

## 68000 **PROGRAMMER'S INSTANT REFERENCE CARD**

**MICRO**<sup>®</sup> CHART

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\*.

PRIVILEGE VIOLATION: Execution of a

PRIVILEE VIOLATION: Execution of a privileged instruction (PI) in User state causes a privilege violation exception (ANDI #dal6.SR; EORI #dal6.SR; MOVE src.SR; MOVE As.USP; MOVE USP,Ad; ORI #dal6.SR; RESET; RTE; STOP #dal6). 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 OHK instructions cause an exception if certain conditions exist when they execute.

PINOUTS 68000 64-Pin DIP, Top View

n. D4= 1 64=D5 . D3= 2 63=D6 D1= 4 63=D8 D0= 5 63=D9 A5\*<6 59=D10 UD5\*<7 58=D11 LD5\*<8 57=D12 R/W\*<9 55=D14 B0AC\*+12 53=AD0 BR\*>13 52>A23 VCC:14 649-VCC

CLK>15 50>A21 CND-16 49-VCC HALT\*=17 48>A20 RESET\*=18 47>A19 WA8\*<19 46>A18 E<20 45>A17 VPA\*>21 44>A16 BERR\*>22 43>A15 FPL2\*>23 42>A14 IPL1\*>24 41>A13 FDL2\*>24 41>A13 FC2<26 39>A11 FC1<27 38>A10 FC0<28 37>A9 A1<29 36>A8

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

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

\* means active low. < and > show direction. = means bidirectional. nc means no connection incide

inside.

#### 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, ord groups. 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 = 5 cm

A7) As = Source Address register (A0-A7) addr = address addr.L = 32-bit absolute address addr.w = 16-bit absolute address - Depend size is byte

B = Operand size is byte BW = Operand size is byte or

EW = Operand size is byte c 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 DEP = Condition C.

CCR = Condition Code

register CLKs = E>

register CLKs = Execution time of instruction in CLK cycles da = Immediate data da3 = 3-bit immediate data da4 = 4-bit immediate vect

numbe da8 = Immediate data byte da16 = Immediate data word da32 = Immediate data long

da32 = Immediate data word Dd = Destination Data Word Immediate Gate 10% Dd = Destination Data register (D0-D7) Gd = 16-bit displacement di16 = 16-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 Le = Operand size is long word LSB = Least significant (low order) bit; Bit 0 MSB = Most significant (low order) bit; Bit 0 MSB = Most significant (low order) bit; Bit 0 MSB = Most significant (low order) bit; Bit 0 MSB = Most significant (low order) bit; Bit 0 RSB = Most significant (low order) bit 0 RSB = Most

cycles during instituction reg = Registers 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

Pointer register T = Trace bit (Bit 15) in SR USP = User Stack Pointer register V = Overflow flag in CCR

USP = User Stack Publics register V = Overflow flag in CCR W = Operand size is word or Long word V Words = Length of instruction in words Writes = Number of write bus cycles during instruction execution 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. if

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. ress

STATUS AND CONDITION CODE REGISTERS: System Byte User Byte (CCR)

		yS	cui		-y	uu		0.	SC.		- 7	uu		50	$\cdot$
								-							
Bit:	15						8	7							0
		-	-	-	-	-	-	-	-	-	-	-	-	-	-
SR·	тΩ	S	Π	Λ	Т	т	т	Π	Λ	Π	х	Ν	7	V	C

T: l = Trace mode, 0 = execute mode S: l = Supervisor state, 0 = User state III: Interrupt priority: l11 = 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, EDRI 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.

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.

C2	FC1	FCO	Cycle Type
			(reserved by Motorola) User Data User Program (reserved for user def.) (reserved by Motorola) Supervisor Data Supervisor Program Interrupt Acknowledge
The	data	a bus	strobes define how the data
ous	is ι	used	
Dat	ta St	trobe	≳s Bus Use
JDSt	* LDS	S* R/	/W* D15-D8 D7-D0

Н HLLHLH n 7-0 7-0 n 15-8 н Н Н Ц L H L L H n 15-8 15-8 7-0m 15-8 n 7-0 7-0 L 15-8m ï \* = Active low signal H = High L = Low x = Don't care \* = Active iow sig 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 addresse; 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 USER STACK: To grow from higher addr 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 destinat move data from a source are written in the form: mnemonic src,dst ation

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 #7,03

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 (dli6(PC)): Address of operand is sum of address of extension word and sign-extended displacement in extension word. 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 DO

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

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

ADDRESS REGISTER INDIRECT WITH 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 (dil6(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 of extension word. Example: ROL.W BIAS(AO,A1.W)

#### ASCII

LSE	MSD	0 000	$^{1}_{001}$	2 010	3 011	4 100	5 101	6 110	7 111
0 1 2 3	0000 0001 0010 0011	NUL SOH STX ETX	DLE DC1 DC2 DC3	SP ! #	0 1 2 3	@ A B C	PQRS	、 a b c	рцна
4 5 6 7	0100 0101 0110 0110 0111	eot enq ack bel	dc4 NAK Syn Etb	\$%&'	4 5 7	D E F G	T U V W	d ef g	tuv w
8 9 A B	1000 1001 1010 1011	BS HT LF VT	CAN EM SUB ESX	( ) +	8 9 :;	H J K	X Y Z [	h i k	x y z {
CDEF	1100 1101 1110 11110 11111	FF CR SO SI	FS GS RS US	• • •/	v = ^?	LMNO	\ ] ^ -	l m n o	 } DEL

#### **EXCEPTION PROCESSING**

e CPU's response to unusual internal or ternal conditions.

external conditions.	
EXCEPTION VECTORS: Number Addr Dec Hex Hex Use CLKs.Reads.Writes	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.
0 00 000000 (Reset SSP; see note below) 10 000004 RESET 40.6.0 2 02 000008 RESET 40.6.0 4 000010 11legal Instruction 34.4.3 5 05 000011 41vide by zero 42.5.4 6 06 000018 CHK operand out of bounds 7 0 00001C TRAPY when $\vee 1 34.4.3$ 8 08 000020 Privilege violation 34.4.3 10 08 000020 Line 1010 emulator 34.4.3 10 08 000020 Line 1010 emulator 34.4.3 11 08 000020 Line 1011 emulator 34.4.3 12 00 000030 (reserved) 13 00 000030 (reserved) 14 06 000030 (reserved) 15 07 000030 (reserved) 15 07 00005C (reserved) 14 18 000066 Spurious inpt 44.5.3 15 19 000066 Ext inpt 2 autovector 44.5.3 27 18 000066 Ext inpt 2 autovector 44.5.3 28 10 000070 Ext inpt 4 autovector 44.5.3 29 10 00070 Ext inpt 4 autovector 44.5.3 29 10 00070 Ext inpt 4 autovector 44.5.3 20 10 000070 Ext inpt 4 autovector 44.5.3 30 1E 000070 Ext inpt 4 autovector 44.5.3 30 1E 000070 Ext inpt 6 autovector 44.5.3 30 1E 000070 Ext inpt 6 autovector 44.5.3 30 1E 000070 Ext inpt 7 autovector 44.5.3 30 1E 000070 Ext inpt 6 autovector 44.5.3 30 1E 000070 Ext inpt 7 autovector 44.5.3 30 1E 000070 Ext inpt 4 autovector 44.5.3 30 1E 000076 Ext inpt 5 autovector 44.5.3 30 1E 000076 Ext inpt 5 autovector 44.5.3 30 1E 000076 Ext inpt 6 autovector 44.5.3 30 1E 000076 Ext inpt 6 autovector 44.5.3 30 1E 000076 Ext inpt 6 autovector 44.5.3 30 1E 0000	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 INTERNUPTS: External logic encodes a priority level on IPL2*, TPL1*, 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 (futerrupt 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. Morcorla reserves each of these for future definition except as follows. Opcodes \$AAFA, §AAFB, and \$AAFC will always cause an Illegal Instruction exception; the first two are reserved for Motorola products, and the third is reserved for Actorola
memory space; all others are in Supervisor Data memory space.	(\$Axxxx) or 1111 (\$Fxxxx) in Bits 15-12 will cause a Line 1010 or Line 1111 Emulator exception, respectively. All
EXCEPTION VECTORS: Each (except 0) holds the long word address of an exception handling routine. Vector 0 is not a	other unimplemented opcodes cause an Illegal Instruction exception.

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 o 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. of

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 stat 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. Pushe 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. . or state Pushes 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

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=III), 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. registers are affected.

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: l=read, ownrite: Bit 3 = 0 if the CPU was exception; Bit 3 = 1 if the CPU was processing any other exception; Bits 2-0: FC2-FC0. processi FC2-FC0.

When HALT\* and BERR\* are both input low, the CPU will abort the cycle, then re-run it when BER\* 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



BCHG

# 68000 **PROGRAMMER'S INSTANT REFERENCE CARD**

### MICRO<sup>®</sup> CHART

INSTRUCTION SET

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 sr.Od the flag code A applies to all addressing modes; cycle code 1 applies to all ADD. B's, Dd and ADD. W DS.Dd; and cycle code 4 applies to ADD.L DS.Dd; and cycle code 4 applies. Oper- Cycle and Inst ands Flag Code ABCD Ds,Dd B3 C ABCD -(As),-(Ad) B42 C Add 2-digit BCD Add 2-digit BCD numbers plus X ADD src.Dd A (add) Ds BWI L4 (As) BW6 L22 - (As) BW1 L4 (As) BW6 L22 - (As) BW18 L45 dil6(As) BW18 L45 dil6(As, Xn) BW25 L53 addr.W BW18 L45 addr.L BW39 L69 dil6(PC) BW18 L45 
 addr.W.
 Bulla La5

 addr.L.
 BW39 L69

 addr.L.
 BW39 L69

 dil6(PC)
 BW18 L45

 dil6(PC), M)
 BW25 L53

 ADD Ds, dst.
 A (add)

 (Ad)
 BW15 L51

 (Ad)
 BW35 L51

 (Ad)
 BW35 L53

 dil6(Ad)
 BW35 L53

 dil6(Ad)
 BW35 L51

 dadtr.W
 BW35 L51

 Addr.L
 BW55 L51

 Addr.L
 BW55 L51

 Addr.L
 BW2

 ADA
 KL4

 ADDA
 TC, Ad
 NL

 Ds
 wL4

 As
 wL4

 (As)
 W13

 L22
 -(As)

 vi16(As)
 W31

 vi16(As)
 W31

 vi16(As, Xn)
 W44

 L53
 addr. N

 w33
 L45

 dif6(PC)
 W33

 dif6(PC)
 W33

 dif6(PC)
 W33

 #da
 W18

 Add to Ad
 W38
 #da W18 Add to Ad ADDI #da,dst A Add to Ad ADDI #da,dst A Dd #da,dst A Dd #da,dst A Dd #da,dst A BW35 L90 (Ad) BW35 L90 (Ad) BW35 L90 (Ad) BW35 L91 di8(Ad,Yn) BW30 L101 di8(Ad,Xn) BW70 L105 addr.W BW59 L101 addr.L BW50 L107 Add immediate ADD #da3,dst A Dd BW11 L4 Ad WL4 AD W35 L73 addr.L BW35 L73 addr.L BW59 L90 Add quick immediate data (3 bits: 1-8); 1 word instr. ADDX 05,00 BW1 L4 A -(As).-(Ad) BW42 L96 Add operands and X (1) AND (See ADD, but no -(A6),-(A0) BWA2 L96 Add operands and X (1) AND (See ADD, but no As, Dd) Logical AND S ANDI (See ADD, See ADD) S ANDI #da8,CCR B52 A (Ad) BW5 L10 A/S #da3,Dd BW5 L10 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 Cc false 4 Bcc di16 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)+ B35 di8(Ad,N) B36 addr.W B35 addr.L B59

BCHG #da8,dst	UMP1 #da,dst   Dd PW9   27
(Ad) B35	(Ad) BW18 L58
(Ad)+ B35	(Ad)+ BW18 L58
dil6(Ad) B59	dil6(Ad) BW39 L79
di8(Ad,Xn) B70	di8(Ad,Xn) BW48 L85
addr.L B80	addr.L BW62 L93
Flip bit specified by	Compare dst to
by dst; put result	(dst - immediate
bit in Z (2)	data)
except:) V	BW14 L50 T
Ds,Dd L9	Compare 2 memory
Clear bit specified	DBcc Dn,dil6 N
by src in location	cc true 17
complement of	cc false & no br. 25
original bit in Z (2)	Decrement and branch
BRA dil6 12 N	l)if cc is true, go
Branch always; di8=0	to next instruction;
BSET (See BCHG) V	decrement low word o
Set bit specified by	Ds; 3)if Ds is -1, g
by dst; put	4)if Ds is not -1,
complement of original bit in 7 (2)	branch (loop); DBRA same as DBF
BSR di8 43 N	DIVS src,Dd T
BSR dil6 47 N Branch (call) to	Ds W131 (As) W132
subroutine; push long	(As)+ W132
word address of next	-(As) W134 di16(As) W135
di8=0 not allowed	di8(Às,Xn) W136
Dd 13	addr.W W135 addr.L W137
(Ad) B6	di16(PC) W135
(AD)+ B6 - (Ad) B13	ulo(PC,XN) W136 #dal6 W133
dil6(Ad) B18	Divide signed long [
addr.W B18	by signed word src, put signed auotient
addr.L B39	in low word and
di8(PC,Xn) B25	as dividend unless
#da8 B8	zero) in high word o
Dd L12	cause Divide By Zero
(Ad) B18	Exception; if
-(Ad) B25	than a signed word,
di16(Ad) B39	set V=1, leave Dd
addr.W B39	early; N and Z
addr.L B62 dil6(PC) B39	describe quotient bu are undefined if V=1
di8(PC,Xn) B48	set C=O always. CLM
complement of bit	max max, min is 70% c
specified by src in location given by	DIVU src,Dd T Ds W124
dst; no change to dst	(As) W125
(2) CHK src,Dd W	(AS)+ W125 -(AS) W127
In-Bounds/Out	dil6(As) W128
(As) W21/W110	addr.W W128
(As)+ W21/W110 -(As) W28/W112	addr.L W130 dil6(PC) W128
dil6(As) W44/W113	di8(PC,Xn) W129
d18(As,Xn) W52/W114 addr.W W44/W113	#dal6 W126 Divide unsigned long
addr.L W68/W115	Dd by unsigned word
	and put quotiont in
dil6(PC) W44/W113 di8(PC,Xn) W52/W114	src, put quotient in low word and
dil6(PC) W44/W113 di8(PC,Xn) W52/W114 #dal6 W25/W111 Check Dd low word	src, put quotient in low word and remainder in high word of Dd: if src i
di16(PC) W44/W113 di8(PC,Xn) W52/W114 #da16 W25/W111 Check Dd low word against 0 and upper	<pre>src, put quotient in low word and remainder in high word of Dd; if src i zero, cause Divide E</pre>
dil6(PC) W44/W113 di8(PC,Xn) W52/W114 #dal6 W25/W111 Check Dd low word against O and upper bound; cause exception if less	src, put quotient ir low word and remainder in high word of Dd; if src i zero, cause Divide E Zero Exception; if dividend is larger
dil6(PC) W44/W113 di8(PC,N) W52/W114 #dal6 W25/W111 Check Dd low word against 0 and upper bound; cause exception if less than 0 (MSB=1) or amotor there wore	src, put quotient in low word and remainder in high word of Dd; if src i zero, cause Divide E Zero Exception; if dividend is larger than a word, set V= lowce Dd urbergerd
dil6(PC) WW44/W113 di8(PC,XN W52/W114 #dal6 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	<pre>src. put quotient in low word and remainder in high word of Dd; if src i zero, cause Divide E Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N anc</pre>
dil6(PC) W44/W113 di8(PC,X) W52/W114 #dal6 W52/W114 fdal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FFF); src is 2's complement (\$2000-\$FFFF); or	src. put quotient if low word and remainder in high word of Dd; if src i Zero Rception; if dividend is larger than a word, set V=J leave Dd unchanged, and end early: N and Z are undefined if V=J: set N=MSR of
dil6(PC) W44/W113 di8(PC,XN W52/W114 #dal6 W52/W114 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	<pre>src, put quotient in low word and remainder in high word of DQ; if src i 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=MSD of quotient; set Z=1 if</pre>
dil6(PC) W44/W113 di8(PC,XN W52/W114 #dal6 W52/W114 fdal6 W52/W114 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 BW1L3 S (Ad) BW15 L51	src, put quotient in low word and remainder in high word of Od; if src i 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=MSD of quotient; set Z=1 if quotient is zero; se C=0 always. CLKS if
dil6(PC) W44/W113 di8(PC,X) W52/W114 #dal6 W52/W114 deal6 W52/W114 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-\$7FFF); or upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51	<pre>src, put quotient in low word and remainder in high word of DQ; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and V=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. Clks is max; min is 90% of</pre>
dil6(PC) W44/W113 di8(PC,XW114 #dal6 W52/W114 #dal6 W52/W114 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) BW35 L73	<pre>src, put quotient in low word and remainder in high word of Dd; if src i 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; se C=0 always. CLKs is max; min is 90% of max EOR (See ADD Ds,dst)</pre>
dil6(PC) W44AW113 di8(PC,NW52/W114 #dal6 W52/W114 #dal6 W52/W114 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-\$7FFF); or upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 -(Ad) BW15 L51 -(Ad) BW35 L53 -(Ad) BW35 L53 -(Ad) BW35 L73 di8(Ad,Xn) BW46 L83 addr W BW5 173	<pre>src, put quotient in low word and remainder in high word of DQ; if src i 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; se C=0 always. CLKs is max; min \$ 90% of max x m is 90% of CBR (See ADD Ds,dst) Logical Exclusive Of like bits of var zero</pre>
dil6(PC) W44/W113 di8(PC,NW52/W114 #dal6 W52/W114 #dal6 W52/W114 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-\$7FFF); or upper bound CLP Dd BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 di8(Ad,NB SL73 di8(Ad,ND BW46 L83 addr.W BW35 L73 addr.W BW35 L70	<pre>src, put quotient in low word and remainder in high word of Dd; if src i 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; se C=0 always. QLKs is max; min is 90% of max EUR (See ADD Ds.dst) Logical Exclusive Of like bits give a zer bit; differing bits</pre>
dil6(PC) WW44/WI13 di8(PC,XW)W52/WI14 #dal6 W52/WI114 #dal6 W52/WI11 Check Dd low word against 0 and upper bound; cause exception if less than 0 (MSB=1) or greater than upper bound; 00-\$FFFF); src is 2's complement (\$8000-\$FFFF) of upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51 (Ad)+ BW15 L51 c(Ad) BW35 L73 di8(Ad,Xm) BW46 L83 addr.W BW35 L73 addr.W BW35 L73 addr.W BW35 L73 addr.W BW35 L73 addr.W BW55 L73 addr.W BW55 L73 addr.W BW55 L73 addr. 2e1 (4)	<pre>src, put quotient in low word and remainder in high word of DQ; if src i 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 is zero; se C=0 always. CLKs is max; min is 90% of max EOR (See ADD Ds,dst) Logical Exclusive OF like bits give a zen bit; differing bits give a 1 bit EOR (See ADDI)</pre>
dil6(PC) WW42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 Check Dd low word against 0 and upper bound; cause exception if less than 0 (MSB=1) or greater than upper bound; 00:37FFF); src is 2's complement (\$8000-\$FFF); or upper bound CLR Dd BW1L3 S (Ad) BW15 L51 (Ad) BW15 L51 c(Ad) BW15 L51 c(Ad) BW15 L51 c(Ad) BW15 L51 c(Ad) BW35 L73 dil6(Ad, Xn) BW46 L83 addr.W BW35 L73 dil6(Ad, Xn) BW45 L73 dil6(Ad, Xn) EW45 L73 dil6(Ad, Xn) BW45 L73 dil6(Ad, X	<pre>src, put quotient in low word and remainder in high word of DQ; if src i 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; se C=0 always. CLKs is max; min is 90% of max EOR (See ADD Ds.dst) Logical Exclusive OF like bits give a zer bit; differing bits give al bit EORI (See ADD) S Logical Exclusive OF like bits give a zer bit; differing bits give al bit</pre>
dil6(PC) W44/W113 di8(PC,XW114 #dal6 W52/W114 #dal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FFF); src is 2's complement (\$8000-\$7FFF); or upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 -(Ad) BW15 L51 -(Ad) BW15 L51 -(Ad) BW35 L53 di8(Ad,Xn) BW46 L83 addr.W BW35 L73 di8(Ad,Xn) BW46 L83 addr.H BW35 L73 di8(Ad,Xn) BW46 L83 addr.L BW59 L90 Clear operand to 0; set N=0, Z=1 (4) CMP src,Dd T Ds BW1 L3	<pre>src, put quotient in low word and remainder in high word of Dd; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N amc Z are undefined if V=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max min is 90% of max EOR (See ADD Ds,dst) Logical Exclusive Of like bits give a zen bit; differing bits give a 1 bit EORI (See ADDI S Logical Exclusive Of immediate EORI #da8,CCR B52 A</pre>
dil6(PC) W44/W113 di8(PC,NW52/W114 #dal6 W52/W114 #dal6 W52/W114 Deek 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-\$7FFF); or upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 -(Ad) BW15 L51 -(Ad) BW35 L73 di8(Ad,Xn) BW46 L83 addr.W BW35 L73 di8(Ad,Xn) BW46 L73 di8(Ad,Xn) BW46 L72 di9(Ad,Xn) BW46 L72 di9(Ad,Xn) BW46 L72 di9(Ad,Xn) BW46 L72 di9(Ad,Xn) BW46 L72 di9(Ad,Xn) BW46 L72 di9(Ad,Xn) BW46 L72 di7(Ad,Xn) BW4	<pre>src, put quotient in low word and remainder in high word of DQ; if src i 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; se C=0 always. CLKs is max; min is 90% of max Gee ADD Ds,dst). Logical Exclusive Of like bits give a zer bit; differing bits give a1 bit EORT (See ADD1 S Logical Exclusive Of immediate EORT #da8_CR H52 A Logical Exclusive Of immediate EORT #da8_CR H52 A Logical Exclusive Of immediate EORT #da8_CR H52 A</pre>
dil6(PC) WW44/WI13 di8(PC,XW)W52/WI14 #dal6 W52/WI14 #dal6 W52/WI14 bound; cause exception if less than 0 (MSB=1) or greater than upper bound; 0c.37FFF); src is 2's complement (\$8000-\$7FFF); or upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW35 L53 dil6(Ad) BW35 L73 dil6(Ad) CL3 Ds BW1 L3 As W1 L3 (As) BW6 L22 (As) BW6 L22 (As) BW6 L22 (As) BW1 L28	<pre>src, put quotient in low word and remainder in high word of Dd; if src i 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; se C=0 always. CLKs is max; min is 90% of max Explicit Exclusive Of like bits give a zel give a l bit give a l bit give a l bit contral Exclusive of limmediatec EORI (See ADD S dst) Logical Exclusive of immediatec EORI fda8,CRR W52 A LORI (Sea KDD S LORI (Sea EADI) S Logical Exclusive of immediatec b SR (PI)</pre>
dil6(PC) W42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 Devextor 10 word against 0 and upper bound; cause exception 1f less than 0 (MSS=1) or greater than upper bound (0-S7FFF); src 1s 2's complement (\$4000-\$47FF7); src 1s 2's complement (\$4000-\$47FF7]; src 1s 2's complement (\$4000-\$47FF7]; src 1s 2's complement (\$4000-\$47FF7]; src 1s 2's complement (\$4000-\$47F77]; src 1s 2's complement (\$400-\$47F77]; src 1s 2's complement (\$400-\$47F777]	<pre>src, put quotient in low word and remainder in high word of DQ; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and quotient is zero; se C=0 always. CLKs is max; min is 90% of max (See ADD Ds.dst) Logical Exclusive Of like bits give a zer bit; differing bits give alb clusive Of like bits give a zer bit; differing bits GPR (See ADD Ds.dst) Logical Exclusive Of like bits give a zer bit; differing bits GPR (See ADD Last) EORT idea CCR B52 A Logical Exclusive Of logical Exclusive Of logical Exclusive Of immediate to SR (PI) or CCR EXC R ER 13 N</pre>
dil6(PC) wW42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 dal6 W52/W114 for the cause exception if less than 0 (MSB=1) or greater than upper bound (0-3/FFF): src is 2's complement (\$8000-\$FFF): or upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW35 L73 di8(Ad,Xn) BW46 L83 addr.W BW35 L73 di8(Ad,Xn) BW46 L53 is SW1 L3 As W1 L3 (As) BW6 L22 (As) BW1 L28 (As) BW1 L28 (As) BW1 L28 (As) BW1 L28 (As) BW1 L28 (As) BW1 L28 (As) BW1 L28 di8(As,Xn) BW25 L53 di8(As,Xn) BW25 L53	<pre>src, put quotient in low word and remainder in high word of Dd; if src i 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; se C=0 always. CLKs is max; min is 90% of max ECR (See ADD Ds,dst) Logical Exclusive OF like bits give a zer bit; differing bits give a 1 bit EORI (See ADD S immediate EORI #da8,CCR B52 A EORI #da8,CCR B54 A EORI #da8,CCR B54 A EORI</pre>
dil6(PC) WW42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 Dence 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-\$7FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (Ad) BW15 L51 -(Ad) BW15 L51 -(Ad) BW15 L51 -(Ad) BW35 L53 dil6(Ad) BW35 L53 dil6(Ad) BW35 L73 addr.L BW59 L90 Clear operand to 0; set N=0, Z=1 (A) CMP src, Dd T Ds BW1 L3 (As) BW6 L22 -(As) BW1 L3 dil6(AS) BW18 L45 dil6(AS, BW18 L45 dil6(AS, BW18 L45	<pre>src, put quotient in low word and remainder in high word of DQ; if src: Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of EOR (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; differing bits give at bit EORI (See ADD J S Logical Exclusive Of immediate EORI #da8,CCR B52 A EORI #da8,CCR B52 A EORI</pre>
dil6(PC) WW42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (\$8000-\$7FFF; src is 2's complement (\$8000-\$7FF; src is 2's complement (\$8000-\$7FFF; src is 2's complement (\$8000-\$7FF; src is 2's complement (\$8	<pre>src, put quotient in low word and remainder in high word of DQ; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max max in 5 90% of max in is 90% of max in is 90% of max is</pre>
dil6(PC) WW42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FFF); src is 2's complement (\$8000-\$7FFF); or upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW35 L53 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad, SM) BW4 L53 As W1 L3 As W1 L3 As BW1 L3 As BW1 L2 (As) BW6 L22 (As) BW6 L22 (As) BW1 L25 dil6(As) BW1 L45 dil6(As) BW1 L45 dil6(Ac) BW25 L53 Compare Dd to src; subtract src from Dd,	<pre>src, put quotient in low word and remainder in high word of Dd; if src i 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; se C=0 always. CLKs is max; min is 90% of max min is 90% of max; min is 90% of EORI (See ADD Ds, dst) EORI (See ADDI) S Logical Exclusive Of immediate to SR (PI) or CCR EXG Rs, Rd L3 N Exchange the content of 2 reg; use Ds, A not As, Dd EXT Dd WL1 S Extend sign; fill higher byte or word</pre>
dil6(PC) W42/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 del6 W25/W114 Check Dd low word against 0 and upper bound; cause exception 1f less than 0 (MSB=1) or greater than upper bound; 0C3/FFF); src 1s 2's complement (\$8000_\$FFF); or Upper bound CLR Dd BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L53 (Ad) BW55 L73 addf(M Xm) BW35 L73 addf(M Xm) BW46 L22 (As)+ BW4 L23 dil8(As, Xm) BW25 L53 addr.W BW18 L45 addr.L BW39 L69 dil8(PC) BW18 L45 compare Dd to src: subtract src from Dd, set flags, don't change operands: Pl T	src, put quotient in low word and remainder in high word of DQ; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N ame quotient is zero; se C=0 always. LIKs is max; min is 90% of max sc is set is set is contained to the set quotient is zero; se C=0 always. LIKs is max; min is 90% of max sc is set is set contained to the set contained to the set set of the set set of the set contained to the set set of the set to the set of the set to the set contained to the set to the set contained to the set contained the set contained to the set contained the set contained to the set containe
dil6(PC) WW42/W114 dil6(PC) WW52/W114 dil6(D) WW52/W114 dil6(D) WW52/W114 dil6(D) WW52/W114 dil6(D) WW52/W114 dil6(D) WW52/W12 dil6(D) W12 L3 Zis complement (\$8000-\$FFF); src ls Zis complement (Ad)+ BW15 L51 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad), BW35 L53 dil6(As), BW18 L45 dil6(As), CBW18 L45 dil6(As), CBW18 L45 dil6(As), BW18 L45 dil6(As	<pre>src, put quotient in low word and remainder in high word of DG; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and quotient is zero; se C=0 always. CLKs is max; min is 90% off max ECR (See ADD Ds.dst) Logical Exclusive OF like bits give a zer bit; differing bits give a 1 bit EORI (See ADD S) SL Logical Exclusive OF immediate EORI fda8,CCR B52 A EORI fda8,CCR B5</pre>
dil6(PC) WW42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FF); src is 2's complement (\$8000-\$7FF); src is 2's complement (Ad) = BW1 L3 S (Ad) = BW1 L3 S (Ad) = BW35 L73 dil6(Ad) = BW35 L73 addr. W BW35 L73 addr. W BW35 L73 addr. W BW35 L73 addr. W BW35 L73 dil6(AS) = BW6 L22 (As) = BW1 L3 addr. W BW18 L45 dil6(AS, Xn) BW25 L53 addr. W BW18 L45 dil6(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	<pre>src, put quotient in low word and remainder in high word of Dd; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and Y=1; set N=MSB of quotient is zero; se C=0 always. CLKs is max; min is 90% of max ECR (See ADD Ds,dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORT (See ADD Js,dst) Logical Exclusive Of immediate EORT #da8,CCR B52 A EORI #da8,CCR B52 A EXT da WLS Extend sign; fill higher byte or word with lowject codes cause Illegal instruction exceptic</pre>
dil6(PC) WW42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (\$8000-\$7FFF; src is 2's complement (\$8000-\$7FF; src is 2's complement (\$8000-\$7FFF; src is 2's complement (\$8000-\$7FFF; src is 2's complement (\$8000-\$7FF; src is 2's complement (\$8000-\$7FF; src is 2's complement (\$8000-\$7500-\$7500-\$7500-\$7500-\$7500-\$75000-\$75000-\$7500-\$7500-\$7500-\$7500-\$75000-\$7500-\$7500-\$75	<pre>src, put quotient in low word and remainder in high word of DQ; if src: Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max ECR (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORI (See ADD D, dst) Logical Exclusive Of like bits give a zer bit; dstfering bits give a 1 bit EORI (See ADD D, dst) Logical Exclusive Of like bits Give Zer DOI immediate EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EXT Dd WL S Extend Str; fill higher byte or word with lower MSB ILLEGAL IO N Invalid bject codest cause Illegal instruction exceptic JMP (Ad) 6 Y</pre>
dil6(PC) W44/W113 di8(PC,Xn) W52/W114 #dal6 W55/W114 #dal6 W55/W114 dbal6 W55/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (\$8000-\$7FFF); or upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW35 L51 (Ad) BW35 L51 (Ad) BW35 L51 (Ad) BW35 L51 (Ad) BW35 L51 (Ad) BW35 L51 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad) BW35 L73 dil6(Ad, SM) BW45 L35 addr.W BW35 L53 dil6(AC) BW18 L45 dil6(AC) BW18 L45 dil6(AC) BW18 L45 dil6(AC) BW18 L45 dil6(AC) BW35 L53 Compare Dd to src; subtract src from Dd, set flags, don't change operands; BLT, for example, branches after CMF if Dd is less than src CMPA src, Ad T Ds WL3	src, put quotient in low word and remainder in high word of DQ; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max min is 90% of max; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max min is 90% of EOR (See ADD Ds, dst). Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORT (See ADD J S Logical Exclusive Of immediate EORT #da8, CCR B52 A Logical Exclusive Of immediate EORT #da8, CCR B52 A Logical Exclusive Of immediate to SR (PI) SEXtend sign; fill higher byte or word with lower MSB ILLEGAL 103 N Invalid object codes cause Illegal instruction exceptid JMP (Ad) 6 N dil6(Ad), Z2
dil6(PC) W42/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 Devextor 10 word against 0 and upper bound; cause exception 1f less than 0 (MSB=1) or greater than upper bound; 0C3/FFF); src 1s 2's complement (\$4000_\$FFF); or Upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW35 L51 dil6(Ad) BW35 L53 dil6(Ad) BW35 L73 dil6(Ad) BW45 L73 dil6(Ad) BW45 L73 dil6(Ad) BW45 L73 dil6(Ad) BW25 L53 Compare Dd to src: subtract src from Dd, set flags, don't change operands: BLT. for exemple, branches after CWP if Dd is less than src CMPA src.Ad T Ds WL3 (As) W11 L22 (As)+ W11 L22	src, put quotient in low word and remainder in high word of DQ; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and quotient: set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max ECR (See ADD Ds.dst) Logical Exclusive Of like bits give a zer bit; differing bits give a l bit EORI (See ADD S) S Logical Exclusive OF like bits give a zer bit; differing bits give a l bit EORI (See ADD S) S Logical Exclusive OF like bits GR SZ A EORI differ a bit EORI (See ADD S) S Logical Exclusive OF like bits give a zer bit; differing bits give a l bit EORI (See ADD S) S Logical Exclusive OF limediate EORI differ and SZ A EORI differ and SZ A SZ A Hong differ and SZ A
dil6(PC) W42/W114 dil6(PC) W42/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 greater than upper bound (0.47FFF); src is 2's complement (%8000.57FFF); src is 2's complement (%8000.57FFF); src LR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 dil6(Ad) BW35 L73 dil6(Ad) BW35 L53 dil6(Ad) BW18 L45 dil6(Ad) BW18 L45 dil6(Ad) BW18 L45 dil6(Ad) BW18 L45 dil6(Ad) BW18 L45 dil6(Ad) BW18 L45 dil6(Ad) SW18 L45 dil6(Ad)	<pre>src, put quotient in low word and remainder in high word of DG; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max ECR (See ADD Ds.dst) Logical Exclusive OF like bits give a zer bit; differing bits give a 1 bit EORI (See ADD DS.dst) Logical Exclusive OF like bits give a zer bit; differing bits give a 1 bit EORI (See ADD S.dst) Logical Exclusive OF like bits give a zer bit; differing bits give a 1 bit EORI (See ADD S.dst) Logical Exclusive OF like bits give a zer bit; differing bits give a 1 bit EORI fda8, CCR B52 A EORI fda8, CCR</pre>
dil6(PC) WW32/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FF); src is 2's complement (\$8000-\$7FF); src is 2's complement (Ad) = BW1 L3 S (Ad) = BW1 L3 S (Ad) = BW35 L73 dil6(Ad) = BW35 L73 dil6(Ad) = BW35 L73 addr. L = BW35 L73 addr. L = BW35 L73 addr. L = BW35 L73 dil6(AS) = BW1 L3 As = W1 L3 (As) = BW6 L22 (As) = BW1 L3 dil6(AS) = BW18 L45 dil6(AS, M) = BW35 L53 dil6(AS, M) = BW35 L53 dil6(AS) = W18 L45 dil6(AS, M) = BW35 L53 dis (As) = W13 L29 dil6(As) W33 L53	<pre>src, put quotient in low word and remainder in high word of Dd; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and Y=1; set N=MSB of quotient is zero; se C=0 always. CLKs is max; min is 90% of max ECR (See ADD Ds,dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORT (See ADD Ds,dst) Logical Exclusive Of immediate EORT (See ADD Ds,dst) Logical Exclusive Of immediate Lost give a zer bit; differing bits give a 1 bit EORT (See ADD Ds, dst) Logical Exclusive Of immediate to SR (PI) or CCR EXG Rs,Rd L3 N Exchange the content of 2 regs; use Ds,Ac not As,Dd EXT Dd WLI S Extend sign; fill hipher byte or word with lowject codes cause Illegal instruction exceptic JMP (Ad) 6 N Lie Qi 2 dig(Ac,Xn) 25 addr.k 12 dig(Ac,Xn) 25 addr.k</pre>
dil6(PC) W42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (Ad) = BW1 L3 S (Ad) BW1 L3 S (Ad) BW35 L51 dil6(Ad) BW35 L53 addr. W BW35 L53 addr. W BW18 L45 dil6(As) BW18 L45 dil6(As) BW18 L45 dil6(As) BW18 L45 dil6(As) BW18 L45 dil6(As) BW18 L45 dil6(As) CM39 L69 dil6(PC, Xn) BW25 L53 addr. W BW18 L45 dil6(As) W13 L22 (As)+ W11 L22 (As)+ W11 L22 (As)+ W11 L22 (As)+ W11 L22 (As)+ W11 L22 (As)+ W13 L29 dil6(As) W25 L45 addr. W W25 L45	<pre>src, put quotient in low word and remainder in high word of Dd; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of EXR (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORI (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORI (See ADD Ds, dst) Logical Exclusive Of immediate EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EXT Dd Wall S Extend stor; fill hipher byte or word with lower MSB ILLEGAL 103 N Invalid object codes cause Illegal instruction exceptic JMP (Ad) 6 Min (Ad) 12 dB(Ad, Xn) 25 addr.w 12 addr.L 20 dB(Ad) 25 Jump unconditional) JSR (Ad) 45 Min (Ad) 45 Min</pre>
dil6(PC) WW42/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 #dal6 W52/W114 bound; cause exception if less than 0 (MSB=1) or greater than upper bound (0-\$7FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (Ad) = BW1 L3 S (Ad) BW15 L51 -(Ad) BW35 L53 dil6(Ad) BW35 L53 dil6(Ad) BW35 L53 dil6(As) BW16 L22 -(As) BW1 L3 dil6(As) BW16 L45 dil6(As) BW18 L45 dil6(As, CM) BW25 L53 compare Dd to src; subtract src from Dd, stfren CMF if Dd is less than src OMFA src, AU I3 (As) W13 L29 dil6(As, Xn) W33 L53 addr. W 25 L45 dil6(As, Xn) W35 L45 dil6	src, put quotient in low word and remainder in high word of DQ; if src: Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max max is 90% of max is
dil6(PC) W42/W114 dil6(PC) W42/W114 dil6(PC, NN) W52/W114 dil6(D low word against 0 and upper bound; cause exception 1f less than 0 (MSB=1) or greater than upper bound; 0:3/FFF); src ls 2's complement (%8000-\$FFF); or Upper bound CLR Dd BW1 L3 S (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW15 L51 dil6(Ad) BW35 L53 dil6(Ad), BW35 L73 dil6(Ad, Nn) BW46 L63 addr. W BW35 L73 dil6(Ad, Nn) BW45 L53 ddf. Nn BW25 L53 addr. W BW18 L45 dil6(CPC) BW18 L45 dil6(	src, put quotient in low word and remainder in high word of DG; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and quotient: set Z=1 if quotient is zero; se C=0 always. CLKs is max min is 90% of max ECR (See ADD Ds.dst) Logical Exclusive Of like bits give a zer bit; differing bits give a l bit EORI (See ADD S) Logical Exclusive OF like bits give a zer bit; differing bits give a l bit EORI (See ADD S) Logical Exclusive OF like bits give a zer bit; differing bits give a l bit EORI fdade CR B52 A EORI fdade CR B52 A EORI fdade CR B52 A Logical Exclusive OF immediate EORI fdade CR B52 A EORI fdade CR B52
dil6(PC) W42/W114 dil6(PC) W42/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W114 dil6(D) W52/W12 dil6(D) W12 L3 dil6(D) W13 L51 dil6(D) W13 L53 dil6(D) W13 L53 dil6(PC) W13 L27 dil6(PC) W13 L37 dil6(PC) W13 L37	src, put quotient in low word and remainder in high word of DG; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early: N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max ECR (See ADD Ds.dst) Logical Exclusive OF like bits give a zer bit; differing bits give a 1 bit EDRI (See ADD Ds.dst) Logical Exclusive OF immediate EDRI fdda8,CCR B52 A EDRI fdd8,CCR B52 A EDRI fdd8,CCR B52 A EDRI f
dil6(PC, WW 42/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 W114 W25/W114 W114 W25/W114 W114 W114 W114 W114 W114 W114 W114	<pre>src, put quotient in low word and remainder in high word of Dd; if src i 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; se C=0 always. CLKs is max; min is 90% of max Excended Exclusive Of like bits give a zer bit; differing bits give a 1 bit EOR (See ADD Ds,dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORI (See ADD Ds,dst) Logical Exclusive Of immediate EORI #da8,CCR B52 A EORI #da8,CCR B52 A E</pre>
dil6(PC, WM 42/W113 di8(PC,XM) W32/W114 #dal6 W32/W114 #dal6 W32/W114 dbal6 W32/W114 dbal6 W32/W114 dbal6 W32/W114 dbal6 W32/W114 dbal6 W32/W114 dbal6 W32/W114 dbal6 W32 dbal6	<pre>src, put quotient in low word and remainder in high word of Dd; if src i Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of EGR (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORI (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORI (See ADD Ds, dst) Logical Exclusive Of immediate EORI #da8.CCR B52 A EORI #da8.CCR B52 A EXT md bit S Extend Str LSUSive Of immediate to SR (PI) or CCR EXG Rs, Rd L3 N Exchange the content of 2 regs; use Ds, Ac not As, Dd EXT Dd WL1 S Extend Syn; fill higher byte or word with lowject codes cause 11legal instruction exceptic JMP (Ad) 6 N dil6(Ad) 12 dil6(PC) 12 dil6(PC) 12 dil6(PC) 27 dil6(PC) 47 dil6(Ad) 43 dil6(Ad) 4</pre>
dil6(PC) W42/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 #dal6 W25/W114 Check Dd low word against 0 and upper bound; cause exception if less than 0 (MSB=1) or greater than upper bound; 0C3/FFF); src is 2's complement (\$8000-\$7FFF); src is 2's complement (Ad) BW15 L51 (Ad) BW15 L51 (Ad) BW35 L53 dil6(Ad) BW35 L53 dil6(Ad) BW35 L53 dil6(As) BW35 L53 dil6(As) BW16 L22 (As) BW1 L3 As W1 L3 (As) BW6 L22 (As) BW1 L3 dil6(As) BW18 L45 dil6(As, BW18 L45 dil6(As, BW18 L45 dil6(As, C) BW35 L53 dil6(As) C) BW35 L53 dil6(As) W13 L29 dil6(As) W13 L29 dil6(As) W13 L53 dil6(As, W25 L45 dil6(As, W25 L45	<pre>src, put quotient in low word and remainder in high word of DQ; if src: Zero Exception; if dividend is larger than a word, set V=1 leave Dd unchanged, and end early; N and Y=1; set N=MSB of quotient; set Z=1 if quotient is zero; se C=0 always. CLKs is max; min is 90% of max ECR (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORI (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; differing bits give a 1 bit EORI (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; dstfering bits give a 1 bit EORI (See ADD Ds, dst) Logical Exclusive Of like bits give a zer bit; dstfering bits give a 1 bit EORI (See ADD Ds, dst) Logical Exclusive Of immediate EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EORI #da8, CCR B52 A EXT Dd WL1 S Extend Stor; fill higher byte or word with lower MSB ILLEGAL 103 N Invalid object codes cause 11legal instruction exceptic JMP (Ad) 6 J dil6(Ad) 12 dil6(PC, N) 25 Jump unconditionally JSR (Ad) 32 N dil6(Ad) 43 dil6(PC, A7) dil6(PC, A7) dil6(PC) 47 dil6(PC, A7) Jump to subroutine; push long word address of next instruction using SF</pre>

т	LEA are Ad N
W8 L27	(As) L1
W18 L58	dil6(As) L8
W25 L69	addr.W L8
W39 L79	addr.L L20
W48 L85 W39 I79	dil6(PC) L8 di8(PC,Xn) L17
W62 L93	Load effective
to	address; calculate
liate	address of operand,
	and put address in Ad
(d)+	for later use
mory	Link and allocate
CMP	stack space; save As
IN	copy new SP into As.
ranch 12	and add dil6 to SP;
o br. 25	SP then points to
true;	highest address +1 of
rue, go	stack space; dil6
ruction;	must be 2's complement of size:
w word of	use at start of
is -1, go	subroutine to reserve
int1.	on stack: undo with
); DBRA	UNLK
	LSL (See ASL) B
131	memory word by 1 bit
1132	or Dd by count in Ds
11.32	or immediate data
135	bit out goes to C and
1136	X. If (Ds)=0 or
1135	#daj=U, set flags
135	LSR (See ASL) B
1136	Logical shift right
d lona Dd	or Dd by count in Ds
rd src,	or immediate data
puotient	(3); zero fill; last
ame sign	X. If (Ds)=0 or
unless	#da3=0, set flags
n word of	ONLY; Flag code is S MOVE src dst S
By Zero	Move from src to dst;
f	see MOVE table,
larger	MOVE src.SR (PI) and
ve Dd	MOVE src,CCR A
ind end	Ds W13
tient but	(As)+ W29
d if V=1;	-(As) W41
iys. CLKs	dil6(As) W53 di8(As Xp) W66
13 70% 01	addr.W W53
104	addr.L W76
124	di8(PC.Xn) W66
125	#dal6 W33
1127	If to CCR, only low
1120	MOVE SR,dst N
1128	Dd W3
1130	(Ad) W15 (Ad)+ W15
129	- (Ad) W23
126	dil6(Ad) W35
ed word	addr.W W35
tient in	addr.L W59
hiah	MOVE AS, USP L1 (PI) N
if src is	MOVEA src,Ad N
Divide By	See Move table
larger	(Ad) W37 L75
set V=1,	-(Ad) W37 L75
hanged,	dil6(Ad) W61 L91 di8(Ad Xp) W71 L98
ned if	addr.W W61 L91
ISB of	addr.L W82 L102
zero: set	(As) W56 188
CLKs is	(As)+ W56 L88
90% of	dil6(As) W/8 LIUU di8(As Xn) W84 L104
Ds,dst) S	addr.W W78 L100
usive OR;	addr.L W92 L106
ng bits	di8(PC,Xn) W84 L104
	Move multiple reas
T) C	(D0 D7 00 07) to on
II) S usive OR	(DO-D7, AO-A7) to or from consecutive
I) S usive OR	(DO-D7, AO-A7) to or from consecutive memory locations; 1s
I) S usive OR 852 A 8 W52 A	(DO-D7, AO-A7) to or from consecutive memory locations; ls in 16-bit reg list bit map rl give regs
II) S Lusive OR 8 B52 A 8 W52 A Lusive OR	(D0-D7, A0-A7) to or from consecutive memory locations; ls in 16-bit reg list bit map rl give regs to move; LSB gives
NI) S USIVE OR 8 B52 A 8 W52 A USIVE OR 0 SR (PI)	(D0-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.
NI) S UUSIVE OR 8 B52 A 8 W52 A UUSIVE OR 0 SR (PI) 3 N	(D0-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-
I) S Usive OR B52 A W52 A Usive OR SR (PI) 3 N contents	(D0-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
NI) S UUSIVE OR B52 A W52 A UUSIVE OR O SR (PI) 3 N Contents use Ds,Ad	(DO-D7, AD-A7) to or from consecutive memory locations: 1s 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 address minus 2
II) S usive OR B52 A W52 A usive OR SR (PI) 3 N contents ise Ds,Ad LL S	(DD-D7. AD-A7) tor from consecutive memory locations; Is 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 addresse, and final
II) S usive OR B52 A W52 A usive OR O SR (PI) 3 N contents use Ds,Ad L1 S fill	(DD-D7, AD-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 Adroints to last
II) S usive OR 2 B52 A 2 W52 A 2 W52 A 2 SR (PI) 3 N ⇒ contents ise Ds.Ad IL1 S fill or word ISB	(DO-D7, AO-A7) to or from consecutive memory locations: 1s 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-AO- D7-D0 from start address minus 2 through lower addresses, and final Ad points to last word written; for other modes order is
I) S usive OR 852 A w52 A w52 A screen SR (PI) 3 N contents ise Ds,Ad L1 S fill or word SB 03 N	(DO-D7, AD-A7) to or remore nonsecutive memory locations: is 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 addresse, and final Ad points to last word written; for other modes order is D0-D7-A0-A7 from
II) S usive OR B52 A W52 A usive OR SR (PI) 3 N contents ise Ds,Ad LL S fill or word USB 03 N cct codes	(DD-D7, AD-A7) to or from consecutive memory locations; Is 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 writen; for other modes order is D0-D7-A0-A7 from start address through bibber addresses.
II) S usive OR B52 A W52 A usive OR SR (PI) 3 N c contents ise Ds,Ad IL1 S fill or word ISB 03 N ext codes i exception	(D1-D7, AD-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 D0-D7-A0-A7 from start address through higher addresses; (AS)+ mode final As
II) S usive OR B52 A W52 A Usive OR O SR (PI) 3 N contents ise Ds.Ad IL1 S fill or word ISB 03 N ct codes ise codes i	(D0-D7, AO-A7) to or from consecutive memory locations: 1s in 16-bit reg list bit map rl give regs to move: LSB gives for the consecutive list: (Ad) mede list: (Ad) mede write Order is A7-A0- D7-D0 from start addresse, and final Ad points to last word written; for other modes order is 00-D7-A0-A7 from start addresses; (As)+ mode final As points to last word
II) S usive OR & B52 A & W52 A usive OR > SR (PI) 3 N = contents ise Ds. Ad L1 S fill or word ISB O3 N exception N 25	(D0-D7, AD-A7) to or from consecutive memory locations; is 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 D0-D7-A0-A7 from start addresses; (AS)+ mode final As points to last word read plus 2; word operands read from
I) S usive OR 1852 A usive OR sSR (PI) 3 N contents see Ds.Ad L1 S fill or word SB 03 N kot codes 1 exception N 2 5 2	(DD-D7. AD-A7) to or from consecutive memory locations; Is 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 D0-D7-A0-A7 from start address through higher addresses; (AS)+ mode final As points to last word operands read from memory are sign-
I) S usive OR E 852 A E 852 A US2 A usive OR SR (PI) 3 N i contents ise DS,Ad LL S I contents ise DS,Ad LL S I contents ise CS, N N C 2 2 2 2 2 2 2	(DO-D7, AD-A7) to or from consecutive memory locations: 1s in 16-bit reg list bit map rl give regs first reg moved. MSB last: - (Ad) mode write order is A7-A0- A7-00 from start addresses, and final Ad points to last word written; for other modes order is O-D7-A0-A7 from start addresses; (As)+ mode final As (points to last word perands read from memory are sign- mextended to 32 bits; l exter word write; letter word write; letter word write; memory are sign- mether mode word; letter word write; letter word write; le
I) S usive OR E 852 A usive OR E 852 A usive OR SR (PI) 3 N contents e Ds, Ad HI S fill or word SB M LI S fill O N N exception N 2 5 5	(D0-D7, AD-A7) too from consecutive memory locations: is 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 D0-D7-A0-A7 from start addresses; (AS)+ mode final As points to last word read plus 2; word operands read from memory are sign- extended to 32 bits; l extra read cycle occurs in memory to
I) S usive OR B52 A usive OR SSR (PI) 3 N contents se Ds, Ad L1 S fill or word SB 03 N exception N 25 2 2 0 2 5 5 2 1 0 2 5 5	<pre>(DD-D7, AD-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 D0-D7-A0-A7 from start addresses; (AS)+ mode final AS points to last word read plus 2; word operands read from memory are sign- extended to 32 bits; l extra read cycle occurs in memory to reg instruction.</pre>
I) S usive OR E B52 A US2 A US2 A US2 A US3 (PI) 3 N : contents :se DS,Ad IL1 S : contents :se DS,Ad IL1 S : contents : con	(DO-D7, AD-A7) toor rom consecutive memory locations: 1s 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- 77-00 from start address minus 2 through lower addresses, and final Ad points to last word written; for other modes order is D0-D7-A0-A7 from start addresses; (As)+ mode final As points to last word operands read from memory are sign- extended to 32 bits; l extra read cycle occurs in memory too. Example: MDYEM.L #\$B007 (AS) = model
I) S usive OR E 852 A usive OR E 852 A usive OR S (PI) 3 N contents ise Ds,Ad L1 S fill or word BB S N C	(D0-D7, AD-A7) tbor remore a prosecutive memory locations: 1s 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 addresse, and final Ad points to last word written; for is 00-D7-A0-A7 from start addresses; (AS)+ mode final As points to last word read plus 2; word operands read from memory are sign- extended to 32 bits; l extra read cycle occurs in memory to reg instruction. Example: MOVEM.L #\$8002,(A5) would move D1 to (A5), then
I) S usive OR E 852 A usive OR SR (PI) 3 N contents see Ds, Ad L1 S fill contents see Ds, Ad L1 S fill CB SB CO SN SN CO SN SN CO SN CO SN CO SN CO SN SN SN CO SN CO SN SN CO SN SN CO SN SN SN SN SN SN SN SN SN SN SN SN SN	<pre>(D0-D7, AD-A7) to or from consecutive memory locations; is 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 D0-D7-A0-A7 from start addresses; (AS)+ mode final As points to last word read plus 2; word operands read from memory are sign- extended to 32 bits; l extra read cycle occurs in memory to reg instruction. Example: MOYEM. #\$8002, (A5) would move D1 to (A5), then move A7 to (A5)+2; A5</pre>
I) S usive OR E B52 A E B52 A US2 A usive OR SR (PI) 3 N contents ise DS,Ad LL S ise DS,Ad LL S SB 03 N exception N 2 5 5 tionally 2 3 4 7 0 7	(DO-D7, AO-A7) to or rhom consecutive memory locations: 1s 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 addresses, and final Ad points to last word written; for other modes order is D0-D7-A0-A7 from start addresses; (As)+ mode final As points to last word operands read from memory are sign- extended to 32 bits; l extra read cycle occurs in memory to (A5), then move A7 to (A5), then move A7 to (A5), then move A7 to (A5)+2; A5 not changed
I) S usive OR E 852 A usive OR E 852 A usive OR S (PI) 3 N contents ise DS,Ad L1 S contents ise DS,Ad L1 S contents ise DS,Ad U3 N Contents ise CS, N CON S CON S S CON S S CON S S CON S S CON S C CON S C CON S C CON S C CON S C CON S C CON S CON S C CON S C CON S C CON S C CON S C CON S C CON S C CON S C CON S C CON S CON S C CON S C CON S C CON S CON S CON S CON S CON S CON S CON S CON S CON S CON S CON S CON S C C CON S C CON S C CON S C C C S C C C C C C C C C C C S C C C C C C	<pre>(D0-D7, AD-A7) tbor memory locations: 1s in 16-bit reg list bit map rl give regs to move: LSB gives fast: cGM onde MSB last: cGM onde MSB last of the MS</pre>
I) S usive OR E 852 A usive OR E 852 A usive OR SR (PI) 3 N contents e Ds, Ad Ll S fill or word SB M to To SN N et codes l exception N 2 2 5 5 2 2 0 2 2 5 5 2 2 0 2 2 5 5 2 2 0 2 3 3 4 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	(D0-D7, AD-A7) too from consecutive memory locations; is in 16-bit reg list bit map r1 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 D0-D7-A0-A7 from start addresses; (AS)+ mode final As points to last word read plus 2; word operands read from memory are sign- extended to 32 bits; l extra read cycle occurs in memory to reg instruction. Example: MOVEM.L #\$8002.(A5) would move DI to (A5), then move A7 to (A5)+2; A5 not changed
I) S usive OR B52 A B52 A US2 A usive OR SR (PI) 3 N contents see DS,Ad LL S fill or word SB 03 N word SB 03 N word SB 03 N to contents see DS,Ad 2 S 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7	(D0-D7, AO-A7) to or memory locations: Is 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 D0-D7-A0-A7 from start address through higher addresses; (AS)+ mode final As points to last word operands read from memory are sign- extended to 32 bits; l extra read cycle occurs in memory to reg instruction. Example: MOYEM.L #\$8002, (A5) would move A1 to (A5), then move A7 to (A5)+2; A5 not changed

MOVEP Ds, dil6(Ad) W36 L74 N MOVEP dil6(As),Dd W36 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 moven upper haddress is even. bytes deven. bytes deven peripherals MOVEQ #da8.Dd Ll S Move byte of immediate (quick) data, sign-extended to 32 bits, to a Data reg (1 word instr.) MULS src.Dd and MULL src.Dd S De W116 MULU src,Dd S Ds W116 (As) W117 (As)+ W117 -(As) W119 di16(As) W120 di8(As,Xn) W121 addr.W W120 addr.W W120 addr.L W122 di16(PC) W120 di8(PC,Xn) W121 #da16 W18 Multi01y word if S W116 W117 W117 W119 W120 Multiply word in Dd Multiply word in Sc by word src; put long product in Dd; signed or unsigned NBCD Dd B3 C (Ad) B15 
 NBCD Dd
 B3 C

 (Ad)
 B15

 (Ad)+
 B15

 -(Ad)
 B23

 dil6(Ad)
 B35

 di8(Ad,Xn)
 B46

 addr.W
 B35

 addr.L
 B59

 Neasta
 B70
 diß(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 NEG (See CLR) A Negate: with extend; subtract operand and X from 0 (1) NO operation NOT (See CLR) S Logical (1's) complement Cee CLR) S Logical (1's) Complement CR (See ADDI) S Logical (0's Logical (0's Logical (0's) ACRI (See ADDI) S ACRI (See dil6(Ad) 136 di8(Ad, X) 135 addr.W L36 addr.L L60 di16(AC, X) 135 addr.L L60 di16(PC) 136 di8(PC, Xn) 155 Push effective address; calculate long word absolute address; calculate long word absolute 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 1eft 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 5 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 5 ROX. (See ASL) B ROL with extend; same flage only: flag code is 5 RDXL (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 RTE 50 (PI) Return from exception; pop SR then PC from stack

RTR 50 A	CYCLE CODES	3
RTR 50 A Return and restore: pop CGR then PC from stack: upper byte of SR is not affected RTS 31 Return from subroutine: pop PC from stack SBCD 0.6 Dd B3 C SBCD -(AD) B42 C SUBC -(AS)(Ad) B42 C Subtract 2-digit BCD src and X from dst Scc Dd cc false B1 Scc Dd cc false B1 Scc Dd cc false B1 Scc Dd cc false B1 Ste dst to SFF if cc is true, \$00 if cc is false (4) STOP #dal6 2 (PT) 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 ADD) A Subtract src from dst SUBA (See ADD) A Subtract src and x from dst (1) SUBA (See ADDA) A Subtract src and X from dst (1) SWAP Dd W1 S Swap Data reg halves (WordS) TAS Dd B1 S (Ad) B23 (Ad) B46 di8(Ad, Xn) B54 addr. W B46 addr. L STO D SUBA (See ADDA) A Subtract src and X from fater from fater	Cycle codes give instruction length in instruction length in time in CLKs. Bus read and write cycles are assumed to be for to the cLKs. The number of read and write bus cycles peach. If bus wait states occur you must add them to the CLKs. The number of read and write bus cycles per instructifi are given for this purpose.Listed in order of increasing words, increasing words, increasing, increasing, increasing, increa	> 1 3 JII (0, 10 10 10 10 10 10 10 10 10 10 10 10 10
addr.L BM99 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	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	
INSTRUCTIO	ON NOTES	
(Applicable where indic (1) Z is cleared if res unchanged otherwise. S start of operation to d especially of repeated (2) Bit number is modul operand, modulo 32 for (3) Shift or rotate cou memory operand, modulo (4) A dst operand in me it is written (5) Elans are not affec.	ated) ult is non-zero, but et or reset Z before etect zero result, operation. o B for memory reg operand nt is modulo 8 for 64 for reg operand mory is read before tor if det is Ad	
except for CMPA	111 11 1000 13 hd,	l

RTR 50 A	CY	CLE CODES	53 54	2.20.4.0 2.20.3.1
pop CCR then PC from stack; upper byte of	Cycle	e codes give	55 56	2.20.2.2 2.12+4r.3+r.0
SR is not affected RTS 31 N Return from	words	in CLKs. Bus	58 59	3.20.4.0 3.20.5.0 3.20.4.1
subroutine; pop PC from stack	read are a	and write cycles assumed to be four	60 61	3.20.3.2 3.12+4r.3.r
SBCD Ds,Dd B3 C SBCD - (As), - (Ad) B42 C	bus v	vait states occur, must add them to	62 63 64	4.20.5.0 4.20.4.1 1.22.2.2
Subtract 2-digit BCD src and X from dst Scc (See NBCD_excent):	the (	CLKs. The numbers and write bus	65 66	1.22.3.2 2.22.4.0
Scc Dd cc true B3 N Scc Dd cc false B1 N	cycle are (	es per instruction given for this	67 68	2.22.2.2 3.22.4.0
Set dst to \$FF if cc is true, \$00 if cc is falco (4)	Liste	ed in order of	70	3.22.3.0 3.22.4.1 3.14+4r.3.r
STOP #dal6 2 (PI) A Move immediate word	incre	easing CLKs, easing words,	72 73	2.24.6.0 2.24.4.2
into SR and stop executing; start if T	reads	easing s+writes, and	74 75 76	2.24.2.4 2.8+8r.2.2r 3.24.5.0
was 1 or after reset or higher priority interrupt occurs	For and 1	ordering only, s=1 r=2 are assumed.	77 78	3.24.4.1 3.16+4r.4+r.0
SUB (See ADD) A Subtract src from dst	r = #	F of regs moved	79 80	4.24.6.0 4.24.5.1
SUBA (See ADDA) N Subtract from Ad	Cvcle	e Words.CLKs.	82 83	4.24.4.2 4.16+4r.4.r 2.26.4.2
Subtract immediate SUBQ (See ADDQ) A	Code	Reads.Writes	84 85	3.18+4r.4+r.0 4.26.6.0
Subtract quick immediate data (3	2 3	1.4.1.0 2.4.0.0 1.6.1.0	86 87 88	4.26.5.1 4.26.4.2 2 12+8r 3+2r 0
bits: 1-8); I word instr. SUBX (See ADDX) A	4 5	1.8.1.0 1.6+2s.1.0	89 90	2.28.4.2 3.28.5.2
Subtract src and X from dst (1)	67	1.8.2.0 1.8.1.1	91 92	3.12+8r.3.2r 4.20+4r.5+r.0
SWAP Dd W1 S Swap Data reg halves	8 9 10	2.8.2.0 1.10.1.0 1.8+2s 1.0	95 94 95	5.28.7.0 5.28.6.1 5.28.5.2
TAS Dd B1 S (Ad) B23	11 12	1.10.2.0 2.10.2.0	96 97	1.30.5.2 3.30.5.2
(Ad)+ B23 -(Ad) B30	13	1.12.2.0 1.12.3.0	98 99	3.14+8r.3.2r 3.32.5.2 3.16.8n 4.2n 0
dil6(Ad) B46 di8(Ad,Xn) B54 addr W B46	16 17	1.12.1.2 2.12.2.0	101 102	4.32.6.2 4.16+8r.4.2r
addr.L B70 Test and set; set N	18 19	2.12.3.0 2.12.2.1	103	1.34.4.3 3.18+8t.4+2r.0
and Z per dst, then set Bit 7 of dst; not	20 21 22	5.12.5.U 1.14.2.0 1.14.3.0	105	4.20+8r.5+2r.0 5.36.7.2
read and write TRAP #da4 108 N	23 24	1.14.2.1 2.14.2.0	108 109	1.38.4.4 1.44.5.4
Cause Trap exception; like breakpoint or	25 26 27	2.14.3.0 2.14.2.1 3.14.3.0	110 111 112	1.48.6.4 2.48.6.4 1.50.6.4
software interrupt; see Exception Processing	28 29	1.16.2.0 1.16.3.0	113 114	2.52.7.4 2.54.7.4
TRAPV Trap 103 N No trap 1	30 31	1.16.2.1 1.16.4.0	115	3.56.8.4 1.70.1.0
Cause TRAPV exception if V=1 TST Dd BWULS	33 34	2.16.3.0 2.16.4.0	118 119	2.74.2.0 1.76.2.0
(Ad) BW6 L14 (Ad)+ BW6 L14	35 36	2.16.3.1 2.16.2.2	120 121	2.78.3.0 2.80.3.0 3.82.4.0
-(Ad) BWII L22 dil6(Ad) BW18 L34 di8(Ad Xp) BW25 L45	38 39	3.16.3.0 3.16.4.0	123 124	1.132.1.0 1.140.1.0
addr.W BW18 L34 addr.L BW39 L58	40 41	3.16.3.1 1.18.4.0	125 126	1.144.2.0 2.144.2.0
Test against 0, set flags only	42 43 44	1.18.2.2	127	1.146.2.0 2.148.3.0 2.150.3.0
Unlink and deallocate stack space; copy As	45 46	2.18.4.0 2.18.3.1	130 131	3.152.4.0 1.158.1.0
to SP and pop long word from stack into	47 48 49	2.18.2.2 3.18.4.0 3.18.3.1	132 133	1.162.2.0 2.162.2.0 1.164.2.0
As; use after LINK to restore stack and As	50 51	1.20.5.0 1.20.3.2	135 136	2.166.3.0 2.168.3.0
	52	2.20.3.0	157	5.1/0.4.0
INSTRUCTIO	ON N	OTES (	CON	DITION DEFS
(Applicable where indica (1) Z is cleared if resu	ated) ult is	non-zero, but CS	Carry	/ Clear C* / Set C
start of operation to de especially of repeated of	etect z poerati	ero result, F	False	e 0 er or Equal
(2) Bit number is module operand, modulo 32 for 1	o B for reg ope	rand GT	Great	(N.V)+(N*.V*) er Than
(3) Shift or rotate cour memory operand, modulo ( (4) A dst operand in mer	nt is m 64 for morv is	reg operand HI reg before HS	Highe Highe	er C*.Z* er Same C*
it is written (5) Flags are not affect	ted if	dst is Ad,	Less	or Equal Z+(N.V*)+(N*.V)
except for CMPA		LS	Lower	c or Same C+Z Than (N.V*)+(N*.V)
MOVE TABLE		MI NE PI	Minus Not E Plus	s N Equal Z* N*
dst-> aa di8(Ad,Xn	ddr.L )	T VC	True Overf	low Clear V*
dil6(Ad)/addr.W   (Ad)/(Ad)+/-(Ad)     src Dd/Ad*		FLAGS	Overf . mear N* mea	'low Set V ns AND; + means OR; ans NOT N: G & Less
MOVE.B and MOVE.W src.ds	st:		are si Lower	gned results; H & are unsigned results
Ds 1 7 19 20 As 1 7 19 20 (As) 6 15 35 40	5 40 5 40 5 59	Affected: A = X N Z V C		
(As)+ 6 15 35 44 -(As) 11 23 46 54	5 59 4 70		Shif BCD	ft and rotate only only; C is decimal
dil6(As) 18 35 59 70 di8(As,Xn) 25 46 70 7 addm W 18 35 50 70	D 80 7 86	S = - N Z VC C	C	ry or borrow (1)
addr.L 39 59 80 80 dil6(PC) 18 35 59 70	5 94 5 80	U = - N Z VC C V = Z	Rota Bit	ops only
di8(PC,Xn) 25 46 70 7 #da 8 19 40 49	7 86 9 63	W = - N Z? V? C N = None A, B. C mean al	1; N n	only means none:
MOVE.L src,dst: Ds 1 16 36 4	7 60	S, T, U, V, W m f? becomes unde	ean so fined	ome
As 1 16 36 4 (As) 14 51 73 8 (As)+ 14 51 73 8	/ 60 3 90 3 90	f set/cleared to z f set/cleared - not channed	ero accoro	ding to result
-(As) 22 65 83 8 dil6(As) 34 73 90 9	9 97 7 101		Cond	dition for flag=1,
di8(As,Xn) 45 83 97 9 addr.W 34 73 90 9 addr.L 58 80 101 20	9 105 7 101 5 107	CCR Flag Bit Flag Nam	e inst	ept as noted in truction descriptions
di16(PC) 34 73 90 9 di8(PC,Xn) 45 83 97 9	7 101 9 105	X 4 Extend N 3 Negative	Carı MSB	of result = 1
#UB32 2U 60 81 8 * MOVEA.W or MOVEA.L s:	/ 95 rc,Ad	V 1 Overflow C 0 Carry	Ovei Cari	rflow occurs ry or borrow occurs