```
*************************************************************************
* Alexander Johnson                                                     *
* Oonagh Geldenhuys                                                     *
* Jason Woodring                                                        *
* --Disassembler Project--                                              *
*************************************************************************
stack           EQU     $00005000       * Stack pointer
begin           EQU     $00000000       * Minimum starting address
end             EQU     $00FFFFFE       * Maximum ending address
*************************************************************************
                ORG     $00001000
*************************************************************************
start           LEA     stack,SP        * Initialize stack pointer
                LEA     wel_1,A1        * Load welcome message 1
                MOVE.B  wel_1_len,D1    * Load welcome message 1 length
                MOVE.L  #0,D0           * Load task code for prompt
                TRAP    #15             * Display user prompt
                LEA     wel_2,A1        * Load welcome message 2
                MOVE.B  wel_2_len,D1    * Load welcome message 2 length
                MOVE.L  #0,D0           * Load task code for prompt
                TRAP    #15             * Display user prompt
prompt_lower    LEA     spc,A1          * Load empty line
                MOVE.B  spc_len,D1      * Load empty line length
                MOVE.L  #0,D0           * Load trap code
                TRAP    #15             * Display empty line
                LEA     lower,A1        * Load lower address prompt
                MOVE.B  lower_length,D1 * Load lower prompt length
                MOVE.L  #1,D0           * Load task code for prompt
                TRAP    #15             * Display user prompt
                LEA     lower_input,A1  * Load address to store user input
                MOVE.L  #2,D0           * Load task code for user input
                TRAP    #15             * Retrieve input
                JSR     decode          * Decode user input string
                CMP.B   #1,D4           * Is the input good?
                BEQ     proceed1        * Yes, so proceed
                LEA     bad_input,A1    * Load bad input message
                MOVE.B  bad_input_len,D1 * Load bad input message length
                MOVE.L  #0,D0           * Load code for bad input message
                TRAP    #15             * Display bad input message
                JSR     prompt_lower    * Ask again for a new input
proceed1        JSR     ck_bnd          * check if the input is odd / even bnd
                CMP.W   #-1,D3          * Is the address odd?
                BEQ     prompt_lower    * Go back and ask for correct input
                MOVE.L  #begin,D6       * Load minimum starting address
                CMP.L   D6,D3           * Is input at or above minimum?
                BLT     prt_low         * Input address is too low, print error
                MOVE.L  #end,D6         * Load maximum starting address
                CMP.L   D6,D3           * Is input at or below maximum?
                BGT     prt_low         * Specified input address is too high
                MOVEA.L D3,A5           * Current address is stored in A5
prompt_upper    LEA     upper1,A1       * Load lower address prompt part 1
                MOVE.B  upper1_length,D1 * Load lower address prompt 1/2 length
                MOVE.L  #1,D0           * Load task code for prompt
                TRAP    #15             * Display first half of upper prompt
                LEA     lower_input,A1  * Load lower address for prompting
                MOVE.L  #8,D1           * Load lower address length
                TRAP    #15             * Display lower address in prompt
                LEA     upper2,A1       * Load lower address prompt part 2
                MOVE.B  upper2_length,D1 * Load lower address prompt 2/2 length
                TRAP    #15             * Display second half of upper prompt
                LEA     upper_input,A1  * Load address to store user input
                MOVE.L  #2,D0           * Load task code for user input
                TRAP    #15             * Retrive input
                JSR     decode          * Decode user input string
                CMP.B   #1,D4           * Is the input good?
                BEQ     proceed2        * Yes, so proceed
                LEA     bad_input,A1    * Load bad input message
                MOVE.B  bad_input_len,D1 * Load bad input message length
                MOVE.L  #0,D0           * Load code for bad input message
                TRAP    #15             * Display bad input message
                BRA     prompt_upper    * Ask again for a new input
```

```
proceed2      JSR      ck_bnd             * check if the input is odd / even bnd
              CMP.W    #-1,D3             * Is the address odd?
              BEQ      prompt_upper       * Go back and ask for correct input
              CMP.L    D6,D3              * Is input at or below maximum?
              BGT      prt_hi             * Specified input address is too high
              CMP.L    A5,D3              * Is input at or above minimum?
              BLT      prt_hi             * Specified input address is too low
              MOVE.L   D3,D6              * Load ending address for calculations
              SUB.L    A5,D6              * Find total length of instructions
              DIVU.W   #$2,D6             * Calculate total instructions
              ANDI.L   #$0000FFFF,D6      * Remove remainder from division
mainloop      MOVE.L   A5,D5              * I want to use D5 to help print address
              MOVEQ    #$1,D4             * Instruction defaults to good
              MOVEQ    #$0,D3             * Move instruction defaults to false
              LEA      good_buf,A4        * Initialize good buffer
              LEA      bad_buf,A3         * Initialize bad buffer
              JSR      wrt_addr           * Write address to good and bad buffers
              MOVEQ    #$1,D4             * Instruction defaults to good
              MOVEQ    #$0,D3             * Move instruction defaults to false
              MOVE.B   #' ',(A4)+         *
              MOVE.B   #' ',(A4)+         *
              MOVE.B   #' ',(A4)+         *
              MOVE.B   #' ',(A4)+         *
              MOVE.B   #' ',(A4)+         *
              MOVE.B   #' ',(A4)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #'D',(A3)+         * D
              MOVE.B   #'A',(A3)+         * A
              MOVE.B   #'T',(A3)+         * T
              MOVE.B   #'A',(A3)+         * A
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #' ',(A3)+         *
              MOVE.B   #'$',(A3)+         *$
              JSR      ld_badbuff         * Load the data for bad buff
              JSR      ck_cnt             * See if we have a full page write
              MOVE.B   #$00,(A3)          * End of bad buffer
              MOVE.W   (A5)+,D7           * Get next instruction
              JSR      disassemble        * Disassemble the instruction
              CMP.B    #$1,D4             * Is the input good?
              BEQ      good               * Yup!
bad           LEA      bad_buf,A1         * We're going to print the bad buffer
              BRA      showbuffer         * Now go print it
good          MOVE.B   #$00,(A4)          * Add terminator to good buffer
              LEA      good_buf,A1        * We're going to print the good buffer
showbuffer    MOVEQ    #13,D0             * Load trap code for buffer printing
              TRAP     #15                * Print output buffer
              DBF      D6,mainloop        * Have we decoded every instruction?
run_agin      LEA      again_prmpt,A1     * Set up A1 ptr for run again prompt
              MOVE.B   again_prmpt_len,D1  * Set up again_prompt length
              MOVE.L   #1,D0              * Load trap code
              TRAP     #15                * Display again_prompt
              LEA      linecount,A2       * Lets reset the counter
              MOVE.B   #00,(A2)           * Reset line counter
              LEA      run_again,A1       * Set up A1 ptr for user input
              MOVE.L   #2,D0              * Load user input trap code 2
              TRAP     #15                * Grab user input
              CMP.B    #$59,(A1)          * Is it a upper case Y?
              BEQ      prompt_lower       * If so, go back to beginning
              CMP.B    #$79,(A1)          * Is it a lower case y?
              BEQ      prompt_lower       * If so, go back to the beginning
```

```
             CMP.B    #$4E,(A1)         * Is it an uppercase N?
             BEQ      finish            * end program
             CMP.B    #$6E,(A1)         * Is it a lowercase n?
             BEQ      finish            * end program
             LEA      again_err,A1      * Load error message for run again
             MOVE.B   again_err_len,D1  * Load length
             MOVE.L   #0,D0             * Load trap code
             TRAP     #15               * Display run again error string
             BRA      run_agin          * Branch to get proper input
finish       JSR      the_end           * Display our graceful ending
             STOP     #$00002700        * End of program
*************************************************************************
* RESERVED LOCATIONS                                                    *
*************************************************************************
lower          DC.B  'Please enter a starting address from $0 to $00FFFFFE: '
lower_length   DC.B  lower_length-lower
upper1         DC.B  'Please enter an ending address from $'
upper1_length  DC.B  upper1_length-upper1
upper2         DC.B  ' to $00FFFFFE: '
upper2_length  DC.B  upper2_length-upper2
bad_input      DC.B  '  Address entered is not within the specified range!'
bad_input_len  DC.B  bad_input_len-bad_input
odd_error      DC.B  '  You must enter an even address!'
odd_lng        DC.B  odd_lng-odd_error
again_prmpt    DC.B  'Would you like to run the program again? (y/n)'
again_prmpt_len DC.B again_prmpt_len-again_prmpt
linecount      DC.B  0
bad_buf        DS.B  80
good_buf       DS.B  80
lower_input    DS.B  80
upper_input    DS.B  80
run_again      DS.B  80
wait           DC.B  'Please hit enter to display the next page. '
wait_len       DC.B  wait_len-wait
wait2          DS.B  80
end1           DC.B  '        (,        ,)'
end1_len       DC.B  end1_len-end1
end2           DC.B  '        |  *_      |  ****** *****  ****   *****  ****  ****  *  * *'
end2_len       DC.B  end2_len-end2
end3           DC.B  '        \\ ((,^,  //  *      *   * *    *  *      *     *     *  * * '
end3_len       DC.B  end3_len-end3
end4           DC.B  '         \`()) _. /   *  *** * *** * ** *  *      ***   ***   *  * * '
end4_len       DC.B  end4_len-end4
end5           DC.B  '          ` )__)-     *    * *  *  *    *  *      *     *     *  * * '
end5_len       DC.B  end5_len-end5
end6           DC.B  '            \ )       ****** *   * *    *  *****  ****  *     **** ****'
end6_len       DC.B  end6_len-end6
end7           DC.B  '             )( '
end7_len       DC.B  end7_len-end7
end8           DC.B  '            /  `_._   ***** *    *  ***  ***  *   *  *****'
end8_len       DC.B  end8_len-end8
end9           DC.B  '           /   /_ , ) *     ***  *  *  *  *   **  *  *'
end9_len       DC.B  end9_len-end9
end10          DC.B  '          /_.-   ( /  **    *  * *  *  *  *   * * *  *   **'
end10_len      DC.B  end10_len-end10
end11          DC.B  '            \ |  ,/   *     *   **  *  *  *   *  **  *   *'
end11_len      DC.B  end11_len-end11
end12          DC.B  '             \ ( /    ***** *    *  ***  ***  *   *  *****'
end12_len      DC.B  end12_len-end12
end13          DC.B  '             ),)/ '
end13_len      DC.B  end13_len-end13
end14          DC.B  '             ( )`'
end14_len      DC.B  end14_len-end14
end15          DC.B  '             \ '
end15_len      DC.B  end15_len-end15
end16          DC.B  '             , '
end16_len      DC.B  end16_len-end16
end17          DC.B  '              |'
end17_len      DC.B  end17_len-end17
end18          DC.B  '             \/ '
end18_len      DC.B  end18_len-end18
```

```
wel_1          DC.B  ' Welcome to the Motorola 68k instruction DIS-ASSEMBLER'
wel_1_len      DC.B  wel_1_len-wel_1
wel_2          DC.B  ' Created by: Alex Johnson, Oonagh Geldenhuys, and Jason Woodring'
wel_2_len      DC.B  wel_2_len-wel_2
spc            DC.B  ' '
spc_len        DC.B  spc_len-spc
again_err      DC.B  'You have not entered a y or n.'
again_err_len DC.B  again_err_len-again_err
*************************************************************************
* DECODE - Converts user input string into a valid memory address       *
*        - Stores decoded address data into D3                          *
*        - Stores good/bad (1/0) data in D4 upon returning              *
*************************************************************************
decode    MOVEQ   #$0,D3     * Clear returned data register
          MOVEQ   #$0,D5     * Set up D5 for null comparisons
          CMP.B   D5,D1      * Has the user entered an input of length 0?
          BEQ     badinput   * No address was specified
          CMPI    #$8,D1     * Has the user entered more than 8 char max?
          BGT     badinput   * The specified address string is too long
examine   MOVE.B  (A1)+,D2   * Examine next character and decode it
          CMP.B   D5,D2      * Check to see if the next character is null (0's)
          BEQ     return     * Character is null, so this is the end of input
          CMP.B   #102,D2    * 102 is the highest valid character code
          BGT     badinput   * Character is above the valid address ranges
          CMP.B   #96,D2     * 96 = lower bound for lower case hex chararacters
          BGT     lohex      * We branch to lower case hexidecimal processing
          CMP.B   #70,D2     * 70 = upper bound for upper case hex characters
          BGT     badinput   * Character is between valid address ranges
          CMP.B   #64,D2     * 64 = lower bound for lower case hex characters
          BGT     uphex      * We branch to upper case hexidecimal processing
          CMP.B   #57,D2     * 57 is the highest decimal character
          BGT     badinput   * Character is between valid address ranges
          CMP.B   #48,D2     * 47 is the lowest decimal character
          BLT     badinput   * Character is below the valid address ranges
          SUBI.B  #48,D2     * Change ascii decimal code to actual value
          BRA     decoded    * We're done decoding this byte
lohex     SUBI.B  #87,D2     * Change ascii lower case code to actual value
          BRA     decoded    * We're done decoding this byte
uphex     SUBI.B  #55,D2     * Change ascii upper case code to actual value
decoded   ASL.L   #4,D3      * Make rooom for new byte in the return register
          ADD.B   D2,D3      * Move the new byte into the return register
          BRA     examine    * We're ready to decode the next byte
badinput  MOVEQ   #$0,D4     * Load bad input code into D4 for returning
          RTS                * Return bad input
return    MOVEQ   #$1,D4     * Load good input code into D4 for returning
          RTS                * Return the decoded user input
*************************************************************************
* PRT_LOW - Prints error message                                        *
*         - Asks again for lower boundary address                       *
*************************************************************************
prt_low  LEA      bad_input,A1      * Load bad input message
         MOVE.B   bad_input_len,D1  * Load bad input message length
         MOVE.L   #0,D0             * Load code for bad input message
         TRAP     #15               * Display bad input message
         JMP      prompt_lower      * Get new input address
*************************************************************************
* PRT_HI - Prints error message                                         *
*         - Asks again for upper boundary address                       *
*************************************************************************
prt_hi   LEA      bad_input,A1      * Load bad input message
         MOVE.B   bad_input_len,D1  * Load bad input message length
         MOVE.L   #0,D0             * Load code for bad input message
         TRAP     #15               * Display bad input message
         JMP      prompt_upper      * Get new input address
*************************************************************************
* CK_BND - Checks the address boundaries to make sure they are even     *

*************************************************************************
ck_bnd   CLR.L          D2              * clear buffer before we use
         MOVE.W         D3,D2           * put address in D2
         MOVE.L         #$00000002,D1   * Load 2 into register for division
```

```
         DIVU           D1,D2           * Divide address by 2
         SWAP           D2              * Swap contents of D2 to get remainder
         CMP.B          #$01,D2         * Check if the remainder is 1
         BEQ            odd_err         * If one, address is odd, branch to
         RTS                            * error routine.  Otherwise, address is
                                        * even, return
odd_err  LEA            odd_error,A1    * Load odd error message into A1
         MOVE.B         odd_lng,D1      * Load eror length to D1
         MOVE.L         #0,D0           * Load task code 0 for printing error
                                        * message in A1
         TRAP           #15             * Display error message
         NOT.L          D5              * Change D5 to -1 to return as a bad
                                        * address
         MOVE.L         D5,D3           * Load -1 into the returned data
                                        * register
         RTS                            * Return
*******************************************************************************
* DISASSEMBLE - Disassembles the code at D7, and appends the resulting string *
*               to the good buffer pointed to by A4                           *
*             - Returns good(1)/bad(0) status of instruction in D4            *
*             - Returns the remaining instruction count in D6                 *
*******************************************************************************
disassemble  MOVE.W  D7,D5         * Copy current instruction for shifting
             LSR.W   #$8,D5        * Get the first 4 characters by shifting 12
             LSR.W   #$4,D5        * Get the first 4 characters by shifting 12
             MULU    #$8,D5        * Form offset for jump table lookup
             LEA     maintable,A0  * Load maintable prior to jumping
             JSR     00(A0,D5)     * Look up function for initial 4 digit code
             RTS                   * We're finished disassembling
*******************************************************************************
* PRIMARY JUMP TABLE - Contains lookup functions for the first 4 bits of the  *
*                      current instruction                                    *
*******************************************************************************
maintable  JSR  main0000  * Call function for codes which begin with 0000
           RTS            * Finished disassembling this code
           JSR  main0001  * Call function for codes which begin with 0001
           RTS            * Finished disassembling this code
           JSR  main0010  * Call function for codes which begin with 0010
           RTS            * Finished disassembling this code
           JSR  main0011  * Call function for codes which begin with 0011
           RTS            * Finished disassembling this code
           JSR  main0100  * Call function for codes which begin with 0100
           RTS            * Finished disassembling this code
           JSR  main0101  * Call function for codes which begin with 0101
           RTS            * Finished disassembling this code
           JSR  main0110  * Call function for codes which begin with 0110
           RTS            * Finished disassembling this code
           JSR  main0111  * Call function for codes which begin with 0111
           RTS            * Finished disassembling this code
           JSR  main1000  * Call function for codes which begin with 1000
           RTS            * Finished disassembling this code
           JSR  main1001  * Call function for codes which begin with 1001
           RTS            * Finished disassembling this code
           JSR  main1010  * Call function for codes which begin with 1010
           RTS            * Finished disassembling this code
           JSR  main1011  * Call function for codes which begin with 1011
           RTS            * Finished disassembling this code
           JSR  main1100  * Call function for codes which begin with 1100
           RTS            * Finished disassembling this code
           JSR  main1101  * Call function for codes which begin with 1101
           RTS            * Finished disassembling this code
           JSR  main1110  * Call function for codes which begin with 1110
           RTS            * Finished disassembling this code
           JSR  main1111  * Call function for codes which begin with 1111
           RTS            * Finished disassembling this code
*******************************************************************************
* MAIN0000 - Further disassembles instructions which begin with 0000          *
*******************************************************************************
main0000   MOVE.W  D7,D5         * Copy current instruction for shifting
           LSR.W   #$8,D5        * Get the 2nd group of 4 characters by shift
           ANDI.W  #$000F,D5     * Eliminate the remaining half of the word
```

```
           MULU    #$8,D5        * Form offset for jump table lookup
           LEA     tabletwo,A0   * Load tabletwo prior to jumping
           JSR     00(A0,D5)     * Look up function for 2nd 4 digit code
           RTS                   * Finished disassembling this code
*******************************************************************************
* MAIN0001 - Instruction is MOVE.B.  Function will further disassemble it     *
*******************************************************************************
main0001   MOVEQ   #$1,D3      * This is a move instruction
           MOVE.B  #'M',(A4)+  * M
           MOVE.B  #'O',(A4)+  * O
           MOVE.B  #'V',(A4)+  * V
           MOVE.B  #'E',(A4)+  * E
           MOVE.B  #'.',(A4)+  * .
           MOVE.B  #'B',(A4)+  * B
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           JSR     effective   * Good luck, and god speed!
           CMPI.B  #%001,D3    * Is the addressing mode valid?
           BNE     ret_moveb   * Addressing mode is valid, so return
           MOVEQ   #$0,D4      * Bad.  Trying to move byte to address register
ret_moveb  RTS                 * Move function decoded
*******************************************************************************
* MAIN0010 - Instruction is MOVE.L/MOVEA.L  Function will disassemble it      *
*******************************************************************************
main0010   MOVEQ   #$1,D3      * This is a move instruction
           MOVE.B  #'M',(A4)+  * M
           MOVE.B  #'O',(A4)+  * O
           MOVE.B  #'V',(A4)+  * V
           MOVE.B  #'E',(A4)+  * E
           JSR     getregmode  * Find out if this is a MOVEA function
           CMPI    #%001,D2    * Well, is it?
           BEQ     movel_a     * It's not, so don't print A
           MOVE.B  #'.',(A4)+  * .
           MOVE.B  #'L',(A4)+  * L
           MOVE.B  #' ',(A4)+  *
movel_con  MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           JSR     effective   * Good luck, and god speed!
           RTS                 * Move function decoded
movel_a    MOVE.B  #'A',(A4)+  * A
           MOVE.B  #'.',(A4)+  * .
           MOVE.B  #'L',(A4)+  * L
           BRA     movel_con   * Continue with normal spacing
*******************************************************************************
* MAIN0011 - Instruction is MOVE.W/MOVEA.W  Function will disassemble it      *
*******************************************************************************
main0011   MOVEQ   #$1,D3      * This is a move instruction
           MOVE.B  #'M',(A4)+  * M
           MOVE.B  #'O',(A4)+  * O
           MOVE.B  #'V',(A4)+  * V
           MOVE.B  #'E',(A4)+  * E
           JSR     getregmode  * Find out if this is a MOVEA function
           CMPI    #%001,D2    * Well, is it?
           BEQ     movew_a     * It's not, so don't print A
           MOVE.B  #'.',(A4)+  * .
           MOVE.B  #'W',(A4)+  * W
           MOVE.B  #' ',(A4)+  *
movew_con  MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           JSR     effective   * Good luck, and god speed!
           RTS                 * Move function decoded
```

```
movew_a     MOVE.B  #'A',(A4)+  * A
            MOVE.B  #'.',(A4)+  * .
            MOVE.B  #'W',(A4)+  * W
            BRA     movew_con   * Continue with normal spacing
*******************************************************************************
* MAIN0100 - Further disassembles instructions which begin with 0100          *
*******************************************************************************
main0100    CMPI.W  #$4E72,D7     * Is this a stop instruction?
            BEQ     stop_inst     * Yes it is!
            MOVE.W  D7,D5         * Copy current instruction for shifting
            LSR.W   #$8,D5        * Get the 2nd group of 4 characters by shift
            ANDI.W  #$000F,D5     * Eliminate the remaining half of the word
            MULU    #$8,D5        * Form offset for jump table lookup
            LEA     tablethree,A0 * Load tablethree prior to jumping
            JSR     00(A0,D5)     * Look up function for 2nd 4 digit code
            RTS                   * Finished disassembling this code
stop_inst   CMPI.L  #$0,D6        * Is the space left to specify stop address?
            BEQ     bad_ins       * Nope
            MOVE.B  #'S',(A4)+    * S
            MOVE.B  #'T',(A4)+    * T
            MOVE.B  #'O',(A4)+    * O
            MOVE.B  #'P',(A4)+    * P
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #'$',(A4)+    * $
            JSR     getword       * Get the word the holds the stop address
            ADD.L   #2,A5         * Increment next address pointer
            SUBI.L  #1,D6         * Decrement remaining word count
            RTS                   * We've decoded the stop instruction
*******************************************************************************
* MAIN0101 - Further disassembles instructions which begin with 0101          *
*******************************************************************************
main0101    MOVE.W  D7,D5         * Copy current instruction for shifting
            LSR.W   #8,D5         * Shift the 8th bit into place
            ANDI.W  #0001,D5      * Isolate the 8th bit
            CMPI.W  #$0,D5        * Does the 8th bit tell us this is an ADDQ?
            BEQ     addq          * Yes it does!
subq        MOVE.B  #'S',(A4)+    * S
            MOVE.B  #'U',(A4)+    * U
            MOVE.B  #'B',(A4)+    * B
            MOVE.B  #'Q',(A4)+    * Q
            MOVE.B  #'.',(A4)+    * .
            JSR     getsize       * Get the size mode
            CMPI.B  #%00,D2       * Is the instruction SUBQ.B?
            BEQ     subqb         * Yes
            CMPI.B  #%01,D2       * Is the instruction SUBQ.W?
            BEQ     subqw         * Yes
            CMPI.B  #%10,D2       * Is the instruction SUBQ.L?
            BEQ     subql         * Yes
            BRA     bad_ins       * The instruction is malformed
subqb       MOVE.B  #'B',(A4)+    * B
            BRA     subq_con      * Continue with instruction disassembly
subqw       MOVE.B  #'W',(A4)+    * W
            BRA     subq_con      * Continue with instruction disassembly
subql       MOVE.B  #'L',(A4)+    * L
            BRA     subq_con      * Continue with instruction disassembly
subq_con    MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #'#',(A4)+    * #
            JSR     gethighreg    * Get the high register code to add to output
            MULU    #$6,D2        * Form offset for reg_mode jump table
```

```
            LEA     reg_mode,A1   * Load reg_mode table prior to jumping
            JSR     00(A1,D2)     * Jump indirect with index
            MOVE.B  #',',(A4)+    * ,
            BRA     effective     * Fill in the effective address
addq        MOVE.B  #'A',(A4)+    * A
            MOVE.B  #'D',(A4)+    * D
            MOVE.B  #'D',(A4)+    * D
            MOVE.B  #'Q',(A4)+    * Q
            MOVE.B  #'.',(A4)+    * .
            JSR     getsize       * Get the size mode
            CMPI.B  #%00,D2       * Is the instruction ADDQ.B?
            BEQ     addqb         * Yes
            CMPI.B  #%01,D2       * Is the instruction ADDQ.W?
            BEQ     addqw         * Yes
            CMPI.B  #%10,D2       * Is the instruction ADDQ.L?
            BEQ     addql         * Yes
            BRA     bad_ins       * The instruction is malformed
addqb       MOVE.B  #'B',(A4)+    * B
            BRA     addq_con      * Continue with instruction disassembly
addqw       MOVE.B  #'W',(A4)+    * W
            BRA     addq_con      * Continue with instruction disassembly
addql       MOVE.B  #'L',(A4)+    * L
            BRA     addq_con      * Continue with instruction disassembly
addq_con    MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #'#',(A4)+    * #
            JSR     gethighreg    * Get the high register code to add to output
            MULU    #$6,D2        * Form offset for reg_mode jump table
            LEA     reg_mode,A1   * Load reg_mode table prior to jumping
            JSR     00(A1,D2)     * Jump indirect with index
            MOVE.B  #',',(A4)+    * ,
            BRA     effective     * Fill in the effective address
*******************************************************************************
* MAIN0110 - Further disassembles instructions which begin with 0110          *
*******************************************************************************
main0110    MOVE.W  D7,D5         * Copy current instruction for shifting
            LSR.W   #$8,D5        * Get the 2nd group of 4 characters by shift
            ANDI.W  #$000F,D5     * Eliminate the remaining half of the word
            MULU    #$8,D5        * Form offset for jump table lookup
            LEA     tablefour,A0  * Load tablefour prior to jumping
            JSR     00(A0,D5)     * Look up function for 2nd 4 digit code
            RTS                   * Finished disassembling this code
*******************************************************************************
* MAIN0111 - Further disassembles instructions which begin with 0111          *
*******************************************************************************
main0111    MOVE.W  D7,D5         * Copy current instruction for shifting
            LSR.W   #8,D5         * Shift the 8th bit into place
            ANDI.W  #0001,D5      * Isolate the 8th bit
            CMPI.W  #$0,D5        * Does the 8th bit tell us this is a MOVEQ?
            BEQ     moveq         * Yes it does!
            MOVEQ   #$0,D4        * Instruction is unknown
            RTS                   * Nothing to return
moveq       MOVE.B  #'M',(A4)+    * M
            MOVE.B  #'O',(A4)+    * O
            MOVE.B  #'V',(A4)+    * V
            MOVE.B  #'E',(A4)+    * E
            MOVE.B  #'Q',(A4)+    * Q
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #' ',(A4)+    *
            MOVE.B  #'#',(A4)+    * #
            JSR     eightbit      * Add the eight bit offset to the ouput buffer
            MOVE.B  #',',(A4)+    * ,
```

```
          MOVE.B  #'D',(A4)+    * D
          JSR     gethighreg    * Get the high register code to add to output
          MULU    #$6,D2        * Form offset for reg_mode jump table
          LEA     reg_mode,A1   * Load reg_mode table prior to jumping
          JSR     00(A1,D2)     * Jump indirect with index
          RTS                   * We've decoded the moveq instruction
*******************************************************************************
* MAIN1000 - Further disassembles instructions which begin with 1000          *
*******************************************************************************
main1000  JSR    getregmode     * Load the reg mod into D2 prior to jumping
          MULU   #$8,D2         * Form offset for jump table lookup
          LEA    tablefive,A0   * Load tablefive prior to jumping
          JSR    00(A0,D2)      * Look up function for corresponding size code
          RTS                   * Finished disassembling this code
*******************************************************************************
* MAIN1001 - Further disassembles instructions which begin with 1001          *
*******************************************************************************
main1001  JSR    getregmode     * Load the reg mod into D2 prior to jumping
          MULU   #$8,D2         * Form offset for jump table lookup
          LEA    tablenine,A0   * Load tablesix prior to jumping
          JSR    00(A0,D2)      * Look up function for corresponding size code
          RTS                   * Finished disassembling this code
*******************************************************************************
* MAIN1010 - Further disassembles instructions which begin with 1010          *
*******************************************************************************
main1010  MOVEQ  #$0,D4         * Instruction is unknown
          RTS                   * Nothing to return
*******************************************************************************
* MAIN1011 - Further disassembles instructions which begin with 1011          *
*******************************************************************************
main1011  JSR    getregmode     * Load the reg mod into D2 prior to jumping
          MULU   #$8,D2         * Form offset for jump table lookup
          LEA    tableseven,A0  * Load tableseven prior to jumping
          JSR    00(A0,D2)      * Look up function for corresponding size code
          RTS                   * Finished disassembling this code
*******************************************************************************
* MAIN1100 - Further disassembles instructions which begin with 1100          *
*******************************************************************************
main1100  JSR     get3to8       * Get the instruction code from bits 3 to 8
          CMPI.B  #%101000,D2   * Is this an exchange data registers instr?
          BEQ     exgdr         * yes
          CMPI.B  #%101001,D2   * Is this an exchange addr registers instr?
          BEQ     exgar         * yes
          CMPI.B  #%110001,D2   * Is this an exchange addr & data instr?
          BEQ     exgdrar       * yes
          JSR     getregmode    * Load the reg mod into D2 prior to jumping
          MULU    #$8,D2        * Form offset for jump table lookup
          LEA     tablesix,A0   * Load tablesix prior to jumping
          JSR     00(A0,D2)     * Look up function for corresponding size code
          RTS                   * Finished disassembling this code
exgdr     MOVE.B  #'E',(A4)+    * E
          MOVE.B  #'X',(A4)+    * X
          MOVE.B  #'G',(A4)+    * G
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #'D',(A4)+    * D
          JSR     gethighreg    * Get the source register
          MULU    #$6,D2        * Form offset for reg_mode jump table
          LEA     reg_mode,A1   * Load reg_mode table prior to jumping
          JSR     00(A1,D2)     * Jump indirect with index
          MOVE.B  #',',(A4)+    * ,
          MOVE.B  #'D',(A4)+    * D
          JSR     gethighreg    * Get the source register
          MULU    #$6,D2        * Form offset for reg_mode jump table
```

```
          LEA     reg_mode,A1   * Load reg_mode table prior to jumping
          JSR     00(A1,D2)     * Jump indirect with index
          RTS                   * We've disassembled this instruction
exgar     MOVE.B  #'E',(A4)+    * E
          MOVE.B  #'X',(A4)+    * X
          MOVE.B  #'G',(A4)+    * G
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #'A',(A4)+    * A
          JSR     gethighreg    * Get the source register
          MULU    #$6,D2        * Form offset for reg_mode jump table
          LEA     reg_mode,A1   * Load reg_mode table prior to jumping
          JSR     00(A1,D2)     * Jump indirect with index
          MOVE.B  #',',(A4)+    * ,
          MOVE.B  #'A',(A4)+    * A
          JSR     gethighreg    * Get the source register
          MULU    #$6,D2        * Form offset for reg_mode jump table
          LEA     reg_mode,A1   * Load reg_mode table prior to jumping
          JSR     00(A1,D2)     * Jump indirect with indexexgdrar
          RTS                   * We've disassembled this instruction
exgdrar   MOVE.B  #'E',(A4)+    * E
          MOVE.B  #'X',(A4)+    * X
          MOVE.B  #'G',(A4)+    * G
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #' ',(A4)+    *
          MOVE.B  #'D',(A4)+    * D
          JSR     gethighreg    * Get the source register
          MULU    #$6,D2        * Form offset for reg_mode jump table
          LEA     reg_mode,A1   * Load reg_mode table prior to jumping
          JSR     00(A1,D2)     * Jump indirect with index
          MOVE.B  #',',(A4)+    * ,
          MOVE.B  #'A',(A4)+    * A
          JSR     gethighreg    * Get the source register
          MULU    #$6,D2        * Form offset for reg_mode jump table
          LEA     reg_mode,A1   * Load reg_mode table prior to jumping
          JSR     00(A1,D2)     * Jump indirect with index
          RTS                   * We've disassembled this instruction
*************************************************************************
* MAIN1101 - Further disassembles instructions which begin with 1101    *
*************************************************************************
main1101  JSR    getregmode     * Load the reg mod into D2 prior to jumping
          MULU   #$8,D2         * Form offset for jump table lookup
          LEA    tableeight,A0  * Load tablesix prior to jumping
          JSR    00(A0,D2)      * Look up function for corresponding size code
          RTS                   * Finished disassembling this code
*************************************************************************
* MAIN1110 - Further disassembles instructions which begin with 1110    *
*************************************************************************
main1110  JSR     getsize        * Get size to determine instruction type
          CMPI.B  #%00,D2        * Is this a byte size shift instruction?
          BEQ     shiftbyte      * Yes it is
          CMPI.B  #%01,D2        * Is this a word size shift instruction?
          BEQ     shiftword      * Yes it is
          CMPI.B  #%10,D2        * Is this a long size shift instruction?
          BEQ     shiftlong      * Yes it is
          JSR     get9to10       * Get the code from bits 9 to 10
          CMPI.B  #%00,D2        * Is this an arithmetic shift instruction?
```

```
            BEQ     shftmasd        * Yes it is
            CMPI.B  #%01,D2         * Is this a logical shift instruction?
            BEQ     shftmlsd        * Yes it is
            CMPI.B  #%10,D2         * Is this an extended roll instruction?
            BEQ     shftmroxd       * Yes it is
            JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this a roll right instruction?
            BEQ     shftmror        * Yes it is
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #' ',(A4)+      *
            BRA     shiftmem        * Continue disassembling shift instruction
shftmror    MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #' ',(A4)+      *
            BRA     shiftmem        * Continue disassembling shift instruction
shftmroxd   JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this an extended roll right instruction?
            BEQ     shftmroxr       * Yes it is
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'X',(A4)+      * X
            MOVE.B  #'L',(A4)+      * L
            BRA     shiftmem        * Continue disassembling shift instruction
shftmroxr   MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'X',(A4)+      * X
            MOVE.B  #'R',(A4)+      * R
            BRA     shiftmem        * Continue disassembling shift instruction
shftmlsd    JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this a logical shift right instruction?
            BEQ     shftmlsr        * Yes it is
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #' ',(A4)+      *
            BRA     shiftmem        * Continue disassembling shift instruction
shftmlsr    MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #' ',(A4)+      *
            BRA     shiftmem        * Continue disassembling shift instruction
shftmasd    JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this an arithmetic shift right?
            BEQ     shftmasr        * Yes it is
            MOVE.B  #'A',(A4)+      * A
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #' ',(A4)+      *
            BRA     shiftmem        * Continue disassembling shift instruction
shftmasr    MOVE.B  #'A',(A4)+      * A
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #' ',(A4)+      *
            BRA     shiftmem        * Continue disassembling shift instruction
shiftmem    MOVE.B  #' ',(A4)+      *
            MOVE.B  #' ',(A4)+      *
            MOVE.B  #' ',(A4)+      *
            MOVE.B  #' ',(A4)+      *
            MOVE.B  #' ',(A4)+      *
            MOVE.B  #' ',(A4)+      *
            MOVE.B  #' ',(A4)+      *
            MOVE.B  #' ',(A4)+      *
            JSR     effective       * Fill out the effective address information
            CMPI.B  #%001,D3        * Is the addressing mode illegal?
            BLE     bad_ins         * Yes, the instruction is malformed
            RTS                     * We've decoded the memory shift instruction
shiftbyte   JSR     get3to4         * Get the code in bits 3 to 4
            CMPI.B  #%00,D2         * Is this an arithmetic shift instruction?
```

```
            BEQ     shftasdb        * Yes it is
            CMPI.B  #%01,D2         * Is this a logical shift instruction?
            BEQ     shftlsdb        * Yes it is
            CMPI.B  #%10,D2         * Is this an extended roll instruction?
            BEQ     shftroxdb       * Yes it is
            JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this a roll right instruction?
            BEQ     shftrorb        * Yes it is
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'B',(A4)+      * B
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftrorb    MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'B',(A4)+      * B
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftroxdb   JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this an extended roll right instruction?
            BEQ     shftroxrb       * Yes it is
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'X',(A4)+      * X
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'B',(A4)+      * B
            BRA     shiftcon        * Continue disassembling shift instruction
shftroxrb   MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'X',(A4)+      * X
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'B',(A4)+      * B
            BRA     shiftcon        * Continue disassembling shift instruction
shftlsdb    JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this a logical shift right instruction?
            BEQ     shftlsrb        * Yes it is
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'B',(A4)+      * B
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftlsrb    MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'B',(A4)+      * B
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftasdb    JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this an arithmetic shift right?
            BEQ     shftasrb        * Yes it is
            MOVE.B  #'A',(A4)+      * A
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'B',(A4)+      * B
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftasrb    MOVE.B  #'A',(A4)+      * A
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'B',(A4)+      * B
```

```
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shiftword   JSR     get3to4         * Get the code in bits 3 to 4
            CMPI.B  #%00,D2         * Is this an arithmetic shift instruction?
            BEQ     shftasdw        * Yes it is
            CMPI.B  #%01,D2         * Is this a logical shift instruction?
            BEQ     shftlsdw        * Yes it is
            CMPI.B  #%10,D2         * Is this an extended roll instruction?
            BEQ     shftroxdw       * Yes it is
            JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this a roll right instruction?
            BEQ     shftrorw        * Yes it is
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'W',(A4)+      * W
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftrorw    MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'W',(A4)+      * W
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftroxdw   JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this an extended roll right instruction?
            BEQ     shftroxrw       * Yes it is
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'X',(A4)+      * X
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'W',(A4)+      * W
            BRA     shiftcon        * Continue disassembling shift instruction
shftroxrw   MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'O',(A4)+      * O
            MOVE.B  #'X',(A4)+      * X
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'W',(A4)+      * W
            BRA     shiftcon        * Continue disassembling shift instruction
shftlsdw    JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this a logical shift right instruction?
            BEQ     shftlsrw        * Yes it is
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'W',(A4)+      * W
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftlsrw    MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'R',(A4)+      * R
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'W',(A4)+      * W
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftasdw    JSR     getbit8         * Get the 8th bit to determine direction
            CMPI.B  #%01,D2         * Is this an arithmetic shift right?
            BEQ     shftasrw        * Yes it is
            MOVE.B  #'A',(A4)+      * A
            MOVE.B  #'S',(A4)+      * S
            MOVE.B  #'L',(A4)+      * L
            MOVE.B  #'.',(A4)+      * .
            MOVE.B  #'W',(A4)+      * W
            MOVE.B  #' ',(A4)+      *
            BRA     shiftcon        * Continue disassembling shift instruction
shftasrw    MOVE.B  #'A',(A4)+      * A
```

```
           MOVE.B  #'S',(A4)+     * S
           MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'.',(A4)+     * .
           MOVE.B  #'W',(A4)+     * W
           MOVE.B  #' ',(A4)+     *
           BRA     shiftcon       * Continue disassembling shift instruction
shiftlong  JSR     get3to4        * Get the code in bits 3 to 4
           CMPI.B  #%00,D2        * Is this an arithmetic shift instruction?
           BEQ     shftasdl       * Yes it is
           CMPI.B  #%01,D2        * Is this a logical shift instruction?
           BEQ     shftlsdl       * Yes it is
           CMPI.B  #%10,D2        * Is this an extended roll instruction?
           BEQ     shftroxdl      * Yes it is
           JSR     getbit8        * Get the 8th bit to determine direction
           CMPI.B  #%01,D2        * Is this a roll right instruction?
           BEQ     shftrorl       * Yes it is
           MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'O',(A4)+     * O
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #'.',(A4)+     * .
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #' ',(A4)+     *
           BRA     shiftcon       * Continue disassembling shift instruction
shftrorl   MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'O',(A4)+     * O
           MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'.',(A4)+     * .
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #' ',(A4)+     *
           BRA     shiftcon       * Continue disassembling shift instruction
shftroxdl  JSR     getbit8        * Get the 8th bit to determine direction
           CMPI.B  #%01,D2        * Is this an extended roll right instruction?
           BEQ     shftroxrl      * Yes it is
           MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'O',(A4)+     * O
           MOVE.B  #'X',(A4)+     * X
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #'.',(A4)+     * .
           MOVE.B  #'L',(A4)+     * L
           BRA     shiftcon       * Continue disassembling shift instruction
shftroxrl  MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'O',(A4)+     * O
           MOVE.B  #'X',(A4)+     * X
           MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'.',(A4)+     * .
           MOVE.B  #'L',(A4)+     * L
           BRA     shiftcon       * Continue disassembling shift instruction
shftlsdl   JSR     getbit8        * Get the 8th bit to determine direction
           CMPI.B  #%01,D2        * Is this a logical shift right instruction?
           BEQ     shftlsrl       * Yes it is
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #'S',(A4)+     * S
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #'.',(A4)+     * .
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #' ',(A4)+     *
           BRA     shiftcon       * Continue disassembling shift instruction
shftlsrl   MOVE.B  #'L',(A4)+     * L
           MOVE.B  #'S',(A4)+     * S
           MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'.',(A4)+     * .
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #' ',(A4)+     *
           BRA     shiftcon       * Continue disassembling shift instruction
shftasdl   JSR     getbit8        * Get the 8th bit to determine direction
           CMPI.B  #%01,D2        * Is this an arithmetic shift right?
           BEQ     shftasrl       * Yes it is
           MOVE.B  #'A',(A4)+     * A
           MOVE.B  #'S',(A4)+     * S
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #'.',(A4)+     * .
```

```
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #' ',(A4)+     *
           BRA     shiftcon       * Continue disassembling shift instruction
shftasrl   MOVE.B  #'A',(A4)+     * A
           MOVE.B  #'S',(A4)+     * S
           MOVE.B  #'R',(A4)+     * R
           MOVE.B  #'.',(A4)+     * .
           MOVE.B  #'L',(A4)+     * L
           MOVE.B  #' ',(A4)+     *
           BRA     shiftcon       * Continue disassembling shift instruction
shiftcon   MOVE.B  #' ',(A4)+     *
           MOVE.B  #' ',(A4)+     *
           MOVE.B  #' ',(A4)+     *
           MOVE.B  #' ',(A4)+     *
           MOVE.B  #' ',(A4)+     *
           MOVE.B  #' ',(A4)+     *
           JSR     getbit5        * Get the 5th bit to determine shift type
           CMPI.B  #%01,D2        * Is the shift using a register for amount?
           BEQ     shiftreg       * Yes it is
           MOVE.B  #'#',(A4)+     * #
           JSR     gethighreg     * Get the shift amount
           CMPI.B  #%000,D2       * Are we shifting 8 bits?
           BEQ     shfteight      * We've read the shift 8 code (000), so yes
           MULU    #$6,D2         * Form offset for reg_mode jump table
           LEA     reg_mode,A1    * Load reg_mode table prior to jumping
           JSR     00(A1,D2)      * Jump indirect with index
           BRA     shiftcon2      * Continue disassembling shift instruction
shfteight  MOVE.B  #'8',(A4)+     * 8
           BRA     shiftcon2      * Continue disassembling shift instruction
shiftreg   MOVE.B  #'D',(A4)+     * D
           JSR     gethighreg     * Get the shift amount
           MULU    #$6,D2         * Form offset for reg_mode jump table
           LEA     reg_mode,A1    * Load reg_mode table prior to jumping
           JSR     00(A1,D2)      * Jump indirect with index
shiftcon2  MOVE.B  #',',(A4)+      * ,
           MOVE.B  #'D',(A4)+      * D
           JSR     getreg         * Load the register to shift
           MULU    #$6,D2         * Form offset for reg_mode jump table
           LEA     reg_mode,A1    * Load reg_mode table prior to jumping
           JSR     00(A1,D2)      * Jump indirect with index
           RTS                    * Finished disassembling instruction
*************************************************************************
* MAIN1111 - Further disassembles instructions which begin with 1111    *
*************************************************************************
main1111   MOVEQ  #$0,D4       * Instruction is unknown
           RTS                 * Nothing to return
*************************************************************************
* SECONDARY JUMP TABLE - Contains lookup functions for the next 4 bits for    *
*                        instructions that begin with 0000.                   *
*************************************************************************
tabletwo   JSR     two0000     * Lookup function with following 4 bits of 0000
           RTS                 * Finished disassembling this code
           JSR     two0001     * Lookup function with following 4 bits of 0001
           RTS                 * Finished disassembling this code
           JSR     two0010     * Lookup function with following 4 bits of 0010
           RTS                 * Finished disassembling this code
           JSR     two0011     * Lookup function with following 4 bits of 0011
           RTS                 * Finished disassembling this code
           JSR     two0100     * Lookup function with following 4 bits of 0100
           RTS                 * Finished disassembling this code
           JSR     two0101     * Lookup function with following 4 bits of 0101
           RTS                 * Finished disassembling this code
           JSR     two0110     * Lookup function with following 4 bits of 0110
           RTS                 * Finished disassembling this code
           JSR     two0111     * Lookup function with following 4 bits of 0111
           RTS                 * Finished disassembling this code
           JSR     two1000     * Lookup function with following 4 bits of 1000
           RTS                 * Finished disassembling this code
           JSR     two1001     * Lookup function with following 4 bits of 1001
           RTS                 * Finished disassembling this code
           JSR     two1010     * Lookup function with following 4 bits of 1010
```

```
           RTS                 * Finished disassembling this code
           JSR     two1011     * Lookup function with following 4 bits of 1011
           RTS                 * Finished disassembling this code
           JSR     two1100     * Lookup function with following 4 bits of 1100
           RTS                 * Finished disassembling this code
           JSR     two1101     * Lookup function with following 4 bits of 1101
           RTS                 * Finished disassembling this code
           JSR     two1110     * Lookup function with following 4 bits of 1110
           RTS                 * Finished disassembling this code
           JSR     two1111     * Lookup function with following 4 bits of 1111
           RTS                 * Finished disassembling this code
*******************************************************************************
* TWO0000 - Instruction is ORI.  This function will further disassemble it    *
*******************************************************************************
two0000    MOVE.B   #'O',(A4)+  * O
           MOVE.B   #'R',(A4)+  * R
           MOVE.B   #'I',(A4)+  * I
           MOVE.B   #'.',(A4)+  * .
           CMPI.L   #$0,D6      * Is there an address left for the immediate
                                * data in this instruction?
           BEQ      bad_ori     * Nope, we won't be able to decode it entirely!
           JSR      getsize     * Figure out what size the ORI instruction is
           CMP.W    #%00,D2     * Byte?
           BEQ      byte_ori    * Yes
           CMP.W    #%01,D2     * Word?
           BEQ      word_ori    * Yes
           CMP.W    #%10,D2     * Long?
           BEQ      long_ori    * Yes
bad_ori    MOVEQ    #$0,D4      * Malformed instruction!
           RTS                  * Instruction is bad, so stop disassembling
con_ori    JSR      getword     * Get the word at A5 and place it in good buf
           ADD.L    #$2,A5      * Increment A5 by one word
           SUBI.L   #$1,D6      * Decrement remaining word count
           MOVE.B   #',',(A4)+  * ,
           JSR      effective   * Get the effective address
           CMPI.L   #0,D4       * Did effective addressing detect a bad
                                * instruction?
           BEQ      dec_ori     * Yes, the instruction is bad
           CMPI.B   #%001,D3    * Is the addressing mode invalid?
           BEQ      dec_ori     * Yes, the addressing mode is invalid
           RTS                  * Return disassembled instruction
dec_ori    ADD.L    #$1,D6      * Because instruction is bad, we re-increment
                                * remaining word count to disassociate the
                                * proceeding word which we interpreted as an
                                * immediate address
           SUB.L    #$2,A5      * For the same reasons we will treat the
                                * proceeding word as its own instruction and
                                * not part of this bad insturction (data)
           BRA      bad_ori     * Instruction is bad
byte_ori   MOVE.B   #'B',(A4)+  * B
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #'#',(A4)+  * #
           MOVE.B   #'$',(A4)+  * $
           BRA      con_ori     * Continue appending immediate information
word_ori   MOVE.B   #'W',(A4)+  * W
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #'#',(A4)+  * #
           MOVE.B   #'$',(A4)+  * $
           BRA      con_ori     * Continue appending immediate information
```

```
long_ori   CMPI.L   #1,D6       * Is there an address left for the immediate
                               * data in this instruction?
           BEQ      bad_ori     * Nope, we won't be able to decode it entirely!
           MOVE.B   #'L',(A4)+  * L
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #'#',(A4)+  * #
           MOVE.B   #'$',(A4)+  * $
           JSR      getlong     * Get the long at A5 and place it in good buf
           ADD.L    #$4,A5      * Increment A5 by two words
           SUBI.L   #$2,D6      * Decrement remaining word count by two
           MOVE.B   #',',(A4)+  * ,
           JSR      effective   * Get the effective address
           CMPI.L   #0,D4       * Did effective addressing detect a bad
                               * instruction?
           BEQ      dec_oril    * Yes, the instruction is bad
           CMPI.B   #%001,D3    * Is the addressing mode invalid?
           BEQ      dec_oril    * Yes, the addressing mode is invalid
           RTS                  * Return from instruction disassembly
dec_oril   ADD.L    #$2,D6      * Because instruction is bad, we re-increment
                               * remaining word count to disassociate the
                               * proceeding long which we interpreted as an
                               * immediate address
           SUB.L    #$4,A5      * For the same reasons we will treat the
                               * proceeding long as its own instruction and
                               * not part of this bad insturction (data)
           BRA      bad_ori     * Instruction is bad
*******************************************************************************
* TWO0001 - Further disassembles instructions that began with 00000001        *
*******************************************************************************
two0001    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO0010 - Further disassembles instructions that began with 00000010        *
*******************************************************************************
two0010    MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'N',(A4)+   * N
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'I',(A4)+   * I
           MOVE.B  #'.',(A4)+   * .
           JSR     immediate    * Rules for all immediate instructions
           RTS                  * Finished decoding
*******************************************************************************
* TWO0011 - Further disassembles instructions that began with 00000011        *
*******************************************************************************
two0011    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO0100 - Instruction is SUBI.  This function will further disassemble it   *
*******************************************************************************
two0100    MOVE.B  #'S',(A4)+   * S
           MOVE.B  #'U',(A4)+   * U
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #'I',(A4)+   * I
           MOVE.B  #'.',(A4)+   * .
           JSR     immediate    * Rules for all immediate instructions
           RTS                  * Finished decoding
*******************************************************************************
* TWO0101 - Further disassembles instructions that began with 00000101        *
*******************************************************************************
two0101    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO0110 - Instruction is ADDI.  This function will further disassemble it   *
*******************************************************************************
two0110    MOVE.B  #'A',(A4)+   * A
```

```
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'I',(A4)+   * I
           MOVE.B  #'.',(A4)+   * .
           JSR     immediate    * Rules for all immediate instructions
           RTS                  * Finished decoding
*******************************************************************************
* TWO0111 - Further disassembles instructions that began with 00000111        *
*******************************************************************************
two0111    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO1000 - Further disassembles instructions that began with 00000111        *
*******************************************************************************
two1000    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO1001 - Further disassembles instructions that began with 00001001        *
*******************************************************************************
two1001    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO1010 - Instruction is EORI.  This function will further disassemble it   *
*******************************************************************************
two1010    MOVE.B  #'E',(A4)+   * E
           MOVE.B  #'O',(A4)+   * O
           MOVE.B  #'R',(A4)+   * R
           MOVE.B  #'I',(A4)+   * I
           MOVE.B  #'.',(A4)+   * .
           JSR     immediate    * Rules for all immediate instructions
           RTS                  * Finished decoding
*******************************************************************************
* TWO1011 - Further disassembles instructions that began with 00001011        *
*******************************************************************************
two1011    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO1100 - Instruction is CMPI.  This function will further disassemble it   *
*******************************************************************************
two1100    MOVE.B  #'C',(A4)+   * C
           MOVE.B  #'M',(A4)+   * M
           MOVE.B  #'P',(A4)+   * P
           MOVE.B  #'I',(A4)+   * I
           MOVE.B  #'.',(A4)+   * .
           JSR     immediate    * Rules for all immediate instructions
           RTS                  * Finished decoding
*******************************************************************************
* TWO1101 - Further disassembles instructions that began with 00001101        *
*******************************************************************************
two1101    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO1110 - Further disassembles instructions that began with 00001110        *
*******************************************************************************
two1110    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TWO1111 - Further disassembles instructions that began with 00001111        *
*******************************************************************************
two1111    MOVEQ  #$0,D4        * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* TERTIARY JUMP TABLE - Contains lookup functions for the next 4 bits for     *
*                       instructions that begin with 0100.                    *
*******************************************************************************
tablethree  JSR     three0000   * Lookup function with following 4 bits of 0000
            RTS                 * Finished disassembling this code
            JSR     three0001   * Lookup function with following 4 bits of 0001
            RTS                 * Finished disassembling this code
            JSR     three0010   * Lookup function with following 4 bits of 0010
            RTS                 * Finished disassembling this code
```

```
            JSR     three0011   * Lookup function with following 4 bits of 0011
            RTS                 * Finished disassembling this code
            JSR     three0100   * Lookup function with following 4 bits of 0100
            RTS                 * Finished disassembling this code
            JSR     three0101   * Lookup function with following 4 bits of 0101
            RTS                 * Finished disassembling this code
            JSR     three0110   * Lookup function with following 4 bits of 0110
            RTS                 * Finished disassembling this code
            JSR     three0111   * Lookup function with following 4 bits of 0111
            RTS                 * Finished disassembling this code
            JSR     three1000   * Lookup function with following 4 bits of 1000
            RTS                 * Finished disassembling this code
            JSR     three1001   * Lookup function with following 4 bits of 1001
            RTS                 * Finished disassembling this code
            JSR     three1010   * Lookup function with following 4 bits of 1010
            RTS                 * Finished disassembling this code
            JSR     three1011   * Lookup function with following 4 bits of 1011
            RTS                 * Finished disassembling this code
            JSR     three1100   * Lookup function with following 4 bits of 1100
            RTS                 * Finished disassembling this code
            JSR     three1101   * Lookup function with following 4 bits of 1101
            RTS                 * Finished disassembling this code
            JSR     three1110   * Lookup function with following 4 bits of 1110
            RTS                 * Finished disassembling this code
            JSR     three1111   * Lookup function with following 4 bits of 1111
            RTS                 * Finished disassembling this code
*************************************************************************
* THREE0000 - Further disassembles instructions that began with 01000000     *
*************************************************************************
three0000   MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
*************************************************************************
* THREE0001 - Further disassembles instructions that began with 01000001     *
*************************************************************************
three0001   JSR     getsize     * Get size to further decode instruction
            CMP.B   #%11,D2     * Is this an LEA instruction?
            BEQ     lea         * Yes
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
*************************************************************************
* THREE0010 - Instruction is CLR.  This function will further disassemble it  *
*************************************************************************
three0010  MOVE.B   #'C',(A4)+  * C
           MOVE.B   #'L',(A4)+  * L
           MOVE.B   #'R',(A4)+  * R
           MOVE.B   #'.',(A4)+  * .
bra_clr    JSR      getsize     * Figure out what size the CLR instruction is
           CMP.W    #%00,D2     * Byte?
           BEQ      byte_clr    * Yes
           CMP.W    #%01,D2     * Word?
           BEQ      word_clr    * Yes
           CMP.W    #%10,D2     * Long?
           BEQ      long_clr    * Yes
bad_clr    MOVEQ    #$0,D4      * Malformed instruction!
           RTS                  * Instruction is bad, so stop disassembling
con_clr    MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           JSR      effective   * Get the effective address
           CMPI.B   #%001,D3    * Is the addressing mode invalid?
           BEQ      bad_clr     * Yes, the addressing mode is invalid
           RTS                  * Return disassembled instruction
byte_clr   MOVE.B   #'B',(A4)+  * B
           BRA      con_clr     * Continue decoding instruction
word_clr   MOVE.B   #'W',(A4)+  * W
           BRA      con_clr     * Continue decoding instruction
long_clr   MOVE.B   #'L',(A4)+  * L
```

```
           BRA      con_clr     * Continue decoding instruction
*******************************************************************************
* THREE0011 - Further disassembles instructions that began with 01000011      *
*******************************************************************************
three0011  JSR      getsize     * Get size to further decode instruction
           CMP.B    #%11,D2     * Is this an LEA instruction?
           BEQ      lea         * Yes
           MOVEQ    #$0,D4      * Instruction is unknown
           RTS                  * Nothing to return
*******************************************************************************
* THREE0100 - Instruction is NEG.  This function will further disassemble it  *
*******************************************************************************
three0100  MOVE.B   #'N',(A4)+  * N
           MOVE.B   #'E',(A4)+  * E
           MOVE.B   #'G',(A4)+  * G
           MOVE.B   #'.',(A4)+  * .
           BRA      bra_clr     * NEG function is identical to CLR after this
*******************************************************************************
* THREE0101 - Further disassembles instructions that began with 01000101      *
*******************************************************************************
three0101   JSR     getsize     * Get size to further decode instruction
            CMP.B   #%11,D2     * Is this an LEA instruction?
            BEQ     lea         * Yes
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
*******************************************************************************
* THREE0100 - Instruction is NOT.  This function will further disassemble it  *
*******************************************************************************
three0110  MOVE.B   #'N',(A4)+  * N
           MOVE.B   #'O',(A4)+  * O
           MOVE.B   #'T',(A4)+  * T
           MOVE.B   #'.',(A4)+  * .
           BRA      bra_clr     * NOT function is identical to CLR after this
*******************************************************************************
* THREE0111 - Further disassembles instructions that began with 01000111      *
*******************************************************************************
three0111   JSR     getsize     * Get size to further decode instruction
            CMP.B   #%11,D2     * Is this an LEA instruction?
            BEQ     lea         * Yes
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
*******************************************************************************
* THREE1000 - Further disassembles instructions that began with 01001000      *
*******************************************************************************
three1000   JSR     get3to7     * Check pattern from 3 to 7 (for SWAP code)
            CMP.B   #%01000,D2  * Is this a SWAP instruction?
            BEQ     swap        * Yes, this is a SWAP instruction
            JSR     getsize     * Check size (for MOVEM codes)
            CMP.B   #%10,D2     * Is this a MOVEM.W instruction?
            BEQ     movem1_w    * Yes, this is a MOVEM.W instruction
            CMP.B   #%11,D2     * Is this a MOVEM.L instruction?
            BEQ     movem1_l    * Yes, this is a MOVEM.L instruction
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
swap        MOVE.B  #'S',(A4)+  * S
            MOVE.B  #'W',(A4)+  * W
            MOVE.B  #'A',(A4)+  * A
            MOVE.B  #'P',(A4)+  * P
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #'D',(A4)+  * D
            JSR     getreg      * Get the register we're swapping
            MULU    #$6,D2      * Form offset for reg_mode jump table
            LEA     reg_mode,A1 * Load reg_mode table prior to jumping
            JSR     00(A1,D2)   * Jump indirect with index
```

```
            RTS                 * We're done decoding the SWAP instruction
movem1_w    MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'O',(A4)+  * O
            MOVE.B  #'V',(A4)+  * V
            MOVE.B  #'E',(A4)+  * E
            MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'.',(A4)+  * .
            MOVE.B  #'W',(A4)+  * W
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            JSR     effective   * Include the effective address
            CMP.B   #$0,D4      * Was the bad flag set?
            BEQ     bad_ins     * Yes, the instruction is malformed
            CMP.B   #%000,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%001,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%100,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            MOVE.B  #',',(A4)+  * ,
            MOVE.B  #'$',(A4)+  * $
            JSR     getword     * Get the word at A5 and place it in good buf
            ADD.L   #$2,A5      * Increment A5 by one word
            SUBI.L  #$1,D6      * Decrement remaining word count by one
            RTS                 * We've finished decoding this instruction
movem1_l    MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'O',(A4)+  * O
            MOVE.B  #'V',(A4)+  * V
            MOVE.B  #'E',(A4)+  * E
            MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'.',(A4)+  * .
            MOVE.B  #'L',(A4)+  * L
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            JSR     effective   * Include the effective address
            CMP.B   #$0,D4      * Was the bad flag set?
            BEQ     bad_ins     * Yes, the instruction is malformed
            CMP.B   #%000,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%001,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%100,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            MOVE.B  #',',(A4)+  * ,
            MOVE.B  #'$',(A4)+  * $
            JSR     getlong     * Get the long at A5 and place it in good buf
            ADD.L   #$4,A5      * Increment A5 by two words
            SUBI.L  #$2,D6      * Decrement remaining word count by two
            RTS                 * We've finished decoding this instruction
*******************************************************************************
* THREE1001 - Further disassembles instructions that began with 01001001      *
*******************************************************************************
three1001   JSR     getsize     * Get size to further decode instruction
            CMP.B   #%11,D2     * Is this an LEA instruction?
            BEQ     lea         * Yes
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
*******************************************************************************
* THREE1010 - Further disassembles instructions that began with 01001010      *
*******************************************************************************
three1010   MOVEQ  #$0,D4       * Instruction is unknown
            RTS                 * Nothing to return
*******************************************************************************
* THREE1011 - Further disassembles instructions that began with 01001011      *
*******************************************************************************
```

```
three1011   JSR     getsize     * Get size to further decode instruction
            CMP.B   #%11,D2     * Is this an LEA instruction?
            BEQ     lea         * Yes
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
*******************************************************************************
* THREE1100 - Further disassembles instructions that began with 01001100      *
*******************************************************************************
three1100   JSR     getsize     * Check size (for MOVEM codes)
            CMP.B   #%10,D2     * Is this a MOVEM.W instruction?
            BEQ     movem2_w    * Yes, this is a MOVEM.W instruction
            CMP.B   #%11,D2     * Is this a MOVEM.L instruction?
            BEQ     movem2_l    * Yes, this is a MOVEM.L instruction
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
movem2_w    MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'O',(A4)+  * O
            MOVE.B  #'V',(A4)+  * V
            MOVE.B  #'E',(A4)+  * E
            MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'.',(A4)+  * .
            MOVE.B  #'W',(A4)+  * W
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #'$',(A4)+  * $
            JSR     getword     * Get the word at A5 and place it in good buf
            ADD.L   #$2,A5      * Increment A5 by one word
            SUBI.L  #$1,D6      * Decrement remaining word count by one
            MOVE.B  #',',(A4)+  * ,
            JSR     effective   * Include the effective address
            CMP.B   #$0,D4      * Was the bad flag set?
            BEQ     bad_incw    * Yes, the instruction is malformed
            CMP.B   #%000,D3    * Is the addressing mode illegal?
            BEQ     bad_incw    * Yes, it's illegal
            CMP.B   #%001,D3    * Is the addressing mode illegal?
            BEQ     bad_incw    * Yes, it's illegal
            CMP.B   #%100,D3    * Is the addressing mode illegal?
            BEQ     bad_incw    * Yes, it's illegal
            RTS                 * We've finished decoding this instruction
movem2_l    MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'O',(A4)+  * O
            MOVE.B  #'V',(A4)+  * V
            MOVE.B  #'E',(A4)+  * E
            MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'.',(A4)+  * .
            MOVE.B  #'W',(A4)+  * W
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #'$',(A4)+  * $
            JSR     getlong     * Get the long at A5 and place it in good buf
            ADD.L   #$2,A5      * Increment A5 by two words
            SUBI.L  #$1,D6      * Decrement remaining word count by two
            MOVE.B  #',',(A4)+  * ,
            JSR     effective   * Include the effective address
            CMP.B   #$0,D4      * Was the bad flag set?
            BEQ     bad_incl    * Yes, the instruction is malformed
            CMP.B   #%000,D3    * Is the addressing mode illegal?
            BEQ     bad_incl    * Yes, it's illegal
            CMP.B   #%001,D3    * Is the addressing mode illegal?
            BEQ     bad_incl    * Yes, it's illegal
            CMP.B   #%100,D3    * Is the addressing mode illegal?
            BEQ     bad_incl    * Yes, it's illegal
            RTS                 * We've finished decoding this instruction
*******************************************************************************
* THREE1101 - Further disassembles instructions that began with 01001101      *
```

```
*******************************************************************************
three1101   JSR     getsize     * Get size to further decode instruction
            CMP.B   #%11,D2     * Is this an LEA instruction?
            BEQ     lea         * Yes
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
*******************************************************************************
* THREE1110 - Further disassembles instructions that began with 01001110      *
*******************************************************************************
three1110   JSR     getsize     * Get size to further decode instruction
            CMP.B   #%11,D2     * Is this a JMP instruction?
            BEQ     jmp         * Yes
            CMP.B   #%10,D2     * Is this a JSR instruction?
            BEQ     jsr         * Yes
            CMP.B   #%01,D2     * Could this be a NOP or RTS instruction?
            BEQ     noprts      * Yes
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
jmp         MOVE.B  #'J',(A4)+  * J
            MOVE.B  #'M',(A4)+  * M
            MOVE.B  #'P',(A4)+  * P
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            JSR     effective   * Include the effective address
            CMP.B   #$0,D4      * Was the bad flag set?
            BEQ     bad_ins     * Yes, the instruction is malformed
            CMP.B   #%000,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%001,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%011,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%100,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            RTS                 * We've finished decoding this instruction
jsr         MOVE.B  #'J',(A4)+  * J
            MOVE.B  #'S',(A4)+  * S
            MOVE.B  #'R',(A4)+  * R
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            JSR     effective   * Include the effective address
            CMP.B   #$0,D4      * Was the bad flag set?
            BEQ     bad_ins     * Yes, the instruction is malformed
            CMP.B   #%000,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%001,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%011,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            CMP.B   #%100,D3    * Is the addressing mode illegal?
            BEQ     bad_ins     * Yes, it's illegal
            RTS                 * We've finished decoding this instruction
noprts      JSR     getreg      * Load the first 3 bits for comparison
            CMP.B   #%001,D2    * Is this a NOP instruction?
            BEQ     nop         * Yes
            CMP.B   #%101,D2    * Is this an RTS instruction?
            BEQ     rts         * Yes
```

```
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
nop         MOVE.B  #'N',(A4)+  * N
            MOVE.B  #'O',(A4)+  * O
            MOVE.B  #'P',(A4)+  * P
            RTS                 * We've finished decoding this instruction
rts         MOVE.B  #'R',(A4)+  * R
            MOVE.B  #'T',(A4)+  * T
            MOVE.B  #'S',(A4)+  * S
            RTS                 * We've finished decoding this instruction
*******************************************************************************
* THREE1111 - Further disassembles instructions that began with 01001111      *
*******************************************************************************
three1111   JSR     getsize     * Get size to further decode instruction
            CMP.B   #%11,D2     * Is this an LEA instruction?
            BEQ     lea         * Yes
            MOVEQ   #$0,D4      * Instruction is unknown
            RTS                 * Nothing to return
*******************************************************************************
* QUATERNARY JUMP TABLE - Contains lookup functions for the next 4 bits for   *
*                         instructions that begin with 0110.                  *
*******************************************************************************
tablefour   JSR     four0000    * Lookup function with following 4 bits of 0000
            RTS                 * Finished disassembling this code
            JSR     four0001    * Lookup function with following 4 bits of 0001
            RTS                 * Finished disassembling this code
            JSR     four0010    * Lookup function with following 4 bits of 0010
            RTS                 * Finished disassembling this code
            JSR     four0011    * Lookup function with following 4 bits of 0011
            RTS                 * Finished disassembling this code
            JSR     four0100    * Lookup function with following 4 bits of 0100
            RTS                 * Finished disassembling this code
            JSR     four0101    * Lookup function with following 4 bits of 0101
            RTS                 * Finished disassembling this code
            JSR     four0110    * Lookup function with following 4 bits of 0110
            RTS                 * Finished disassembling this code
            JSR     four0111    * Lookup function with following 4 bits of 0111
            RTS                 * Finished disassembling this code
            JSR     four1000    * Lookup function with following 4 bits of 1000
            RTS                 * Finished disassembling this code
            JSR     four1001    * Lookup function with following 4 bits of 1001
            RTS                 * Finished disassembling this code
            JSR     four1010    * Lookup function with following 4 bits of 1010
            RTS                 * Finished disassembling this code
            JSR     four1011    * Lookup function with following 4 bits of 1011
            RTS                 * Finished disassembling this code
            JSR     four1100    * Lookup function with following 4 bits of 1100
            RTS                 * Finished disassembling this code
            JSR     four1101    * Lookup function with following 4 bits of 1101
            RTS                 * Finished disassembling this code
            JSR     four1110    * Lookup function with following 4 bits of 1110
            RTS                 * Finished disassembling this code
            JSR     four1111    * Lookup function with following 4 bits of 1111
            RTS                 * Finished disassembling this code
*******************************************************************************
* FOUR0000 - Instruction is BRA.  This will further disassemble it            *
*******************************************************************************
four0000    MOVE.B  #'B',(A4)+  * B
            MOVE.B  #'R',(A4)+  * R
            MOVE.B  #'A',(A4)+  * A
            BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR0001 - Instruction is BSR.  This will further disassemble it            *
*******************************************************************************
four0001    MOVE.B  #'B',(A4)+  * B
            MOVE.B  #'S',(A4)+  * S
            MOVE.B  #'R',(A4)+  * R
            BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR0010 - Instruction is BHI.  This will further disassemble it            *
*******************************************************************************
```

```
four0010   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'H',(A4)+  * H
           MOVE.B  #'I',(A4)+  * I
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR0011 - Instruction is BLS.  This will further disassemble it            *
*******************************************************************************
four0011   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'L',(A4)+  * L
           MOVE.B  #'S',(A4)+  * S
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR0100 - Instruction is BCC.  This will further disassemble it            *
*******************************************************************************
four0100   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'C',(A4)+  * C
           MOVE.B  #'C',(A4)+  * C
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR0101 - Instruction is BCS.  This will further disassemble it            *
*******************************************************************************
four0101   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'C',(A4)+  * C
           MOVE.B  #'S',(A4)+  * S
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR0110 - Instruction is BNE.  This will further disassemble it            *
*******************************************************************************
four0110   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'N',(A4)+  * N
           MOVE.B  #'E',(A4)+  * E
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR0111 - Instruction is BEQ.  This will further disassemble it            *
*******************************************************************************
four0111   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'E',(A4)+  * E
           MOVE.B  #'Q',(A4)+  * Q
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR1000 - Instruction is BVC.  This will further disassemble it            *
*******************************************************************************
four1000   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'V',(A4)+  * V
           MOVE.B  #'C',(A4)+  * C
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR1001 - Instruction is BVS.  This will further disassemble it            *
*******************************************************************************
four1001   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'V',(A4)+  * V
           MOVE.B  #'S',(A4)+  * S
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR1010 - Instruction is BPL.  This will further disassemble it            *
*******************************************************************************
four1010   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'P',(A4)+  * P
           MOVE.B  #'L',(A4)+  * L
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR1011 - Instruction is BMI.  This will further disassemble it            *
*******************************************************************************
four1011   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'M',(A4)+  * M
           MOVE.B  #'I',(A4)+  * I
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR1100 - Instruction is BGE.  This will further disassemble it            *
*******************************************************************************
four1100   MOVE.B  #'B',(A4)+  * B
```

```
           MOVE.B  #'G',(A4)+  * G
           MOVE.B  #'E',(A4)+  * E
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR1101 - Instruction is BLT.  This will further disassemble it            *
*******************************************************************************
four1101   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'L',(A4)+  * L
           MOVE.B  #'T',(A4)+  * T
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR1110 - Instruction is BGT.  This will further disassemble it            *
*******************************************************************************
four1110   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'G',(A4)+  * G
           MOVE.B  #'T',(A4)+  * T
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* FOUR1111 - Instruction is BLE.  This will further disassemble it            *
*******************************************************************************
four1111   MOVE.B  #'B',(A4)+  * B
           MOVE.B  #'L',(A4)+  * L
           MOVE.B  #'E',(A4)+  * E
           BRA     branch      * Go fill in the displacement
*******************************************************************************
* QUINARY JUMP TABLE - Contains lookup functions for instructions beginning   *
*                      with 1000 based off of the corresponding opcode        *
*******************************************************************************
tablefive  JSR     five000    * Lookup function with corresponding code of 000
           RTS                * Finished disassembling this code
           JSR     five001    * Lookup function with corresponding code of 001
           RTS                * Finished disassembling this code
           JSR     five010    * Lookup function with corresponding code of 010
           RTS                * Finished disassembling this code
           JSR     five011    * Lookup function with corresponding code of 011
           RTS                * Finished disassembling this code
           JSR     five100    * Lookup function with corresponding code of 100
           RTS                * Finished disassembling this code
           JSR     five101    * Lookup function with corresponding code of 101
           RTS                * Finished disassembling this code
           JSR     five110    * Lookup function with corresponding code of 110
           RTS                * Finished disassembling this code
           JSR     five111    * Lookup function with corresponding code of 111
           RTS                * Finished disassembling this code
*******************************************************************************
* FIVE000 - Instruction is OR.B.  This function will further disassemble it   *
*******************************************************************************
five000    MOVE.B  #'O',(A4)+   * O
           MOVE.B  #'R',(A4)+   * R
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
```

```
*******************************************************************************
* FIVE001 - Instruction is OR.W.  This function will further disassemble it   *
*******************************************************************************
five001    MOVE.B  #'O',(A4)+   * O
           MOVE.B  #'R',(A4)+   * R
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'W',(A4)+   * W
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* FIVE010 - Instruction is OR.L.  This function will further disassemble it   *
*******************************************************************************
five010    MOVE.B  #'O',(A4)+   * O
           MOVE.B  #'R',(A4)+   * R
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'L',(A4)+   * L
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* FIVE011 - Further disassembles instructions that began with 1000XXX011      *
*******************************************************************************
five011    MOVEQ  #$0,D4       * Instruction is unknown
           RTS                 * Nothing to return
*******************************************************************************
* FIVE100 - Instruction is OR.B.  This function will further disassemble it   *
*******************************************************************************
five100    MOVE.B  #'O',(A4)+   * O
           MOVE.B  #'R',(A4)+   * R
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
```

```
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* FIVE101 - Instruction is OR.W.  This function will further disassemble it   *
*******************************************************************************
five101   MOVE.B  #'O',(A4)+   * O
          MOVE.B  #'R',(A4)+   * R
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'W',(A4)+   * W
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* FIVE110 - Instruction is OR.L.  This function will further disassemble it   *
*******************************************************************************
five110   MOVE.B  #'O',(A4)+   * O
          MOVE.B  #'R',(A4)+   * R
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'L',(A4)+   * L
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* FIVE111 - Further disassembles instructions that began with 1000XXX111      *
```

```
*******************************************************************************
five111    MOVEQ  #$0,D4       * Instruction is unknown
           RTS                 * Nothing to return
*******************************************************************************
* SENARY JUMP TABLE - Contains lookup functions for instructions beginning    *
*                     with 1100 based off of the corresponding opcode         *
*******************************************************************************
tablesix   JSR     six000     * Lookup function with corresponding code of 000
           RTS                * Finished disassembling this code
           JSR     six001     * Lookup function with corresponding code of 001
           RTS                * Finished disassembling this code
           JSR     six010     * Lookup function with corresponding code of 010
           RTS                * Finished disassembling this code
           JSR     six011     * Lookup function with corresponding code of 011
           RTS                * Finished disassembling this code
           JSR     six100     * Lookup function with corresponding code of 100
           RTS                * Finished disassembling this code
           JSR     six101     * Lookup function with corresponding code of 101
           RTS                * Finished disassembling this code
           JSR     six110     * Lookup function with corresponding code of 110
           RTS                * Finished disassembling this code
           JSR     six111     * Lookup function with corresponding code of 111
           RTS                * Finished disassembling this code
*******************************************************************************
* SIX000 - Instruction is AND.B.  This function will further disassemble it   *
*******************************************************************************
six000     MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'N',(A4)+   * N
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* SIX001 - Instruction is AND.W.  This function will further disassemble it   *
*******************************************************************************
six001     MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'N',(A4)+   * N
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'W',(A4)+   * W
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
```

```
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          RTS                  * Instruction has been decoded
*******************************************************************************
* SIX010 - Instruction is AND.L.  This function will further disassemble it   *
*******************************************************************************
six010    MOVE.B  #'A',(A4)+   * A
          MOVE.B  #'N',(A4)+   * N
          MOVE.B  #'D',(A4)+   * D
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'L',(A4)+   * L
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          MOVE.B  #',',(A4)+   * ,
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          RTS                  * Instruction has been decoded
*******************************************************************************
* SIX011 - Further disassembles instructions that began with 1100XXX011       *
*******************************************************************************
six011    MOVEQ  #$0,D4       * Instruction is unknown
          RTS                 * Nothing to return
*******************************************************************************
* SIX100 - Instruction is AND.B.  This function will further disassemble it   *
*******************************************************************************
six100    MOVE.B  #'A',(A4)+   * A
          MOVE.B  #'N',(A4)+   * N
          MOVE.B  #'D',(A4)+   * D
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'B',(A4)+   * B
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* SIX101 - Instruction is AND.W.  This function will further disassemble it   *
*******************************************************************************
six101    MOVE.B  #'A',(A4)+   * A
          MOVE.B  #'N',(A4)+   * N
          MOVE.B  #'D',(A4)+   * D
```

```
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'W',(A4)+   * W
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* SIX110 - Instruction is AND.L.  This function will further disassemble it   *
*******************************************************************************
six110    MOVE.B  #'A',(A4)+   * A
          MOVE.B  #'N',(A4)+   * N
          MOVE.B  #'D',(A4)+   * D
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'L',(A4)+   * L
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* SIX111 - Further disassembles instructions that began with 1100XXX111       *
*******************************************************************************
six111    MOVEQ  #$0,D4       * Instruction is unknown
          RTS                 * Nothing to return
*******************************************************************************
* SEPTENARY JUMP TABLE - Contains lookup functions for instructions beginning *
*                        with 1011 based off of the corresponding opcode      *
*******************************************************************************
tableseven  JSR     seven000   * Lookup function with corresponding code of 000
            RTS                * Finished disassembling this code
            JSR     seven001   * Lookup function with corresponding code of 001
            RTS                * Finished disassembling this code
            JSR     seven010   * Lookup function with corresponding code of 010
            RTS                * Finished disassembling this code
            JSR     seven011   * Lookup function with corresponding code of 011
            RTS                * Finished disassembling this code
            JSR     seven100   * Lookup function with corresponding code of 100
            RTS                * Finished disassembling this code
            JSR     seven101   * Lookup function with corresponding code of 101
            RTS                * Finished disassembling this code
            JSR     seven110   * Lookup function with corresponding code of 110
            RTS                * Finished disassembling this code
```

```
           JSR     seven111   * Lookup function with corresponding code of 111
           RTS                * Finished disassembling this code
*******************************************************************************
* SEVEN000 - Instruction is CMP.B.  This function will further disassemble it *
*******************************************************************************
SEVEN000   MOVE.B  #'C',(A4)+   * C
           MOVE.B  #'M',(A4)+   * M
           MOVE.B  #'P',(A4)+   * P
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* SEVEN001 - Instruction is CMP.W.  This function will further disassemble it *
*******************************************************************************
SEVEN001   MOVE.B  #'C',(A4)+   * C
           MOVE.B  #'M',(A4)+   * M
           MOVE.B  #'P',(A4)+   * P
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'W',(A4)+   * W
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* SEVEN010 - Instruction is CMP.L.  This function will further disassemble it *
*******************************************************************************
SEVEN010   MOVE.B  #'C',(A4)+   * C
           MOVE.B  #'M',(A4)+   * M
           MOVE.B  #'P',(A4)+   * P
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'L',(A4)+   * L
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
```

```
          BEQ     bad_ins      * Yes, the instruction is malformed
          MOVE.B  #',',(A4)+   * ,
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          RTS                  * Instruction has been decoded
*******************************************************************************
* SEVEN011 - Instruction is CMPA.W.  This will further disassemble it         *
*******************************************************************************
SEVEN011  MOVE.B  #'C',(A4)+   * C
          MOVE.B  #'M',(A4)+   * M
          MOVE.B  #'P',(A4)+   * P
          MOVE.B  #'A',(A4)+   * A
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'W',(A4)+   * W
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          MOVE.B  #',',(A4)+   * ,
          MOVE.B  #'A',(A4)+   * A
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          RTS                  * Instruction has been decoded
*******************************************************************************
* SEVEN100 - Instruction is EOR.B.  This function will further disassemble it *
*******************************************************************************
seven100  MOVE.B  #'E',(A4)+   * E
          MOVE.B  #'O',(A4)+   * O
          MOVE.B  #'R',(A4)+   * R
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'B',(A4)+   * B
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* SEVEN101 - Instruction is EOR.W.  This function will further disassemble it *
*******************************************************************************
seven101  MOVE.B  #'E',(A4)+   * E
          MOVE.B  #'O',(A4)+   * O
          MOVE.B  #'R',(A4)+   * R
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'W',(A4)+   * W
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
```

```
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           MOVE.B  #',',(A4)+   * ,
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           RTS                  * Instruction has been decoded
*******************************************************************************
* SEVEN110 - Instruction is EOR.L.  This function will further disassemble it *
*******************************************************************************
seven110   MOVE.B  #'E',(A4)+   * E
           MOVE.B  #'O',(A4)+   * O
           MOVE.B  #'R',(A4)+   * R
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'L',(A4)+   * L
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           MOVE.B  #',',(A4)+   * ,
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           RTS                  * Instruction has been decoded
*******************************************************************************
* SEVEN111 - Instruction is CMPA.L.  This will further disassemble it         *
*******************************************************************************
SEVEN111   MOVE.B  #'C',(A4)+   * C
           MOVE.B  #'M',(A4)+   * M
           MOVE.B  #'P',(A4)+   * P
           MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'L',(A4)+   * L
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'A',(A4)+   * A
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* OCTONARY JUMP TABLE - Contains lookup functions for instructions beginning  *
*                       with 1101 based off of the corresponding opcode       *
```

```
*************************************************************************
tableeight  JSR     eight000   * Lookup function with corresponding code of 000
            RTS                * Finished disassembling this code
            JSR     eight001   * Lookup function with corresponding code of 001
            RTS                * Finished disassembling this code
            JSR     eight010   * Lookup function with corresponding code of 010
            RTS                * Finished disassembling this code
            JSR     eight011   * Lookup function with corresponding code of 011
            RTS                * Finished disassembling this code
            JSR     eight100   * Lookup function with corresponding code of 100
            RTS                * Finished disassembling this code
            JSR     eight101   * Lookup function with corresponding code of 101
            RTS                * Finished disassembling this code
            JSR     eight110   * Lookup function with corresponding code of 110
            RTS                * Finished disassembling this code
            JSR     eight111   * Lookup function with corresponding code of 111
            RTS                * Finished disassembling this code
*************************************************************************
* EIGHT000 - Instruction is ADD.B.  This function will further disassemble it *
*************************************************************************
eight000    MOVE.B  #'A',(A4)+   * A
            MOVE.B  #'D',(A4)+   * D
            MOVE.B  #'D',(A4)+   * D
            MOVE.B  #'.',(A4)+   * .
            MOVE.B  #'B',(A4)+   * B
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            JSR     effective    *
            CMPI.B  #0,D4        * Was a bad flag returned?
            BEQ     bad_ins      * Yes, the instruction is malformed
            CMPI.B  #%001,D3     * Is the addressing mode illegal?
            BEQ     bad_ins      * Yes, the instruction is malformed
            MOVE.B  #',',(A4)+   * ,
            MOVE.B  #'D',(A4)+   * D
            JSR     gethighreg   * Load the register to append to the buffer
            MULU    #$6,D2       * Form offset for reg_mode jump table
            LEA     reg_mode,A1  * Load reg_mode table prior to jumping
            JSR     00(A1,D2)    * Jump indirect with index
            RTS                  * Instruction has been decoded
*************************************************************************
* EIGHT001 - Instruction is ADD.W.  This function will further disassemble it *
*************************************************************************
eight001    MOVE.B  #'A',(A4)+   * A
            MOVE.B  #'D',(A4)+   * D
            MOVE.B  #'D',(A4)+   * D
            MOVE.B  #'.',(A4)+   * .
            MOVE.B  #'W',(A4)+   * W
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            MOVE.B  #' ',(A4)+   *
            JSR     effective    *
            CMPI.B  #0,D4        * Was a bad flag returned?
            BEQ     bad_ins      * Yes, the instruction is malformed
            MOVE.B  #',',(A4)+   * ,
            MOVE.B  #'D',(A4)+   * D
            JSR     gethighreg   * Load the register to append to the buffer
            MULU    #$6,D2       * Form offset for reg_mode jump table
            LEA     reg_mode,A1  * Load reg_mode table prior to jumping
            JSR     00(A1,D2)    * Jump indirect with index
            RTS                  * Instruction has been decoded
*************************************************************************
* EIGHT010 - Instruction is ADD.L.  This function will further disassemble it *
```

```
*******************************************************************************
eight010   MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'L',(A4)+   * L
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* EIGHT011 - Instruction is ADDA.W.  This will further disassemble it         *
*******************************************************************************
eight011   MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'W',(A4)+   * W
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'A',(A4)+   * A
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* EIGHT100 - Instruction is ADD.B.  This function will further disassemble it *
*******************************************************************************
eight100   MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           MOVE.B  #',',(A4)+   * ,
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
```

```
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           RTS                  * Instruction has been decoded
*******************************************************************************
* EIGHT101 - Instruction is ADD.W.  This function will further disassemble it *
*******************************************************************************
eight101   MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'W',(A4)+   * W
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           MOVE.B  #',',(A4)+   * ,
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           RTS                  * Instruction has been decoded
*******************************************************************************
* EIGHT110 - Instruction is ADD.L.  This function will further disassemble it *
*******************************************************************************
eight110   MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'L',(A4)+   * L
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           MOVE.B  #',',(A4)+   * ,
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           RTS                  * Instruction has been decoded
*******************************************************************************
* EIGHT111 - Instruction is ADDA.L.  This will further disassemble it         *
*******************************************************************************
eight111   MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'D',(A4)+   * D
           MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'L',(A4)+   * L
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
```

```
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'A',(A4)+   * A
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* NONARY JUMP TABLE - Contains lookup functions for instructions beginning    *
*                     with 1001 based off of the corresponding opcode         *
*******************************************************************************
tablenine   JSR     nine000   * Lookup function with corresponding code of 000
            RTS               * Finished disassembling this code
            JSR     nine001   * Lookup function with corresponding code of 001
            RTS               * Finished disassembling this code
            JSR     nine010   * Lookup function with corresponding code of 010
            RTS               * Finished disassembling this code
            JSR     nine011   * Lookup function with corresponding code of 011
            RTS               * Finished disassembling this code
            JSR     nine100   * Lookup function with corresponding code of 100
            RTS               * Finished disassembling this code
            JSR     nine101   * Lookup function with corresponding code of 101
            RTS               * Finished disassembling this code
            JSR     nine110   * Lookup function with corresponding code of 110
            RTS               * Finished disassembling this code
            JSR     nine111   * Lookup function with corresponding code of 111
            RTS               * Finished disassembling this code
*******************************************************************************
* NINE000 - Instruction is SUB.B.  This function will further disassemble it  *
*******************************************************************************
nine000     MOVE.B  #'S',(A4)+   * S
           MOVE.B  #'U',(A4)+   * U
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           CMPI.B  #%001,D3     * Is the addressing mode illegal?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'D',(A4)+   * D
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* NINE001 - Instruction is SUB.W.  This function will further disassemble it  *
*******************************************************************************
nine001     MOVE.B  #'S',(A4)+   * S
           MOVE.B  #'U',(A4)+   * U
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'W',(A4)+   * W
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
```

```
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          MOVE.B  #',',(A4)+   * ,
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          RTS                  * Instruction has been decoded
*******************************************************************************
* NINE010 - Instruction is SUB.L.  This function will further disassemble it  *
*******************************************************************************
nine010   MOVE.B  #'S',(A4)+   * S
          MOVE.B  #'U',(A4)+   * U
          MOVE.B  #'B',(A4)+   * B
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'L',(A4)+   * L
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          MOVE.B  #',',(A4)+   * ,
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          RTS                  * Instruction has been decoded
*******************************************************************************
* NINE011 - Instruction is SUBA.W.  This will further disassemble it          *
*******************************************************************************
nine011   MOVE.B  #'S',(A4)+   * S
          MOVE.B  #'U',(A4)+   * U
          MOVE.B  #'B',(A4)+   * B
          MOVE.B  #'A',(A4)+   * A
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'W',(A4)+   * W
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          MOVE.B  #',',(A4)+   * ,
          MOVE.B  #'A',(A4)+   * A
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          RTS                  * Instruction has been decoded
*******************************************************************************
* NINE100 - Instruction is SUB.B.  This function will further disassemble it  *
*******************************************************************************
nine100   MOVE.B  #'S',(A4)+   * S
          MOVE.B  #'U',(A4)+   * U
          MOVE.B  #'B',(A4)+   * B
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'B',(A4)+   * B
```

```
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* NINE101 - Instruction is SUB.W.  This function will further disassemble it  *
*******************************************************************************
nine101   MOVE.B  #'S',(A4)+   * S
          MOVE.B  #'U',(A4)+   * U
          MOVE.B  #'B',(A4)+   * B
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'W',(A4)+   * W
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
          BEQ     bad_ins      * Yes, the instruction is malformed
          RTS                  * Instruction has been decoded
*******************************************************************************
* NINE110 - Instruction is SUB.L.  This function will further disassemble it  *
*******************************************************************************
nine110   MOVE.B  #'S',(A4)+   * S
          MOVE.B  #'U',(A4)+   * U
          MOVE.B  #'B',(A4)+   * B
          MOVE.B  #'.',(A4)+   * .
          MOVE.B  #'L',(A4)+   * L
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #' ',(A4)+   *
          MOVE.B  #'D',(A4)+   * D
          JSR     gethighreg   * Load the register to append to the buffer
          MULU    #$6,D2       * Form offset for reg_mode jump table
          LEA     reg_mode,A1  * Load reg_mode table prior to jumping
          JSR     00(A1,D2)    * Jump indirect with index
          MOVE.B  #',',(A4)+   * ,
          JSR     effective    *
          CMPI.B  #0,D4        * Was a bad flag returned?
          BEQ     bad_ins      * Yes, the instruction is malformed
          CMPI.B  #%001,D3     * Is the addressing mode illegal?
```

```
           BEQ     bad_ins      * Yes, the instruction is malformed
           RTS                  * Instruction has been decoded
*******************************************************************************
* NINE111 - Instruction is SUBA.L.  This will further disassemble it          *
*******************************************************************************
nine111    MOVE.B  #'S',(A4)+   * S
           MOVE.B  #'U',(A4)+   * U
           MOVE.B  #'B',(A4)+   * B
           MOVE.B  #'A',(A4)+   * A
           MOVE.B  #'.',(A4)+   * .
           MOVE.B  #'L',(A4)+   * L
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           MOVE.B  #' ',(A4)+   *
           JSR     effective    *
           CMPI.B  #0,D4        * Was a bad flag returned?
           BEQ     bad_ins      * Yes, the instruction is malformed
           MOVE.B  #',',(A4)+   * ,
           MOVE.B  #'A',(A4)+   * A
           JSR     gethighreg   * Load the register to append to the buffer
           MULU    #$6,D2       * Form offset for reg_mode jump table
           LEA     reg_mode,A1  * Load reg_mode table prior to jumping
           JSR     00(A1,D2)    * Jump indirect with index
           RTS                  * Instruction has been decoded
*******************************************************************************
* BRANCH - Fills out branch address of branch instructions                    *
*******************************************************************************
branch     MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #' ',(A4)+  *
           MOVE.B  #'$',(A4)+  * $
           CMP.B   #$0,D7       * Does the branch specify a 16 bit offset?
           BEQ     branchw      * Yes it does
           CMP.B   #$FF,D7      * Does the branch specify a 32 bit offset?
           BEQ     branchl      * Yes it does
           BRA     eightbit     * Branch must be default 8 bit offset
branchw    CMPI.L  #$0,D6       * Is there a 16-bit word remaining?
           BLE     bad_ins      * Nope
           JSR     getword      * Get the 16 bit offset word
           ADD.L   #$2,A5       * Increment next instruction address
           SUBI.L  #$1,D6       * Decrement remaining instructino count
           RTS                  * We've decoded the branch instruction
branchl    CMPI.L  #1,D6        * Is there a 32-bit long remaining?
           BLE     bad_ins      * Nope
           JSR     getlong      * Get the 32 bit offset long
           ADD.L   #$4,A5       * Increment next instruction address
           SUBI.L  #$2,D6       * Decrement remaining instruction count
           RTS                  * We've decoded the branch instruction
*******************************************************************************
* EIGHTBIT - Gets the eight bit offset from the end of the instruction in D7, *
*            encodes the offset in ASCII, and writes the resulting address to *
*            the good buffer in A4                                            *
*******************************************************************************
eightbit   MOVE.W  D7,D5        * Load data prior to masking and shifting
           ANDI.W  #$00FF,D5    * Mask out first two characters of the word
           LSR.W   #4,D5        * Get the third character of the word
           JSR     codechar     * Go encode and write the third character
           MOVE.W  D7,D5        * Reload data prior to masking and shifting
           ANDI.W  #$000F,D5    * Mask out first three characters of the word
           JSR     codechar     * Go encode and write the fourth character
           RTS                  * We've decoded the eight bit offset
*******************************************************************************
```

```
* LEA - Instruction is LEA, this will further disassemble it                  *
*******************************************************************************
lea         MOVE.B  #'L',(A4)+  * L
            MOVE.B  #'E',(A4)+  * E
            MOVE.B  #'A',(A4)+  * A
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            JSR     effective   * Get the effective address
            CMP.B   #%000,D3    * Is this an illegal addressing mode?
            BEQ     lea_bad     * Yes
            CMP.B   #%001,D3    * Is this an illegal addressing mode?
            BEQ     lea_bad     * Yes
            CMP.B   #%011,D3    * Is this an illegal addressing mode?
            BEQ     lea_bad     * Yes
            CMP.B   #%100,D3    * Is this an illegal addressing mode?
            BEQ     lea_bad     * Yes
            MOVE.B  #',',(A4)+  * ,
            MOVE.B  #'A',(A4)+  * A
            JSR     gethighreg  * Get the register
            MULU    #$6,D2      * Form offset for reg_mode jump table
            LEA     reg_mode,A1 * Load reg_mode table prior to jumping
            JSR     00(A1,D2)   * Jump indirect with index
            RTS                 * We're done decoding the LEA instruction
lea_bad     MOVEQ   #$0,D4      * Set instruction bad flag
            RTS                 * Return bad instruction
*******************************************************************************
* IMMEDIATE - All immediate instructions follow this process when decoding    *
*******************************************************************************
immediate   CMPI.L  #$0,D6      * Is there an address left for the immediate
                                * data in this instruction?
            BEQ     bad_im      * Nope, we won't be able to decode it entirely!
            JSR     getsize     * Figure out what size the ORI instruction is
            CMP.W   #%00,D2     * Byte?
            BEQ     byte_im     * Yes
            CMP.W   #%01,D2     * Word?
            BEQ     word_im     * Yes
            CMP.W   #%10,D2     * Long?
            BEQ     long_im     * Yes
bad_im      MOVEQ   #$0,D4      * Malformed instruction!
            RTS                 * Instruction is bad, so stop disassembling
con_im      JSR     getword     * Get the word at A5 and place it in good buf
            ADD.L   #$2,A5      * Increment A5 by one word
            SUBI.L  #$1,D6      * Decrement remaining word count
            MOVE.B  #',',(A4)+  * ,
            JSR     effective   * Get the effective address
            CMPI.L  #0,D4       * Did effective addressing detect a bad
                                * instruction?
            BEQ     dec_im      * Yes, the instruction is bad
            CMPI.B  #%001,D3    * Is the addressing mode invalid?
            BEQ     dec_im      * Yes, the addressing mode is invalid
            RTS                 * Return disassembled instruction
dec_im      ADD.L   #$1,D6      * Because instruction is bad, we re-increment
                                * remaining word count to disassociate the
                                * proceeding word which we interpreted as an
                                * immediate address
            SUB.L   #$2,A5      * For the same reasons we will treat the
                                * proceeding word as its own instruction and
                                * not part of this bad insturction (data)
            BRA     bad_im      * Instruction is bad
byte_im     MOVE.B  #'B',(A4)+  * B
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
            MOVE.B  #' ',(A4)+  *
```

```
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #'#',(A4)+  * #
           MOVE.B   #'$',(A4)+  * $
           BRA      con_im      * Continue appending immediate information
word_im    MOVE.B   #'W',(A4)+  * W
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #'#',(A4)+  * #
           MOVE.B   #'$',(A4)+  * $
           BRA      con_im      * Continue appending immedaite information
long_im    CMPI.L   #1,D6       * Is there an address left for the immediate
                                * data in this instruction?
           BEQ      bad_im      * Nope, we won't be able to decode it entirely!
           MOVE.B   #'L',(A4)+  * L
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #' ',(A4)+  *
           MOVE.B   #'#',(A4)+  * #
           MOVE.B   #'$',(A4)+  * $
           JSR      getlong     * Get the long at A5 and place it in good buf
           ADD.L    #$4,A5      * Increment A5 by two words
           SUBI.L   #$2,D6      * Decrement remaining word count by two
           MOVE.B   #',',(A4)+  * ,
           JSR      effective   * Get the effective address
           CMPI.L   #0,D4       * Did effective addressing detect a bad
                                * instruction?
           BEQ      dec_iml     * Yes, the instruction is bad
           CMPI.B   #%001,D3    * Is the addressing mode invalid?
           BEQ      dec_im      * Yes, the addressing mode is invalid
           RTS                  * Return from instruction disassembly
dec_iml    ADD.L    #$2,D6      * Because instruction is bad, we re-increment
                                * remaining word count to disassociate the
                                * proceeding long which we interpreted as an
                                * immediate address
           SUB.L    #$4,A5      * For the same reasons we will treat the
                                * proceeding long as its own instruction and
                                * not part of this bad insturction (data)
           BRA      bad_im      * Instruction is bad
*******************************************************************************
* GETSIZE - Determines the instruction size stored in bits 6 and 7 of the     *
*           instruction at D7, then places the results in D2                  *
*******************************************************************************
getsize    MOVE.W   D7,D2       * Copy current instruction before shifting
           LSR.W    #6,D2       * Move the size bits to the 1st two positions
           ANDI.W   #$0003,D2   * Eliminate the remaining bits
           RTS                  * Size will be returned in D2
*******************************************************************************
* GETBYTE - Gets the word at A5 and stores it as data in the good buffer A4+  *
*         - Does not advance A5 or decrement remaining word count D6          *
*******************************************************************************
getbyte    MOVEQ    #$0,D5      * Clear D5 prior to use
           MOVE.W   (A5),D5     * Load data from the next address
           LSR.B    #4,D5       * Get the first character
           JSR      codechar    * Go encode and write the first character
           MOVE.B   (A5),D5     * Reload data from the next address
           ANDI.B   #$F0,D5     * Mask out the first character
           JSR      codechar    * Go encode and write the second character
           RTS                  * We've decoded the word
*******************************************************************************
* GETWORD - Gets the word at A5 and stores it as data in the good buffer A4+  *
*         - Does not advance A5 or decrement remaining word count D6          *
*******************************************************************************
getword    MOVEQ    #$0,D5      * Clear D5 prior to use
```

```
           MOVE.W   (A5),D5    * Load data from the next address
           LSR.W    #8,D5      * Get the first character
           LSR.W    #4,D5      * Get the first character
           JSR      codechar   * Go encode and write the first character
           MOVE.W   (A5),D5    * Reload data from the next address
           ANDI.W   #$0FFF,D5  * Mask out the first character
           LSR.W    #8,D5      * Get the second character
           JSR      codechar   * Go encode and write the second character
           MOVE.W   (A5),D5    * Reload data from the next address
           ANDI.W   #$00FF,D5  * Mask out the first two character
           LSR.W    #4,D5      * Get the third character
           JSR      codechar   * Go encode and write the third character
           MOVE.W   (A5),D5    * Reload data from the next address
           ANDI.W   #$000F,D5  * Mask out the first three characters
           JSR      codechar   * Go encode and write the fourth character
           RTS                 * We've decoded the word
*******************************************************************************
* GETLONG - Gets the long at A5 and stores it as data in the good buffer A4+  *
*         - Does not advance A5 or decrement remaining word count D6          *
*******************************************************************************
getlong    MOVEQ    #$0,D5         * Clear D5 prior to use
           MOVE.L   (A5),D5        * Load data from the next address
           LSR.L    #8,D5          * Get the first character
           LSR.L    #8,D5          * Get the first character
           LSR.L    #8,D5          * Get the first character
           LSR.L    #4,D5          * Get the first character
           JSR      codechar       * Go encode and write the first character
           MOVE.L   (A5),D5        * Reload data from the next address
           ANDI.L   #$0FFFFFFF,D5  * Mask out the first character
           LSR.L    #8,D5          * Get the second character
           LSR.L    #8,D5          * Get the second character
           LSR.L    #8,D5          * Get the second character
           JSR      codechar       * Go encode and write the third chracter
           MOVE.L   (A5),D5        * Reload data from the next address
           ANDI.L   #$00FFFFFF,D5  * Mask out the first two characters
           LSR.L    #8,D5          * Get the third character
           LSR.L    #8,D5          * Get the third character
           LSR.L    #4,D5          * Get the third character
           JSR      codechar       * Go encode and write the third chracter
           MOVE.L   (A5),D5        * Reload data from the next address
           ANDI.L   #$000FFFFF,D5  * Mask out the first three characters
           LSR.L    #8,D5          * Get the fourth character
           LSR.L    #8,D5          * Get the fourth character
           JSR      codechar       * Go encode and write the fourth chracter
           MOVE.L   (A5),D5        * Reload data from the next address
           ANDI.L   #$0000FFFF,D5  * Mask out the first four characters
           LSR.L    #8,D5          * Get the fifth character
           LSR.L    #4,D5          * Get the fifth character
           JSR      codechar       * Go encode and write the fifth chracter
           MOVE.L   (A5),D5        * Reload data from the next address
           ANDI.L   #$00000FFF,D5  * Mask out the first five characters
           LSR.L    #8,D5          * Get the sixth character
           JSR      codechar       * Go encode and write the sixth chracter
           MOVE.L   (A5),D5        * Reload data from the next address
           ANDI.L   #$000000FF,D5  * Mask out the first six characters
           LSR.L    #4,D5          * Get the seventh character
           JSR      codechar       * Go encode and write the seventh chracter
           MOVE.L   (A5),D5        * Reload data from the next address
           ANDI.L   #$0000000F,D5  * Mask out the first seven characters
           JSR      codechar       * Go encode and write the eigth chracter
           RTS                     * We've decoded the long
*******************************************************************************
* CODECHAR - Determines if the current character in D5 is numeric or          *
*            alphabetic, and writes the corresponding character code to A4+   *
*******************************************************************************
codechar   CMP.B    #$A,D5     * Is the character hexadecimal?
           BGE      alpha      * Yes
           ADDI.B   #48,D5     * Add code to make this a decimal ascii char
           BRA      writechar  * Time to write this character to the buffer
alpha      ADDI.B   #55,D5     * Add code to make this an alphabetic ascii char
writechar  MOVE.B   D5,(A4)+   * Write the character to the good buffer
```

```
          RTS                 * We've encoded and written the character
*******************************************************************************
* GETREGMODE - Determines the register mode stored in bits 6, 7, & 8 of the   *
*              instruction at D7, then places the results in D2               *
*******************************************************************************
getregmode  MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
            LSR.W   #6,D2      * Shift the instruction into first 3 bits
            ANDI.W  #$0007,D2  * Eliminate the remaining bits
            RTS                * Register mode will be returned in D2
*******************************************************************************
* GETHIGHREG - Determines the register stored in bits 9, 10, & 11 of the      *
*              instruction at D7, then places the results in D2               *
*******************************************************************************
gethighreg  MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
            LSR.W   #8,D2      * Shift the instruction into first 3 bits
            LSR.W   #1,D2      * Shift the instruction into first 3 bits
            ANDI.W  #$0007,D2  * Eliminate the remaining bits
            RTS                * Register code will be returned in D2
*******************************************************************************
* GETMODE - Determines the mode stored in bits 3, 4, & 5 of the               *
*           instruction at D7, then places the results in D2                  *
*******************************************************************************
getmode     MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
            LSR.W   #3,D2      * Shift the instruction into first 3 bits
            ANDI.W  #$0007,D2  * Eliminate the remaining bits
            RTS                * Register code will be returned in D2
*******************************************************************************
* GETREG - Determines the register stored in bits 0, 1, & 2 of the            *
*          instruction at D7, then places the results in D2                   *
*******************************************************************************
getreg      MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
            ANDI.W  #$0007,D2  * Eliminate the remaining bits
            RTS                * Register code will be returned in D2
*******************************************************************************
* GET3TO7 - Determines the code stored in bits 3, 4, 5, 6, & 7 of the         *
*           instruction at D7, then places the results in D2                  *
*******************************************************************************
get3to7     MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
            LSR.W   #3,D2      * Shift the instruction into first 3 bits
            ANDI.W  #$001F,D2  * Eliminate the remaining bits
            RTS                * Register code will be returned in D2
*******************************************************************************
* GET9TO10 - Determines the code stored in bits 9 and 10 of the               *
*            instruction at D7, then places the results in D2                 *
*******************************************************************************
get9to10    MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
            LSR.W   #8,D2      * Shift the code into first 2 bits
            LSR.W   #1,D2      * Shift the code into first 2 bits
            ANDI.W  #$0003,D2  * Eliminate the remaining bits
            RTS                * Register code will be returned in D2
*******************************************************************************
* GETBIT8 - Determines the value of the 8th bit in the code of the            *
*           instruction at D7, then places the result in D2                   *
*******************************************************************************
getbit8     MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
            LSR.W   #7,D2      * Shift the 8th bit into the first position
            ANDI.W  #$0001,D2  * Eliminate the remaining bits
            RTS                * Register code will be returned in D2
*******************************************************************************
* GET3TO4 - Determines the value of the 3rd and 4th bits in the code of the   *
*           instruction at D7, then places the results in D2                  *
*******************************************************************************
get3to4     MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
            LSR.W   #3,D2      * Shift the 8th bit into the first position
            ANDI.W  #$0003,D2  * Eliminate the remaining bits
            RTS                * Register code will be returned in D2
*******************************************************************************
* GETBIT5 - Determines the value of the 5th bit in the code of the            *
*           instruction at D7, then places the result in D2                   *
*******************************************************************************
getbit5     MOVE.W  D7,D2      * Copy the current instruction from D7 to D2
```

```
            LSR.W   #5,D2       * Shift the 8th bit into the first position
            ANDI.W  #$0001,D2   * Eliminate the remaining bits
            RTS                 * Register code will be returned in D2
*******************************************************************************
* GET3TO8 - Determines the value of the 3rd to 8th bits from the instruction  *
*           at D7, then places the results in D2                              *
*******************************************************************************
get3to8     MOVE.W  D7,D2       * Copy the current instruction from D7 to D2
            LSR.W   #3,D2       * Shift the 8th bit into the first position
            ANDI.W  #$003F,D2   * Eliminate the remaining bits
            RTS                 * Register code will be returned in D2
*******************************************************************************
* EFFECTIVE - Determines the effective address type, adds the effective       *
*             address to the good buffer in A4, advances the good buffer, and *
*             stores the mode in D3                                           *
*           - Always checks D3 beforehand to see if it's a move code (1),     *
*             otherwise assumes it's a standard effective address in bits 0-5 *
*           - Changes flag in D4 to bad (0) if address mode is illegal        *
*           - Changes flag in D4 to bad (0) if there are no remaining words   *
*             to decode as addresses (end of user specified addresses)        *
*           - Decrements remaining word count in D6 and updates current       *
*             instruction pointer in A5 if additional words are required for  *
*             decoding the current address(es)                                *
*           - Returns remaining word count in D6                              *
*           - Returns good buffer pointer in A4                               *
*           - Maintains current instruction data in D7                        *
*******************************************************************************
effective   MOVE.B  D7,D5       * Move instruction to D5
            LSL.B   #2,D5       * Eliminate most significant two bits
            LSR.B   #5,D5       * To isolate bits 3 thru 5
            MOVE.B  D7,D2       * Move instruction to D2
            LSL.B   #5,D2       * Eliminate most significant five bits
            LSR.B   #5,D2       * Isolage bits 0 thru 2
            CMP.B   #$1,D3      * If D3 is equal to 1...
            BEQ     moveea      * This is a MOVE instruction
            CMP.B   #$0,D3      * If D3 is equal to 0...
            BEQ     anyea       * This is any other kind of instruction
            RTS
*******************************************************************************
* anyea - Puts the effective address into the buffer                          *
*******************************************************************************
anyea       MULU    #$6,D5      * Form offset for jump table
            LEA     ea_mode,A0  * Load ea_mode table prior to jumping
            JSR     00(A0,D5)   * Jump indirect with index
            JSR     getregmode  * For testing (not part of implementation)
            MOVE.B  D5,D3       * For testing (not part of implementation)
            RTS
*******************************************************************************
* moveea - Puts the source and destination into the buffer                    *
*******************************************************************************
moveea      MULU    #$6,D5      * Form offset for jump table
            LEA     ea_mode,A0  * Load ea_mode table prior to jumping
            JSR     00(A0,D5)   * Jump indirect with index
            MOVE.B  #',',(A4)+  * Prints , to good buffer/increments
            MOVE.W  D7,D2       * Move whole instruction to D2
            LSL.W   #4,D2       * Eliminate most significant four bits
            LSR.W   #8,D2       * Shift by 13 total (here 8)
            LSR.W   #5,D2       * Shift by 13 total (here 5)
            MOVE.W  D7,D5       * Move whole instruction to D5
            LSL.W   #7,D5       * Eliminate most significant seven bits
            LSR.W   #8,D5       * Shift by 13 total (here 8)
            LSR.W   #5,D5       * Shift by 5 total (here 5)
            MULU    #$6,D5      * Form offset for jump table
            LEA     ea_mode,A0  * Load ea_mode table prior to jumping
            JSR     00(A0,D5)   * Jump indirect with index
            JSR     getregmode  * For testing (not part of implementation)
            MOVE.B  D5,D3       * For testing (not part of implementation)
            RTS
*******************************************************************************
* EA_MODE Jump Table - Lookup table for the effective address mode            *
*******************************************************************************
```

```
ea_mode  JMP    eamode000       * Register is Dn
         JMP    eamode001       * Register is An
         JMP    eamode010       * Register is (An)
         JMP    eamode011       * Register is (An)+
         JMP    eamode100       * Register is -(An)
         JMP    eamode101       * Register is (d16,An)
         JMP    eamode110       * Register is (d8,An,Xn)
         JMP    eamode111       * Register is (XXX).W or (XXX).L
*******************************************************************************
* REG_MODE Jump Table - Lookup table for the effective address mode           *
*******************************************************************************
reg_mode JMP    regmode000      * Register is 0
         JMP    regmode001      * Register is 1
         JMP    regmode010      * Register is 2
         JMP    regmode011      * Register is 3
         JMP    regmode100      * Register is 4
         JMP    regmode101      * Register is 5
         JMP    regmode110      * Register is 6
         JMP    regmode111      * Register is 7
*******************************************************************************
* AB_MODE Jump Table - Lookup table for absolute addressing modes             *
*******************************************************************************
ab_mode  JMP    abmode000       * Register mode is 000
         JMP    abmode001       * Register mode is 001
         JMP    abmode010       * Register mode is 010
         JMP    abmode011       * Register mode is 011
         JMP    abmode100       * Register mode is 100
         JMP    abmode101       * Register mode is 101
         JMP    abmode110       * Register mode is 110
         JMP    abmode111       * Register mode is 111
*******************************************************************************
* EAMODE000 - Instructions for putting eamode000 into the buffer              *
*******************************************************************************
eamode000       MOVE.B  #'D',(A4)+      * Prints D for data register
                MULU    #$6,D2          * Form offset for reg_mode jump table
                LEA     reg_mode,A1     * Load reg_mode table prior to jumping
                JSR     00(A1,D2)       * Jump indirect with index
                RTS
*******************************************************************************
* EAMODE001 - Instructions for putting eamode001 into the buffer              *
*******************************************************************************
eamode001       MOVE.B  #'A',(A4)+      * Prints A for data register
                MULU    #$6,D2          * Form offset for reg_mode jump table
                LEA     reg_mode,A1     * Load reg_mode table prior to jumping
                JSR     00(A1,D2)       * Jump indirect with index
                RTS
*******************************************************************************
* EAMODE010 - Instructions for putting eamode010 into the buffer              *
*******************************************************************************
eamode010       MOVE.B  #'(',(A4)+      * Prints ( to good buffer/increments
                MOVE.B  #'A',(A4)+      * Prints A to good buffer/increments
                MULU    #$6,D2          * Form offset for reg_mode jump table
                LEA     reg_mode,A1     * Load reg_mode table prior to jumping
                JSR     00(A1,D2)       * Jump indirect with index
                MOVE.B  #')',(A4)+      * Prints ) to good buffer/increments
                RTS
*******************************************************************************
* EAMODE011 - Instructions for putting eamode011 into the buffer              *
*******************************************************************************
eamode011       MOVE.B  #'(',(A4)+      * Prints ( to good buffer/increments
                MOVE.B  #'A',(A4)+      * Prints A to good buffer/increments
                MULU    #$6,D2          * Form offset for reg_mode jump table
                LEA     reg_mode,A1     * Load reg_mode table prior to jumping
                JSR     00(A1,D2)       * Jump indirect with index
                MOVE.B  #')',(A4)+      * Prints ) to good buffer/increments
                MOVE.B  #'+',(A4)+      * Prints + to good buffer/increments
                RTS
*******************************************************************************
* EAMODE100 - Instructions for putting eamode100 into the buffer              *
*******************************************************************************
eamode100       MOVE.B  #'-',(A4)+      * Prints - to good buffer/increments
```

```
                MOVE.B  #'(',(A4)+      * Prints ( to good buffer/increments
                MOVE.B  #'A',(A4)+      * Prints A to good buffer/increments
                MULU    #$6,D2          * Form offset for reg_mode jump table
                LEA     reg_mode,A1     * Load reg_mode table prior to jumping
                JSR     00(A1,D2)       * Jump indirect with index
                MOVE.B  #')',(A4)+      * Prints ) to good buffer/increments
                RTS
*******************************************************************************
* EAMODE101 - Instructions for putting eamode101 into the buffer              *
*             Need another word, 16 binary bits maximum                       *
*******************************************************************************
eamode101       MOVE.B  #'(',(A4)+      * Prints ( to good buffer/increments
                MOVE.W  (A5),D0         * Store next word in D0
                CMPI.L  #$0,D6          * If D6 is equal to 0...
                BEQ     bad_ins         * Then the instruction is bad
                SUBI.L  #$1,D6          * Decrement remaining word count
                MOVE.B  #'$',(A4)+      * Prints $ to good buffer/increments
                LSR.W   #8,D0           * Shift by 12 total (here 8)
                LSR.W   #4,D0           * Shift by 12 total (here 4)
                JSR     prt_digs        * Print the first digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #4,D0           * Shift left 4 digits
                LSR.W   #8,D0           * Shift by 12 total (here 8)
                LSR.W   #4,D0           * Shift by 12 total (here 4)
                JSR     prt_digs        * Print the second digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #8,D0           * Shift left 8 digits
                LSR.W   #8,D0           * Shift by 12 total (here 8)
                LSR.W   #4,D0           * Shift by 12 total (here 4)
                JSR     prt_digs        * Print the third digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #8,D0           * Shift left by 12 total (here 8)
                LSL.W   #4,D0           * Shift left by 12 total (here 4)
                LSR.W   #8,D0           * Shift by 12 total (here 8)
                LSR.W   #4,D0           * Shift by 12 total (here 4)
                JSR     prt_digs        * Print the fourth digit
                MOVE.W  (A5),D0         * Reload word in D0
                CMP.W   #$7FFF,D0       * Is the value above 16 binary digits
                BGT     bad_inc         * Then the instruction is bad,
                                        * reincrement word count
                MOVE.B  #',',(A4)+      * Prints , to good buffer/increments
                MOVE.B  #'A',(A4)+      * Prints A to good buffer/increments
                MULU    #$6,D2          * Form offset for reg_mode jump table
                LEA     reg_mode,A1     * Load reg_mode table prior to jumping
                JSR     00(A1,D2)       * Jump indirect with index
                MOVE.B  #')',(A4)+      * Prints ) to good buffer/increments
                ADDA.L  #2,A5           * Increment address to the next word
                RTS
*******************************************************************************
* EAMODE110 - Instructions for putting eamode110 into the buffer              *
*             Need another word                                               *
*******************************************************************************
eamode110       MOVE.B  #'(',(A4)+      * Prints ( to good buffer/increments
                MOVE.W  (A5),D0         * Store next word in D5
                CMPI.L  #$0,D6          * If D6 is equal to 0...
                BEQ     bad_ins         * Then the instruction is bad
                SUBI.L  #$1,D6          * Decrement remaining word count
                MOVE.B  #'$',(A4)+      * Prints $ to good buffer/increments
                LSL.W   #8,D0           * Shift left 8
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the first digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #8,D0           * Shift left total 12 (8 here)
                LSL.W   #4,D0           * Shift left total 12 (4 here)
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the second digit
                MOVE.B  #',',(A4)+      * Prints , to good buffer/increments
                MOVE.B  #'A',(A4)+      * Prints , to good buffer/increments
                MULU    #$6,D2          * Form offset for reg_mode jump table
```

```
                LEA     reg_mode,A1     * Load reg_mode table prior to jumping
                JSR     00(A1,D2)       * Jump indirect with index
                MOVE.B  #',',(A4)+      * Prints , to good buffer/increments
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #8,D0           * Shift left total 15 (8 here)
                LSL.W   #7,D0           * Shift left total 15 (7 here)
                JSR     prt_ad          * Print A or D for register
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #1,D0           * Shift right once
                LSR.W   #8,D0           * Shift left total 13 (8 here)
                LSR.W   #5,D0           * Shift left total 13 (5 here)
                MULU    #$6,D0          * Form offset for reg_mode jump table
                LEA     reg_mode,A1     * Load reg_mode table prior to jumping
                JSR     00(A1,D0)       * Jump indirect with index
                MOVE.B  #')',(A4)+      * Prints ) to good buffer/increments
                ADDA.L  #2,A5           * Increment address to the next word
                RTS
*******************************************************************************
* EAMODE111 - Instructions for putting eamode111 into the buffer              *
*******************************************************************************
eamode111       MULU    #$6,D2          * Form offset for reg_mode jump table
                LEA     ab_mode,A1      * Load reg_mode table prior to jumping
                JSR     00(A1,D2)       * Jump indirect with index
                RTS
*******************************************************************************
* ABMODE000 - Loads absolute data addressing .W (XXX).W into the buffer       *
*******************************************************************************
abmode000       MOVE.B  #'$',(A4)+      * $
                MOVE.W  (A5),D0         * Store next word in D5
                CMPI.L  #$0,D6          * If D6 is equal to 0...
                BEQ     bad_ins         * Then the instruction is bad
                SUBI.L  #$1,D6          * Decrement remaining word count
                LSR.W   #8,D0           * Shift right total 12 (8 here)
                LSR.W   #4,D0           * Shift right total 12 (4 here)
                JSR     prt_digs        * Print the first digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #4,D0           * Shift left 4
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the second digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #8,D0           * Shift left 8
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the third digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #8,D0           * Shift left total 12 (8 here)
                LSL.W   #4,D0           * Shift left total 12 (4 here)
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the fourth digit
                ADD.L   #$2,A5          * Increment next address location
                RTS
*******************************************************************************
* ABMODE001 - Loads absolute data addressing .L (XXX).L into the buffer       *
*             Gets next two words (long) to do so                             *
*******************************************************************************
abmode001       MOVE.B  #'$',(A4)+      * $
                MOVE.W  (A5),D0         * Store next word in D5
                CMPI.L  #$1,D6          * If D6 is equal to 0...
                BLE     bad_ins         * Then the instruction is bad
                SUBI.L  #$2,D6          * Decrement remaining word count
                LSR.W   #8,D0           * Shift right total 12 (8 here)
                LSR.W   #4,D0           * Shift right total 12 (4 here)
                JSR     prt_digs        * Print the first digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #4,D0           * Shift left 4
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the second digit
                MOVE.W  (A5),D0         * Reload word in D0
```

```
                LSL.W   #8,D0           * Shift left 8
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the third digit
                MOVE.W  (A5)+,D0        * Reload word in D0, and increment
                LSL.W   #8,D0           * Shift left total 12 (8 here)
                LSL.W   #4,D0           * Shift left total 12 (4 here)
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the fourth digit
                MOVE.W  (A5),D0         * Store next word in D5
                LSR.W   #8,D0           * Shift right total 12 (8 here)
                LSR.W   #4,D0           * Shift right total 12 (4 here)
                JSR     prt_digs        * Print the first digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #4,D0           * Shift left 4
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the second digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #8,D0           * Shift left 8
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the third digit
                MOVE.W  (A5),D0         * Reload word in D0
                LSL.W   #8,D0           * Shift left total 12 (8 here)
                LSL.W   #4,D0           * Shift left total 12 (4 here)
                LSR.W   #8,D0           * Shift back total 12 (8 here)
                LSR.W   #4,D0           * Shift back total 12 (4 here)
                JSR     prt_digs        * Print the fourth digit
                ADD.L   #$2,A5          * Increment next address location
                RTS
*******************************************************************************
* ABMODE010 - Decodes instructions with register mode of 010                  *
*******************************************************************************
abmode010       MOVEQ.L  #$0,D4         * Register mode is unrecognized
                RTS                     * Nothing to return!
*******************************************************************************
* ABMODE011 - Decodes instructions with register mode of 011                  *
*******************************************************************************
abmode011       MOVEQ.L  #$0,D4         * Register mode is unrecognized
                RTS                     * Nothing to return!
*******************************************************************************
* ABMODE100 - Decodes instructions with register mode of 100                  *
*******************************************************************************
abmode100       MOVE.B  #'#',(A4)+      * #
                CMPI.L  #$0,D6          * Is there space left for the data?
                BEQ     bad_ins         * Nope!
                JSR     getword         * Get the word for the address
                SUBI.L  #$1,D6          * Decrement remaining instruction count
                ADD.L   #$2,A5          * Increment next instruction address
                RTS                     * Immediate address decoded!
*******************************************************************************
* ABMODE101 - Decodes instructions with register mode of 101                  *
*******************************************************************************
abmode101       MOVEQ.L  #$0,D4         * Register mode is unrecognized
                RTS                     * Nothing to return!
*******************************************************************************
* ABMODE110 - Decodes instructions with register mode of 110                  *
*******************************************************************************
abmode110       MOVEQ.L  #$0,D4         * Register mode is unrecognized
                RTS                     * Nothing to return!
*******************************************************************************
* ABMODE111 - Decodes instructions with register mode of 111                  *
*******************************************************************************
abmode111       MOVEQ.L  #$0,D4         * Register mode is unrecognized
                RTS                     * Nothing to return!
*******************************************************************************
* REGMODE000 - Puts the register number in the good buffer and increments ptr *
*******************************************************************************
regmode000      MOVE.B  #'0',(A4)+      * Prints 0 for the register number
```

```
                RTS
*******************************************************************************
* REGMODE001 - Puts the register number in the good buffer and increments ptr *
*******************************************************************************
regmode001      MOVE.B  #'1',(A4)+      * Prints 1 for the register number
                RTS
*******************************************************************************
* REGMODE010 - Puts the register number in the good buffer and increments ptr *
*******************************************************************************
regmode010      MOVE.B  #'2',(A4)+      * Prints 2 for the register number
                RTS
*******************************************************************************
* REGMODE011 - Puts the register number in the good buffer and increments ptr *
*******************************************************************************
regmode011      MOVE.B  #'3',(A4)+      * Prints 3 for the register number
                RTS
*******************************************************************************
* REGMODE100 - Puts the register number in the good buffer and increments ptr *
*******************************************************************************
regmode100      MOVE.B  #'4',(A4)+      * Prints 4 for the register number
                RTS
*******************************************************************************
* REGMODE101 - Puts the register number in the good buffer and increments ptr *
*******************************************************************************
regmode101      MOVE.B  #'5',(A4)+      * Prints 5 for the register number
                RTS
*******************************************************************************
* REGMODE110 - Puts the register number in the good buffer and increments ptr *
*******************************************************************************
regmode110      MOVE.B  #'6',(A4)+      * Prints 6 for the register number
                RTS
*******************************************************************************
* REGMODE111 - Puts the register number in the good buffer and increments ptr *
*******************************************************************************
regmode111      MOVE.B  #'7',(A4)+      * Prints 7 for the register number
                RTS
*******************************************************************************
* BAD_INS - The instrcution is bad, change bad/good flag to bad and return    *
*******************************************************************************
bad_ins         MOVEQ   #$0,D4          * Change bad/good flag to bad
                RTS
*******************************************************************************
* BAD_INC - The instrcution is bad, change to bad and increment word count    *
*******************************************************************************
bad_inc         MOVEQ   #$0,D4          * Change bad/good flag to bad
                ADDI.L  #$1,D6          * Increment remaining word count
                RTS
*******************************************************************************
* BAD_INC1 - The instrcution is bad, change to bad and increment word count    *
*******************************************************************************
bad_inc1        MOVEQ   #$0,D4          * Change bad/good flag to bad
                ADDI.L  #$1,D6          * Increment remaining word count
                SUB.L   #$2,A5          * Restore old next address
                RTS
*******************************************************************************
* BAD_INCW - The instrcution is bad, change to bad and increment word count   *
*******************************************************************************
bad_incw        MOVEQ   #$0,D4          * Change bad/good flag to bad
                ADDI.L  #$1,D6          * Increment remaining word count
                SUB.L   #$2,A5          * Restore old next address
                RTS
*******************************************************************************
* BAD_INCL - The instrcution is bad, change to bad and increment word count   *
*******************************************************************************
bad_incl        MOVEQ   #$0,D4          * Change bad/good flag to bad
                ADDI.L  #$2,D6          * Increment remaining word count
                SUB.L   #$4,A5          * Restore old next address
                RTS
*******************************************************************************
* PRT_DIGS - Prints the digits in ASCII                                       *
*******************************************************************************
```

```
prt_digs        CMP.B   #$0A,D0         * Check to see if it's a number
                BLT     num             * It's a number, print out in ASCII
                CMP.B   #$9,D0          * Check to see if it's a letter
                BGT     char            * It's a letter, print out in ASCII
num             ADDI.B   #48,D0         * Add 48 to print in ASCII
                MOVE.B   D0,(A4)+       * Put in good buffer
                MOVE.B   D0,(A3)+       * Put in bad buffer
                RTS
char            ADDI.B   #55,D0         * Add 55 to print in ASCII
                MOVE.B   D0,(A4)+       * Put in good buffer
                MOVE.B   D0,(A3)+       * Put in bad buffer
                RTS
*******************************************************************************
* PRT_AD - Checks bit, and prints either D for data or A for Address          *
*******************************************************************************
prt_ad          CMP.B   #$0,D0          * Check to see if it's a 0
                BEQ     prt_d           * If so, print D
                CMP.B   #$1,D0          * Check to see if it's a 1
                BEQ     prt_a           * If so, print A
prt_d           MOVE.B  #'D',(A4)+      * Prints A to good buffer/increments
                RTS
prt_a           MOVE.B  #'A',(A4)+      * Prints A to good buffer/increments
                RTS
*******************************************************************************
* wrt_addr - We are going to write the address to the good and bad buffers    *
*******************************************************************************
wrt_addr      MOVE.B   #$30,D3          * Load 0
              MOVE.B   D3,(A4)+         * Load leading zero into good buffer
              MOVE.B   D3,(A3)+         * Load leading zero into bad buffer
              MOVE.B   D3,(A4)+         * Load 2nd zero into good buffer
              MOVE.B   D3,(A3)+         * Load 2nd zero into bad buffer
              MOVE.L   D5, D3           * Prepare for most significant
                                        * digit handling
              LSL.L    #8,D3            * Shift 8 to the left
              JSR      hexTo_char       * Decode the value
              MOVE.L   D5, D3           * Prepare for 2nd most significant
                                        * digit handling
              LSL.L    #8,D3            * Shift 12 to the left (8 now)
              LSL.L    #4,D3            * Shift 12 to the left (4 now)
              JSR      hexTo_char       * Decode the value
              MOVE.L   D5, D3           * Prepare for 3rd most significant
                                        * digit handling
              LSL.L    #8,D3            * Shift 16 to the left (8 now)
              LSL.L    #8,D3            * Shift 16 to the left (4 now)
              JSR      hexTo_char       * Decode the value
              MOVE.L   D5, D3           * Prepare for 4rd most significant
                                        * digit handling
              LSL.L    #8,D3            * Shift 20 to the left (8 now)
              LSL.L    #8,D3            * Shift 20 to the left (8 now)
              LSL.L    #4,D3            * Shift 20 to the left (4 now)
              JSR      hexTo_char       * Decode the value
              MOVE.L   D5, D3           * Prepare for 5th most significant
                                        * digit handling
              LSL.L    #8,D3            * Shift 24 to the left (8 now)
              LSL.L    #8,D3            * Shift 24 to the left (8 now)
              LSL.L    #8,D3            * Shift 24 to the left (8 now)
              JSR      hexTo_char       * Decode the value
              MOVE.L   D5, D3           * Prepare for 6th most significant
                                        * digit handling
              LSL.L    #8,D3            * Shift 28 to the left (8 now)
              LSL.L    #8,D3            * Shift 28 to the left (8 now)
              LSL.L    #8,D3            * Shift 28 to the left (8 now)
              LSL.L    #4,D3            * Shift 28 to the left (4 now)
              JSR      hexTo_char       * Decode the value
              RTS                       * Return to main loop
*******************************************************************************
* hexTo_char - this subroutine assists the wrt_addr subroutine. It shifts the *
*              hex value all the way to the right, then figures out whether   *
*              the hex value is an int or char, then manipulates it           *
*              accordingly                                                    *
*******************************************************************************
```

```
hexTo_char  LSR.L   #8,D3       * Shift 28 to the right (8 now)
            LSR.L   #8,D3       * Shift 28 to the right (8 now)
            LSR.L   #8,D3       * Shift 28 to the right (8 now)
            LSR.L   #4,D3       * Shift 28 to the right (4 now)
            CMP.B   #$09,D3     * Check to see if the char is less than A
            BGT     get_char
            ADDI.B  #48,D3      * The char must be A-F so add 55 to
                                * get ascii value
            MOVE.B  D3,(A4)+    * Load ascii value into good buffer
            MOVE.B  D3,(A3)+    * Load ascii value into bad buffer
            RTS                 * Return to wrt_addr
get_char    ADDI.B  #55,D3      * The char must be A-F so add 55 to get
                                * ascii value
            MOVE.B  D3,(A4)+    * Load ascii value into good buffer
            MOVE.B  D3,(A3)+    * Load ascii value into bad buffer
            RTS                 * Return to wrt_addr
*************************************************************************
* ck_cnt -  this subroutine handles the counting and reseting of the line     *
*           counter. When the line counter is 31, it is reset to zero and     *
*           a user prompt displays asking for the user to hit enter to display*
*           the next page                                                     *
*************************************************************************
ck_cnt  lea      linecount,A2  * Set up A2 ptr as linecount
        ADDI.B   #$1,(A2)      * Increment our line counter
        CMP.B    #31,(A2)      * Has our counter hit 31 yet?
        BEQ      pse_pg        * Pause the page if it is full
        RTS                    * If counter isnt at 31 then go back to
                               * main loop

pse_pg  MOVE.B   #00,(A2)      * Reset the counter then display message
        LEA      wait,A1       * Load lower address prompt
        MOVE.B   wait_len,D1   * Load lower prompt length
        MOVE.L   #1,D0         * Load task code for prompt
        TRAP     #15           * Display user prompt
        LEA      wait2,A1      * Load address to store user input
        MOVE.L   #2,D0         * Load task code for user input
        TRAP     #15           * Retrieve input
        RTS                    * Return to the main program loop
*************************************************************************
* the_end -  Displays our "graceful ending" ascii art by using trap 15 code 0 *
*************************************************************************
the_end  LEA      end1,A1           * Set up our A1 pointer to first ascii line
         MOVE.B   end1_len,D1       * Load 1st ascii line length
         MOVE.L   #0,D0             * Load task code
         TRAP     #15               * Display the first ascii art line
         LEA      end2,A1           * Set up our A1 pointer to 2nd ascii line
         MOVE.B   end2_len,D1       * Load 2nd ascii line length
         MOVE.L   #0,D0             * Load task code
         TRAP     #15               * Display the 2nd ascii art line
         LEA      end3,A1           * Set up A1 ptr to 3rd ascii line
         MOVE.B   end3_len,D1       * Load 3rd ascii line length
         MOVE.L   #0,D0             * Load task code
         TRAP     #15               * Display the 3rd ascii art line
         LEA      end4,A1           * Set up our A1 ptr to 4th ascii line
         MOVE.B   end4_len,D1       * Load 4th ascii line length
         MOVE.L   #0,D0             * Load task code
         TRAP     #15               * Display 4th ascii art line
         LEA      end5,A1           * Set up our A1 ptr to 5th ascii line
         MOVE.B   end5_len,D1       * Load 5th ascii line length
         MOVE.L   #0,D0             * Load task code
         TRAP     #15               * Display 5th ascii art line
         LEA      end6,A1           * Set up our A1 ptr to 6th ascii line
         MOVE.B   end6_len,D1       * Load 6th ascii line length
         MOVE.L   #0,D0             * Load task code
         TRAP     #15               * Display 6th ascii art line
         LEA      end7,A1           * Set up our A1 ptr to 7th ascii line
         MOVE.B   end7_len,D1       * Load 7th ascii line length
         MOVE.L   #0,D0             * Load task code
         TRAP     #15               * Display 7th ascii art line
         LEA      end8,A1           * Set up our A1 ptr to 8th ascii line
         MOVE.B   end8_len,D1       * Load 8th ascii line length
```

```
          MOVE.L   #0,D0             * Load task code
          TRAP     #15               * Display 8th ascii art line
          LEA      end9,A1           * Set up our A1 ptr to 9th ascii line
          MOVE.B   end9_len,D1       * Load 9th ascii line length
          MOVE.L   #0,D0             * Load task code
          TRAP     #15               * Display 9th ascii art line
          LEA      end10,A1          * Set up our A1 ptr to 10th ascii line
          MOVE.B   end10_len,D1      * Load 10th ascii line length
          MOVE.L   #0,D0             * Load task code
          TRAP     #15               * Display 10th ascii art line
          LEA      end11,A1          * Set up our A1 ptr to 11th ascii line
          MOVE.B   end11_len,D1      * Load 11th ascii line length
          MOVE.L   #0,D0             * Load task code
          TRAP     #15               * Display 11th ascii art line
          LEA      end12,A1          * Set up A1 ptr to 12th ascii line
          MOVE.B   end12_len,D1      * Load 12th ascii line length
          MOVE.L   #0,D0             * Load task code
          TRAP     #15               * Display 12th ascii art line
          LEA      end13,A1          * Set up A1 ptr to 13th ascii line
          MOVE.B   end13_len,D1      * Load 13th ascii line length
          MOVE.L   #0,D0             * Load task code
          TRAP     #15               * Display 13th ascii art line
          LEA      end14,A1          * Set up A1 ptr to 14th ascii line
          MOVE.B   end14_len,D1      * Load 14th ascii line length
          MOVE.L   #0,D0             * Load task code
          TRAP     #15               * Display 14th ascii art line
          LEA      end15,A1          * Set up A1 ptr to 15th ascii line
          MOVE.B   end15_len,D1      * Load 15th ascii line length
          MOVE.L   #0,D0             * Load task code
          TRAP     #15               * Display 15th ascii art line
*************************************************************************
* ld_badbuff - this subroutine handles the loading of the memory into the bad *
*              buffer A3                                                *
*************************************************************************
ld_badbuff  MOVE.W   (A5),D1       * Load D1 for manipulating 1st hex char
            JSR      bb_assist     * Handle the memory conversion for I/O
            MOVE.W   (A5),D1       * Load D1 for manipulating 2nd hex char
            LSL.W    #4,D1         * Handle
            JSR      bb_assist     * Handle the memory conversion for I/O
            MOVE.W   (A5),D1       * Load D1 for manipulating 3rd hex char
            LSL.W    #8,D1         * Shift 8 to the left for hex manipulation
            JSR      bb_assist     * Handle the memory conversion for I/O
            MOVE.W   (A5),D1       * Load D1 for manipulating 4th hex char
            LSL.W    #8,D1         * Shift 8 left (12 total)
            LSL.W    #4,D1         * Shift 4 left (12 total)
            JSR      bb_assist     * Handle the memory conversion for I/O
            MOVE.B   #$00,(A3)     * End of bad buffer
            RTS
*************************************************************************
* bb_assist -  bad buffer load assist                                   *
*              handles the conversion of hex characters in memory       *
*              to the proper values to display the hex characters to the *
*              output window                                            *
*************************************************************************
bb_assist   LSR.W    #8,D1    * Shift hex digit to the right 8 (12 total)
            LSR.W    #4,D1    * Shift hex digit to the right 4 (12 total)
            CMP.B    #$0A,D1  * Compare hex A char to whats in D1
            BLT      int      * If its less than, BRA to handle into conversion
            ADDI.B   #55,D1   * Otherwise handle hex char conversion, add 55
            MOVE.B   D1,(A3)+ * Add to bad buffer and increment it
            RTS               * Return back to ld_buffer subroutine
int         ADDI.B   #48,D1   * handle int char conversion, add 48
            MOVE.B   D1,(A3)+ * Add to bad buffer and increment it
            RTS               * Return back to ld_buffer subroutine

testing     MOVE.L      (A6)+,D2

            END     $00001000
```