANDI.W
ASL
ASL.B

ASL.L
ASL.W
ASR
ASR.B
ASR.L
ASR.W
BCC
BCC.S
BCHG
BCLR
BCS
BCS.S
BEQ
BEQ.S
BGE
BGE.S
BGT
BGT.S
BHI
BHI.S
BLE
BLE.S
BLS
BLS.S
BLT
BLT.S
BMI
BMI.S
BNE
BNE.S

BPL
BPL.S
BRA
BRA.S
BSET
BSR
BSR.S
BTST
BVC
BVC.S
BVS
BVS.S
CHK
CLR
CLR.B
CLR.L
CLR.W
CMP
CMP.B
CMP.L
CMP.W
CMPA
CMPA.L
CMPA.W
CMPI
CMPI.B
CMPI.L
CMPI.W
CMPM
CMPM.B

CMPM.L
CMPM.W
DBCC
DBCS
DBEQ
DBF
DBGE
DBGT
DBHI
DBLE
DBLS
DBLT
DBMI
DBNE
DBPL
DBRA
DBT
DBVC
DBVS
DIVS
DIVU
EOR
EOR.B
EOR.L
EOR.W
EORI
EORI.B
EORI.L
EORI.W
EXG

EXT
EXT.L
EXT.W
JMP
JSR
LEA
LINK
LSL
LSL.B
LSL.L
LSL.W
LSR
LSR.B
LSR.L
LSR.W
MOVE
MOVE.B
MOVE.L
MOVE.W
MOVEA
MOVEA.L
MOVEA.W
MOVEM
MOVEM.L
MOVEM.W
MOVEP
MOVEP.L
MOVEP.W
MOVEQ
MULS

MULU
NBCD
NEG
NEG.B
NEG.L
NEG.W
NEGX
NEGX.B
NEGX.L
NEGX.W
NOP
NOT
NOT.B
NOT.L
NOT.W
OR
OR.B
OR.L
OR.W
ORI
ORI.B
ORI.L
ORI.W
PEA
RESET
ROL
ROL.B
ROL.L
ROL.W
ROR

ROR.B
ROR.L
ROR.W
ROXL
ROXL.B
ROXL.L
ROXL.W
ROXR
ROXR.B
ROXR.L
ROXR.W
RTE
RTR
RTS
SBCD
SCC
SCS
SEQ
SF
SGE
SGT
SHI
SLE
SLS
SLT
SMI
SNE
SPL
ST
STOP

SUB
SUB.B
SUB.L
SUB.W
SUBA
SUBA.L
SUBA.W
SUBI
SUBI.B
SUBI.L
SUBI.W
SUBQ
SUBQ.B
SUBQ.L
SUBQ.W
SUBX
SUBX.B
SUBX.L
SUBX.W
SVC
SVS
SWAP
TAS
TRAP
TRAPV
TST
TST.B
TST.L
TST.W
UNLK

68000 Register Names

A0
A0.L
A0.W
A1
A1.L
A1.W
A2
A2.L
A2.W
A3
A3.L
A3.W
A4
A4.L
A4.W
A5
A5.L
A5.W
A6
A6.L
A6.W
A7
A7.L
A7.W
CCR
D0
D0.L
D0.W
D1
D1.L
D1.W
D2
D2.L
D2.W
D3
D3.L
D3.W
D4
D4.L
D4.W
D5
D5.L
D5.W
D6
D6.L
D6.W
D7
D7.L
D7.W
SP
SR
USP

Tektronix Assembler Directives, Options, and Operators

ABSOLUTE
ADDRESS
ALIGN
ASCII
ASET
BASE
BITS
BLOCK
BYTE
CLASS
CND
COMMON
CON
DBG
DEF
ELSE
ELSEIF
END
ENDIF
ENDM
ENDOF
ENDR
EQU
EXITM
EXITR
FLOAT
GLOBAL
HI
IF
INCLUDE
INPAGE
LINE
LIST
LO
LONG
MACRO
ME
MEG
MOD
NAME
NCHR
NOLIST
ORG
PAGE
REPEAT
RESERVE
RESUME
SCALAR
SECTION
SEG
SHL
SHR
SPACE
STITLE
STRING
STRINGOF
SYM
TIMES
TITLE
WARNING
WORD
XREF

Tektronix Special Assembler Directives

GEN.R GEN.S GEN.L

PAGE SIZE

The page size for the 68000 assembler is 256 bytes.

DEFAULT RELOCATION TYPE

The default relocation type for the 68000 is word-relocatable.

ERROR MESSAGES

The following error messages apply only to the 68000 assembler.

*****ASM: 245 (E) Invalid register list.** The register list on the MOVEM instruction is invalid. Common causes are specifying both a data register and an address register (A1 – D2), or specifying a bad range (A7 – A3).

*****ASM: 246 (W) Invalid bit expression.** Immediate expression in a BCLR, BCHG, BSET, or BTST is not in the range 0 – 65535. The value is truncated to 16 bits.

*****ASM: 247 (E) Invalid quick expression.** Quick expression is not a scalar or not in the range 1 – 8.

*****ASM: 248 (W) Word value too large; truncated.** The expression used exceeds 65535.

*****ASM: 249 (W) Byte value too large; truncated.** The expression used exceeds 255.

*****ASM: 250 (E) Illegal effective address form.** The address form used is not allowed with this instruction.

*****ASM: 251 (W) Address not even.** It is illegal to use an odd address in a word, long, or branch instruction.

*****ASM: 252 (E) Branch out of range.** The branch address is not in the range $ – 32676 to $ + 32679.

*****ASM: 253 (E) Invalid branch expression.** Expression on a short branch is:

1. Not in the range $ – 126 to $ + 129.
2. A branch to the next instruction (displacement is zero).
3. Not in the current section.

*****ASM: 254 (E) Invalid TRAP expression.** The expression in the TRAP instruction is not a scalar or not in the range 0 – 15.

IRREGULARITIES

The method used to select the absolute short, absolute long, or relative forms of addressing for the Tektronix assembler is different from the method used by the Motorola assembler. See the discussion of "Specifying Addressing Modes" in this section for information relating to the method used by the Tektronix assembler.

The Tektronix assembler allows relocatable expressions to be used as immediates except for the immediates in the ADDQ and SUBQ instructions (must be a scalar in the range 1—8) and the vector of the TRAP instruction (must be a scalar in the range 0—15). The Motorola assembler requires all immediates to be scalars.

SERIES A 68000 ASSEMBLER—SERIES B 68000 ASSEMBLER DIFFERENCES

Table 9C-3 lists the differences between the Series A and Series B assemblers. This information is intended for users of the TEKTRONIX 8300AXX (Series A).

Table 9C-3
Series A 68000 Assembler—Series B 68000 Assembler Differences

Series A	Series B
Long operands (32 bits) not supported	Long operands (32 bits) supported.
Long branches allowed only in current section.	Long branches out of current section allowed.
The address (A), quick (Q), and immediate (I) forms of the ADD, CMP EOR, MOVE, OR, and SUB instructions must be chosen by the user. Therefore, only data alterable addressing modes are allowed in the destination of the MOVE instruction.	The address (A), quick (Q), and immediate (I) forms of the ADD, CMP EOR, MOVE, OR, and SUB instructions are chosen automatically. Therefore, all alterable addressing modes are allowed in the destination of the MOVE instruction.

In addition to the differences listed in Table 9C-3, the method used to distinguish relative addressing with displacement from long and short absolute addressing is not the same with the Series A assembler as it is with the Series B assembler. The is also true of address register indirect addressing with displacement and relative addressing with index. In the Series A assembler the user could not easily control which addressing mode was used. With the Series B assembler, the user has full control with the use of 3 special assembler directives, GEN.R, GEN.L, and GEN.S. Of course, these are reserved words for the Series B assembler. See the discussion of "Specifying Addressing Modes" in this section for more information about these directives.

MANUAL CHANGE INFORMATION

At Tektronix, we continually strive to keep up with latest electronic developments by adding circuit and component improvements to our instruments as soon as they are developed and tested.

Sometimes, due to printing and shipping requirements, we can't get these changes immediately into printed manuals. Hence, your manual may contain new change information on following pages.

A single change may affect several sections. Since the change information sheets are carried in the manual until all changes are permanently entered, some duplication may occur. If no such change pages appear following this page, your manual is correct as printed.

Tektronix®
COMMITTED TO EXCELLENCE

MANUAL CHANGE INFORMATION

Date: 2-2-82 Change Reference: C1/282

Product: 8500 MDL: 68000 Assembler Specifics Users Manual Part No.: 070-3855-00

DESCRIPTION

TEXT CORRECTIONS

Page 9C-6 Under the heading "Special Address Modes", change the first paragraph to read:

The Special Address Modes include Absolute Short Address, Absolute Long Address, Relative, Relative with Index, Immediate, and Status Register addressing.

Page 9C-7 Under the heading "Specifying Addressing Modes", replace the first paragraph with the following:

The syntax for absolute short, absolute long, and relative is the same. The syntax for address register indirect with displacement and relative with index is also the same.

In the second paragraph, change 9C-2 to 9C-1.

Page 9C-8 Table 9C-1 should appear as follows:

Table 9C-1
Special Assembler Directives

Directive	Address Form Used	
	Syntax: d or addr	Syntax: d(An)
GEN.R	Relative with displacement	Relative with index
GEN.S (default)	Absolute short	Address register indirect with displacement
GEN.L	Absolute short if d (or addr) is a backward reference 0-32767 in current section or a scalar 0-32767. Otherwise absolute long.	Address register indirect with displacement

DESCRIPTION

Page 9C-8 Under the heading, "Address, Quick, and Immediate Forms", remove the word "NEG" from the second line.

Page 9C-26 Before the TRAP instruction, add the following:

ILLEGAL	**none**	**(Illegal instruction Push PC, SR; initiate illegal instruction processing)**	**none**
	–	2	ILLEGAL

Page 9C-30 Under the heading "Error Messages" replace the entire list of error messages with the following:

The following error messages apply only to the 68000 assembler.

*****ASM: 245 (W) Address too large.** A short absolute address is greater than 32767 or a long absolute address is greater than 16,777,215.

*****ASM: 246 (E) Expression is complex or has HI, LO, BITS or ENDOF applied.** An expression that is the difference of two relocatable symbols or has the HI, LO, BITS, or ENDOF function applied may not be used with the following:

- Absolute short addressing
- Absolute long addressing
- Program counter with index addressing
- Program counter with displacement addressing
- Branch instructions
- Test condition, decrement, and branch instructions
- The operand(s) of the ADDRESS directive

DESCRIPTION

***ASM: 247 (E) Invalid register list. A register list is invalid. Either the registers are of different types (A1-D2), or the range is illegal (A7-A3).

***ASM: 248 (E) Invalid quick expression. A quick expression is not a scalar or not in the range 1-8.

***ASM: 249 (W) Word value too large; truncated. The value of the expression used exceeds 65535.

***ASM: 250 (W) Byte value too large; truncated. The value of the expression used exceeds 255.

***ASM: 251 (E) Illegal effective address form. The address form used is not allowed with this instruction.

***ASM: 252 (W) Address not even. It is illegal to use an odd address in a word, long, or branch instruction.

***ASM: 253 (E) Long branch out of range. The branch address is not in the range $-32666 to $+32669.

***ASM: 254 (E) Invalid short branch expression. Expression on a short branch is either:

- Not in the range $-126 to $+129.
- A branch to the next instruction (displacement is zero).
- Not in the current section.
- The difference of two relocatable symbols.

***ASM: 255 (E) Invalid TRAP expression. The expression in the TRAP instruction is not a scalar or not in the range 0-15.

DESCRIPTION

After page 9C-31, add the following discussion:

68000 LINKER SPECIFICS

The 68000 microprocessor can address four memory spaces:

1. User Data space -- The most significant byte of the 32-bit address is 01.
2. User Program space -- The most significant byte of the 32-bit address is 02.
3. Supervisor Data space -- The most significant byte of the 32-bit address is 04.
4. Supervisor Program space -- The most significant byte of the 32-bit address is 08.

Since the top byte is reserved for memory space mapping, each memory space has a maximum range of 0-16,777,215 (24 bits). You may specify the memory space into which a particular section of code is to be loaded. This may be done in the source code with an ORG directive or during the linking operation. If this is done in the source code, the section must be ABSOLUTE and the high byte in operand of the ORG directive must contain the proper memory space mapping information. If the High byte is zero then the code will be loaded into the default memory space.

NOTE

See the 8500 Modular MDL Series 68000 Emulator Specifics Users Manual for examples and more information about memory partitioning and linking.