MC680x0

COLLABORATORS

	TITLE : MC680x0	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 24, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

MC680x0

1.1 MC680x0 Reference

```
Instrction types:
  @{ "Move Instructions            " link MovesInst }
  @{ "Mathematical Instructions    " link Math  }
  @{ "Logic Instructions           " link Logic }
  @{ "Flow Control Instructions    " link Flow  }
  @{ "Shift and Rotate Instructions" link Shift }
  @{ "Bit Manipulation Instructions" link Bit   }
  @{ "Miscellaneous Instructions   " link Misc  }
```

1.2 moveins

```
Normal:
  @{ "MOVE " link Move  } @{ "MOVEA" link Movea } @{ "MOVEC" link Movec } @{ " ↩
     MOVEM" link Movem } @{ "MOVEP" link Movep } @{ "MOVEQ" link Moveq } @{ "MOVES ↩
     " link Moves }

  @{ "MOVE to CCR  " link MoveToCCR    } @{ "MOVE from CCR" link MoveFromCCR  }
  @{ "MOVE to SR   " link MoveToSR     } @{ "MOVE from SR " link MoveFromSR   }
  @{ "MOVE USP     " link MoveUSP  }

Special:
  @{ "LEA  " link Lea   } @{ "PEA  " link Pea   }
```

1.3 math

```
Integer:
  @{ "ADD " link Add  } @{ "ADDI" link Addi } @{ "ADDQ" link Addq } @{ "ADDA" link ↩
      Adda } @{ "SUB " link Sub  } @{ "SUBI" link Subi } @{ "SUBQ" link Subq } @{  ↩
     "SUBA" link Suba }
  @{ "DIVS" link Divs } @{ "DIVU" link Divu } @{ "MULS" link Mul  } @{ "MULU" link ↩
      Mul  }
  @{ "EXT " link Ext  }
  @{ "NEG " link Neg  }
```

```
  @{ "CMP " link Cmp  } @{ "CMPI" link Cmpi } @{ "CMPA" link Cmpa }

Multi-Precision Integer:
  @{ "ABCD" link Abcd } @{ "ADDX" link Addx } @{ "SBCD" link Sbcd } @{ "SUBX" link ↩
      Subx }
  @{ "NBCD" link Nbcd }
  @{ "CMPM" link Cmpm }
```

1.4 logic

```
Bit-wise:
  @{ "AND " link And } @{ "ANDI" link Andi } @{ "ANDI to CCR" link AndiCCR } @{ " ↩
     ANDI to SR" link AndiSR }
  @{ "EOR " link Eor } @{ "EORI" link Eori } @{ "EORI to CCR" link EoriCCR } @{ " ↩
     EORI to SR" link EoriSR }
  @{ "OR  " link Or  } @{ "ORI " link Ori  } @{ "ORI  to CCR" link OriCCR } @{ " ↩
     ORI  to SR" link OriSR }

Byte-wise:
  @{ "TST " link Tst } @{ "TAS " link Tas }
  @{ "Scc " link Scc }
  @{ "NOT " link Not }
```

1.5 flow

```
Subroutine:
  @{ "BSR    " link Bsr     } @{ "JSR    " link Jsr     }
  @{ "RTS    " link Rts     } @{ "RTD    " link Rtd     } @{ "RTR    " link Rtr  ↩
         }

Local:
  @{ "BRA    " line Bra     } @{ "JMP    " link Jmp     }

Conditional:
  @{ "Bcc    " link Bcc     } @{ "DBcc   " link Dbcc    }

System:
  @{ "ILLEGAL" link Illegal } @{ "BKPT   " link Bkpt    }
  @{ "TRAP   " link Trap    } @{ "TRAPV  " link Trapv   }
  @{ "RESET  " link Reset   } @{ "STOP   " link Stop    }
  @{ "RTE    " link Rte     }
```

1.6 misc

```
Stack Frame Maintainance:
  @{ "LINK" link Link } @{ "UNLK" link Unlk }

Processor:
  @{ "NOP " link Nop  }
```

1.7 shift

```
Shifts:
  @{ "ASd " link ASd  } @{ "LSd " link Lsd  }

Rotates:
  @{ "ROd " link Rod  } @{ "ROXd" link Roxd }
```

1.8 bit

```
Single Bit:
  @{ "BCHG  " link Bchg   } @{ "BCLR  " link Bclr   } @{ "BSET  " link Bset   } @{ ↩
       "BTST  " link Btst   }

Bit Field:
  @{ "BFCHG " link BFCHG  } @{ "BFCLR " link Bfclr  } @{ "BFSET " link Bfset  }
  @{ "BFINS " link Bfins  } @{ "BFEXTS" link bfexts } @{ "BFEXTU" link Bfextu }
  @{ "BFFFO " link Bfffo  } @{ "BFTST " link Bftst  }
```

1.9 abcd

```
NAME
  ABCD -- Add binary coded decimal

SYNOPSIS
  ABCD  Dy,Dx
  ABCD  -(Ay),-(Ax)

  Size = (Byte)

FUNCTION
  Adds the source operand to the destination operand along with
  the extend bit, and stores the result in the destination location.  The
  addition is performed using binary coded decimal arithmetic.  The
  operands, which are packed BCD numbers, can be addressed in two dif-
  ferent ways:

  1. Data register to data register: The operands are contained in the
     data registers specified in the instruction.

  2. Memory to memory: The operands are addressed with the predecrement
     addressing mode using the address registers specified in the
     instruction.

  This operation is a byte operation only.

  Normally the Z condition code bit is set via programming before the
  start of an operation.  That allows successful tests for zero results
  upon completion of multiple-precision operations.

RESULT
  X - Set the same as the carry bit.
```

```
  N - Undefined
  Z - Cleared if the result is non-zero. Unchanged otherwise.
  V - Undefined
  C - Set if a decimal carry was generated.  Cleared otherwise.

SEE ALSO
  @{ "ADD " link Add  } @{ "ADDI" link Addi } @{ "ADDQ" link Addq }
  @{ "ADDX" link Addx }
  @{ "SUB " link Sub  } @{ "SUBI" link Subi } @{ "SUBQ" link Subq }
  @{ "SBCD" link Sbcd } @{ "SUBX" link Subx }
```

1.10 add

```
NAME
  ADD -- Add integer

SYNOPSIS
  ADD <ea>,Dn
  ADD Dn,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Adds the source operand to the destination operand using
  binary addition, and stores the result in the destination location.
  The size of the operation may be specified as byte, word, or long.
  The mode of the instruction indicates which operand is the source and
  which is the destination as well as the operand size.

RESULT
  X - Set the same as the carry bit.
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if an overflow is generated.  Cleared otherwise.
  C - Set if a carry is generated.  Cleared otherwise.

SEE ALSO
  @{ "ADDI" link Addi } @{ "ADDQ" link Addq } @{ "ADDX" link Addx }
  @{ "SUB " link Sub  } @{ "SUBI" link Subi } @{ "SUBQ" link Subq }
  @{ "SUBX" link Subx }
```

1.11 adda

```
NAME
  ADDA -- Add address

SYNOPSIS
  ADDA  <ea>,An

  Size = (Word, Long)

FUNCTION
```

```
  Adds the source operand to the destination address register,
  and stores the result in the destination address register.  The size
  of the operation may be specified as word or long.  The entire
  destination operand is used regardless of the operation size.

RESULT
  None.

SEE ALSO
  @{ "ADDQ" link Addq } @{ "SUBQ" link Subq } @{ "SUBA" link Suba }
```

1.12 addi

```
NAME
  ADDI -- Add immediate

SYNOPSIS
  ADDI  #<data>,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Adds the immediate data to the destination operand, and
  stores the result in the destination location.  The size of the
  operation may be specified as byte, word, or long.  The size of the
  immediate data matches the operation size.

RESULT
  X - Set the same as the carry bit.
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if an overflow is generated.  Cleared otherwise.
  C - Set if a carry is generated.  Cleared otherwise.

SEE ALSO
  @{ "ADD " link Add  } @{ "ADDQ" link Addq } @{ "ADDX" link Addx }
  @{ "SUB " link Sub  } @{ "SUBI" link Subi } @{ "SUBQ" link Subq }
  @{ "SUBX" link Subx }
```

1.13 addq

```
NAME
  ADDQ -- Add 3-bit immediate quick


SYNOPSIS
  ADDQ  #<data>,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Adds the immediate value of 1 to 8 to the operand at the
```

```
  destination location.  The size of the operation may be specified as
  byte, word, or long.  When adding to address registers, the condition
  codes are not altered, and the entire destination address register is
  used regardless of the operation size.

RESULT
  X - Set the same as the carry bit.
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if an overflow is generated.  Cleared otherwise.
  C - Set if a carry is generated.  Cleared otherwise.

SEE ALSO
  @{ "ADD " link Add  } @{ "ADDI" link Addi }
  @{ "SUB " link Sub  } @{ "SUBI" link Subi } @{ "SUBQ" link Subq }
```

1.14 addx

```
NAME
  ADDX -- Add integer with extend

SYNOPSIS
  ADDX  Dy,Dx
  ADDX  -(Ay),-(Ax)

  Size = (Byte, Word, Long)

FUNCTION
  Adds the source operand to the destination operand along with
  the extend bit, and stores the result in the destination location.  The
  addition is performed using binary coded decimal arithmetic.  The
  operands, which are packed BCD numbers, can be addressed in two dif-
  ferent ways:

  1. Data register to data register: The operands are contained in the
     data registers specified in the instruction.

  2. Memory to memory: The operands are addressed with the predecrement
     addressing mode using the address registers specified in the
     instruction.

  The size of operation can be specified as byte, word, or long.

  Normally the Z condition code bit is set via programming before the
  start of an operation.  That allows successful tests for zero results
  upon completion of multiple-precision operations.

RESULT
  X - Set the same as the carry bit.
  N - Set if the result is negative.  Cleared otherwise.
  Z - Cleared if the result is non-zero. Unchanged otherwise.
  V - Set if an overflow is generated.  Cleared otherwise.
  C - Set if a carry is generated.  Cleared otherwise.

SEE ALSO
```

```
  @{ "ADD" link Add  } @{ "ADDI" link Addi }
  @{ "SUB" link Sub  } @{ "SUBI" link Subi } @{ "SUBX" link Subx }
```

1.15 and

```
NAME
  AND -- Logical AND

SYNOPSIS
  AND <ea>,Dn
  AND Dn,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Performs a bit-wise AND operation with the source operand and
  the destination operand and stores the result in the destination.  The
  size of ther operation can be specified as byte, word, or long.  The
  contents of an address register may not be used as an operand.

RESULT
  X - Not affected
  N - Set if the most-significant bit of the result was set. Cleared
      otherwise.
  Z - Set if the result was zero.  Cleared otherwise.
  V - Always cleared.
  C - Always cleared.

SEE ALSO
  @{ "ANDI" link Andi }
```

1.16 andi

```
NAME
  ANDI -- Logical AND immediate

SYNOPSIS
  ANDI  #<data>,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Performs a bit-wise AND operation with the immediate data and
  the destination operand and stores the result in the destination.  The
  size of ther operation can be specified as byte, word, or long.  The
  size of the immediate data matches the operation size.

RESULT
  X - Not affected
  N - Set if the most-significant bit of the result was set. Cleared
      otherwise.
  Z - Set if the result was zero.  Cleared otherwise.
```

```
  V - Always cleared.
  C - Always cleared.

SEE ALSO
  @{ "AND " link And } @{ "ANDI to CCR" link AndiCCR } @{ "ANDI to SR" link AndiSR ↩
      }
```

1.17 andiccr

```
NAME
  ANDI to CCR -- Logical AND immediate to condition code register

SYNOPSIS
  ANDI  #<data>,CCR

  Size = (Byte)

FUNCTION
  Performs a bit-wise AND operation with the immediate data and
  the lower byte of the status register.

RESULT
  X - Cleared if bit 4 of immed. operand is zero. Unchaned otherwise.
  N - Cleared if bit 3 of immed. operand is zero. Unchaned otherwise.
  Z - Cleared if bit 2 of immed. operand is zero. Unchaned otherwise.
  V - Cleared if bit 1 of immed. operand is zero. Unchaned otherwise.
  C - Cleared if bit 0 of immed. operand is zero. Unchaned otherwise.

SEE ALSO
  @{ "AND " link And } @{ "ANDI" link Andi } @{ "ANDI to SR" link AndiSR }
```

1.18 andisr

```
NAME
  ANDI to SR -- Logical AND immediate to status register (privileged)

SYNOPSIS
  ANDI  #<data>,SR

  Size = (Word)

FUNCTION
  Performs a bit-wise AND operation with the immediate data and
  the status register.  All implemented bits of the status register are
  affected.

RESULT
  X - Cleared if bit 4 of immed. operand is zero. Unchaned otherwise.
  N - Cleared if bit 3 of immed. operand is zero. Unchaned otherwise.
  Z - Cleared if bit 2 of immed. operand is zero. Unchaned otherwise.
  V - Cleared if bit 1 of immed. operand is zero. Unchaned otherwise.
  C - Cleared if bit 0 of immed. operand is zero. Unchaned otherwise.
```

```
SEE ALSO
  @{ "AND " link And } @{ "ANDI" link Andi } @{ "ANDI to CCR" link AndiCCR }
```

1.19 asd

```
NAME
  ASL, ASR -- Arithmetic shift left and arithmetic shift right

  ASd #<data>,Dy
  ASd <ea>
  where 'd' is the direction, L or R

  Size = (Byte, Word, Long)

FUNCTION


RESULT
  X - Set according to the list bit shifted out of the operand.
      Unaffected for a shift count of zero.
  N - Set if the most-significant bit of the result is set.  Cleared
      otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if the most significant bit is changed at any time during the
      shift operation.  Cleared otherwise.
  C - Set according to the list bit shifted out of the operand.
      Cleared for a shift count of zero.

SEE ALSO
  @{ "ROd " link Rod } @{ "ROXd" link Roxd }
```

1.20 bcc

```
NAME
  Bcc -- Conditional branch

SYNOPSIS
  Bcc <label>

  Size = (Byte, Word)

FUNCTION

  CC  carry clear C'    LS low or same    C+Z
  CS  carry set   C   LT less than    N'·V+N'V
  EQ  equal   Z   MI  minus   N
  GE  greater or equal  N·V+N'V'  NE  not equal   Z'
  GT  greater than  N·V·Z'+N'V'·Z'  PL  plus    N'
  HI  high    C'·Z'   VC  overflow clear  V'
  LE  less or equal Z+N'·V+N'V  VS  overflow set  V
```

```
RESULT
  None.

SEE ALSO
  @{ "BRA " link Bra } @{ "DBcc" link Dbcc } @{ "Scc " link Scc }
```

1.21 bchg

```
NAME
  BCHG -- Bit change

SYNOPSIS
  BCHG  Dn,<ea>
  BCHG  #<data>,<ea>

  Size = (Byte, Long)

FUNCTION
  Tests a bit in the destination operand and sets the Z
  condition code appropriately, then inverts the bit in the destination.
  If the destination is a data register, any of the 32 bits can be
  specifice by the modulo 32 number.  When the distination is a memory
  location, the operation must be a byte operation, and therefore the
  bit number is modulo 8.  In all cases, bit zero is the least
  significant bit.  The bit number for this operation may be specified
  in either of two ways:

  1. Immediate -- The bit number is specified as immediate data.
  2. Register  -- The specified data register contains the bit number.

RESULT
  X - not affected
  N - not affected
  Z - Set if the bit tested is zero.  Cleared otherwise.
  V - not affected
  C - not affected

SEE ALSO
  @{ "BCLR " link Bclr  } @{ "BSET " link Bset  } @{ "BTST " link Btst  }
  @{ "EOR  " link Eor   } @{ "BFCHG" link Bfchg }
```

1.22 bclr

```
NAME
  BCLR -- Bit clear

SYNOPSIS
  BCLR  Dn,<ea>
  BCLR  #<data>,<ea>

  Size = (Byte, Long)
```

```
FUNCTION
  Tests a bit in the destination operand and sets the Z
  condition code appropriately, then clears the bit in the destination.
  If the destination is a data register, any of the 32 bits can be
  specifice by the modulo 32 number.  When the distination is a memory
  location, the operation must be a byte operation, and therefore the
  bit number is modulo 8.  In all cases, bit zero is the least
  significant bit.  The bit number for this operation may be specified
  in either of two ways:

  1. Immediate -- The bit number is specified as immediate data.
  2. Register  -- The specified data register contains the bit number.

RESULT
  X - not affected
  N - not affected
  Z - Set if the bit tested is zero.  Cleared otherwise.
  V - not affected
  C - not affected

SEE ALSO
  @{ "BCHG " link Bchg  } @{ "BSET " link Bset  } @{ "BTST " link Btst  }
  @{ "AND  " link And   } @{ "BFCLR" link Bfclr }
```

1.23 bkpt

```
NAME
  BKPT -- Break-point

SYNOPSIS
  BKPT  #<data>

FUNCTION
  This instruction is used to support the program breakpoint
  function for debug monitors and real-time hardware emulators, and
  the operation will be dependent on the implementation.  Execution of
  this instruction will cause the MC68010 to run a breakpoint
  acknowledge bus cycle and zeros on all address lines, but an
  MC68020 will place the immediate data on lines A2, A3, and A4, and
  zeros on lines A0 and A1.

  Whether the breakpoint acknowledge cycle is terminaled with
  (DTACK)', (BERR)', or (VPA)' the processor always takes an illegal
  instruction exception.  During exception processing, a debug monitor
  can distinguish eight different software breakpoints by decoding the
  field in the BKPT instruction.

  For the MC68000 and the MC68HC000, this instruction causes an illegal
  instruction exception, but does not run the breakpoint acknowledge
  bus cycle.

  There are two possible responses on an MC68020: normal and exception.
  The normal response is an operation word (typically the instruction
  the BKPT originally replaced) on the data lines with the (DSACKx)'
  signal asserted.  The operation word is the executed in place of the
```

```
  breakpoint instruction.

  For the exception response, a bus error signal will cause the MC68020
  to take an illegal instruction execpetion, just as an MC68010 or
  MC68000 would do.

RESULT
  None.

SEE ALSO
  @{ "ILLEGAL   " link Illegal    }
  @{ "Exceptions" link Exceptions }
```

1.24 bra

```
NAME
  BRA -- Unconditional branch

SYNOPSIS
  BRA <label>

  Size = (Byte, Word)
  Size = (Byte, Word, Long) (68020+)

FUNCTION
  Program execution continues at location (PC) + displacement.
  The PC contains the address of the instruction word of the BRA
  instruction puls two.  The displacement is a twos complement integer
  that represents the relative distance in bytes from the current PC
  to the destination PC.

RESULT
  None.

SEE ALSO
  @{ "JMP" link Jmp } @{ "Bcc" link Bcc }
```

1.25 bset

```
NAME
  BSET -- Bit set

SYNOPSIS
  BCLR  Dn,<ea>
  BCLR  #<data>,<ea>

  Size = (Byte, Long)

FUNCTION
  Tests a bit in the destination operand and sets the Z
  condition code appropriately, then sets the bit in the destination.
  If the destination is a data register, any of the 32 bits can be
```

```
  specifice by the modulo 32 number.  When the distination is a memory
  location, the operation must be a byte operation, and therefore the
  bit number is modulo 8.  In all cases, bit zero is the least
  significant bit.  The bit number for this operation may be specified
  in either of two ways:

  1. Immediate -- The bit number is specified as immediate data.
  2. Register  -- The specified data register contains the bit number.

RESULT
  X - not affected
  N - not affected
  Z - Set if the bit tested is zero.  Cleared otherwise.
  V - not affected
  C - not affected

SEE ALSO
  @{ "BCHG " link Bchg  } @{ "BCLR " link Bclr  } @{ "BTST " link Btst  }
  @{ "OR   " link Or    } @{ "BFSET" link Bfset } @{ "BFINS" link Bfins }
```

1.26 bsr

```
NAME
  BSR -- Branch to subroutine

SYNOPSIS
  BSR <label>

  Size = (Byte, Word)
  Size = (Byte, Word, Long) (68020+)

FUNCTION
  Pushes the long word address of the instruction immediately
  following the BSR instruction onto the system stack.  The PC contains
  the address of the instruction word plus two.  Program execution
  continues at location (PC) + displacement.

RESULT
  None.

SEE ALSO
  @{ "JSR" link Jsr } @{ "BRA" link Bra }
  @{ "RTS" link Rts } @{ "RTD" link Rts } @{ "RTR" link Rts }
```

1.27 btst

```
NAME
  BTST -- Bit test

SYNOPSIS
  BTST  Dn,<ea>
  BTST  #<data>,<ea>
```

```
  Size = (Byte, Long)

FUNCTION
  Tests a bit in the destination operand and sets the Z
  condition code appropriately.  If the destination is a data register,
  any of the 32 bits can be specified by the modulo 32 number.  When
  the distination is a memory location, the operation must be a byte
  operation, and therefore the bit number is modulo 8.  In all cases,
  bit zero is the least significant bit.  The bit number for this
  operation may be specified in either of two ways:

  1. Immediate -- The bit number is specified as immediate data.
  2. Register  -- The specified data register contains the bit number.

RESULT
  X - not affected
  N - not affected
  Z - Set if the bit tested is zero.  Cleared otherwise.
  V - not affected
  C - not affected

SEE ALSO
  @{ "BFTST" link Bftst } @{ "BFFFO" link Bfffo }
```

1.28 chk

```
NAME
  CHK -- Check bounds

SYNOPSIS
  CHK <ea>,Dn

  Size = (Word)

FUNCTION
  Compares the value in the data register specified to
  zero and to the upper bound.  The upper bound is a twos complement
  integer.  If the register value is less than zero or greater than
  the upper bound, a CHK instruction, vector number 6, occours.

RESULT
  X - Not affected
  N - Set if Dn < 0; cleared if Dn > <ea>.  Undefined otherwise.
  Z - Undefined.
  V - Undefined.
  C - Undefined.

SEE ALSO
  @{ "CMP " link Cmp  } @{ "CMPI" link Cmpi } @{ "CMPA" link Cmpa }
  @{ "CHK2" link Chk2 }
```

1.29 clr

```
NAME
  CLR -- Clear

SYNOPSIS
  CLR <ea>

  Size = (Byte, Word, Long)

FUNCTION
  Clears the destination operand to zero.

  On an MC68000 and MC68HC000, a CLR instruction does both a
  read and a write to the destination.  Because of this, this
  instruction should never be used on custom chip registers.

RESULT
  X - Not affected
  N - Always cleared
  Z - Always set
  V - Always cleared
  C - Always cleared

SEE ALSO
  @{ "MOVE " link Move  } @{ "MOVEQ" link Moveq }
  @{ "BCLR " link Bclr  } @{ "BFCLR" link Bfclr }
```

1.30 cmp

```
NAME
  CMP -- Compare

SYNOPSIS
  CMP <ea>,Dn

  Size = (Byte, Word, Long)

FUNCTION
  Subtracts the source operand from the destination data
  register and sets the condition codes according to the result.  The
  data register is NOT changed.

RESULT
  X - Not affected
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if an overflow occours.  Cleared otherwise.
  C - Set if a borrow occours.  Cleared otherwise.

SEE ALSO
  @{ "CMPI" link Cmpi } @{ "CMPA" link Cmpa } @{ "CMPM" link Cmpm } @{ "CMP2" link ↩
      Cmp2 }
  @{ "TST " link Tst  } @{ "CHK " link Chk  } @{ "CHK2" link Chk2 }
```

1.31 cmpa

```
NAME
  CMPA -- Compare address

SYNOPSIS
  CMPA  <ea>,An

  Size = (Word, Long)

FUNCTION
  Subtracts the source operand from the destination address
  register and sets the condition codes according to the result.  The
  address register is NOT changed.  Word sized source operands are
  sign extended to long for comparison.

RESULT
  X - Not affected
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if an overflow occours.  Cleared otherwise.
  C - Set if a borrow occours.  Cleared otherwise.

SEE ALSO
  @{ "CMP " link Cmp  } @{ "CMPI" link Cmpi } @{ "CMP2" link Cmp2 }
```

1.32 cmpi

```
NAME
  CMPI -- Compare immediate

SYNOPSIS
  CMP #<data>,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Subtracts the source operand from the destination operand
  and sets the condition codes according to the result.  The destination
  is NOT changed.  The size of the immediate data matches the operation
  size.

RESULT
  X - Not affected
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if an overflow occours.  Cleared otherwise.
  C - Set if a borrow occours.  Cleared otherwise.

SEE ALSO
  @{ "CMP " link Cmp  } @{ "CMPA" link Cmpa } @{ "CMPM" link Cmpm } @{ "CMP2" link ↩
      Cmp2 }
  @{ "TST " link Tst  } @{ "CHK " link Chk  } @{ "CHK2" link Chk2 }
```

1.33 cmpm

```
NAME
  CMPM -- Compare memory

SYNOPSIS
  CMPM  (Ay)+,(Ax)+

  Size = (Byte, Word, Long)

FUNCTION
  Subtracts the source operand from the destination operand
  and sets the condition codes according to the result.  The destination
  operand is NOT changed.  Operands are always addressed with the
  postincrement mode.

RESULT
  X - Not affected
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if an overflow occours.  Cleared otherwise.
  C - Set if a borrow occours.  Cleared otherwise.

SEE ALSO
  @{ "CMP " link Cmp  } @{ "CMPI" link Cmpi } @{ "CMPA" link Cmpa } @{ "CMP2" link ↩
      Cmp2 }
  @{ "TST " link Tst  } @{ "CHK " link Chk  } @{ "CHK2" link Chk2 }
```

1.34 dbcc

```
NAME
  DBcc -- Decrement and branch conditionally

SYNOPSIS
  DBcc  Dn,<label>

  Size = (Word)

FUNCTION
  Controls a loop of instructions.  The parameters are: a
  condition code, a data register (counter), and a displacement value.
  The insctruction first tests the condition (for termination); if it
  is true, no operation is performed.  If the termination condition is
  not true, the low-order 16 bits of the counter are decremented by
  one.  If the result is -1, execution continues at the next
  instruction, otherwise, execution continues at the specified
  address.

  Condition code 'cc' specifies one of the following:

  CC  carry clear C'    LS low or same    C+Z
  CS  carry set   C   LT less than    N'·V+N'V
  EQ  equal   Z   MI  minus   N
  GE  greater or equal  N·V+N'V'  NE  not equal   Z'
```

```
  GT  greater than  N·V·Z'+N'V'·Z'  PL  plus    N'
  HI  high    C'·Z'   VC  overflow clear  V'
  LE  less or equal Z+N'·V+N'V  VS  overflow set  V


  Keep the following in mind when using DBcc instructions:
  1. A DBcc acts as the UNTIL loop contruct in high level
     languages. E.g., DBMI would be "decrement and branch until
     minus".
  2. Most assemblers accept DBRA or DBF for use when no condition
     is required for termination of a loop.

RESULT
  None.

SEE ALSO
  @{ "Bcc" link Bcc } @{ "Scc" link Scc }
```

1.35 divs

```
NAME
  DIVS, DIVSL -- Signed divide

SYNOPSIS
  DIVS.W  <ea>,Dn     32/16 -> 16r:16q
  DIVS.L  <ea>,Dq     32/32 -> 32q      (68020+)
  DIVS.L  <ea>,Dr:Dq  64/32 -> 32r:32q  (68020+)
  DIVSL.L <ea>,Dr:Dq  32/32 -> 32r:32q  (68020+)

  Size = (Word, Long)

FUNCTION
  Divides the signed destination operand by the signed source
  operand and stores the signed result in the destination.

  The instruction has a word form and three long forms.  For the
  word form, the destination operand is a long word and the source
  operand is a word.  The resultant quotient is placed in the lower
  word of the destination and the resultant remainder is placed in the
  upper word of the destination.  The sign of the remainder is the
  same as the sign of the dividend.

  In the first long form, the destination and the source are both
  long words.  The quotient is placed in the longword of the destination
  and the remaineder is discarded.

  The second long form has the destination as a quadword (eight bytes),
  specified by any two data registers, and the source is a long word.
  The resultant remainder and quotient are both long words and are
  placed in the destination registers.

  The final long form has both the source and the destination as long
  words and the resultant quotient and remainder as long words.

RESULT
```

```
  X - Not affected
  N - Set if the quotient is negative, cleared otherwise. Undefined if
      overflow or divide by zero occurs.
  Z - Set if the quotient is zero, cleared otherwise. Undefined if
      overflow or divide by zero occurs.
  V - Set if overflow occurs, cleared otherwise. Undefined if divide by
      zero occurs.
  C - Always cleared.

  Notes:
  1. If divide by zero occurs, an exception occurs.
  2. If overflow occurs, neither operand is affected.

SEE ALSO
  @{ "DIVU" link Divu } @{ "MULS" link Muls } @{ "MULU" link Mulu }
  @{ "Exceptions" link Exceptions }
```

1.36 divu

```
NAME
  DIVU, DIVUL -- Unsigned divide

SYNOPSIS
  DIVU.W  <ea>,Dn     32/16 -> 16r:16q
  DIVU.L  <ea>,Dq     32/32 -> 32q      (68020+)
  DIVU.L  <ea>,Dr:Dq  64/32 -> 32r:32q  (68020+)
  DIVUL.L <ea>,Dr:Dq  32/32 -> 32r:32q  (68020+)

  Size = (Word, Long)

FUNCTION
  Divides the unsigned destination operand by the unsigned
  source operand and stores the unsigned result in the destination.

  The instruction has a word form and three long forms.  For the
  word form, the destination operand is a long word and the source
  operand is a word.  The resultant quotient is placed in the lower
  word of the destination and the resultant remainder is placed in the
  upper word of the destination.  The sign of the remainder is the
  same as the sign of the dividend.

  In the first long form, the destination and the source are both
  long words.  The quotient is placed in the longword of the destination
  and the remaineder is discarded.

  The second long form has the destination as a quadword (eight bytes),
  specified by any two data registers, and the source is a long word.
  The resultant remainder and quotient are both long words and are
  placed in the destination registers.

  The final long form has both the source and the destination as long
  words and the resultant quotient and remainder as long words.

RESULT
```

```
  X - Not affected
  N - See below.
  Z - Set if the quotient is zero, cleared otherwise. Undefined if
      overflow or divide by zero occurs.
  V - Set if overflow occurs, cleared otherwise. Undefined if divide by
      zero occurs.
  C - Always cleared.

  Notes:
  1. If divide by zero occurs, an exception occurs.
  2. If overflow occurs, neither operand is affected.

  According to the Motorola data books, the N flag is set if the
  quotient is negative, but in an unsigned divide, this seems to
  be impossible.

SEE ALSO
  @{ "DIVS" link Divs } @{ "MULS" link Muls } @{ "MULU" link Mulu }
  @{ "Exceptions" link Exceptions }
```

1.37 eor

```
NAME
  EOR -- Exclusive logical OR

SYNOPSIS
  EOR Dn,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Performs an exclusive OR operation on the destination operand
  with the source operand.

RESULT
  X - Not Affected
  N - Set to the value of the most significant bit.
  Z - Set if the result is zero.
  V - Always cleared
  C - Always cleared

SEE ALSO
  EORI BCHG
```

1.38 eori

```
NAME
  EORI -- Exclusive OR immediate

SYNOPSIS
  EORI  #<data>,<ea>
```

```
  Size = (Byte, Word, Long)

FUNCTION
  Performs an exclusive OR operation on the destination operand
  with the source operand.

RESULT
  X - Not Affected
  N - Set to the value of the most significant bit.
  Z - Set if the result is zero.
  V - Always cleared
  C - Always cleared

SEE ALSO
  @{ "EOR " link Eor  } @{ "EORI to CCR" link EoriCCR } @{ "EORI to SR" link  ↩
     EoriSR }
  @{ "BCHG" link Bchg }
```

1.39 eoriccr

```
NAME
  EORI to CCR -- Exclusive OR immediate to the condition code register

SYNOPSIS
  EORI  #<data>,CCR

  Size = (Byte)

FUNCTION
  Performs an exclusive OR operation on the condition codes
  register with the source operand.

RESULT
  X - Changed if bit 4 of the source is set, cleared otherwise.
  N - Changed if bit 3 of the source is set, cleared otherwise.
  Z - Changed if bit 2 of the source is set, cleared otherwise.
  V - Changed if bit 1 of the source is set, cleared otherwise.
  C - Changed if bit 0 of the source is set, cleared otherwise.

SEE ALSO
  @{ "EOR " link Eor  } @{ "EORI" link Eori } @{ "EORI to SR" link EoriSR }
```

1.40 eorisr

```
NAME
  EORI to SR -- Exclusive OR immediated to the status register (privileged)

SYNOPSIS
  EORI  #<data>,SR

  Size = (Word)
```

```
FUNCTION
  Performs an exclusive OR operation on the status register
  with the source operand.

RESULT
  X - Changed if bit 4 of the source is set, cleared otherwise.
  N - Changed if bit 3 of the source is set, cleared otherwise.
  Z - Changed if bit 2 of the source is set, cleared otherwise.
  V - Changed if bit 1 of the source is set, cleared otherwise.
  C - Changed if bit 0 of the source is set, cleared otherwise.

SEE ALSO
  EOR  EORI EORI to CCR
```

1.41 exg

```
NAME
  EXG -- Register exchange

SYNOPSIS
  EXG Rx,Ry

  Size = (Long)

FUNCTION
  Exchanges the contents of any two registers.

RESULT
  None.

SEE ALSO
  @{ "SWAP" link Swap }
```

1.42 ext

```
NAME
  EXT, EXTB -- Sign extend

SYNOPSIS
  EXT.W Dn  Extend byte to word
  EXT.L Dn  Extend word to long word
  EXTB.L  Dn  Extend byte to long word  (68020+)

  Size = (Word, Long)

FUNCTION
  Extends a byte to a word, or a word to a long word in a data
  register by copying the sign bit through the upper bits.  If the
  operation is from byte to word, bit 7 is copied to bits 8 through
  15.  If the operation is from word to long word, bit 15 is copied
  to bits 16 through 31.  The EXTB copies bit 7 to bits 8 through 31.
```

```
RESULT
  X - Not affected
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Always cleared
  C - Always cleared

SEE ALSO
```

1.43 illegal

```
NAME
  ILLEGAL -- Illegal processor instruction

SYNOPSIS
  ILLEGAL

FUNCTION
  This instruction forces an Illegal Instruction exception,
  vector number 4.  All other illegal instruction bit patterns,
  including, but not limited to, $fxxx and $axxx, are reserved for
  future expansion.

RESULT
  None.


SEE ALSO
  @{ "BKPT      " link Bkpt }
  @{ "Exceptions" link Exceptions }
```

1.44 jmp

```
NAME
  JMP -- Unconditional far jump

SYNOPSIS
  JMP <ea>

FUNCTION
  Program execution continues at the address specified by
  the operand.

RESULT
  None.

SEE ALSO
  @{ "BRA" link Bra } @{ "JSR" link Jsr }
```

1.45 jsr

```
NAME
  JSR -- Jump to far subroutine

SYNOPSIS
  JSR <ea>

FUNCTION
  Pushes the long word address of the instruction immediately
  following the JSR instruction onto the stack.  The PC contains
  the address of the instruction word plus two.  Program execution
  continues at location specified by <ea>.

RESULT
  None.

SEE ALSO
  @{ "BSR" link Jsr } @{ "BRA" link Bra }
  @{ "RTS" link Rts } @{ "RTD" link Rts } @{ "RTR" link Rts }
```

1.46 lea

```
NAME
  LEA -- Load effective address

SYNOPSIS
  LEA <ea>,An

  Size = (Long)

FUNCTION
  Places the specified address into the destination address
  register.  Note:  All 32 bits of An are affected by this instruction.

RESULT
  None.

SEE ALSO
  @{ "MOVEA" link Movea } @{ "ADDA " link Adda  } @{ "SUBA " link Suba  }
```

1.47 link

```
NAME
  LINK -- Create local stack frame

SYNOPSIS
  LINK  An,#<data>

  Size = (Word)
  Size = (Word, Long)    (68020+)

FUNCTION
```

```
  This instruction saves the specified address register onto
  the stack, then places the new stack pointer in that register.  It
  then adds the specificed immediate data to the stack pointer.  To
  allocate space on the stack for a local data area, a negative value
  should be used for the second operand.

RESULT
  None.

SEE ALSO
  @{ "UNLK               " link Unlk       }
  @{ "Local Stack Frames" link LocalStack }
```

1.48 lsd

```
NAME
  LSL, LSR -- Logical shift left and logical shift right

SYNOPSIS
  LSd Dx,Dy
  LSd #<data>,Dy
  LSd <ea>
  where d is directoin, L or R

  Size = (Byte, Word, Long)

FUNCTION
  Shift the bits of the operand in the specified direction.
  The carry bit set set to the last bit shifted out of the operand.
  The shift count for the shifting of a register may be specified in
  two different ways:

  1. Immediate - the shift count is specified in the instruction (shift
                 range 1-8).
  2. Register  - the shift count is contained in a data register
                 specified in the instruction (shift count mod 64)

  For a register, the size may be byte, word, or long, but for a memory
  location, the size must be a word.  The shift count is also restricted
  to one for a memory location.

RESULT
  X - Set according to the last bit shifted out of the operand.
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Always cleared
  C - Set according to the last bit shifted out of the operand.

SEE ALSO
```

1.49 move

```
NAME
  MOVE -- Source -> Destination

SYNOPSIS
  MOVE  <ea>,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Move the content of the source to the destination location.
  The data is examied as it is moved, and the condition codes
  set accordingly.

RESULT
  X - Not affected.
  N - Set if the result is negative. Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Always cleared.
  C - Always cleared.

SEE ALSO
```

1.50 movea

```
NAME
  MOVEA -- Source -> Destination

SYNOPSIS
  MOVEA <ea>,An

  Size = (Word, Long)

FUNCTION
  Move the content of the source to the destination address
  register.  Word sized operands are sign extended to 32 bits
  before the operation is done.

RESULT
  None.

SEE ALSO
  MOVE LEA
```

1.51 movefromccr

```
NAME
  MOVE from CCR -- CCR -> Destination

SYNOPSIS
  MOVE  CCR,<ea>
```

```
  Size = (Word)

FUNCTION
  The content of the status register is moved to the
  destination location.  The source operand is a word,
  but only the low order byte contains the condition
  codes.  The high order byte is set to all zeros.

RESULT
  None.

SEE ALSO
```

1.52 movetoccr

```
NAME
  MOVE to CCR -- Source -> CCR

SYNOPSIS
  MOVE  <ea>,CCR

  Size = (Word)

FUNCTION
  The content of the source operand is moved to the
  condition codes.  The source operand is a word, but
  only the low order byte is used to update the condition
  codes.  The high order byte is ignored.

RESULT
  X - Set the same as bit 4 of the source operand.
  N - Set the same as bit 3 of the source operand.
  Z - Set the same as bit 2 of the source operand.
  V - Set the same as bit 1 of the source operand.
  C - Set the same as bit 0 of the source operand.

SEE ALSO
```

1.53 movefromsr

```
NAME
  MOVE from SR -- Move from status register (privileged)

SYNOPSIS
  MOVE  SR,<ea>

  Size = (Word)

FUNCTION
  The content of the status register is moved to the
  destination location.  The operand size is a word.
```

```
RESULT
  None.

SEE ALSO
```

1.54 movetosr

```
NAME
  MOVE to SR -- Move to status register (privileged)

SYNOPSIS
  MOVE  <ea>,SR

  Size = (Word)

FUNCTION
  The content of the source operand is moved to the
  status register.  The source operand size is a word
  and all bits of the status register are affected.

RESULT

  X - Set the same as bit 4 of the source operand.
  N - Set the same as bit 3 of the source operand.
  Z - Set the same as bit 2 of the source operand.
  V - Set the same as bit 1 of the source operand.
  C - Set the same as bit 0 of the source operand.

SEE ALSO
  MOVE to CCR
```

1.55 moveusp

```
NAME
  MOVE USP -- Move to/from user stack pointer (privileged)

SYNOPSIS
  MOVE  USP,An
  MOVE  An,USP

  Size = (Long)

FUNCTION
  The contents of the user stack pointer are transferred
  either to or from the specified address register.

RESULT
  None.

SEE ALSO
```

1.56 movec

```
NAME
  MOVEC -- Move to/from control register

SYNOPSIS
  MOVEC Rc,Rn
  MOVEC Rn,Rc

  Size = (Long)

FUNCTION
  Copy the contents of the specified control register
  to the specified general register or copy from the
  general register to the control register.  This is
  always a 32-bit transfer even though the the control
  register may be implemented with fewer bits.

RESULT
  None.

SEE ALSO
```

1.57 movem

```
NAME
  MOVEM -- Move multiple registers

SYNOPSIS
  MOVEM register list,<ea>
  MOVEM <ea>,register list

  Size = (Word, Long)

FUNCTION
  Registers in the register list are either moved to or
  fetched from consecutive memory locations at the specified
  address.  Data can be either word or long word, but if
  the register list is destination and the size is word,
  each register is filled with the source word sign extended
  to 32-bits.

  Also, in the case that the register list is the destination,
  register indirect with predecrement is not a valid source
  mode.  If the register list is the source, then the
  destination may not be register indirect with postincrement.

    MOVEM.L D0/D1/A0,(A2)+    ; invalid
    MOVEM.W -(A1),D5/D7/A4    ; invalid

  The register list is accessed with D0 first through D7, followed
  by A0 through A7.

RESULT
```

```
  None.

SEE ALSO
```

1.58 movep

```
NAME
  MOVEP -- Move peripheral data

SYNOPSIS
  MOVEP Dx,(d,Ay)
  MOVEP (d,Ay),Dx

  Size = (Word, Long)

FUNCTION
  Data is transferred between a data register and ever-other
  byte of memory at the selected address. Example:

    LEA port0,A0  ; A0 -> $FFFFFFFFFFFFFFFF
    MOVEQ #0,D0
    MOVEP.L D0,(0,A0) ; A0 -> $FF00FF00FF00FF00
    MOVE.L  #$55554444,D0
    MOVEP.L D0,(1,A0) ; A0 -> $FF55FF55FF44FF44

RESULT
  None.

SEE ALSO
```

1.59 moveq

```
NAME
  MOVEQ -- Move signed 8-bit data quick

SYNOPSIS
  MOVEQ #<data:8>,Dn

  Size = (Long)

FUNCTION
  Move signed 8-bit data to the specified data register.
  The specified data is sign extended to 32-bits before
  it is moved to the register

RESULT
  X - Not affected.
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Always cleared.
  C - Always cleared.
```

```
SEE ALSO
```

1.60 moves

```
NAME
  MOVES -- Move address space (privileged)

SYNOPSIS
  MOVES Rn,<ea>
  MOVES <ea>,Rn

FUNCTION

RESULT
  None.

SEE ALSO
```

1.61 mul

```
NAME
  MULS -- Signed multiply
  MULU -- Unsigned multiply

SYNOPSIS
  MULS.W  <ea>,Dn   16*16->32
  MULS.L  <ea>,Dn   32*32->32 68020+
  MULS.L  <ea>,Dh:Dl  32*32->64 68020+

  MULU.W  <ea>,Dn   16*16->32
  MULU.L  <ea>,Dn   32*32->32 68020+
  MULU.L  <ea>,Dh:Dl  32*32->64 68020+

  Size = (Word)
  Size = (Word, Long)     68020+

FUNCTION
  Multiply two signed (MULS) or unsigned (MULU) integers
  to produce either a signed or unsigned, respectivly,
  result.

  This instruction has three forms.  They are basically
  word, long word, and quad word.  The first version is
  the only one available on a processore lower than a
  68020.  It will multiply two 16-bit integers are produce
  a 32-bit result.  The second will multiply two 32-bit
  integers and produce a 32-bit result.

  The third form needs some special consideration.  It
  will multiply two 32-bit integers, specified by <ea>
  and Dl, the result is (sign) extended to 64-bits with
  the low order 32 being placed in Dl and the high order
```

```
  32 being placed in Dh.

RESULT
  X - Not affected.
  N - Set if the result is negative.  Cleared otherwise.
  Z - Set if the result is zero.  Cleared otherwise.
  V - Set if overflow.  Cleared otherwise.
  C - Always cleared.

SEE ALSO
```

1.62 nbcd

```
NAME
  NBCD -- Negate binary coded decimal with extend

SYNOPSIS
  NBCD  <ea>

  Size = (Byte)

FUNCTION
  The specified BCD number and the extend bit are subtracted
  from zero.  Therefore, if the extend bit is set a nines
  complement is performed, else a tens complement is performed.
  The result is placed back in the specified <ea>.

  It can be use full to set the zero flag before performing
  this operation so that multi precision operations can
  be correctly tested for zero.

RESULT
  X - Set the same as the carry bit.
  N - Undefined.
  Z - Cleared it the result is non-zero, unchanged otherwise.
  V - Undefined.
  C - Set if a borrow was generated, cleared otherwise.

SEE ALSO
  NEG NEGX
```

1.63 neg

```
NAME
  NEG -- Negate

SYNOPSIS
  NEG <ea>

  Size = (Byte, Word, Long)

FUNCTION
```

```
  The operand specified by <ea> is subtracted from
  zero. The result is stored in <ea>.

RESULT
  X - Set the same as the carry bit.
  N - Set if the result is negative, otherwise cleared.
  Z - Set if the result is zero, otherwise cleared.
  V - Set if overflow, otherwise cleared.
  C - Cleared if the result is zero, otherwise set.

SEE ALSO
  NBCD NEGX
```

1.64 negx

```
NAME
  NEGX -- Negate with extend

SYNOPSIS
  NEGX  <ea>

  Size = (Byte, Word, Long)

FUNCTION
  Perform an operation similar to a NEG, with the
  exception that the operand and the extend bit are both
  subtracted from zero.  The result then being place in
  the given ea.

  As with ADDX, SUBX, ABCD,
  SBCD, and NBCD, it can be useful to set
  the zero flag before performing this operation so that
  multi precision operations can be tested for zero.

RESULT
  X - Set the same as the carry bit.
  N - Set if the result is negative, otherwise cleared.
  Z - Cleared if the result is non-zero, otherwise unchanged.
  V - Set if an overflow is generated, cleared otherwise.
  C - Set if a borrow is generated, otherwise cleared.

SEE ALSO
  NEG NBCD ADDX SUBX
```

1.65 nop

```
NAME
  NOP -- No operation

SYNOPSIS
  NOP
```

```
FUNCTION
  Nothing happens! Well, sort of.  This instruction will
  basically wait until all pending bus activity is
  completed.  This allows synchronization of the pipeline
  and prevents instruction overlap.

RESULT
  None.

SEE ALSO
```

1.66 not

```
NAME
  NOT -- Logical complement

SYNOPSIS
  NOT <ea>

  Size = (Byte, Word, Long)

FUNCTION
  All bits of the specified operand are inverted and placed
  back in the operand.

RESULT
  X - Not affected.
  N - Set if the result is negative, otherwise cleared.
  Z - Set if the result is zero, otherwise cleared.
  V - Always cleared.
  C - Always cleared.

SEE ALSO
```

1.67 or

```
NAME
  OR -- Logical OR

SYNOPSIS
  OR  Dn,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Performs an OR operation on the destination operand
  with the source operand.

RESULT
  X - Not Affected
  N - Set to the value of the most significant bit.
  Z - Set if the result is zero.
```

```
  V - Always cleared
  C - Always cleared

SEE ALSO
  ORI BSET
```

1.68 ori

```
NAME
  ORI -- Logical OR immediate

SYNOPSIS
  ORI #<data>,<ea>

  Size = (Byte, Word, Long)

FUNCTION
  Performs an OR operation on the destination operand
  with the source operand.

RESULT
  X - Not Affected
  N - Set to the value of the most significant bit.
  Z - Set if the result is zero.
  V - Always cleared
  C - Always cleared

SEE ALSO
  @{ "OR  " link Eor  } @{ "ORI to CCR" link oriCCR } @{ "ORI to SR" link oriSR }
  @{ "BSET" link Bchg }
```

1.69 oriccr

```
NAME
  ORI to CCR -- Logical OR immediate to the condition code register

SYNOPSIS
  ORI #<data>,CCR

  Size = (Byte)

FUNCTION
  Performs an OR operation on the condition codes
  register with the source operand.

RESULT
  X - Set if bit 4 of the source is set, cleared otherwise.
  N - Set if bit 3 of the source is set, cleared otherwise.
  Z - Set if bit 2 of the source is set, cleared otherwise.
  V - Set if bit 1 of the source is set, cleared otherwise.
  C - Set if bit 0 of the source is set, cleared otherwise.
```

```
SEE ALSO
  @{ "OR " link or  } @{ "ORI" link ori } @{ "ORI to SR" link oriSR }
```

1.70 orisr

```
NAME
  ORI to SR -- Logical OR immediated to the status register (privileged)

SYNOPSIS
  ORI #<data>,SR

  Size = (Word)

FUNCTION
  Performs an OR operation on the status register with
  the source operand, and leaves the result in the status
  register.

RESULT
  X - Set if bit 4 of the source is set, cleared otherwise.
  N - Set if bit 3 of the source is set, cleared otherwise.
  Z - Set if bit 2 of the source is set, cleared otherwise.
  V - Set if bit 1 of the source is set, cleared otherwise.
  C - Set if bit 0 of the source is set, cleared otherwise.

SEE ALSO
  OR  ORI ORI to CCR
```

1.71 unnamed.1

```
NAME
  PACK -- Pack binary coded decimal (68020+)

SYNOPSIS
  PACK  -(Ax),-(Ay),#<adjustment>
  PACK  Dx,Dy,#<adjustment>

FUNCTION
  Convert byte-per-digit unpacked BCD to packed two-digit-
  per-byte BCD.

RESULT
  None.

SEE ALSO
```

1.72 unnamed.2

```
NAME
  PEA -- Push effective address

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.73 unnamed.3

```
NAME
  RESET -- Reset external devices

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.74 rod

```
NAME
  ROL, ROR -- Rotate left and rotate right

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.75 roxd

```
NAME
  ROXL, ROXD -- Rotate left with extend and rotate right with extend
```

```
SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.76 rtd

```
NAME
  RTD -- Return and deallocate parameter stack frame  (68010+)

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.77 rte

```
NAME
  RTE -- Return from exception

SYNOPSIS

FUNCTION

RESULT

SEE ALSO
```

1.78 rtm

```
NAME
  RTM -- Return from process module (68020 only)

SYNOPSIS

FUNCTION

RESULT
```

SEE ALSO

1.79 rtr

NAME
  RTR -- Return and restore condition code register

SYNOPSIS

FUNCTION

RESULT

SEE ALSO

1.80 rts

NAME
  RTS -- Return from subroutine

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO

1.81 sbcd

NAME
  SBCD -- Subtract binary coded decimal with extend

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO

1.82 scc

```
NAME
  Scc -- Conditional set

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.83 stop

```
NAME
  STOP -- Stop processor execution (privileged)

SYNOPSIS

FUNCTION

RESULT

SEE ALSO
```

1.84 sub

```
NAME
  SUB -- Subtract

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.85 suba

```
NAME
  SUBA -- Subtract address
```

```
SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.86 subi

```
NAME
  SUBI -- Subtract immediate

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.87 subq

```
NAME
  SUBQ -- Subtract 3-bit immediate quick

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.88 subx

```
NAME
  SUBX -- Subtract with extend

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.89 swap

```
NAME
  SWAP -- Swap register upper and lower words

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.90 tas

```
NAME
  TAS -- Test and set operand

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.91 trap

```
NAME
  TRAP -- Initiate processor trap

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.92 trapcc

```
NAME
  TRAPcc -- Conditional trap    (68020+)
  TRAPv  -- Trap on overflow

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.93 tst

```
NAME
  TST -- Test operand for zero

SYNOPSIS

FUNCTION

RESULT
  X -
  N -
  Z -
  V -
  C -

SEE ALSO
```

1.94 unlk

```
NAME
  UNLK -- Free stack frame created by LINK

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.95 unpk

```
NAME
  UNPK -- Unpack binary coded decimal (68020+)

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.96 exceptions

```
NAME
  Processor exception initiation and handling

SYNOPSIS

FUNCTION

RESULT
  None.

SEE ALSO
```

1.97 localstack

```
NAME
  Local stack frame maintaince

SYNOPSIS
    LINK  An,#<data>
    ...
    UNLK  An
```

```
    RTS

FUNCTION
  The use of a local stack frame is critically important to the
  programmer who wishes to write re-entrant or recursive functions.
  The creation of a local stack frame on the MC680x0 family is done
  through the use of the LINK and UNLK instructions.  The LINK
  instruction saves the frame pointer onto the stack, and places a
  pointer to the new stack frame in it.  The UNLK instruction
  restores the old stack frame.  For example:

    link  a5,#-8    ; a5 is chosen to be the frame
          ; pointer.  8 bytes of local stack
          ; frame are allocated.
    ...
    unlk  a5    ; a5, the old frame pointer, and the
          ; old SP are restored.

  Since the LINK and UNLK instructions maintain both the frame pointer
  and the stack pointer, the following code segment is perfectly
  legal:

    link  a5,#-4

    movem.l d0-a4,-(sp)
    pea (500).w
    move.l  d2,-(sp)
    bsr.b Routine

    unlk  a5
    rts

RESULT
  None.

SEE ALSO
  @{ "LINK" link Link } @{ "UNLK" link Unlk }
```