Instruction	Syntax	SIZE	Description	Operation	X N Z V C
ABCD	ABCD Dy,Dx	B	Add decimal with extend	[destination] 10 ← [source] 10 + [destination] 10 + [X]	* U * U *
ADD	ADD <ea>,Dn	B,W,L	Add binary	[destination] ← [source] + [destination]	* * * * *
ADDA	ADDA <ea>,An	W,L	Add address	[destination] ← [source] + [destination]	- - - - -
ADDI	ADDI #<data>,<ea>	B,W,L	Add immediate	[destination] ← <literal> + [destination]	* * * * *
ADDQ	ADDQ #6,D3	B,W,L	Add quick	[destination] ← <literal> + [destination]	* * * * *
ADDX	ADDX Dy,Dx	B,W,L	Add extended (ADC)	[destination] ← [source] + [destination] + [X]	* * * * *
AND	AND <ea>,Dn	B,W,L	AND logical	[destination] ← [source].[destination]	- * * 0 0
ANDI	ANDI #<data>,<ea>	B,W,L	AND immediate	[destination] ← <literal>.[destination]	- * * 0 0
ANDI to CCR	ANDI #<data>,CCR	W,L	AND immediate to condition code register	[CCR] ← <data>.[CCR]	* * * * *
ANDI to SR	ANDI #<data>,SR	W,L	AND immediate to status register	IF [S] = 1 THEN [SR] ← <literal>.[SR] ELSE TRAP	* * * * *
ASL	ASL Dx,Dy	B,W,L	Arithmetic shift left	[destination] ← [destination] shifted by <count>	* * * * *
ASR	ASR Dx,Dy	B,W,L	Arithmetic shift right	[destination] ← [destination] shifted by <count>	* * * * *
BCC	BCC <label>	8/16 bit dis	branch on carry clear C	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BCS	BCS <label>	8/16 bit dis	branch on carry set C	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BEQ	BEQ <label>	8/16 bit dis	branch on equal Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BGE	BGE <label>	8/16 bit dis	branch on greater than or equal N.V + N.V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BGT	BGT <label>	8/16 bit dis	branch on greater than N.V.Z + N.V.Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BHI	BHI <label>	8/16 bit dis	branch on higher than C.Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BLE	BLE <label>	8/16 bit dis	branch on less than or equal Z + N.V + N.V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BLS	BLS <label>	8/16 bit dis	branch on lower than or same C + Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BLT	BLT <label>	8/16 bit dis	branch on less than N.V + N.V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BMI	BMI <label>	8/16 bit dis	branch on minus	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BNE	BNE <label>	8/16 bit dis	branch on not equal Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BPL	BPL <label>	8/16 bit dis	branch on plus	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BVC	BVC <label>	8/16 bit dis	branch on overflow clear V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BVS	BVS <label>	8/16 bit dis	branch on overflow set V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *
BCHG	BCHG Dn,<ea>	B,L	Test a bit and change	[Z] ← <bit number> OF [destination]	- - * - -
BCLR	BCLR Dn,<ea>	B,L	Test a bit and clear (RES)	[Z] ← <bit number> OF [destination]	- - * - -
BRA	BRA <label>	8/16 bit dis	Branch always	[PC] ← [PC] + d	- - - - -
BSET	BSET Dn,<ea>	B,L	Test a bit and set (SET)	[Z] ← <bit number> OF [destination]	- - * - -
BSR	BSR <label>	8/16 bit dis	Branch to subroutine (CALL relative addr)	[SP] ← [SP] - 4; [M([SP])] ← [PC]; [PC] ← [PC] + d	- - - - -
BTST	BTST Dn,<ea>	B,L	Test a bit	[Z] ← <bit number> OF [destination]	- - * - -
CHK	CHK <ea>,Dn	B,W,L	Check register against bounds (Trap 6 if failed)	IF [Dn] < 0 OR [Dn] > [<ea>] THEN TRAP	- * U U U
CLR	CLR <ea>	B,W,L	Clear an operand	[destination] ← 0	- 0 1 0 0
CMP	CMP <ea>,Dn	B,W,L	Compare	[destination] - [source]	- * * * *
CMPA	CMPA <ea>,An	W,L	Compare address	[destination] - [source]	
CMPI	CMPI #<data>,<ea>	B,W,L	Compare immediate	[destination] - <immediate data>	- * * * *
CMPM	CMPM (Ay)+,(Ax)+	B,W,L	Compare memory with memory	[destination] - [source]	- * * * *
Dbcc	DBcc Dn,<label>	8/16 bit dis	Test condition cc, decrement, and branch	IF(condition false) THEN [Dn] ← [Dn] - 1 {decrement loop counter}	- - - - -
DIVS	DIVS <ea>,Dn	W	Signed divide	[destination] ← [destination]/[source]	- * * * 0
DIVU	DIVU <ea>,Dn	W	unsigned divide	[destination] ← [destination]/[source]	- * * * 0
EOR	EOR Dn,<ea>	B,W,L	Exclusive OR logical (XOR)	[destination] ← [source] ⊕ [destination]	- * * 0 0
EORI	EORI #<data>,<ea>	B,W,L	EOR immediate	[destination] ← <literal> ⊕ [destination]	- * * 0 0
EORI to CCR	EORI #<data>,CCR	W	EOR immediate to CCR	[CCR] ← <literal> ⊕ [CCR]	* * * * *
EXG	EXG Rx,Ry	L	Exchange registers (EXX)	[Rx] ← [Ry]; [Ry] ← [Rx]	- - - - -
EXT	EXT.W Dn	W,L	Sign-extend a data register	[destination] ← sign-extended[destination]	- * * 0 0
ILLEGAL	ILLEGAL		Illegal instruction	[SSP] ← [SSP] - 4; [M([SSP])] ← [PC]; [SSP] ← [SSP] - 2; [M([SSP])] ← [SR];	- - - - -
JMP	JMP <ea>		Jump (unconditionally)	[PC] ← destination	- - - - -
JSR	JSR <ea>		Jump to subroutine (CALL)	[SP] ← [SP] - 4; [M([SP])] ← [PC]	- - - - -
LEA	LEA <ea>,An	L	Load effective address	[An] ← <ea>	- - - - -
LINK	LINK An,#<displacement>	W,L	Link and allocate ram from stackpointer	[SP] ← [SP] - 4; [M([SP])] ← [An];	- - - - -
LSL	LSL Dx,Dy	B,W,L	Logical shift left	[destination] ← [destination] shifted by <count>	* * * 0 *
LSR	LSR #<data>,Dy	B,W,L	Logical shift right	[destination] ← [destination] shifted by <count>	
MOVE	MOVE <ea>,<e>	B,W,L	Copy data from source to destination	[destination] ← [source]	- * * 0 0
MOVEA	MOVEA <ea>,An	W,L	Move address	[An] ← [source]	- - - - -
MOVE to CCR	MOVE <ea>,CCR	W	Copy data to CCR from source	[CCR] ← [source]	* * * * *
MOVE from SR	MOVE SR,<ea>	W	Copy data from SR to destination	[destination] ← [SR]	- - - - -
MOVE to SR	MOVE <ea>,SR	W	Copy data to SR from source	IF [S] = 1 THEN [SR] ← [source] ELSE TRAP	* * * * *
MOVE USP	MOVE USP,An	L	Copy data to or from USP	MOVE USP,An form: IF [S] = 1 THEN [USP] ← [An] ELSE TRAP	- - - - -
MOVEM	MOVEM <ea>,<register list>	W,L	Move multiple registers	REPEAT [destination_register] ← [source] UNTIL all registers in list moved	- - - - -
MOVEP	MOVEP Dx,(d,Ay)	W,L	Move peripheral data	[destination] ← [source]	- - - - -
MOVEQ	MOVEQ #<data>,Dn	L	Move quick (copy a small literal to a destination)	[destination] ← <literal>	- * * 0 0
MULS	MULS <ea>,Dn	W,L	Signed multiply	[destination] ← [destination] * [source]	- * * 0 0
MULU	MULU <ea>,Dn	W,L	unsigned multiply	[destination] ← [destination] * [source]	- * * 0 0
NBCD	NBCD <ea>	B	Negate decimal with sign extend	[destination] 10 ← 0 − [destination] 10 - [X]	* U * U *
NEG	NEG <ea>	B,W,L	Negate	[destination] ← 0 - [destination]	* * * * *
NEGX	NEGX <ea>	B,W,L	Negate with extend	[destination] ← 0 - [destination] - [X]	* * * * *
NOP	NOP		No operation	None	- - - - -
NOT	NOT <ea>	B,W,L	Logical complement	[destination] ← [destination]	
OR	OR <ea>,Dn	B,W,L	OR logical	[destination] ← [source] + [destination]	- * * 0 0
ORI	ORI #<data>,<ea>	B,W,L	OR immediate	[destination] ← <literal> + [destination]	- * * 0 0
ORI to CCR	ORI #<data>,CCR	W	Inclusive OR immediate to CCR	[CCR] ← <literal> + [CCR]	* * * * *
ORI to SR	ORI #<data>,SR	W	Inclusive OR immediate to status register	IF [S] = 1 THEN [SR] ← <literal> + [SR] ELSE TRAP	* * * * *
PEA	PEA <ea>	L	Push effective address	[SP] ← [SP] - 4; [M([SP])] ← <ea>	- - - - -
RESET	RESET		Reset external devices	IF [S] = 1 THEN Assert RESET* line ELSE TRAP	- - - - -
ROL	ROL Dx,Dy	B,W,L	Rotate left	[destination] ← [destination] rotated by <count>	- * * 0 *
ROR	ROR #<data>,Dy	B,W,L	Rotate right	[destination] ← [destination] rotated by <count>	- * * 0 *
ROXL	ROXL Dx,Dy	B,W,L	Rotate left with extend	[destination] ← [destination] rotated by <count>	* * * 0 *
ROXR	ROXR #<data>,Dy	B,W,L	Rotate right with extend	[destination] ← [destination] rotated by <count>	* * * 0 *
RTE	RTE		Return from exception	IF [S] = 1 THEN [SR] ← [M([SP])]; [SP] ← [SP] + 2	* * * * *
RTR	RTR		Return and restore condition codes	[CCR] ← [M([SP])]; [SP] ← [SP] + 2	* * * * *
RTS	RTS		Return from subroutine (RET)	[PC] ← [M([SP])]; [SP] ← [SP] + 4	- - - - -
SBCD	SBCD Dy,Dx	B	Subtract decimal with extend	[destination] 10 ← [destination] 10 - [source] 10 - [X]	* U * U *
Scc			Set according to condition cc		
SF	SF <ea>		set on false (i.e. set never) 0	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 16	- - - - -
ST	ST <ea>		set on true (i.e. set always) 1	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 17	- - - - -
STOP	STOP #<data>		Load status register and stop	IF [S] = 1 THEN [SR] ← <data> STOP ELSE TRAP	* * * * *
SUB	SUB <ea>,Dn	B,W,L	Subtract binary	[destination] ← [destination] - [source]	* * * * *
SUBA	SUBA <ea>,An	W,L	Subtract address	[destination] ← [destination] - [source]	- - - - -
SUBI	SUBI #<data>,<ea>	B,W,L	Subtract immediate	[destination] ← [destination] - <literal>	* * * * *
SUBQ	SUBQ #<data>,<ea>	B,W,L	Subtract quick	[destination] ← [destination] - <literal>	* * * * *
SUBX	SUBX Dx,Dy	B,W,L	Subtract extended (SBC)	[destination] ← [destination] - [source] - [X]	* * * * *
SWAP	SWAP Dn	W	Swap register halves	[Register(16:31)] ← [Register(0:15)];	- * * 0 0
TAS	TAS <ea>	B	Test and set an operand	[CCR] ← tested([operand]); [destination(7)] ← 1	- * * 0 0
TRAP	TRAP #<vector>		Trap to Vectors 32-47 at address (32+Vector)*4	S ← 1; [SSP] ← [SSP] - 4; [M([SSP])] ← [PC];	- - - - -
TRAPV	TRAPV		Trap on overflow to Trap 7	IF V = 1 THEN: [SSP] ← [SSP] - 4; [M([SSP])] ← [PC];	- - - - -
TST	TST <ea>	B,W,L	Test an operand	[CCR] ← tested([operand])	- * * 0 0
UNLK	Unlink		UNLK An	[SP] ← [An]; [An] ← [M([SP])]; [SP] ← [SP] + 4	- - - - -

Instruction	Syntax	SIZE	Description	Operation	X N Z V C	Sample		
ABCD	ABCD Dy,Dx ABCD -(Ay),-(Ax)	B B	Add decimal with extend	[destination] 10 ← [source] 10 + [destination] 10 + [X]	* U * U *			
ADD	ADD <ea>,Dn ADD Dn,<ea>	B,W,L B,W,L	Add binary	[destination] ← [source] + [destination]	* * * * *			
ADDA	ADDA <ea>,An	W,L	Add address	[destination] ← [source] + [destination]	- - - - -			
ADDI	ADDI #<data>,<ea>	B,W,L	Add immediate	[destination] ← <literal> + [destination]	* * * * *			
ADDQ	ADDQ #6,D3	B,W,L	Add quick	[destination] ← <literal> + [destination]	* * * * *	ADDQ #6,D3		
ADDX	ADDX Dy,Dx ADDX -(Ay),-(Ax)	B,W,L B,W,L	Add extended (ADC)	[destination] ← [source] + [destination] + [X]	* * * * *			
AND	AND <ea>,Dn AND Dn,<ea>	B,W,L B,W,L	AND logical	[destination] ← [source].[destination]	- * * 0 0			
ANDI	ANDI #<data>,<ea>	B,W,L	AND immediate	[destination] ← <literal>.[destination]	- * * 0 0			
ANDI to CCR	ANDI #<data>,CCR	W,L	AND immediate to condition code register	[CCR] ← <data>.[CCR]	* * * * *			
ANDI to SR	ANDI #<data>,SR	W,L	AND immediate to status register	IF [S] = 1 THEN [SR] ← <literal>.[SR] ELSE TRAP	* * * * *			
ASL	ASL Dx,Dy ASL #<data>,Dy ASL <ea>	B,W,L B,W,L W	Arithmetic shift left	[destination] ← [destination] shifted by <count>	* * * * *			
ASR	ASR Dx,Dy ASR #<data>,Dy ASR <ea>	B,W,L B,W,L L	Arithmetic shift right	[destination] ← [destination] shifted by <count>	* * * * *			
BCC	BCC <label>	8/16 bit disp	branch on carry clear C	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BCS	BCS <label>	8/16 bit disp	branch on carry set C	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BEQ	BEQ <label>	8/16 bit disp	branch on equal Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *	BEQ Loop_4		
BGE	BGE <label>	8/16 bit disp	branch on greater than or equal N.V + N.V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BGT	BGT <label>	8/16 bit disp	branch on greater than N.V.Z + N.V.Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BHI	BHI <label>	8/16 bit disp	branch on higher than C.Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BLE	BLE <label>	8/16 bit disp	branch on less than or equal Z + N.V + N.V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BLS	BLS <label>	8/16 bit disp	branch on lower than or same C + Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BLT	BLT <label>	8/16 bit disp	branch on less than N.V + N.V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BMI	BMI <label>	8/16 bit disp	branch on minus (i.e.	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BNE	BNE <label>	8/16 bit disp	branch on not equal Z	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BPL	BPL <label>	8/16 bit disp	branch on plus (i.e.	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BVC	BVC <label>	8/16 bit disp	branch on overflow clear V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *	BVC *+8		
BVS	BVS <label>	8/16 bit disp	branch on overflow set V	If cc = 1 THEN [PC] ← [PC] + d	* * * * *			
BCHG	BCHG Dn,<ea> BCHG #<data>,<ea>	B,L B,L	Test a bit and change	[Z] ← <bit number> OF [destination] <bit number> OF [destination] ← <bit number> OF [destination]	- - * - -			
BCLR	BCLR Dn,<ea> BCLR #<data>,<ea>	B,L B,L	Test a bit and clear (RES)	[Z] ← <bit number> OF [destination] <bit number> OF [destination] ← 0	- - * - -			
BRA	BRA <label> BRA <literal>	8/16 bit disp 8/16 bit disp	Branch always	[PC] ← [PC] + d	- - - - -			
BSET	BSET Dn,<ea> BSET #<data>,<ea>	B,L B,L	Test a bit and set (SET)	[Z] ← <bit number> OF [destination] <bit number> OF [destination] ← 1	- - * - -			
BSR	BSR <label> BSR <literal>	8/16 bit disp 8/16 bit disp	Branch to subroutine (CALL relative addr)	[SP] ← [SP] - 4; [M([SP])] ← [PC]; [PC] ← [PC] + d	- - - - -			
BTST	BTST Dn,<ea> BTST #<data>,<ea>	B,L B,L	Test a bit	[Z] ← <bit number> OF [destination]	- - * - -			
CHK	CHK <ea>,Dn	B,W,L	Check register against bounds (Trap 6 if failed)	IF [Dn] < 0 OR [Dn] > [<ea>] THEN TRAP	- * U U U			
CLR	CLR <ea>	B,W,L	Clear an operand	[destination] ← 0	- 0 1 0 0	CLR (A4)+		
CMP	CMP <ea>,Dn	B,W,L	Compare	[destination] - [source]	- * * * *	CMP (Test,A6,D3.W),D2		
CMPA	CMPA <ea>,An	W,L	Compare address	[destination] - [source]		CMPA.L #$1000,A4	CMPA.W (A2)+,A6	CMPA.L D5,A2
CMPI	CMPI #<data>,<ea>	B,W,L	Compare immediate	[destination] - <immediate data>	- * * * *			
CMPM	CMPM (Ay)+,(Ax)+	B,W,L	Compare memory with memory	[destination] - [source]	- * * * *	CMPM.B (A3)+,(A4)+		

DBcc	DBcc Dn,<label>	8/16 bit disp	Test condition, decrement, and branch	IF(condition false)
				THEN [Dn] ← [Dn] - 1 {decrement loop counter}
				IF [Dn] = -1 THEN [PC] ← [PC] + 2 {fall through to next instruction}
				ELSE [PC] ← [PC] + d {take branch}
				ELSE [PC] ← [PC] + 2 {fall through to next instruction}
					- - - - -	DBEQ D0,AGAIN		

DIVS	DIVS <ea>,Dn	W	Signed divide	[destination] ← [destination]/[source]	- * * * 0			
DIVU	DIVU <ea>,Dn	W	unsigned divide	[destination] ← [destination]/[source]	- * * * 0			
EOR	EOR Dn,<ea>	B,W,L	Exclusive OR logical (XOR)	[destination] ← [source] ⊕ [destination]	- * * 0 0	EOR D3,-(A3)		
EORI	EORI #<data>,<ea>	B,W,L	EOR immediate	[destination] ← <literal> ⊕ [destination]	- * * 0 0			
EORI to CCR	EORI #<data>,CCR	W	EOR immediate to CCR	[CCR] ← <literal> ⊕ [CCR]	* * * * *			
EXG	EXG Rx,Ry	L	Exchange registers (EXX)	[Rx] ← [Ry]; [Ry] ← [Rx]	- - - - -	EXG D3,D4	EXG D2,A0	EXG A7,D5
EXT	EXT.W Dn EXT.L Dn	W,L	Sign-extend a data register	[destination] ← sign-extended[destination]	- * * 0 0			

ILLEGAL	ILLEGAL		Illegal instruction	[SSP] ← [SSP] - 4; [M([SSP])] ← [PC];
				[SSP] ← [SSP] - 2; [M([SSP])] ← [SR];
				[PC] ← Illegal instruction vector
					- - - - -			

JMP	JMP <ea>		Jump (unconditionally)	[PC] ← destination	- - - - -			
JSR	JSR <ea>		Jump to subroutine (CALL)	[SP] ← [SP] - 4; [M([SP])] ← [PC] [PC] ← destination	- - - - -	JSR (Ai) JSR (Ai,Dj)		
LEA	LEA <ea>,An	L	Load effective address	[An] ← <ea>	- - - - -	LEA Table,A0 LEA (Table,PC),A0	LEA (-6,A0,D0.L),A6 LEA (Table,PC,D0),A6	
LINK	LINK An,#<displacement>	W,L	Link and allocate ram from stackpointer	[SP] ← [SP] - 4; [M([SP])] ← [An]; [An] ← [SP]; [SP] ← [SP] + d	- - - - -	LINK A6,#-12		
LSL	LSL Dx,Dy LSL #<data>,Dy LSL <ea>	B,W,L B,W,L W	Logical shift left	[destination] ← [destination] shifted by <count>	* * * 0 *			
LSR	LSR #<data>,Dy LSR Dx,Dy LSR <ea>	B,W,L B,W,L W	Logical shift right	[destination] ← [destination] shifted by <count>				
MOVE	MOVE <ea>,<e>	B,W,L	Copy data from source to destination	[destination] ← [source]	- * * 0 0	MOVE (A5),-(A2) MOVE -(A5),(A2)+	MOVE #$123,(A6)+ MOVE Temp1,Temp2	
MOVEA	MOVEA <ea>,An	W,L	Move address	[An] ← [source]	- - - - -	MOVEA.L #$1234,A0		
MOVE to CCR	MOVE <ea>,CCR	W	Copy data to CCR from source	[CCR] ← [source]	* * * * *			
MOVE from SR	MOVE SR,<ea>	W	Copy data from SR to destination	[destination] ← [SR]	- - - - -			
MOVE to SR	MOVE <ea>,SR	W	Copy data to SR from source	IF [S] = 1 THEN [SR] ← [source] ELSE TRAP	* * * * *			
MOVE USP	MOVE USP,An MOVE An,USP	L L	Copy data to or from USP	MOVE USP,An form: IF [S] = 1 THEN [USP] ← [An] ELSE TRAP MOVE An,USP form: IF [S] = 1 THEN [An] ← [USP] ELSE TRAP	- - - - -			
MOVEM	MOVEM <ea>,<register list> MOVEM <register list>,<ea>	W,L W,L	Move multiple registers	REPEAT [destination_register] ← [source] UNTIL all registers in list moved REPEAT [destination] ← [source_register] UNTIL all registers in list moved	- - - - -	MOVEM.L D0-D5/A0-A3,-(SP) MOVEM.L (SP)+,D0-D5/A0-A3		
MOVEP	MOVEP Dx,(d,Ay) MOVEP (d,Ay),Dx	W,L W,L	Move peripheral data	[destination] ← [source]	- - - - -	MOVEP D3,(Control,A0) MOVEP (Input,A6),D5		
MOVEQ	MOVEQ #<data>,Dn	L	Move quick (copy a small literal to a destination)	[destination] ← <literal>	- * * 0 0	MOVEQ #12,D0		
MULS	MULS <ea>,Dn	W,L	Signed multiply	[destination] ← [destination] * [source]	- * * 0 0			
MULU	MULU <ea>,Dn	W,L	unsigned multiply	[destination] ← [destination] * [source]	- * * 0 0	MULU #$1234,D3		
NBCD	NBCD <ea>	B	Negate decimal with sign extend	[destination] 10 ← 0 − [destination] 10 - [X]	* U * U *			
NEG	NEG <ea>	B,W,L	Negate	[destination] ← 0 - [destination]	* * * * *			
NEGX	NEGX <ea>	B,W,L	Negate with extend	[destination] ← 0 - [destination] - [X]	* * * * *			
NOP	NOP		No operation	None	- - - - -			
NOT	NOT <ea>	B,W,L	Logical complement	[destination] ← [destination]				
OR	OR <ea>,Dn OR Dn,<ea>	B,W,L B,W,L	OR logical	[destination] ← [source] + [destination]	- * * 0 0	OR.L #$F0000000,D0		
ORI	ORI #<data>,<ea>	B,W,L	OR immediate	[destination] ← <literal> + [destination]	- * * 0 0	ORI.B #%00000011,(A0)+		
ORI to CCR	ORI #<data>,CCR	W	Inclusive OR immediate to CCR	[CCR] ← <literal> + [CCR]	* * * * *	ORI #$04,CCR		
ORI to SR	ORI #<data>,SR	W	Inclusive OR immediate to status register	IF [S] = 1 THEN [SR] ← <literal> + [SR] ELSE TRAP	* * * * *	ORI #$8000,SR		
PEA	PEA <ea>	L	Push effective address	[SP] ← [SP] - 4; [M([SP])] ← <ea>	- - - - -	PEA (TABLE,PC)		
RESET	RESET		Reset external devices	IF [S] = 1 THEN Assert RESET* line ELSE TRAP	- - - - -			
ROL	ROL Dx,Dy ROL #<data>,Dy ROL <ea>	B,W,L B,W,L W	Rotate left	[destination] ← [destination] rotated by <count>	- * * 0 *			
ROR	ROR #<data>,Dy ROR Dx,Dy ROR <ea>	B,W,L B,W,L W	Rotate right	[destination] ← [destination] rotated by <count>	- * * 0 *			
ROXL	ROXL Dx,Dy ROXL #<data>,Dy ROXL <ea>	B,W,L B,W,L W	Rotate left with extend	[destination] ← [destination] rotated by <count>	* * * 0 *			
ROXR	ROXR #<data>,Dy ROXR Dx,Dy ROXR <ea>	B,W,L B,W,L W	Rotate right with extend	[destination] ← [destination] rotated by <count>	* * * 0 *			

RTE	RTE		Return from exception	IF [S] = 1 THEN
				[SR] ← [M([SP])]; [SP] ← [SP] + 2
				[PC] ← [M([SP])]; [SP] ← [SP] + 4
				ELSE TRAP
					* * * * *			

RTR	RTR		Return and restore condition codes	[CCR] ← [M([SP])]; [SP] ← [SP] + 2 [PC] ← [M([SP])]; [SP] ← [SP] + 4	* * * * *			
RTS	RTS		Return from subroutine (RET)	[PC] ← [M([SP])]; [SP] ← [SP] + 4	- - - - -			
SBCD	SBCD Dy,Dx SBCD -(Ay),-(Ax)	B B	Subtract decimal with extend	[destination] 10 ← [destination] 10 - [source] 10 - [X]	* U * U *			
Scc			Set according to condition cc					

SCC	SCC <ea>		set on carry clear C	IF cc = 1 THEN [destination] ← 11111111 2
				ELSE [destination] ← 00000000 2
					- - - - -			

SCS	SCS <ea>		set on carry set C	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 3	- - - - -			
SEQ	SEQ <ea>		set on equal Z	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 4	- - - - -			
SGE	SGE <ea>		set on greater than or equal N.V + N.V	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 5	- - - - -			
SGT	SGT <ea>		set on greater than N.V.Z + N.V.Z	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 6	- - - - -			
SHI	SHI <ea>		set on higher than C.Z	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 7	- - - - -			
SLE	SLE <ea>		set on less than or equal Z + N.V + N.V	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 8	- - - - -			
SLS	SLS <ea>		set on lower than or same C + Z	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 9	- - - - -			
SLT	SLT <ea>		set on less than N.V + N.V	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 10	- - - - -			
SMI	SMI <ea>		set on minus (i.e. negative) N	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 11	- - - - -			
SNE	SNE <ea>		set on not equal Z	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 12	- - - - -			
SPL	SPL <ea>		set on plus (i.e. positive) N	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 13	- - - - -			
SVC	SVC <ea>		set on overflow clear V	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 14	- - - - -			
SVS	SVS <ea>		set on overflow set V	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 15	- - - - -			
SF	SF <ea>		set on false (i.e. set never) 0	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 16	- - - - -			
ST	ST <ea>		set on true (i.e. set always) 1	IF cc = 1 THEN [destination] ← 11111111 2ELSE [destination] ← 00000000 17	- - - - -			
STOP	STOP #<data>		Load status register and stop	IF [S] = 1 THEN [SR] ← <data> STOP ELSE TRAP	* * * * *	STOP #$2700 STOP #SetUp		
SUB	SUB <ea>,Dn SUB Dn,<ea>	B,W,L B,W,L	Subtract binary	[destination] ← [destination] - [source]	* * * * *			
SUBA	SUBA <ea>,An	W,L	Subtract address	[destination] ← [destination] - [source]	- - - - -			
SUBI	SUBI #<data>,<ea>	B,W,L	Subtract immediate	[destination] ← [destination] - <literal>	* * * * *			
SUBQ	SUBQ #<data>,<ea>	B,W,L	Subtract quick	[destination] ← [destination] - <literal>	* * * * *			
SUBX	SUBX Dx,Dy SUBX -(Ax),-(Ay)	B,W,L B,W,L	Subtract extended (SBC)	[destination] ← [destination] - [source] - [X]	* * * * *			
SWAP	SWAP Dn	W	Swap register halves	[Register(16:31)] ← [Register(0:15)]; [Register(0:15)] ← [Register(16:31]	- * * 0 0			
TAS	TAS <ea>	B	Test and set an operand	[CCR] ← tested([operand]); [destination(7)] ← 1	- * * 0 0			

TRAP	TRAP #<vector>		Trap to Vectors 32-47 at address (32+Vector)*4	S ← 1;
				[SSP] ← [SSP] - 4; [M([SSP])] ← [PC];
				[SSP] ← [SSP] - 2; [M([SSP])] ← [SR];
				[PC] ← vector
					- - - - -	TRAP #15		

TRAPV	TRAPV		Trap on overflow to Trap 7	IF V = 1 THEN:
				[SSP] ← [SSP] - 4; [M([SSP])] ← [PC];
				[SSP] ← [SSP] - 2; [M([SSP])] ← [SR];
				[PC] ← [M($01C)]
				ELSE no action
					- - - - -			

TST	TST <ea>	B,W,L	Test an operand	[CCR] ← tested([operand]) i.e., [operand] - 0; update CCR	- * * 0 0			
UNLK	Unlink		UNLK An	[SP] ← [An]; [An] ← [M([SP])]; [SP] ← [SP] + 4	- - - - -			

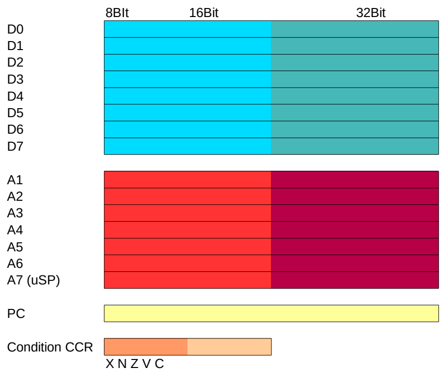
Dn	An Data and address register direct.
(An)	Address register indirect.
(An)+, -(An)	Address register indirect with post-incrementing or pre-decrementing.
(d,An), (d,An,Xi)	Address register indirect with displacement, and address register indirect with indexing and a displacement.
ABS.W, ABS.L	Absolute addressing with a 16-bit or a 32-bit address.
(d,PC), (d,PC,Xi)	Program counter relative addressing with a 16-bit offset,or with an 8-bit offset plus the contents of an index register.
imm	An immediate value (i.e., literal) which may be 16 or 32 bits, depending on the instruction.

Old notation Current notation
d(An), d(An,Xi) (d,An), (d,An,Xi)
d(PC), d(PC,Xi) (d,PC), (d,PC,Xi)

Data stored Big Endian

<ea> - effective address
<rea> - register effective address
<dea> - data effective address
<mea> - Memory effective address
<cea> - control effective address
<aea> - alterable effective address (data or memory)

	MODE	MODE	ea	rea	dea	mea	cea	aea
Data Register Direct	Dn	Dn	X	X	X			X
Address Register Direct	An	An	X	X				X
Address Register Indirect	(An)	(An)	X		X	X	X	X
Address Register Indirect with Postincrement	(An)+	(An)+	X		X	X		X
Address Register Indirect with Predecrement	-(An)	-(An)	X		X	X		X
Address Register Indirect with Displacement	d(An)	(d,An)	X		X	X	X	X
Address Register Indirect with Index	d(An,Xn)	(d,An,Xn)	X		X	X	X	X
Absolute Short	A16	A16	X		X	X	X	X
Absolute Long	A32	A32	X		X	X	X	X
Program Counter with Displacement	d(PC)	(d,PC)	X		X	X	X	
Program Counter with Index	d(PC,Xn)	(d,PC,Xn)	X		X	X	X	
Immediate	#<data>	#<data>	X		X	X		



CCR Bit		
X	Extend	Set to the value of the C-bit for arithmetic operations; otherwise not affected or set to a specified result.
N	Negative	Set if the most significant bit of the result is set; otherwise clear.
Z	Zero	Set if the result equals zero; otherwise clear.
V	Overflow	Set if an arithmetic overflow occurs implying that the result cannot be represented in the operand size; otherwise clear.
C	Carry	Set if a carry out of the most significant bit of the operand occurs for an addition, or if a borrow occurs in a subtraction; otherwise clear.

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

IS-I/IS Memory Indirect Action Encodings

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

EFFECTIVE ADDRESSING MODES

Mode	Generation	Syntax	Mode Field	Reg Field	Ext Words
Data Register Direct Mode	EA = Dn	Dn	000	REG. NO.	0
Address Register Direct Mode	EA = An	An	001	REG. NO.	0
Address Register Indirect Mode	EA = (An)	(An)	010	REG. NO.	0
Address Register Indirect with Postincrement Mode	EA = (An) + S	(An) +	011	REG. NO.	0
Address Register Indirect with Predecrement Mode	EA = (An)–SIZ	– (An)	100	REG. NO.	0
Address Register Indirect with Displacement Mode	EA = (An) + d	(d16, An)	101	REG. NO.	1
Address Register Indirect with Index (8-Bit Displacement) Mode	EA = (An) + (X	(d8 ,An, Xn.S	110	REG. NO.	1

Vector	Dec	Addressx	Space	Assignment
0	0	000	SP	Reset: Initial SSP?
1	4	004	SP	Reset: Initial PC2
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace (every instruction calls this if T flag is set)
10	40	028	SD	Axxx – Line 1010 Emulator
11	44	02C	SD	Fxxx – Line 1111 Emulator
12	48	030	SD	(Unassigned. Reserved)
13	52	034	SD	(Unassigned, Reserved)
14	56	038	SD	Format Error**
15	60	03C	SD	Uninitialized Interrupt Vector
16-23'	64	040	SD	(Unassigned, Reserved)
95		05F		-
24	96	060	SD	Spurious Interrupt3
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SO	Level 2 Interrupt Autovector
27	108	06C	SO	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	191	0BF		
48-631	192	0C0	SD	(Unassigned, Reserved)
	256	0FF		-
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		-