2 68000 INSTRUCTIONS AND ADDRESSING MODES

In 1971 Intel introduced the 4004, which had 46 instructions. Later on, Intel came out with the 8008. This had 48 instructions. In 1973 the Intel 8080 appeared with 78 basic instruction types, giving well over 200 instructions. The year 1974 saw the release of the Motorola 6800, and within a further couple of years the Zilog Z80 processor appeared with 158 basic instruction types and around 700 instructions. The dawn of the 1980s saw the release of the 68000. This processor has 56 basic instruction mnemonics and over a thousand instructions!

A decade of microelectronic research has clearly produced some amazing achievements. One of the most interesting points, as far as the 68000 is concerned, is that the number of instructions available has increased greatly, with only a minor increase in the number of instruction mnemonics. This has important consequences. First, the instruction set is not hard to learn. Second, the instructions must be capable of multiple addressing modes, of which the 68000 has 14. Additionally, the 68000 has the ability to deal with five basic data types. It is the merging of these basic instruction mnemonics, addressing modes, and data types that give the 68000 its large, yet simple to learn, instruction repertoire.

In this chapter we will look at the basic instructions that are available, together with the addressing modes which the instructions may variably use. Only an overview is presented, for reasons already mentioned (see Introduction). The emphasis is toward a companion to the 68000 so that anyone, with some knowledge of assembly language programming, can quickly assimilate the capabilities of the processor.

When discussing these topics it becomes important to know how the resultant instructions are actually presented to an assembler. Assembler packages vary in the form of syntax that they will allow, and therefore descriptions in this chapter will relate to the McGraw-Hill assembler documented in Chapter 14. This particular assembler was used to create all the example programs given in this book. Clearly if some other assembler package is used there will almost certainly be some syntax changes required.

In particular it will be noticed, at least by those of you who have looked also at a detailed text on the 68000 using Motorola nomenclature, that there appear to be less instruction types described (e.g., ADDI is not listed). The McGraw-Hill assembler is a full implementation and, therefore, all instructions do exist; it is simply that certain instructions, listed by Motorola as being variations, have been included here as a natural member of the parent instruction. For example, Motorola's ADDI instruction is simply an ADD instruction using immediate data as the source operand.

2.1 Addressing modes

Six basic addressing modes in the 68000 give rise to 14 actual modes. The modes of addressing are shown in Fig.2.1, together with the appropriate assembler syntax. Not every addressing mode can be used with all instructions, though the homogeneity is remarkable.

MODE	SYNTAX
Implied	
Register	SR, CCR, USP, PC
Immediate	
Immediate	#n
Quick immediate	#b
Absolute	
Short	a16
Long	a32
Register Direct	
Data register	Dn
Address register direct	An
Register Indirect	
Address register	(An)
Postincrement	(An)+
Predecrement	-(An)
Address register with offset	d16(An)
Register with index and offset	d8(An,i)
Program Counter Relative	
Address register with offset	d16(PC)
Register with index and offset	d8(PC,i)

Notes:

b = 3, 4 or 8 bits
n = 8,16, or 32 bits
d8 = 8 bit offset
d16 = 16 bit offset
a16 = 16 bit address
a32 = 32 bit address

i = An or Dn
An = address register
Dn = data register
PC = current location
SR = status register
CCR = condition codes
USP = user stack ptr

Figure 2.1 68000 addressing modes

Perhaps the major deviation is with respect to address registers as operands. When these registers are used within direct addressing modes, bit or byte data types are never permitted. There are also some instructions which are severely limited in their choice of operands. For example, the Bcc (Branch) and DBcc (Decrement and Branch) instructions may only have an absolute address as their operand. This absolute address is normally specified in terms of a label. The value entered into the instruction opcodes will be an offset, so that the instructions provide relative addressing.

Note that there is also what is termed inherent addressing. Instructions using this form of addressing mode require no operands. There are six instructions which fall into this category (viz., NOP, RESET, RTE, RTR, RTS, and TRAPV).

PROGRAM COUNTER RELATIVE ADDRESSING

A special mention must be given to the way in which the assembler will handle this form of addressing. In principle, program (counter) relative addressing modes are the same as address register indirect with offset (or offset and index), and the overall syntax is therefore the same. The difference, of course, is that the program counter (PC) is being used instead of an address register for the indirection.

However, the major purpose in using this particular form of addressing is to obtain position independent code. The offsets given to the instruction are, therefore, normally labels within the program. Clearly the assembler should not allocate the value of the label as being the offset; rather it should allocate the offset required to get from where the program counter is at the moment to where the label is. As such, the assembler will manipulate the instructions in one of two ways. First, if the offset expression starts with an integer, the absolute value of the expression will be allocated as the offset:

```
Opcode          Instruction
4BFA1040        lea $1040(PC),a5
```

Second, if the offset expression starts with a symbol, the symbol will be treated as a label (whether it was or not; the assembler has no way of knowing) and the true offset will be allocated:

```
Address    Opcode        Instruction
282AE      44FA0012      move lab(PC),CCR
```

The actual value of the symbol (label) 'lab' is $282C2, and $0012 is the offset required to reach it from the given instruction.

Note that program counter relative addressing modes can never be used as destination operands.

SOURCE AND DESTINATION OPERANDS

Instructions for the 68000 may require no operands at all, a single source or destination operand, both a source operand and a destination operand, or (in one case only - BTST) two source operands. Whenever two operands are required, the source operand is always specified first, and separated from the destination operand by a comma (,). For example:

1. **No operand:** rts
2. **Source operand:** clr (a0)
3. **Source and destination operand:** divs (a1)+,d1

The result of any operation involving two operands will be stored in the destination operand effective address, if this is pertinent. Source operand effective address contents will not be changed.

This declaration order for the operands is normally very helpful and readable. An exception to this is in the CMP instruction. For example, 'CMP.L D1,D2' compares D2 against D1. If D1 is less than D2, this comparison will yield the condition GT (greater than; signed) or HI (higher; unsigned).

2.2 Condition codes

In the following descriptions of the 68000 instructions there are three instructions (Bcc, DBcc, and Scc) which use a set of conditional tests. The tests are given 'one/two character' mnemonics and the full instruction mnemonic consists of the above names with 'cc' replaced by the test mnemonic (e.g., BHI, BF, DBEQ, SNE, and so on). Each test produces a true or false result depending on the state of given condition flags in the 68000 CCR register. The tests, their mnemonics, and their interpretation, are as follows:

Mnemonic	Test	Interpretation
T	1	true (always)
F	0	false (always)
HI	not(C).not(Z)	higher (unsigned)
LS	C+Z	less than or same (unsigned)
CC	not(C)	carry clear (unsigned)
CS	C	carry set (unsigned)
NE	not(Z)	not equal
EQ	Z	equal
VC	not(V)	overflow clear
VS	V	overflow set
PL	not(N)	plus
MI	N	minus
GE	not(N xor V)	greater than or equal (signed)
LT	N xor V	less than (signed)
GT	not(Z+(N xor V))	greater than
LE	Z+(N xor V)	less than or equal

Some of the above mnemonics have alternative mnemonics, in order to improve their readability under given instances (see Sec.2.4).

2.3 Condition code flag handling

The condition code flags (X, N, V, Z, and C) are manipulated, at various times and in various ways, by the instruction set of the 68000. The handling of these flags may seem a little irregular, and indeed it is, but the irregularity is not some strange quirk of the processor; rather it is a positive phenomenon. In general, the condition codes are set according to the value being sent to the destination operand. This is also true of the operand in the TST instruction and the second operand in the CMP instruction, even though these 'destination' operands are not altered. A near general exception to this rule is in the use of address registers as destination operands. In this case the condition codes are not altered. This enables the adjustment of stack pointers, and the calculation of addresses to be performed without wiping out condition flags set by a previous operation. However, note that the condition codes are set when an address register is the 'destination' register of a CMP instruction (it would be a poor state of affairs if they were not!).

The handling of the Z flag, in particular, is even more variable. The extended operations (ABCD, ADDX, NEGX, SBCD, and SUBX) cause the Z flag to be cleared if the result is non-zero, or left alone in all other cases. This means that at the end of a series of extended operations the flag will only be set if all the results were zero. For bit operations (BCHG, BCLR, BSET, BTST, and TAS) the Z flag is set according to the state of the specified bit **before** the operation.

2.4 Alternative mnemonics

A set of alternative mnemonics exists within the assembler to aid the programmer both in terms of style and readability. First is the mnemonic for 'exclusive-or' operations. There are two widely used mnemonics for this instruction and both are supported:

Standard	**Alternative**
EOR	**XOR**

Only the mnemonic EOR is listed in the following descriptions of the 68000 instructions.

Second, there is the common confusion, especially with processors which cater for signed and unsigned arithmetic, as to the true interpretation of the 'carry-clear' and 'carry-set' conditional statements. As such the assembler provides the following:

Standard	Alternative
BCC, BCS	BHS, BLO
DBCC, DBCS	DBHS, DBLO
SCC, SCS	SHS, SLO

The mnemonic part 'HS' stands for 'higher or same', and 'LO' stands for 'lower'. They differ from the 'greater or equal' (GE) and 'less than' (LT) mnemonics in that they refer to conditions set after an unsigned operation.

2.5 68000 instructions

There are 56 basic instructions for the 68000 processor. The assembler allocates a further seven variations, bringing the total to 63 instruction mnemonics. Each instruction mnemonic is discussed briefly giving, amongst other things, details of its addressing modes. The instructions are covered in alphabetical order for quick reference. Appendix A contains a summary of the instructions, showing their effect upon the CCR flags.

References will be found to items called data qualifiers. These are qualifiers that can be given to certain instructions, which specify what type of data size is to be used. For example, let us look at the MOVE instruction. We can move bytes (8 bits), words (16 bits) or long-words (32 bits). The same MOVE instruction mnemonic is used in all three cases; it is the qualifier which determines the actual instruction operation. This gives rise to the following three forms:

MOVE.B MOVE.W MOVE.L

The qualifier '.L', and the additional qualifier '.S', may be found also when looking at instructions that use a label as their operand (e.g., BSR - branch to subroutine). In this context, '.L' stands for long, and '.S' for short. A label addressed as being short must be within +127 bytes or -128 bytes of the current program counter position. A long label can be up to within +32767 or -32768 bytes of the current position. Short branch instructions use less bytes of opcode, and therefore it is worth specifying them as such if you know a label is in range but that it is as yet unknown to the assembler (i.e., because it is a forward reference during pass 1). There is no need to specify a qualifier for backward references because the assembler will always use a short addressing mode wherever possible.

ABCD ADD DECIMAL WITH EXTEND

Addressing modes: -(An),-(An)
Dn,Dn

Flags affected: X N Z V C

Privileged instruction: no

Byte data size only. Adds two BCD digits in source byte, with extend, to two BCD digits in destination byte.

ADD ADD

Addressing modes:

	#n	
	a16	
	a32	
	Dn	
	(An)	
	(An)+	,Dn
	-(An)	,An
	d16(An)	
	d8(An,i)	
	d16(PC)	
	d8(PC,i)	
#n,		a16
		a32
		(An)
Dn,		(An)+
		-(An)
		d16(An)
		d8(An,i)

An,An
An,Dn

Flags affected: X N Z V C

Privileged instruction: no

Register An may not be used as destination for byte operations. Adds source to destination.

ADDQ ADD QUICK

Addressing modes:

#b, { Dn, An, a16, a32, (An), (An)+, -(An), d16(An), d8(An,i) }

Flags affected: X N Z V C

Privileged instruction: no

Register An may not be used as destination for byte operations. Word and long-word operations are identical. Adds data (1 to 8) to destination.

ADDX ADD WITH EXTEND

Addressing modes: -(An),-(An)
Dn,Dn

Flags affected: X N Z V C

Privileged instruction: no

Adds source, with extend, to destination.

AND LOGICAL AND

Addressing modes:

{ #n, a16, a32, Dn, (An), (An)+, -(An), d16(An), d8(An,i), d16(PC), d8(PC,i) } ,Dn

	#n, Dn,	a16 a32 (An) (An)+ -(An) d16(An) d8(An,i)
	#n,SR	

Flags affected: N Z V C

Privileged instruction: no, except for AND.W #n,SR

Register SR may not be used with long-word operations. CCR is least significant byte of SR, and accessed by 'AND.B #n,SR'. Logically ANDs all bits of source with corresponding bits of destination.

ASL ARITHMETIC SHIFT LEFT

Addressing modes:

a16	(no data qualifier)
a32	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
#b,Dn	(data qualifier used)
Dn,Dn	

Flags affected: X N Z V C

Privileged instruction: no

Data size is always word. Destination shift is always by one bit when no qualifier is present, or by a count of up to 63. An immediate shift count can be given of 0 to 7 only. Zero signifies a shift of eight places. Sets V flag if sign bit changes at any time during shift.

ASR ARITHMETIC SHIFT RIGHT

Addressing modes:

a16	(no data qualifier)
a32	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	

#b,Dn (data qualifier used)
Dn,Dn

Flags affected: X N Z V C

Privileged instruction: no

Data size is always word. Destination shift is always by one bit when no qualifier is present, or by a count of up to 63. An immediate shift count can be given of 0 to 7 only. Zero signifies a shift of eight places. Sign bit is replicated.

Bcc BRANCH CONDITIONALLY

Addressing modes: label

Flags affected: none

Privileged instruction: no

Label may be declared short (.S) or long (.L). Byte or word offsets are used.

BCHG BIT TEST AND CHANGE

Addressing modes:

#n,	a16
Dn,	a32
	Dn
	(An)
	(An)+
	-(An)
	d16(An)
	d8(An,i)

Flags affected: Z

Privileged instruction: no

Byte operations only, except when data register is destination. If Dn is destination then data size is long-word. Tests (setting Z flag) and then inverts the specified bit of destination.

BCLR BIT TEST AND CLEAR

Addressing modes:

#n, Dn,	a16 a32 Dn (An) (An)+ -(An) d16(An) d8(An,i)

Flags affected: Z

Privileged instruction: no

Byte operations only, except when data register is destination. If Dn is destination then data size is long-word. Tests (setting Z flag) and then clears the specified bit of destination to zero.

BRA BRANCH ALWAYS

Addressing modes: label

Flags affected: none

Privileged instruction: no

Label may be declared short (.S) or long (.L). Byte or word offsets are used.

BSET BIT TEST AND SET

Addressing modes:

#n, Dn,	a16 a32 Dn (An) (An)+ -(An) d16(An) d8(An,i)

Flags affected: Z

Privileged instruction: no

Byte operations only, except when data register is destination. If Dn is destination then data size is long-word. Tests (setting Z flag) and then sets the specified bit of destination to one.

BSR BRANCH TO SUBROUTINE

Addressing modes: label

Flags affected: none

Privileged instruction: no

Label may be declared short (.S) or long (.L). Pushes address of next instruction and then branches by a byte or word offset.

BTST BIT TEST

Addressing modes:

#n, Dn,	a16
	a32
	Dn
	(An)
	(An)+
	-(An)
	d16(An)
	d8(An,i)
	d16(PC)
	d8(PC,i)

Flags affected: Z

Privileged instruction: no

Byte operations only, except when data register is destination. If Dn is destination then data size is long-word. Tests (setting Z flag) the specified bit.

CHK CHECK REGISTER AGAINST BOUNDS

Addressing modes:

#n	,Dn
a16	
a32	
Dn	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
d16(PC)	
d8(PC,i)	

Flags affected: N Z V C

Privileged instruction: no

Data size is word only. Will generate an exception if Dn is less than zero or greater than operand contents.

CLR CLEAR OPERAND

Addressing modes:

a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flags affected: N Z V C

Privileged instruction: no

Operand data size is cleared to zero.

CMP COMPARE

Addressing modes:

#n
a16
a32
Dn
(An)
(An)+ ,Dn
-(An) ,An
d16(An)
d8(An,i)
d16(PC)
d8(PC,i)

#n, a16
a32
(An)
(An)+
-(An)
d16(An)
d8(An,i)

An,An
An,Dn

Flags affected: N Z V C

Privileged instruction: no

Register An may not be used as destination for byte operations. Subtracts source from destination but does not store the result.

CMPM COMPARE MEMORY

Addressing modes: (An)+,(An)+

Flags affected: N Z V C

Privileged instruction: no

Subtracts source from destination but does not store the result. Not an extended operation.

DBcc DECREMENT AND BRANCH CONDITIONALLY

Addressing modes: Dn,label

Flags affected: none

Privileged instruction: no

If condition is not met, data register word is decremented, then if result is not -1, branches by word offset. (DBT is a 4-byte no-op).

DBRA DECREMENT AND BRANCH ALWAYS

Addressing modes: Dn,label

Flags affected: none

Privileged instruction: no

Decrements data register word and then branches by word offset.

DIVS SIGNED DIVIDE

Addressing modes:

#n	,Dn
a16	
a32	
Dn	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
d16(PC)	
d8(PC,i)	

Flags affected: N Z V C

Privileged instruction: no

Divides destination long-word by source word. Quotient is put in low order word, remainder (same sign as dividend!) is put in high order word.

DIVU UNSIGNED DIVIDE

Addressing modes:

#n	,Dn
a16	
a32	
Dn	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
d16(PC)	
d8(PC,i)	

Flags affected: N Z V C

Privileged instruction: no

Divides destination long-word by source word. Quotient is put in low order word, remainder in high order word.

EOR EXCLUSIVE OR

Addressing modes:

#n, Dn,	Dn a16 a32 (An) (An)+ -(An) d16(An) d8(An,i)
#n,SR	

Flags affected: N Z V C

Privileged instruction: no, except for EOR.W #n,SR

Register SR may not be used with long-word operations. CCR is least significant byte of SR, and accessed by 'AND.B #n,SR'. Exclusive ORs all bits of source with corresponding bits of destination.

EXG EXCHANGE REGISTERS

Addressing modes: An,Dn
Dn,Dn
An,An
Dn,An

Flags affected: none

Privileged instruction: no

Long-word operations only. Exchanges complete contents of two registers.

EXT SIGN EXTEND

Addressing modes: Dn

Flags affected: N Z V C

Privileged instruction: no

Byte operations not allowed. Extends sign bit of low order half of destination, through the entire high order half of destination.

JMP JUMP

Addressing modes: a16
a32
(An)
d16(An)
d8(An,i)
d16(PC)
d8(PC,i)

Flags affected: none

Privileged instruction: no

Sets program counter to destination address.

JSR JUMP TO SUBROUTINE

Addressing modes: a16
a32
(An)
d16(An)
d8(An,i)
d16(PC)
d8(PC,i)

Flags affected: none

Privileged instruction: no

Pushes address of next instruction, and sets program counter to destination address.

LEA LOAD EFFECTIVE ADDRESS

Addressing modes: a16
a32
(An)
d16(An) ,An
d8(An,i)
d16(PC)
d8(PC,i)

Flags affected: none

Privileged instruction: no

Puts the effective address of the source into the destination register.

LINK LINK STACK

Addressing modes: An,#n

Flags affected: none

Privileged instruction: no

The contents of An are pushed onto the stack. Register An is then loaded from the updated stack pointer. Finally, the sign-extended dispacement is added to the stack pointer.

LSL LOGICAL SHIFT LEFT

Addressing modes:

a16	(no data qualifier)
a32	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
#b,Dn	(data qualifier used)
Dn,Dn	

Flags affected: X N Z V C

Privileged instruction: no

Data size is always word. Shift is always by one bit when no qualifier is present, or by a count of up to 63. An immediate shift count can be given of 0 to 7 only. Zero signifies a shift of eight places.

LSR LOGICAL SHIFT RIGHT

Addressing modes:

a16	(no data qualifier)
a32	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
#b,Dn	(data qualifier used)
Dn,Dn	

Flags affected: X N Z V C

Privileged instruction: no

Data size is always word. Shift is always by one bit when no qualifier is present, or by a count of up to 63. An immediate shift count can be given of 0 to 7 only. Zero signifies a shift of eight places.

MOVE MOVE

Two categories of MOVE instruction exist; those that use a data qualifier and those that do not.

1. MOVE instructions that require a data qualifier

Addressing modes:

#n			
a16			a16
a32			a32
Dn			Dn
** An			An **
(An)			(An)
(An)+	,		(An)+
-(An)			-(An)
d16(An)			d16(An)
d8(An,i)			d8(An,i)
d16(PC)			
d8(PC,i)			

(** An address register may not be used as a source or destination operand if the data type is byte)

Flags affected: N Z V C
(No flags are affected if the destination is An)

Privileged instruction: no

2. MOVE instructions that do not use a data qualifier

Addressing modes:

#n	
a16	
a32	
Dn	
(An)	,CCR
(An)+	,SR
-(An)	
d16(An)	
d8(An,i)	
d16(PC)	
d8(PC,i)	

An,USP
USP,An

SR, { a16, a32, Dn, (An), (An)+, -(An), d16(An), d8(An,i) }

Flags affected: X N Z V C
(No flags are affected if the source is SR or USP)

Privileged instruction: yes, unless moving from SR or to CCR

Moving to CCR or SR is always word. When moving to CCR only least significant byte is used to update condition codes. Moving from SR is always word. USP operations are always long-word.

MOVEM MOVE MULTIPLE REGISTERS

Addressing modes: { a16, a32, (An), (An)+, d16(An), d8(An,i), d16(PC), d8(PC,i) } ,<reg-list>

<reg-list>, { a16, a32, (An), -(An), d16(An), d8(An,i) }

Flags affected: none

Privileged instruction: no

Data size is word or long-word. Register list can be any list of data or address registers separated by a comma (no ranges allowed by assembler).
For example: MOVEM.L locstore, A1,A2,A3,D4,D6
Organization in memory is D0 at lowest address, A7 at highest.

MOVEP MOVE PERIPHERAL DATA

Addressing modes: Dn,d16(An)
d16(An),Dn

Flags affected: none

Privileged instruction: no

Data size is word or long-word only. Bytes are transferred to/from alternate memory locations. If address is even, transfer is on high order half of data bus (68000 only; 68008 has 8-bit data bus).

MOVEQ MOVEQ

Addressing modes: #b,Dn

Flags affected: N Z V C

Privileged instruction: no

Moves data (-128 to +127) to complete data register.

MULS SIGNED MULTIPLY

Addressing modes:

#n	,Dn
a16	
a32	
Dn	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
d16(PC)	
d8(PC,i)	

Flags affected: N Z V C

Privileged instruction: no

The low order half of destination long-word is multiplied by source word.

MULU UNSIGNED MULTIPLY

Addressing modes:

#n	
a16	
a32	
Dn	
(An)	
(An)+	,Dn
-(An)	
d16(An)	
d8(An,i)	
d16(PC)	
d8(PC,i)	

Flags affected: N Z V C

Privileged instruction: no

The low order half of destination long-word is multiplied by source word.

NBCD NEGATE DECIMAL WITH EXTEND

Addressing modes:

a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flags affected: X N Z V C

Privileged instruction: no

Byte data size only. Subtracts the two BCD digits of destination, with extend, from zero.

NEG NEGATE

Addressing modes:

a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flags affected: X N Z V C

Privileged instruction: no

Subtracts the destination from zero.

NEGX NEGATE WITH EXTEND

Addressing modes:
a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flags affected: X N Z V C

Privileged instruction: no

Subtracts the destination, with extend, from zero.

NOP NO OPERATION

Addressing modes: inherent

Flags affected: none

Privileged instruction: no

NOT ONE'S COMPLEMENT

Addressing modes:
a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flags affected: N Z V C

Privileged instruction: no

Inverts all bits of the destination.

OR LOGICAL OR

Addressing modes:

	#n	
	a16	
	a32	
	Dn	
	(An)	
	(An)+	,Dn
	-(An)	
	d16(An)	
	d8(An,i)	
	d16(PC)	
	d8(PC,i)	

		a16
		a32
		(An)
#n,		(An)+
Dn,		-(An)
		d16(An)
		d8(An,i)

#n,SR

Flags affected: N Z V C

Privileged instruction: no, except for OR.W #n,SR

Register SR may not be used with long-word operations. CCR is least significant byte of SR, and accessed by 'AND.B #n,SR'. Logically ORs all bits of the source with corresponding bits of destination.

PEA PUSH EFFECTIVE ADDRESS

Addressing modes:

a16
a32
(An)
d16(An)
d8(An,i)
d16(PC)
d8(PC,i)

Flags affected: none

Privileged instruction: no

Pushes the effective address of the source.

RESET RESET EXTERNAL DEVICES

Addressing modes: inherent

Flags affected: none

Privileged instruction: yes

Asserts the reset pin.

ROL ROTATE LEFT

Addressing modes:

a16	(no data qualifier)
a32	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
#b,Dn	(data qualifier used)
Dn,Dn	

Flags affected: N Z V C

Privileged instruction: no

Data size is always word. Rotate is always by one bit when no qualifier is present, or by a count of up to 63. An immediate rotate count can be given of 0 to 7 only. Zero signifies a rotate of eight places. Does not set extend flag.

ROR ROTATE RIGHT

Addressing modes:

a16	(no data qualifier)
a32	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
#b,Dn	(data qualifier used)
Dn,Dn	

Flags affected: N Z V C

Privileged instruction: no

Data size is always word. Rotate is always by one bit when no qualifier is present, or by a count of up to 63. An immediate rotate count can be given of 0 to 7 only. Zero signifies a rotate of eight places. Does not set extend flag.

ROXL ROTATE LEFT THROUGH EXTEND

Addressing modes:

a16	(no data qualifier)
a32	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
#b,Dn	(data qualifier used)
Dn,Dn	

Flags affected: X N Z V C

Privileged instruction: no

Data size is always word. Rotate is always by one bit when no qualifier is present, or by a count of up to 63. An immediate rotate count can be given of 0 to 7 only. Zero signifies a rotate of eight places. Rotates through extend flag.

ROXR ROTATE RIGHT THROUGH EXTEND

Addressing modes:

a16	(no data qualifier)
a32	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	
#b,Dn	(data qualifier used)
Dn,Dn	

Flags affected: X N Z V C

Privileged instruction: no

Data size is always word. Rotate is always by one bit when no qualifier is present, or by a count of up to 63. An immediate rotate count can be given of 0 to 7 only. Zero signifies a rotate of eight places. Rotates through extend flag.

RTE RETURN FROM EXCEPTION

Addressing modes: inherent

Flags affected: X N Z V C

Privileged instruction: yes

Pops status register and program counter.

RTR RETURN AND RESTORE CCR

Addressing modes: inherent

Flags affected: X N Z V C

Privileged instruction: no

Pops condition code register and program counter.

RTS RETURN FROM SUBROUTINE

Addressing modes: inherent

Flags affected: none

Privileged instruction: no

Pops program counter.

SBCD SUBTRACT DECIMAL WITH EXTEND

Addressing modes: -(An),-(An)
Dn,Dn

Flags affected: X N Z V C

Privileged instruction: no

Byte data size only. Subtracts two BCD digits in source, with extend, from two BCD digits in destination.

Scc SET CONDITIONAL

Addressing modes:

a16	
a32	
Dn	
(An)	
(An)+	
-(An)	
d16(An)	
d8(An,i)	

Flags affected: none

Privileged instruction: no

Byte operations only. If condition is true, sets destination byte to $FF, else clears destination byte to zero.

STOP STOP

Addressing modes: #n

Flags affected: X N Z V C

Privileged instruction: yes

Loads status register and stops until interrupt or reset.

SUB SUBTRACT

Addressing modes:

#n	
a16	
a32	
Dn	
(An)	
(An)+	,Dn
-(An)	,An
d16(An)	
d8(An,i)	
d16(PC)	
d8(PC,i)	

	a16
	a32
#n,	(An)
Dn,	(An)+
	-(An)
	d16(An)
	d8(An,i)

An,An
An,Dn

Flags affected: X N Z V C

Privileged instruction: no

Register An may not be used as destination for byte operations. Subtracts source from destination.

SUBQ SUBTRACT QUICK

Addressing modes:

#b, { Dn, An, a16, a32, (An), (An)+, -(An), d16(An), d8(An,i) }

Flags affected: X N Z V C

Privileged instruction: no

Register An may not be used as destination for byte operations. Word and long-word operations are identical. Subtracts data (1 to 8) from destination.

SUBX SUBTRACT WITH EXTEND

Addressing modes: -(An),-(An)
Dn,Dn

Flags affected: X N Z V C

Privileged instruction: no

Subtracts source, with extend, from destination.

SWAP SWAP DATA REGISTER HALVES

Addressing modes: Dn

Flags affected: N Z V C

Privileged instruction: no

Swaps low order word with high order word.

TAS TEST AND SET BIT 7

Addressing modes:
a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flags affected: N Z V C

Privileged instruction: no

Byte operations only. Tests bit 7 of byte (setting N and Z flags) and then sets bit 7 to one.

TRAP TRAP

Addressing modes: #b

Flags affected: none

Privileged instruction: no

Immediate value is vector between 0 and 15. Generates the specified TRAP exception.

TRAPV TRAP ON OVERFLOW

Addressing modes: inherent

Flags affected: none

Privileged instruction: no

If overflow flag (V) is set, generates a TRAPV exception.

TST TEST

Addressing modes: a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flags affected: N Z V C

Privileged instruction: no

Tests the destination. Destination is not altered.

UNLK UNLINK

Addressing modes: An

Flags affected: none

Privileged instruction: no

Stack pointer is loaded from register An. Register An is then loaded from long-word pulled off stack.