ColdFire Reference Card

<table>
<tr><th>Mneumonic</th><th>Addressing modes</th><th>Bit Pattern</th><th>Condition Codes</th></tr>
<tr><td>ADD (L)
add two values</td><td>Source + destination -> destination
ADD <ea>y + Dx --> Dx; ADD Dx + <ea>y --> <ea>y

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table>Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr></table></td><td>1101 Register[3] Opmode[3] EA-mode[3] EA-register[3]

Opmode:
010: <ea>y + Dx --> Dx
110: Dx + <ea>y --> <ea>y</td><td>X N Z V C</td></tr>
<tr><td>ADDA (L)
add address</td><td>Source + destination -> destination
ADDA <ea>y,Ax

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table></td><td>1101 R[3] 111 EA-mode[3] EA-register[3]</td><td>not affected</td></tr>
<tr><td>ADDI (L)
add immediate</td><td>Immediate Data + destination -> destination
ADDI #<data>,Dx</td><td>0000 0110 1000 0 R[3]
upper word of immediate data[16]
lower word of immediate data[16]</td><td>X N Z V C</td></tr>
<tr><td>ADDQ (L)
add quick</td><td>Immediate Data + destination -> destination
ADDQ #<data>,<ea>x

Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr></table></td><td>0101 DATA[3] 010 EA-mode[3] EA-register[3]</td><td>X N Z V C</td></tr>
<tr><td>ADDX (L)
add with extend</td><td>Source + destination + X -> destination
ADDX Dy,Dx</td><td>1101 Register Dx[3] 11 0000 Register Dy[3]</td><td>X N Z V C</td></tr>
<tr><td>AND (L)
logical AND</td><td>Source + destination -> destination
AND Dy,<ea>x; AND <ea>y,Dx

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table>Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr></table></td><td>1100 Register[3] Opmode[3] EA-mode[3] EA-register[3]

Opmode:
110: AND Dy,<ea>x
010: AND <ea>y,Dx</td><td>N Z V=0
C=0</td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr></table></td><td></td><td></td></tr>
<tr><td>ANDI (L)
logical AND
immediate</td><td>Immediate Data + destination -> destination
ADDI #<data>,Dx</td><td>0000 0010 1000 0 Register[3]</td><td>N Z V=0
C=0</td></tr>
<tr><td>ASL, ASR
arithmetic
shift left and
right</td><td>Destination shifted by count -> destination
ASd Dy, Dx; ASd #<data>,Dx
where d is direction: L or R</td><td>1110 count register[3] dr[1] 1 0 ir[1] 00 register[3]

count register: specifies shift count
dr: direction: 0 shift right, 1 shift left
ir: 0 immediate shift count, 1 register shift</td><td>X N Z V=0
C</td></tr>
<tr><td>Bcc (B,W)
branch
conditionally</td><td>If condtion true, then PC + dn -> PC
Bcc <label>; where cc is a condition code from the condition code table</td><td>0110 condition[4] displacement[8]

or

0110 condition[4] 0000 0000 displacement[16]</td><td>not affected</td></tr>
<tr><td>BCHG
(B,L)
test bit and
change</td><td>1. (<bit number> of Destination) -> Z
2. (<bit number> of Destination) -> <bit number> of destination
BCHG Dy,<ea>x; BCHG #<data>,<ea>x

Destination (1 and 2)
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>L</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr><tr><td>L</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>1. 0000 Register[3] 101 EA-mode[3] EA-register[3]

2. 0000 1000 01 EA-mode[3] EA-register[3]
0000 0000 bit number[8]</td><td>Z</td></tr>
<tr><td>BCLR (B,L)
test bit and
clear</td><td>1. (<bit number> of Destination) -> Z
2. 0 -> <bit number> of destination
BCLR Dy,<ea>x; BCLR #<data>,<ea>x

Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>L</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr><tr><td>L</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>1. 0000 Register[3] 110 EA-mode[3] EA-register[3]

2. 0000 1000 10 EA-mode[3] EA-register[3]
0000 0000 bit number[8]</td><td>Z</td></tr>
<tr><td>BRA (B,W)
branch</td><td>PC + dn -> PC
BRA <label></td><td>0110 0000 displacement[8]

or

0110 0000 0000 0000 displacement[16]</td><td>not affected</td></tr>
<tr><td>BSET (B,L)
test bit and
set</td><td>1.TEST (<bit number> of Destination) -> Z
2. 1 -> <bit number> of destination
BSET Dy,<ea>x; BSET #<data>,<ea>x

Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>L</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr><tr><td>L</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>1. 0000 Register[3] 111 EA-mode[3] EA-register[3]

2. 0000 1000 11 EA-mode[3] EA-register[3]
0000 0000 bit number[8]</td><td>Z</td></tr>
</table>

<table>
<tr><td>BSR (B,W)
branch to subroutine</td><td>SP - 4 -> SP; Next PC -> (SP); PC + dn -> PC
BSR <label></td><td>0110 0001 displacement[8]

or

0110 0001 0000 0000
displacement[16]</td><td>not affected</td></tr>
<tr><td>BTST (B,L)
test bit</td><td>1.(<bit number> of Destination) -> Z
BTST Dy,<ea>x; BTST #<data>,<ea>x

Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>L</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr><tr><td>L</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>1. 0000 Register[3] 100 EA-mode[3] EA-register[3]

2. 0000 1000 00 EA-mode[3] EA-register[3]
0000 0000 bit number[8]</td><td>Z</td></tr>
<tr><td>CLR (B,W,L)
clear</td><td>0 -> destination

Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr></table></td><td>0100 0010 size[2] EA-mode[3] EA-register[3]

size: 00 byte, 01 word, 10 long</td><td>N=0 Z=1
V=0 C=0</td></tr>
<tr><td>CMP (L)
compare</td><td>Destination - source -> cc
CMP <ea>y, Dx

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table></td><td>1011 Register[3] 010 EA-mode[3] EA-register[3]</td><td>N Z V C</td></tr>
<tr><td>CMPA (L)
compare address</td><td>Destination - Source -> cc
CMPA <ea>y,Ax

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table></td><td>1011 Register[3] 111 EA-mode[3] EA-register[3]</td><td>N Z V C</td></tr>
<tr><td>CMPI (L)
compare immediate</td><td>Destination - immediate data -> cc
CMPI #<data>,Dx</td><td>0000 1100 1000 0 Register[3]
Upper word of immediate data[16]
Lower word of immediate data[16]</td><td>N Z V C</td></tr>
<tr><td>DIVS (W,L)
signed divide</td><td>Source / destination -> destination
1. DIVS.L <ea>y,Dx (32 Dx/ 32 -> 32q in Dx)
2. DIVS.W <ea>y,Dx (16 Dx/ 16 -> 16r:16q in Dx)

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>1. Word
1000 Register[3] 111 Ea-mode[3] EA-register[3]

2. Long
0100 1100 01 EA-mode[3] EA-register[3]
0 Register Dx[3] 1000 0000 0
Register Dx[3]</td><td>N Z V C=0</td></tr>
</table>

<table>
<tr>
<td>DIVU (W,L)
unsigned
divide</td>
<td>Source / destination -> destination
1. DIVU.L <ea>y,Dx (32 Dx/ 32 -> 32q in Dx)
2. DIVU.W <ea>y,Dx (16 Dx/ 16 -> 16r:16q in Dx)

Source
<table>
<tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr>
<tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr>
<tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td>1. Word
1000 Register[3] 011 Ea-mode[3] EA-register[3]

2. Long
0100 1100 01 EA-mode[3] EA-register[3]
0 Register Dx[3] 0000 0000 0 Register Dx[3]</td>
<td>N Z V C=0</td>
</tr>
<tr>
<td>EOR (L)
logical XOR</td>
<td>Source XOR destination -> destination
EOR Dy,<ea>x

Source
<table>
<tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr>
<tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr>
</table>
</td>
<td>1011 Register[3] 110 EA-mode[3] EA-register[3]</td>
<td>N Z V=0
C=0</td>
</tr>
<tr>
<td>EORI (L)
logical XOR
immediate</td>
<td>Immediate data XOR destination -> destination
EORI #<data>,Dx</td>
<td>0000 1010 1000 0 Register[3]
Upper word of immediate data[16]
Lower word of immediate data[16]</td>
<td>N Z V=0
C=0</td>
</tr>
<tr>
<td>EXT, EXTB
(W,L)
sign extend</td>
<td>Destination sign - extended -> destination
EXT.W Dx (extend byte to word)
EXT.L Dx (extend word to longword)
EXTB.L Dx (extend byte to longword)</td>
<td>0100 100 Opmode[3] 000 Register[3]

Opmode:
010: EXT.W Dx
011: EXT.L Dx
111: EXTB.L Dx</td>
<td>N Z V=0
C=0</td>
</tr>
<tr>
<td>JMP
(unsized)
jump</td>
<td>Target address -> PC
JMP <ea>y

Destination
<table>
<tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr>
<tr><td></td><td></td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td></tr>
</table>
</td>
<td>0100 1110 11 EA-mode[3] EA-register[3]</td>
<td>not affected</td>
</tr>
<tr>
<td>JSR
(unsized)
jump to
subroutine</td>
<td>SP-4 -> SP; Next PC -> (SP); Target address -> PC

Destination
<table>
<tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr>
<tr><td></td><td></td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td></tr>
</table>
</td>
<td>0100 1110 10 EA-mode[3] EA-register[3]</td>
<td>not affected</td>
</tr>
<tr>
<td>LEA (L)
load
effective
address</td>
<td>Effective address -> destination
LEA <ea>y, Ax

Destination
<table>
<tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr>
<tr><td></td><td></td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td></tr>
</table>
</td>
<td>0100 Register[3] 111 EA-mode[3] EA-register[3]</td>
<td>not affected</td>
</tr>
<tr>
<td>LINK (W)
link and</td>
<td>SP - 4 -> SP; Ax -> (SP); SP -> Ax; SP + d16 -> SP
LINK Ax,#<displacement></td>
<td>0100 1110 0101 0 Register[3]</td>
<td>not affected</td>
</tr>
</table>

<table>
<tr><td>allocate</td><td></td><td></td><td></td></tr>
<tr><td>LSL, LSR (L)
logical shift left and right</td><td>Destination shifted by count -> destination
LSd Dy,Dx; LSD #<data>,Dx (where d is direction: L or R)</td><td>1111 count/register[3] dr[1] 10 ir[1] 01 register[3]</td><td>X N Z V=0 C</td></tr>
<tr><td>MOVE, MOVEA (B,W,L)
move</td><td>Source -> Destination
Move <ea>y,<ea>x; MOVEA <ea>y,Ax

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr></table>Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table>Restricted modes
<table><tr><th>Source addressing mode</th><th>Destination Addressing mode</th></tr><tr><td>Dy, Ay, (Ay), (Ay)+,-(Ay)</td><td>All possible</td></tr><tr><td>(d16,Ay), (d16,PC)</td><td>All except (d8,Ay,Xi), (xxx).W, (xxx).L</td></tr><tr><td>(d8,Ay,Xi), (d8,PC,Xi), (xxx).W, xxx.(L), #<xxx></td><td>All except (d8,Ay,Xi), (d16,Ay), (xxx).W, (xxx).L</td></tr></table></td><td>00 Size[2] Destination-Register[3] Destination-Mode[3] Source-Mode[3] Source-Register[3]

Size: 01 byte; 11 word; 10 long</td><td>N Z V=0 C=0</td></tr>
<tr><td>MOVE from CCR (W)
move from condition code register</td><td>CCR -> destination
MOVE CCR,Dx</td><td>0100 0010 1100 0 Register[3]</td><td>not affected</td></tr>
<tr><td>MOVE to CCR (W)
move to condition code register</td><td>Source -> CCR
MOVE Dy,CCR; MOVE #<data>,CCR

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td></tr></table></td><td>0100 0100 11 EA-mode[3] EA-register[3]</td><td>X N Z V C</td></tr>
<tr><td>MOVEM (L)
move multiple registers</td><td>Registers -> destination or Source -> Registers
MOVEM <list>,<ea>x; MOVEM <ea>y,<list>

Register to Memory or Memory to Register
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td></td><td></td><td>•</td><td></td><td></td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>0100 1 dr[1] 0011 EA-mode[3] EA-register[3]
registers to transfer/modify bit mask[16]

dr: 0 register to memory; 1 memory to register</td><td>not affected</td></tr>
<tr><td>MOVEQ (L)</td><td>Immediate data -> destination
MOVEQ #<data>,Dx</td><td>0111 Register[3] 0 Data[8]

Data is 8 bits, sign-extended to a long</td><td>N Z V=0 C=0</td></tr>
</table>

<table>
<tr><td>move quick</td><td></td><td></td><td></td></tr>
<tr><td>MULS (W,L) signed multiply</td><td>Source x destination -> destination
1. MULS.L <ea>y,Dx (32 x 32 -> 32)
2. MULS.W <ea>y,Dx (16 x 16 -> 32)

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>1. Word
1100 Register[3] 111 Ea-mode[3] EA-register[3]

2. Long
0100 1100 00 EA-mode[3] EA-register[3]
0 Register Dx[3] 1000 0000 0000</td><td>N Z V=0
C=0</td></tr>
<tr><td>MULU (W,L) unsigned multiply</td><td>Source x destination -> destination
1. MULU.L <ea>y,Dx (32 x 32 -> 32)
2. MULU.W <ea>y,Dx (16 x 16 -> 32)

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>1. Word
1100 Register[3] 011 Ea-mode[3] EA-register[3]

2. Long
0100 1100 00 EA-mode[3] EA-register[3]
0 Register Dx[3] 0000 0000 0000</td><td>N Z V=0
C=0</td></tr>
<tr><td>NEG (L) negate</td><td>0 - destination -> destination
NEG Dx</td><td>0100 0100 1000 0 Register[3]</td><td>X N Z V C</td></tr>
<tr><td>NEGX (L) negate with extend</td><td>0 - destination - X -> destination
NEGX Dx</td><td>0100 0000 1000 0 Register[3]</td><td>X N Z V C</td></tr>
<tr><td>NOP (unsized) no operation</td><td>No operation
NOP</td><td>0100 1110 0111 0001 = $4E71</td><td>not affected</td></tr>
<tr><td>NOT (L) logical complement</td><td>~ destination -> destination
NOT Dx</td><td>0100 0110 1000 0 Register[3]</td><td>N Z V=0
C=0</td></tr>
<tr><td>OR (L) logical inclusive-OR</td><td>Source OR destination -> destination
OR Dy,<ea>x; OR <ea>y,Dy

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table>Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr></table></td><td>1000 Register[3] Opmode[3] EA-mode[3] EA-register[3]

Opmode:
110: OR Dy,<ea>x
010: OR <ea>y,Dy</td><td>N Z V=0
C=0</td></tr>
<tr><td>ORI (L) logical inclusive-OR immediate</td><td>Immediate data OR destination -> destination
ORI #<data>,Dx</td><td>0000 0000 1000 0 Register[3]
Upper word of immediate data[16]
Lower word of immediate data[16]</td><td>N Z V=0
C=0</td></tr>
</table>

<table>
<tr>
<td>PEA (L)
push effective address onto stack</td>
<td>SP - 4 -> SP; <ea>y -> (SP)
PEA <ea>y

Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td></td><td></td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td></tr></table></td>
<td>0100 1000 01 EA-mode[3] EA-register[3]</td>
<td>not affected</td>
</tr>
<tr>
<td>REMS (L)
signed divide remainder (if Dw=Dx then DIVU is executed instead)</td>
<td>Destination / Source -> Remainder
1. REMU.L <ea>y,Dw:Dx (32 Dx/ 32 -> 32r in Dw)

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Long
0100 1100 01 EA-mode[3] EA-register[3]
0 Register Dx[3] 1000 0000 0 Register Dw[3]</td>
<td>N Z V C=0</td>
</tr>
<tr>
<td>REMU (L)
unsigned divide remainder (if Dw=Dx then DIVU is executed instead)</td>
<td>Destination / Source -> Remainder
1. REMU.L <ea>y,Dw:Dx (32 Dx/ 32 -> 32r in Dw)

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Long
0100 1100 01 EA-mode[3] EA-register[3]
0 Register Dx[3] 0000 0000 0 Register Dw[3]</td>
<td>N Z V C=0</td>
</tr>
<tr>
<td>RTS (unsized)
return from subroutine</td>
<td>(SP) -> PC; SP + 4 -> SP
RTS</td>
<td>0100 1110 0111 0101 = $4E75</td>
<td>not affected</td>
</tr>
<tr>
<td>SUB (L)
subtract</td>
<td>Destination - source -> destination
SUB Dy,<ea>x; SUB <ea>y,DY

Source
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table>Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr></table></td>
<td>1001 Register[3] Opmode[3] EA-mode[3] EA-register[3]

Opmode:
010: SUB Dy,<ea>x
110: SUB <ea>y,DY</td>
<td>X N Z V C</td>
</tr>
<tr>
<td>SUBA (L)
subtract address</td>
<td>Destination - source -> destination
SUBA <ea>y,Ax

Destination
<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table></td>
<td>1001 Register[3] 111 EA-mode[3] EA-register[3]</td>
<td>not affected</td>
</tr>
<tr>
<td>SUBI (L)
subtract immediate</td>
<td>Destination - immediate data -> destination
SUBI #<data>,Dx</td>
<td>0000 0100 1000 0 Register[3]
Upper word of immediate data[16]
Lower word of immediate data[16]</td>
<td>X N Z V C</td>
</tr>
</table>

<table>
<tr><td>SUBQ (L)
subtract quick</td><td>Destination - immediate data -> destination
SUBQ #<data>,<ea>x

Destination<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td></td></tr></table></td><td>0101 data[3] 110 EA-mode[3] EA-register[3]</td><td>X N Z V C</td></tr>
<tr><td>SUBX (L)
subtract with extend</td><td>Destination - source -> destination
SUBX Dy,Dx</td><td>1001 Dx[3] 1 1000 0 Dy[3]</td><td>X N Z V C</td></tr>
<tr><td>Scc (B)
set according to condition</td><td>If condition true then 1s -> destination
else 0s -> destination
Scc Dx (cc is condition code, see condition codes table)</td><td>0101 Condtion[3] 1100 0 Register[3]</td><td>not affected</td></tr>
<tr><td>SWAP (W)
swap register words</td><td>Register bits 31 to 16 <--> Register bits 15 to 0
SWAP Dx</td><td>0100 1000 0100 0 Register[3]</td><td>N Z V=0
C=0</td></tr>
<tr><td>TRAP
(unsized)
trap</td><td>TRAP #<vector></td><td>0100 1110 0100 Vector[4]</td><td>not affected</td></tr>
<tr><td>TST
(B,W,L)
test operand</td><td>Destination tested -> condition codes
TST <ea>y

Destination<table><tr><th>Dn</th><th>An</th><th>(An)</th><th>(An)+</th><th>-(An)</th><th>(d16,An)</th><th>(d8,An,Xi)</th><th>(d16,PC)</th><th>(d8,PC,Xi)</th><th>(xxx).W</th><th>(xxx).L</th><th>#<xxx></th></tr><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr></table></td><td>0100 1010 Size[2] EA-mode[3] EA-register[3]</td><td>N Z V=0
C=0</td></tr>
<tr><td>UNLK
(unsized)
unlink</td><td>Ax -> SP; (SP) -> Ax; SP+4 -> SP
UNLK Ax</td><td>0100 1110 0101 1 Register[3]</td><td>not affected</td></tr>
</table>

Effective Addressing Modes

Addressing Modes	Syntax	Mode Field	Register Field
Data Register Direct	**Dn**	000	register number
Address Register Direct	**An**	001	register number
Address Register Indirect	**(An)**	010	register number
Postincrement Address Register Indirect	**(An)+**	011	register number
Predecrement Address Register Indirect	**-(An)**	100	register number
8-bit index displacement Address Register Indirect	**(d8,An,Xi)**	110	register number
Program counter indirect with displacement	**(d16,PC)**	111	010
Program counter indirect with index 8-bit displacement	**(d8,PC,Xi)**	111	011
Absolute Short	**(xxx).W**	111	000

Absolute Long	**(xxx).L**	111	001
Immediate	**#<xxx>**	111	100

Condition Codes Register

The CCR is the least significant byte of the status register (SR). Bits 4-0 are the condition code flags:

- X: Extend condition code (bit 4)
- N: Negative condition code (bit 3)
- Z: Zero condition code (bit 2)
- V: Overflow condition code (bit 1)
- C: Carry condition code (bit 0)

Condition Codes Table

Mnemonic	Condition	Encoding	Test
T (**BRA**)	True	0000	1
F (**BSR**)	False	0001	0
HI	High	0010	C' * Z'
LS	Lower or Same	0011	C + Z
CC (HI)	Carry Clear	0100	C'
CS (LO)	Carry Set	0101	C
NE	Not Equal	0110	Z'
EQ	Equal	0111	Z
VC	Overflow Clear	1000	V'
VS	Overflow Set	1001	V
PL	Plus	1010	N'
MI	Minus	1011	N
GE	Greater or Equal	1100	N * V + N' * V'
LT	Less Than	1101	N * V' + N' * V
GT	Greater Than	1110	N * V * Z' + N' * V' * Z'
LE	Less or Equal	1111	Z + N * V' + N' * V
A prime, ' indicates the complement.			
A plus, + indicates an or operation.			
A star, * indicates an and operation.			

ASCII Chart

```
    0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
0  NUL SOH STX ETX EOT ENQ ACK BEL BS  HT  LF  VT  FF  CR  SO  SI
1  DLE DC1 DC2 DC3 DC4 NAK SYN ETB CAN EM  SUB ESC FS  GS  RS  US
2   SP  !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /
3   0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
4   @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
```

```
5   P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
6   `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
7   p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ DEL
```

Notes

Supervisor (privileged) instructions not included in chart.
While we have tried to be accurate, there are no guarantees that this chart is completely correct. If you find an error, please send mail to Roger.

Developed and produced by Ashish Mishra, E&CE Dept., U of W, 1999. Updated E. Praetzel 2004

Added DIVU, DIVS, REMS, REMU for 5307 processor. The V4 processors add a lot more.