MICRO LOGIC CORP.

HACKENSACK, NJ

68000

PROGRAMMER'S INSTANT REFERENCE CARD

MICRO® CHART

HOW TO USE THIS MICRO CHART

The INSTRUCTION SET section describes each instruction and gives its addressing modes, assembler syntax, size, execution time, and effect on the flags.

The OPERANDS AND ADDRESSING section has general information on operand sizes, data organization in memory and registers, addressing modes, stacks, and queues.

The EXCEPTION PROCESSING section explains the 68000's response to errors, traps, interrupts, and other unusual conditions and its use of reserved memory locations.

The PINOUTS section lists the IC package pin numbers and signal names.

The ABBREVIATIONS section defines abbreviations used throughout this Micro Chart.

ABBREVIATIONS

* = Active low signal name suffix, or boolean inversion
$ = Hexadecimal
Ad = Destination Address register (A0-A7)
An = Address register (A0-A7)
As = Source Address register (A0-A7)
addr = address
addr.L = 32-bit absolute address
addr.W = 16-bit absolute address
B = Operand size is byte
BW = Operand size is byte or word
BWL = Operand size is byte, word, or long word
C = Carry flag in CCR
cc = Any of the sixteen condition codes: CC, CS, EQ, F, GE, GT, HI, LE, LS, LT, MI, NE, PL, T, VC, or VS
CCR = Condition Code register
CLKs = Execution time of instruction in CLK cycles
da = Immediate data
da3 = 3-bit immediate data
da4 = 4-bit immediate vector number
da8 = Immediate data byte
da16 = Immediate data word
da32 = Immediate data long word
Dd = Destination Data register (D0-D7)
di8 = 8-bit displacement
di16 = 16-bit displacement
Dn = Data register (D0-D7)
Ds = Source Data register (D0-D7)
dst = Destination operand
ea = Effective address
Hex = Hexadecimal
III = Interrupt mask (Bits 10,9,8) in SR
L = Operand size is long word
LSB = Least significant (low order) bit; Bit 0
MSB = Most significant (high order) bit
N = Negative flag in CCR
PC = Program Counter register
PI = Privileged instuction
Rd = Destination register (A0-A7 or D0-D7)
Reads = Number of read bus cycles during instruction execution
reg = Register
regs = Registers
Rs = Source register (A0-A7 or D0-D7)
S = Supervisor bit (Bit 13) in SR
SP = Stack Pointer register (SSP or USP)
SR = Status register, including CCR
src = Source operand
SSP = Supervisor Stack Pointer register
T = Trace bit (Bit 15) in SR
USP = User Stack Pointer register
V = Overflow flag in CCR
W = Operand size is word
WL = Operand size is word or long word
Words = Length of instruction in words
Writes = Number of write bus cycles during instruction execution
X = Extend flag in CCR
Xn = Index register (A0-A7 or D0-D7)
Z = Zero flag in CCR

OPERANDS AND ADDRESSING

INSTRUCTIONS: 1 to 5 words. Operation, register, length, and sometimes operand are given in first (Operation) word. 0-4 Extension words specify immediate data, source address, and destination address operands in that order; each, if present, is 1-2 words.

REGISTERS: Sixteen 32-bit general purpose registers consisting of eight Data registers (D0-D7) and eight Address registers (A0-A7), one 32-bit Program Counter (PC), and one 16-bit Status Register (SR). The Condition Code register (CCR) is the lower byte of the SR. A7 is the system Stack Pointer. One of two registers, SSP or USP, is used as A7; when one is active, the other is inaccessible; see Supervisor and User States below.

STATUS AND CONDITION CODE REGISTERS:

```
        System Byte   User Byte (CCR)
      ---------------  --------------
Bit: 15             8 7             0
     - - - - - - - - - - - - - - - -
SR:  T 0 S 0 0 I I I 0 0 0 X N Z V C

T: 1 = Trace mode, 0 = execute mode
S: 1 = Supervisor state, 0 = User state
III: Interrupt priority:
      111 = 7 (highest and non-maskable)
      000 = 0 (lowest)
X,N,Z,V,C - See Flags
Other bits are usually zero
```

SUPERVISOR STATE: The CPU is in Supervisor state when S=1. A7 is the SSP. All memory accesses are to the Supervisor memory space. All instructions are allowed. Only these privileged instructions can switch the CPU to User state by clearing the S bit: ANDI to SR, EORI to SR, MOVE to SR, or RTE.

USER STATE: The CPU is in User state when the S=0. A7 is the USP. All memory accesses are to the User memory space. An attempt to execute a Privileged instruction will cause an exception. Only an exception can switch the CPU to the Supervisor state.

OPERANDS

BIT NUMBERS: Low order (least significant) bit is numbered 0.

OPERAND SIZES: Add suffix .B, .W, or .L to instruction mnemonic for Byte (8 bits), Word (16), or Long Word (32). The default size is Word.

DATA REGISTER OPERANDS (D0-D7): can be 1, 8, 16, or 32 bits. Only low order part of register is used or changed for byte and word operands; high order part is not affected. Only one bit is used or changed for bit operations.

ADDRESS REGISTER OPERANDS (A0-A7): If destination, all 32 bits are affected, and SOURCE WORD OPERAND IS SIGN-EXTENDED to 32 bits before operation. If source, all or low order half is used.

INDEX REGISTER (A0-A7 or D0-D7): Any address or data register can be used as a word (Xn.W, sign-extended low order word) or a long word (Xn.L) index register.

MEMORY OPERANDS: can be 1, 8, 16, or 32 bits. 1 byte per address. High order byte of word has same address (always even) as word; low order byte has next higher address (odd). Instructions and multibyte data start on even addresses. Long word at address N has second word at address N+2; second long word is at address N+4. Most significant digit of BCD byte is in high order bits; less significant digits are in bytes at higher addresses. The FC2-FC0 outputs distinguish program references from data references; all writes are data references; all operand reads except PC relative are data references.

FC2	FC1	FC0	Cycle Type
L	L	L	(reserved by Motorola)
L	L	H	User Data
L	H	L	User Program
L	H	H	(reserved for user def.)
H	L	L	(reserved by Motorola)
H	L	H	Supervisor Data
H	H	L	Supervisor Program
H	H	H	Interrupt Acknowledge

The data bus strobes define how the data bus is used:

Data Strobes			Bus Use	
UDS*	LDS*	R/W*	D15-D8	D7-D0
H	H	x	n	n
L	L	H	15-8	7-0
H	L	H	n	7-0
L	H	H	15-8	n
L	L	L	15-8	7-0
H	L	L	7-0m	7-0
L	H	L	15-8	15-8m

* = Active low signal
H = High
L = Low
x = Don't care
n = No valid data
m = Maybe

STACKS AND QUEUES

SYSTEM STACK: A7 is the system Stack Pointer used for subroutine calls. See Operands and Addressing. The stack grows from higher to lower addresses; SP points to last word pushed on stack; SP decrements before push, increments after pop. Any instruction using -(A7) as the destination operand is a push; any instruction using (A7)+ as the source operand is a pop.

USER STACK: To grow from higher address to lower address, use -(Ad) to push, (As)+ to pop; An points to top item. To grow from lower address to higher address, use (Ad)+ to push, -(As) to pop; An points to next free spot.

USER QUEUE: A FIFO list. To grow from lower address to higher address, use (Ad)+ to put, -(As) to get. To grow from higher address to lower address, use -(Ad) to put, (As)+ to get.

ADDRESSING MODES

SOURCE,DESTINATION: Instructions that move data from a source to a destination are written in the form:

```
     mnemonic  src,dst
```

IMPLIED: Operand is in one of these registers: CCR, PC, SR, SP, SSP, or USP.
Example: TRAPV

QUICK IMMEDIATE (_Q #_): 3-bit operand (1 to 8) is in operation word for ADDQ and SUBQ; 8-bit operand (-128 to +127) is in operation word for MOVEQ.
Example: ADDQ #7,D3

IMMEDIATE (#da): Byte operand is in low order byte of extension word; word operand is in extension word; long word operand is in 2 extension words.
Example: ORI.B #$7F,D6

ABSOLUTE SHORT (addr.W): Extension word, sign-extended to 32 bits, is address of operand. Example: ASL VAR6.W

ABSOLUTE LONG (addr.L): Two extension words are 32-bit address of operand.
Example: CLR COUNT.L

PROGRAM COUNTER RELATIVE WITH DISPLACEMENT (di16(PC)): Address of operand is sum of address of extension word and sign-extended displacement in extension word.
Example: LEA LOOKUP(PC),A4

PROGRAM COUNTER RELATIVE WITH INDEX AND DISPLACEMENT (di8(PC,Xn)): Address of operand is sum of address of extension word, contents of index register, and sign-extended displacement in low byte of extension word. Index register can be any Address or Data register.
Example: JMP NEXT(PC,D1.L)

DATA REGISTER DIRECT (Dn): Operand is in data register. Example: CLR.B D0

ADDRESS REGISTER DIRECT (An): Operand is in address register.
Example: CMPA.L D0,A0

ADDRESS REGISTER INDIRECT ((An)): Address of operand is in address register. Example: LSR (A5)

ADDRESS REGISTER INDIRECT WITH PREDECREMENT (-(An)) OR POSTINCREMENT ((An)+): Address of operand is in address register. Address register is decremented before use or incremented after use by 1, 2, or 4 depending on operand size. If size is byte and register is SP, adjustment is by 2, not 1. Examples: TAS -(A1) NEG.B (A6)+

ADDRESS REGISTER INDIRECT WITH DISPLACEMENT (di16(An)): Address of operand is sum of sign-extended extension word and address register contents.
Example: EORI.B #$55,LIGHTS(A2)

ADDRESS REGISTER INDIRECT WITH INDEX AND DISPLACEMENT (di8(An,Xn)):Address of operand is sum of address register contents, index register contents, and sign-extended displacement in LOW BYTE of extension word.
Example: ROL.W BIAS(A0,A1.W)

ASCII

LSD \ MSD	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 0000	NUL	DLE	SP	0	@	P	`	p
1 0001	SOH	DC1	!	1	A	Q	a	q
2 0010	STX	DC2	"	2	B	R	b	r
3 0011	ETX	DC3	#	3	C	S	c	s
4 0100	EOT	DC4	$	4	D	T	d	t
5 0101	ENQ	NAK	%	5	E	U	e	u
6 0110	ACK	SYN	&	6	F	V	f	v
7 0111	BEL	ETB	'	7	G	W	g	w
8 1000	BS	CAN	(	8	H	X	h	x
9 1001	HT	EM	)	9	I	Y	i	y
A 1010	LF	SUB	*	:	J	Z	j	z
B 1011	VT	ESX	+	;	K	[	k	{
C 1100	FF	FS	,	<	L	\	l	|
D 1101	CR	GS	-	=	M	]	m	}
E 1110	SO	RS	.	>	N	^	n	~
F 1111	SI	US	/	?	O	_	o	DEL

EXCEPTION PROCESSING

The CPU's response to unusual internal or external conditions.

EXCEPTION VECTORS:

Number Dec	Number Hex	Addr Hex	Use	CLKs.Reads.Writes
0	00	000000	(Reset SSP; see note below)	
1	01	000004	RESET* 40.6.0	
2	02	000008	Bus Error (BERR*) 50.4.7	
3	03	00000C	Address Error 50.4.7	
4	04	000010	Illegal Instruction 34.4.3	
5	05	000014	Divide by zero 42.5.4	
6	06	000018	CHK operand out of bounds	
7	07	00001C	TRAPV when V=1 34.4.3	
8	08	000020	Privilege violation 34.4.3	
9	09	000024	Trace 34.4.3	
10	0A	000028	Line 1010 emulator 34.4.3	
11	0B	00002C	Line 1111 emulator 34.4.3	
12	0C	000030	(reserved)	
13	0D	000034	(reserved)	
14	0E	000038	(reserved)	
15	0F	00003C	Unitialized irpt 44.5.3	
16	10	000040	(reserved)	
to	to	to		
23	17	00005C	(reserved)	
24	18	000060	Spurious irpt 44.5.3	
25	19	000064	Ext irpt 1 autovector 44.5.3	
26	1A	000068	Ext irpt 2 autovector 44.5.3	
27	1B	00006C	Ext irpt 3 autovector 44.5.3	
28	1C	000070	Ext irpt 4 autovector 44.5.3	
29	1D	000074	Ext irpt 5 autovector 44.5.3	
30	1E	000078	Ext irpt 6 autovector 44.5.3	
31	1F	00007C	Ext irpt 7 autovector 44.5.3	
32	20	000080	TRAP #0 instruction 38.4.4	
to	to	to		
47	2F	0000BC	TRAP #15 instruction 38.4.4	
48	30	0000C0	(reserved)	
to	to	to		
63	3F	0000FC	(reserved)	
64	40	000100	0th User interrupt 44.5.3	
to	to	to		
255	FF	0003FC	191st User interrupt 44.5.3	

Vectors 0 and 1 are in Supervisor Program memory space; all others are in Supervisor Data memory space.

EXCEPTION VECTORS: Each (except 0) holds the long word address of an exception handling routine. Vector 0 is not a vector; it is the value loaded into the SSP after a RESET*.

VECTOR NUMBER: Provided by CPU or external logic. When multiplied by four, gives address of vector.

EXCEPTION PROCESSING TIMES: CLKs is the number of CPU CLK cycles to process the exception and fetch the first two words of the handler routine. Assumes a four CLK interrupt acknowledge bus cycle and no wait states. If CLKs are not shown here, see the Instruction Set section.

EXCEPTION PRIORITIES (Highest to Lowest): Reset; bus error and halt; address error; trace; external (user) interrupts 7 through 1; illegal instruction; privilege violation; trap, check, and divide by zero.

EXCEPTION PROCESSING: All exception processing is done in the Supervisor state including use of the SSP for stacking. Except as noted below, the CPU: 1. Saves SR internally. 2. Forces S=1 and T=0 in SR. 3. Gets the vector number. 4. Pushes the saved SR then the PC onto the stack using the SSP. 5. Loads the PC from the exception vector. 6. Executes ahandler routine. The saved PC is usually the address of the first word of the next instruction.

EXCEPTION DESCRIPTIONS

Listed in order of decreasing priority.

(reserved): Reserved for future use by Motorola; do not use

RESET*: If RESET* and HALT* are BOTH input low, the current bus cycle is aborted, and exception processing begins when they return high. The interrupt mask is set to 7 (III=111), no stacking occurs, and the SSP and PC are loaded from Vectors 0 and 1. No other CPU registers are affected. The CPU outputs RESET* low when it executes the RESET instruction, but no registers are affected.

BUS ERROR: When BERR* is input low, the CPU aborts the current bus cycle and floats the address and data busses. When the BERR* input returns high, the CPU stacks the Program Counter (unpredictable value), the Status Register, and four more words in this order: 1. The first word of the executing instruction; 2. The lower 16 bits of the aborted bus address; 3. The upper 16 bits of the address; 4. Five bits of bus cycle information: Bit 4: 1=read, 0=write; Bit 3 = 0 if the CPU was executing an instruction or processing a TRAP, TRAPV, CHK or divide by zero exception; Bit 3 = 1 if the CPU was processing any other exception; Bits 2-0: FC2-FC0.

When HALT* and BERR* are both input low, the CPU will abort the cycle, then re-run it when BERR* then HALT* return high. If a bus error occurs during bus or address error exception processing or while reading the vector table, the CPU halts.

HALT*: When HALT* is input low (with RESET* and BERR* high), the CPU finishes the current bus cycle, stops, and floats the address and data lines. Bus arbitration operates normally during halt. The CPU will continue when HALT* returns high. The CPU outputs HALT* low when it stops because of double bus fault. Then only a low input on RESET* can restart the CPU. See RESET* and BERR*.

ADDRESS ERROR: When the CPU fetches a word from an odd address, it responds as it does for a bus error. If a bus error occurs during address error exception processing, the CPU halts.

TRACE: When T=1 in the SR, an exception is forced after each instruction executes. An exception caused by an instruction is processed before the Trace exception is.

EXTERNAL INTERRUPTS: External logic encodes a priority level on IPL2*, IPL1*, IPL0* (level sensitive). Level 7 is highest and not maskable. Level 1 is lowest. Level 0 is no interrupt. If the encoded level is 7, or greater than III, the CPU starts exception processing after it completes the current instruction. The CPU sets III to the encoded value when it forces S=1 and T=0 in the SR. The vector number is supplied internally (autovector) if VPA* is low or externally (Interrupt Acknowledge bus cycle) if VPA* is high; if BERR* is low, the Spurious Interrupt vector is used. Uninitialized 68000 support chips give vector number 15.

USER INTERRUPTS: These are external interrupts for which external logic provides an 8-bit vector ($40-$FF) during the Interrupt Acknowledge bus cycle.

ILLEVAL, EMULATOR, AND UNIMPLEMENTED INSTRUCTIONS: Any invalid instruction opcode will cause an exception. Motorola reserves each of these for future definition except as follows. Opcodes $4AFA, $4AFB, and $4AFC will always cause an Illegal Instruction exception; the first two are reserved for Motorola products, and the third is reserved for customer use. An opcode with 1010 ($Axxxx) or 1111 ($Fxxxx) in Bits 15-12 will cause a Line 1010 or Line 1111 Emulator exception, respectively. All other unimplemented opcodes cause an Illegal Instruction exception.

PRIVILEGE VIOLATION: Execution of a privileged instruction (PI) in User state causes a privilege violation exception (ANDI #da16,SR; EORI #da16,SR; MOVE src,SR; MOVE As,USP; MOVE USP,Ad; ORI #da16,SR; RESET; RTE; STOP #da16). The saved PC is the address of the first word of the PI.

TRAP, TRAPV, AND CHK: The TRAP instruction always causes a trap exception, and four bits in the instruction word provide part of the vector number. The TRAPV and CHK instructions cause an exception if certain conditions exist when they execute.

BUS ARBITRATION: determined by the BR*, BG*, and BGACK* signals.

PINOUTS

68000
64-Pin DIP, Top View

```
* means active low.
< and > show direction.   D4= 1 64=D5
= means bidirectional.    D3= 2 63=D6
nc means no connection    D2= 3 62=D7
inside.                   D1= 4 61=D8
                          D0= 5 60=D9
                      AS*< 6 59=D10
                     UDS*< 7 58=D11
                     LDS*< 8 57=D12
                     R/W*< 9 56=D13
                   DTACK*>10 55=D14
                      BD*<11 54=D15
                   BGACK*>12 53-GND
                      BR*>13 52>A23
                     VCC-14 51>A22
                     CLK>15 50>A21
                     GND-16 49-VCC
                    HALT*=17 48>A20
                   RESET*=18 47>A19
                     VMA*<19 46>A18
                        E<20 45>A17
                     VPA*>21 44>A16
                    BERR*>22 43>A15
                    IPL2*>23 42>A14
                    IPL1*>24 41>A13
                    IPL0*>25 40>A12
                      FC2<26 39>A11
                      FC1<27 38>A10
                      FC0<28 37>A9
                        A1<29 36>A8
                        A2<30 35>A7
                        A3<31 34>A6
                        A4<32 33>A5
```


Author: Curtis A. Ingraham
Micro Logic Corp (©1987)
SVG version by RetroParla (2024)

www.retroparla.com

MICRO LOGIC CORP.

HACKENSACK, NJ

68000

PROGRAMMER'S INSTANT REFERENCE CARD

MICRO® CHART

INSTRUCTION SET

This table gives the addressing modes for each instruction. The Cycle and Flag Code column gives a code for operand sizes, instruction length, timing, and affect on flags. Example: Under ADD src,Dd the flag code A applies to all addressing modes; cycle code 1 applies to ADD.B Ds,Dd and ADD.W Ds,Dd; and cycle code 4 applies to ADD.L Ds,Dd. See Flag and Cycle Code tables.

Inst	Oper-ands	Cycle and Flag Code
ABCD	Ds,Dd	B3 C
ABCD	-(As),-(Ad)	B42 C
Add 2-digit BCD numbers plus X		
ADD	src,Dd	A (add)
	Ds	BW1 L4
	As	W1 L4
	(As)	BW6 L22
	(As)+	BW6 L22
	-(As)	BW11 L29
	dl16(As)	BW18 L45
	d18(As,Xn)	BW25 L53
	addr.W	BW18 L45
	addr.L	BW39 L69
	dl16(PC)	BW18 L45
	d18(PC,Xn)	BW25 L53
ADD	Ds,dst	A (add)
	(Ad)	BW15 L51
	(Ad)+	BW15 L51
	-(Ad)	BW23 L65
	dl16(Ad)	BW35 L73
	d18(Ad,Xn)	BW46 L83
	addr.W	BW35 L73
	addr.L	BW59 L90
ADDA	src,Ad	N
	Ds	WL4
	As	WL4
	(As)	W13 L22
	(As)+	W13 L22
	-(As)	W21 L29
	dl16(As)	W33 L45
	d18(As,Xn)	W44 L53
	addr.W	W33 L45
	addr.L	W57 L69
	dl16(PC)	W33 L45
	d18(PC,Xn)	W44 L53
	#da	W18 L38
Add to Ad		
ADDI	#da,dst	A
	Dd	BW8 L38
	(Ad)	BW35 L90
	(Ad)+	BW35 L90
	-(Ad)	BW46 L97
	dl16(Ad)	BW59 L101
	d18(Ad,Xn)	BW70 L105
	addr.W	BW59 L101
	addr.L	BW80 L107
Add immediate		
ADDQ	#da3,dst	A
	Dd	BW1 L4
	Ad	WL4
	(Ad)	BW15 L51
	(Ad)+	BW15 L51
	-(Ad)	BW23 L65
	dl16(Ad)	BW35 L73
	d18(Ad,Xn)	BW46 L83
	addr.W	BW35 L73
	addr.L	BW59 L90
Add quick immediate data (3 bits: 1-8); 1 word instr.		
ADDX	Ds,Dd	BW1 L4 A
	-(As),-(Ad)	BW42 L96
Add operands and X (1)		
AND	(See ADD, but no As,Dd)	Logical AND S
ANDI	(See ADDI)	S
ANDI	#da8,CCR	B52 A
ANDI	#da16,SR	W52 A (PI)
Logical AND immediate		
ASL	Ds,Dd	BW5 L10 A/S
	#da3,Dd	BW5 L10 A/S
	(Ad)	W15 A
	(Ad)+	W15 A
	-(Ad)	W23 A
	dl16(Ad)	W35 A
	d18(Ad,Xn)	W46 A
	addr.W	W35 A
	addr.L	W59 A
Arithmetic shift left memory word by 1 bit or Dd by count in Ds or immediate data (3); zero fill; last bit out goes to C and X; set V=1 if MSB changes, else V=0. If (Ds)=0 or #da3=0, set flags only; flag code is S		
ASR	(See ASL)	B/S
Arithmetic shift right memory word by 1 bit or Dd by count in Ds or immediate data (3); MSB fill; last bit out goes to C and X. If (Ds)=0 or #da3=0, set flags only; flag code is S		
Bcc	di8 cc true 11 N	
	cc false 4	
Bcc	dl16 cc true 12	
	cc false 17	
Branch if cc is true; di8=0 not allowed; cc=T,F not allowed		
BCHG	Ds,dst	V
	Dd	L4
	(Ad)	B15
	(Ad)+	B15
	-(Ad)	B23
	dl16(Ad)	B35
	d18(Ad,Xn)	B46
	addr.W	B35
	addr.L	B59
BCHG	#da8,dst	
	Dd	L17
	(Ad)	B35
	(Ad)+	B35
	-(Ad)	B46
	dl16(Ad)	B59
	d18(Ad,Xn)	B70
	addr.W	B59
	addr.L	B80
Flip bit specified by src in location given by dst; put result bit in Z (2)		
BCLR	(See BCHG, except:)	V
	Ds,Dd	L9
	#da8,Dd	L24
Clear bit specified by src in location given by dst; put complement of original bit in Z (2)		
BRA	di8	11 N
BRA	dl16	12 N
Branch always; di8=0 not allowed		
BSET	(See BCHG)	V
Set bit specified by src in location given by dst; put complement of original bit in Z (2)		
BSR	di8	43 N
BSR	dl16	47 N
Branch (call) to subroutine; push long word address of next instruction using SP; di8=0 not allowed		
BTST	Ds,dst	V
	Dd	L3
	(Ad)	B6
	(Ad)+	B6
	-(Ad)	B11
	dl16(Ad)	B18
	d18(Ad,Xn)	B25
	addr.W	B18
	addr.L	B39
	dl16(PC)	B18
	d18(PC,Xn)	B25
	#da8	B8
BTST	#da8,dst	V
	Dd	L12
	(Ad)	B18
	(Ad)+	B18
	-(Ad)	B25
	dl16(Ad)	B39
	d18(Ad,Xn)	B48
	addr.W	B39
	addr.L	B62
	dl16(PC)	B39
	d18(PC,Xn)	B48
Set Z equal to complement of bit specified by src in location given by dst; no change to dst (2)		
CHK	src,Dd	W
	In-Bounds/Out	
	Ds	W9/W109
	(As)	W21/W110
	(As)+	W21/W110
	-(As)	W28/W112
	dl16(As)	W44/W113
	d18(As,Xn)	W52/W114
	addr.W	W44/W113
	addr.L	W68/W115
	dl16(PC)	W44/W113
	d18(PC,Xn)	W52/W114
	#da16	W25/W111
Check Dd low word against 0 and upper bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-$7FFF); src is 2's complement ($8000-$FFFF) of upper bound		
CLR	Dd	BW1 L3 S
	(Ad)	BW15 L51
	(Ad)+	BW15 L51
	-(Ad)	BW23 L65
	dl16(Ad)	BW35 L73
	d18(Ad,Xn)	BW46 L83
	addr.W	BW35 L73
	addr.L	BW59 L90
Clear operand to 0; set N=0, Z=1 (4)		
CMP	src,Dd	T
	Ds	BW1 L3
	As	W1 L3
	(As)	BW6 L22
	(As)+	BW6 L22
	-(As)	BW11 L28
	dl16(As)	BW18 L45
	d18(As,Xn)	BW25 L53
	addr.W	BW18 L45
	addr.L	BW39 L69
	dl16(PC)	BW18 L45
	d18(PC,Xn)	BW25 L53
Compare Dd to src; subtract src from Dd, set flags, don't change operands; BLT, for example, branches after CMP if Dd is less than src		
CMPA	src,Ad	T
	Ds	WL3
	As	WL3
	(As)	W11 L22
	(As)+	W11 L22
	-(As)	W13 L29
	dl16(As)	W25 L45
	d18(As,Xn)	W33 L53
	addr.W	W25 L45
	addr.L	W48 L69
	dl16(PC)	W25 L45
	d18(PC,Xn)	W33 L53
	#da	W12 L27
Compare Ad to src as CMP (5)		
CMPI	#da,dst	T
	Dd	BW8 L27
	(Ad)	BW18 L58
	(Ad)+	BW18 L58
	-(Ad)	BW25 L69
	dl16(Ad)	BW39 L79
	d18(Ad,Xn)	BW48 L85
	addr.W	BW39 L79
	addr.L	BW62 L93
Compare dst to immediate data as CMP (dst - immediate data)		
CMPM	(As)+,(Ad)+	
	BW14 L50	T
Compare 2 memory operands as CMP		
DBcc	Dn,dl16	N
	cc true 17	
	cc false & branch 12	
	cc false & no br. 25	
Decrement and branch until cc is true; 1)if cc is true, go to next instruction; 2)if cc is false, decrement low word of Ds; 3)if Ds is -1, go to next instruction; 4)if Ds is not -1, branch (loop); DBRA same as DBF		
DIVS	src,Dd	T
	Ds	W131
	(As)	W132
	(As)+	W132
	-(As)	W134
	dl16(As)	W135
	d18(As,Xn)	W136
	addr.W	W135
	addr.L	W137
	dl16(PC)	W135
	d18(PC,Xn)	W136
	#da16	W133
Divide signed long Dd by signed word src, put signed quotient in low word and remainder (same sign as dividend unless zero) in high word of Dd; if src is zero, cause Divide By Zero Exception; if dividend is larger than a signed word, set V=1, leave Dd unchanged, and end early; N and Z describe quotient but are undefined if V=1; set C=0 always. CLKs is max; min is 90% of max		
DIVU	src,Dd	T
	Ds	W124
	(As)	W125
	(As)+	W125
	-(As)	W127
	dl16(As)	W128
	d18(As,Xn)	W129
	addr.W	W128
	addr.L	W130
	dl16(PC)	W128
	d18(PC,Xn)	W129
	#da16	W126
Divide unsigned long Dd by unsigned word src, put quotient in low word and remainder in high word of Dd; if src is zero, cause Divide By Zero Exception; if dividend is larger than a word, set V=1, leave Dd unchanged, and end early; N and Z are undefined if V=1; set N=MSB of quotient; set Z=1 if quotient is zero; set C=0 always. CLKs is max; min is 90% of max		
EOR	(See ADD Ds,dst)	S
Logical Exclusive OR; like bits give a zero bit; differing bits give a 1 bit		
EORI	(See ADDI)	S
Logical Exclusive OR immediate		
EORI	#da8,CCR	B52 A
EORI	#da16,SR	W52 A
Logical Exclusive OR immediate to SR (PI) or CCR		
EXG	Rs,Rd	L3 N
Exchange the contents of 2 regs; use Ds,Ad not As,Dd		
EXT	Dd	WL1 S
Extend sign; fill higher byte or word with lower MSB		
ILLEGAL		103 N
Invalid object codes cause Illegal instruction exception		
JMP	(Ad)	6 N
	dl16(Ad)	12
	d18(Ad,Xn)	25
	addr.W	12
	addr.L	20
	dl16(PC)	12
	d18(PC,Xn)	25
Jump unconditionally		
JSR	(Ad)	32 N
	dl16(Ad)	43
	d18(Ad,Xn)	64
	addr.W	47
	addr.L	60
	dl16(PC)	47
	d18(PC,Xn)	67
Jump to subroutine; push long word address of next instruction using SP		
LEA	src,Ad	N
	(As)	L1
	dl16(As)	L8
	d18(As,Xn)	L17
	addr.W	L8
	addr.L	L20
	dl16(PC)	L8
	d18(PC,Xn)	L17
Load effective address; calculate long word absolute address of operand, and put address in Ad for later use		
LINK	As,dl16	36 N
Link and allocate stack space; save As contents on stack, copy new SP into As, and add dl16 to SP; SP then points to lowest and As to highest address +1 of stack space; dl16 must be 2's complement of size; use at start of subroutine to reserve temporary data space on stack; undo with UNLK		
LSL	(See ASL)	B
Logical shift left memory word by 1 bit or Dd by count in Ds or immediate data (3); zero fill; last bit out goes to C and X. If (Ds)=0 or #da3=0, set flags only; flag code is S		
LSR	(See ASL)	B
Logical shift right memory word by 1 bit or Dd by count in Ds or immediate data (3); zero fill; last bit out goes to C and X. If (Ds)=0 or #da3=0, set flags only; flag code is S		
MOVE	src,dst	S
Move from src to dst; see MOVE table, except as follows		
MOVE	src,SR (PI)	and
MOVE	src,CCR	A
	Ds	W13
	(As)	W29
	(As)+	W29
	-(As)	W41
	dl16(As)	W53
	d18(As,Xn)	W66
	addr.W	W53
	addr.L	W76
	dl16(PC)	W53
	d18(PC,Xn)	W66
	#da16	W33
If to CCR, only low byte is affected		
MOVE	SR,dst	N
	Dd	W3
	(Ad)	W15
	(Ad)+	W15
	-(Ad)	W23
	dl16(Ad)	W35
	d18(Ad,Xn)	W46
	addr.W	W35
	addr.L	W59
MOVE	As,USP	L1 (PI) N
MOVE	USP,Ad	L1 (PI) N
MOVEA	src,Ad	N
See Move table		
MOVEM	rl,dst	N
	(Ad)	W37 L75
	-(Ad)	W37 L75
	dl16(Ad)	W61 L91
	d18(Ad,Xn)	W71 L98
	addr.W	W61 L91
	addr.L	W82 L102
MOVEM	src,rl	N
	(As)	W56 L88
	(As)+	W56 L88
	dl16(As)	W78 L100
	d18(As,Xn)	W84 L104
	addr.W	W78 L100
	addr.L	W92 L106
	dl16(PC)	W78 L100
	d18(PC,Xn)	W84 L104
Move multiple regs (DO-D7, A0-A7) to or from consecutive memory locations; ls in 16-bit reg list bit map rl give regs to move; LSB gives first reg moved, MSB last; -(Ad) mode write order is A7-A0-D7-D0 from start address minus 2 through lower addresses, and final Ad points to last word written; for other modes order is DO-D7-A0-A7 from start address through higher addresses; (As)+ mode final As points to last word read plus 2; word operands read from memory are sign-extended to 32 bits; 1 extra read cycle occurs in memory to reg instruction. Example: MOVEM.L #$80D2,(A5) would move D1 to (A5), then move A7 to (A5)+2; A5 not changed		
MOVEP	Ds,dl16(Ad)	
	W36 L74	N
MOVEP	dl16(As),Dd	
	W34 L72	N
Move 2 or 4 bytes of peripheral data between a Data reg and alternate bytes of memory; high order reg byte moves to/from address, next lower byte to address+2, etc.; bytes move on upper half of data bus if address is even, lower half if address is odd; An does not change; for 8-bit peripherals		
MOVEQ	#da8,Dd	L1 S
Move byte of immediate (quick) data, sign-extended to 32 bits, to a Data reg (1 word instr.)		
MULS	src,Dd	and
MULU	src,Dd	S
	Ds	W116
	(As)	W117
	(As)+	W117
	-(As)	W119
	dl16(As)	W120
	d18(As,Xn)	W121
	addr.W	W120
	addr.L	W122
	dl16(PC)	W120
	d18(PC,Xn)	W121
	#da16	W118
Multiply word in Dd by word src; put long product in Dd; signed or unsigned		
NBCD	Dd	B3 C
	(Ad)	B15
	(Ad)+	B15
	-(Ad)	B23
	dl16(Ad)	B35
	d18(Ad,Xn)	B46
	addr.W	B35
	addr.L	B59
Negate BCD byte with extend; subtract operand and X from zero; gives 10's complement if X=0, 9's complement if X=1		
NEG	(See CLR)	A
Negate; 2's complement		
NEGX	(See CLR)	A
Negate with extend; subtract operand and X from 0 (1)		
NOP		1 N
No operation		
NOT	(See CLR)	S
Logical (1's) complement		
OR	(See ADD, except no As,Ad)	S
Logical OR		
ORI	(See ADDI)	S
Logical OR immediate		
ORI	#da8,CCR	B52 A
ORI	#da16,SR	W52 (PI) A
Logical OR immediate to SR or CCR		
PEA	(Ad)	L16 N
	dl16(Ad)	L36
	d18(Ad,Xn)	L55
	addr.W	L36
	addr.L	L60
	dl16(PC)	L36
	d18(PC,Xn)	L55
Push effective address; calculate long word absolute address of operand and push address onto stack for later use		
RESET		123 (PI) N
Output RESET* line low for 124 CLKs; no CPU regs are affected		
ROL	(See ASL)	U
Rotate left memory word by 1 bit or Dd by count in Ds or immediate data (3); last bit out goes back in and to C. If (Ds)=0 or #da3=0, set flags only; flag code is S		
ROR	(See ASL)	U
Rotate right memory word by 1 bit or Dd by count in Ds or immediate data (3); last bit out goes back in and to C. If (Ds)=0 or #da3=0, set flags only; flag code is S		
ROXL	(See ASL)	B
ROL with extend; same as ROL except last bit out goes to C and X; X goes back in. If (Ds)=0 or #da3=0, set flags only; flag code is U; set C to X		
ROXR	(See ASL)	B
ROR with extend; same as ROR except last bit out goes to C and X; X goes back in. If (Ds)=0 or #da3=0, set flags only; flag code is U; set C to X		
RTE		50 (PI) A
Return from exception; pop SR then PC from stack		
RTR		50 A
Return and restore; pop CCR then PC from stack; upper byte of SR is not affected		
RTS		31 N
Return from subroutine; pop PC from stack		
SBCD	Ds,Dd	B3 C
SBCD	-(As),-(Ad)	B42 C
Subtract 2-digit BCD src and X from dst		
Scc	(See NBCD, except):	
	Scc Dd	cc true B3 N
	Scc Dd	cc false B1 N
Set dst to $FF if cc is true, $00 if cc is false (4)		
STOP	#da16	2 (PI) A
Move immediate word into SR and stop executing; start if T was 1 or after reset or higher priority interrupt occurs		
SUB	(See ADD)	A
Subtract src from dst		
SUBA	(See ADDA)	N
Subtract from Ad		
SUBI	(See ADDI)	A
Subtract immediate		
SUBQ	(See ADDQ)	A
Subtract quick immediate data (3 bits: 1-8); 1 word instr.		
SUBX	(See ADDX)	A
Subtract src and X from dst (1)		
SWAP	Dd	W1 S
Swap Data reg halves (words)		
TAS	Dd	B1 S
	(Ad)	B23
	(Ad)+	B23
	-(Ad)	B30
	dl16(Ad)	B46
	d18(Ad,Xn)	B54
	addr.W	B46
	addr.L	B70
Test and set; set N and Z per dst, then set Bit 7 of dst; not interruptible between read and write		
TRAP	#da4	108 N
Cause Trap exception; like breakpoint or software interrupt; see Exception Processing		
TRAPV		Trap 103 N
		No trap 1
Cause TRAPV exception if V=1		
TST	Dd	BWL1 S
	(Ad)	BW6 L14
	(Ad)+	BW6 L14
	-(Ad)	BW11 L22
	dl16(Ad)	BW18 L34
	d18(Ad,Xn)	BW25 L45
	addr.W	BW18 L34
	addr.L	BW39 L58
Test against 0, set flags only		
UNLK	As	14 N
Unlink and deallocate stack space; copy As to SP and pop long word from stack into As; use after LINK to restore stack and As		

CYCLE CODES

Cycle codes give instruction length in words and execution time in CLKs. Bus read and write cycles are assumed to be four CLK cycles each. If bus wait states occur, you must add them to the CLKs. The numbers of read and write bus cycles per instruction are given for this purpose.

Listed in order of increasing CLKs, increasing words, increasing reads+writes, and increasing writes. For ordering only, s=1 and r=2 are assumed.

r = # of regs moved
s = shift count

Cycle Code	Words.CLKs. Reads.Writes
1	1.4.1.0
2	2.4.0.0
3	1.6.1.0
4	1.8.1.0
5	1.6+2s.1.0
6	1.8.2.0
7	1.8.1.1
8	2.8.2.0
9	1.10.1.0
10	1.8+2s.1.0
11	1.10.2.0
12	2.10.2.0
13	1.12.2.0
14	1.12.3.0
15	1.12.2.1
16	1.12.1.2
17	2.12.2.0
18	2.12.3.0
19	2.12.2.1
20	3.12.3.0
21	1.14.2.0
22	1.14.3.0
23	1.14.2.1
24	2.14.2.0
25	2.14.3.0
26	2.14.2.1
27	3.14.3.0
28	1.16.2.0
29	1.16.3.0
30	1.16.2.1
31	1.16.4.0
32	1.16.2.2
33	2.16.3.0
34	2.16.4.0
35	2.16.3.1
36	2.16.2.2
37	2.8+4r.2.r
38	3.16.3.0
39	3.16.4.0
40	3.16.3.1
41	1.18.4.0
42	1.18.3.1
43	1.18.2.2
44	2.18.3.0
45	2.18.4.0
46	2.18.3.1
47	2.18.2.2
48	3.18.4.0
49	3.18.3.1
50	1.20.5.0
51	1.20.3.2
52	2.20.3.0
53	2.20.4.0
54	2.20.3.1
55	2.20.2.2
56	2.12+4r.3+r.0
57	3.20.4.0
58	3.20.5.0
59	3.20.4.1
60	3.20.3.2
61	3.12+4r.3.r
62	4.20.5.0
63	4.20.4.1
64	1.22.2.2
65	1.22.3.2
66	2.22.4.0
67	2.22.2.2
68	3.22.4.0
69	3.22.5.0
70	3.22.4.1
71	3.14+4r.3.r
72	2.24.6.0
73	2.24.4.2
74	2.24.2.4
75	2.8+8r.2.2r
76	3.24.5.0
77	3.24.4.1
78	3.16+4r.4+r.0
79	4.24.6.0
80	4.24.5.1
81	4.24.4.2
82	4.16+4r.4.r
83	2.26.4.2
84	3.18+4r.4+r.0
85	4.26.6.0
86	4.26.5.1
87	4.26.4.2
88	2.12+8r.3+2r.0
89	2.28.4.2
90	3.28.5.2
91	3.12+8r.3.2r
92	4.20+4r.5+r.0
93	5.28.7.0
94	5.28.6.1
95	5.28.5.2
96	1.30.5.2
97	3.30.5.2
98	3.14+8r.3.2r
99	3.32.5.2
100	3.16+8r.4+2r.0
101	4.32.6.2
102	4.16+8r.4.2r
103	1.34.4.3
104	3.18+8t.4+2r.0
105	4.34.6.2
106	4.20+8r.5+2r.0
107	5.36.7.2
108	1.38.4.4
109	1.44.5.4
110	1.48.6.4
111	2.48.6.4
112	1.50.6.4
113	2.52.7.4
114	2.54.7.4
115	3.56.8.4
116	1.70.1.0
117	1.74.2.0
118	2.74.2.0
119	1.76.2.0
120	2.78.3.0
121	2.80.3.0
122	3.82.4.0
123	1.132.1.0
124	1.140.1.0
125	1.144.2.0
126	2.144.2.0
127	1.146.2.0
128	2.148.3.0
129	2.150.3.0
130	3.152.4.0
131	1.158.1.0
132	1.162.2.0
133	2.162.2.0
134	1.164.2.0
135	2.166.3.0
136	2.168.3.0
137	3.170.4.0

INSTRUCTION NOTES

(Applicable where indicated)
(1) Z is cleared if result is non-zero, but unchanged otherwise. Set or reset Z before start of operation to detect zero result, especially of repeated operation.
(2) Bit number is modulo B for memory operand, modulo 32 for reg operand
(3) Shift or rotate count is modulo 8 for memory operand, modulo 64 for reg operand
(4) A dst operand in memory is read before it is written
(5) Flags are not affected if dst is Ad, except for CMPA

CONDITION DEFS

CC	Carry Clear	C*
CS	Carry Set	C
EQ	Equal	Z
F	False	0
GE	Greater or Equal	(N.V)+(N*.V*)
GT	Greater Than	(N.V.Z*)+(N*.V*.Z*)
HI	Higher	C*.Z*
HS	Higher or Same	C*
LE	Less or Equal	Z+(N.V*)+(N*.V)
LO	Lower	C
LS	Lower or Same	C+Z
LT	Less Than	(N.V*)+(N*.V)
MI	Minus	N
NE	Not Equal	Z*
PL	Plus	N*
T	True	1
VC	Overflow Clear	V*
VS	Overflow Set	V

. means AND; + means OR; N* means NOT N; G & Less are signed results; H & Lower are unsigned results

MOVE TABLE

dst->	addr.L				
	d18(Ad,Xn)				
	dl16(Ad)/addr.W				
(Ad)/(Ad)+/-(Ad)					
src Dd/Ad*					
MOVE.B and MOVE.W src,dst:					
Ds	1	7	19	26	40
As	1	7	19	26	40
(As)	6	15	35	46	59
(As)+	6	15	35	46	59
-(As)	11	23	46	54	70
dl16(As)	18	35	59	70	80
d18(As,Xn)	25	46	70	77	86
addr.W	18	35	59	70	80
addr.L	39	59	80	86	94
dl16(PC)	18	35	59	70	80
d18(PC,Xn)	25	46	70	77	86
#da	8	19	40	49	63
MOVE.L src,dst:					
Ds	1	16	36	47	60
As	1	16	36	47	60
(As)	14	51	73	83	90
(As)+	14	51	73	83	90
-(As)	22	65	83	89	97
dl16(As)	34	73	90	97	101
d18(As,Xn)	45	83	97	99	105
addr.W	34	73	90	97	101
addr.L	58	90	101	105	107
dl16(PC)	34	73	90	97	101
d18(PC,Xn)	45	83	97	99	105
#da32	20	60	81	87	95

* MOVEA.W or MOVEA.L src,Ad

FLAGS

Flags Affected:						
A =	X	N	Z	V	C	Shift and rotate only
B =	X	N	Z	VC	C	BCD only; C is decimal carry or borrow (1)
C =	X	N?	Z	V?	C	
S =	-	N	Z	VC	C	Compare and divide only
T =	-	N	Z	V	C	Rotate only
U =	-	N	Z	VC	C	Bit ops only
V =	-	-	Z	-	-	CHK only
W =	-	N	Z?	V?	C?	
N = None						

A, B, C mean all; N means none; S, T, U, V, W mean some
f? becomes undefined
fC cleared to zero
f set/cleared according to result
- not changed

Flag	CCR Bit	Flag Name	Condition for flag=1, except as noted in instruction descriptions
X	4	Extend	Carry
N	3	Negative	MSB of result = 1
Z	2	Zero	Result=0
V	1	Overflow	Overflow occurs
C	0	Carry	Carry or borrow occurs