```
*-----------------------------------------------------------------------------
* Title      : 422 Disassembler Final Project
* Written by : Amy Meyers, Mariah Files, Mustafa Majeed
* Date       : 3/10/2019
*
* Description: This program is written to scan sections of memory and attempt
*               convert the contents in memory to a valid string that can be
*               used as assembly language code for the 68K chip.
*
*               The program parses the op-code word of the instruction at
*               a specified space in memory and then decides how many
*               additional words of memory need to be read in order to
*               complete the instruction, then proceeds to read everything
*               that is needed to extract a valid 68K instruction.
*
*               For each valid 68K machine code instruction that resides in
*               memory, this program will print out a complete instruction in
*               ASCII readable format.
*
*               The required Op-Codes Instructions for this program are:
*               MOVE,MOVEA,MOVEM        OR,ORI          ROL,ROR
*               ADD,ADDA                NEG             BCLR
*               SUB,SUBQ                EOR             CMP,CMPI
*               MULS,DIVS               LSR,LSL         Bcc (BCS,BGE,BLT,BVC)
*               LEA                     ASR,ASL         BRA,JSR,RTS
*
*
*               The required Effective Addressing Modes for this program are:
*               Data Register Direct        Address Reg Indirect w/ Post Increment
*               Address Register Direct     Address Reg Indirect w/ Pre decrement
*               Address Register Indirect   Absolute Long Address
*               Immediate Addressing        Absolute Word Address
*
*
*-----------------------------------------------------------------------------
Areg_list   REG A0-A6
Dreg_list   REG D0-D7
*-----------------------------------------------------------------------------
    ORG    $1000    *Don't change this; see hints and tips above
*-----------------------------------------------------------------------------

START:
*-----------------------------------------------------------------------------
* TESTING INSTRUCTIONS

*--------NOTES - Amy --------
* I cannot figure out the second data register for ADD commands
* Code goes into an infinite loop or problem at 9020
* My absolute EA modes do not work...I think the registers and/or shifts need to be modified
*-----------------------------------------------------------------------------

    *MOVE.W  #$DA31,$9002 *ADD.B
    *MOVE.W  #$DA71,$9004 *ADD.W
    *MOVE.W  #$DAB1,$9006 *ADD.L
    *MOVE.W  #%1101101100110001,$9008    *ADD.B
    *MOVE.W  #%1101101101110001,$900A    *ADD.W
    *MOVE.W  #%1101101110110001,$900C    *ADD.L
    *MOVE.W  #%1101101011110001,$900E    *ADDA.W
    * 9020
    * DATA
    *MOVE.W  #%1110101100101001,$9022    *LSLB
    *MOVE.W  #%1110101101101001,$9024    *LSLW
    *MOVE.W  #%1110101110101001,$9026    *LSLL
    * DATA
    *MOVE.W  #%1110101000101001,$902A    *LSRB
    *MOVE.W  #%1110101001101001,$902C    *LSRW
    *MOVE.W  #%1110101010101001,$902E    *LSRL
    * 9030
    *DATA
    *MOVE.W  #%0110010100000000,$9032    *BCS
    *MOVE.W  #%0110110000000000,$9034    *BGE
```

```
    *MOVE.W  #%0110110100000000,$9036    *BLT
    *MOVE.W  #%0110100000000000,$9038    *BVC
    *MOVE.W  #%0000101001110001,$903A    *BCLR
    *MOVE.W  #%1011101000110001,$903C    *CMPB
    *MOVE.W  #%1011101001110001,$903E    *CMPW
    * 9040
    *MOVE.W  #%1011101010110001,$9040    *CMPL
    *MOVE.W  #%1011101100110001,$9042    *EORB
    *MOVE.W  #%1011101101110001,$9044    *EORW
    *MOVE.W  #%1011101110110001,$9046    *EORL
    *MOVE.W  #%1000101111110001,$9048    *DIVSW
    *MOVE.W  #%1000101000110001,$904A    *ORB1
    *MOVE.W  #%1000101001110001,$904C    *ORW1
    *MOVE.W  #%1000101010110001,$904E    *ORL1
    * 9050
    *MOVE.W  #%1000101100110001,$9050    *ORB2
    *MOVE.W  #%0010101001110001,$9060    *MOVEAL
    *MOVE.W  #%1100101001110001,$9062    *MULS
    *MOVE.W  #%1001101000110001,$9064    *SUBB
    *MOVE.W  #%1001101001110001,$9066    *SUBW
    *MOVE.W  #%1001101010110001,$9068    *SUBL
    *MOVE.W  #%0100100010111000,$9070 *MOVEMW
    *MOVE.W  #%0100100011111000,$9072 *MOVEML
    *MOVE.W  #%0110000010100111,$9074 *BRA
    *MOVE.W  #%0100111010100111,$9076 *JSR


*-------------------END TEST----------------

    LEA begMSG1, A1     *load "Welcome to CSS 422 Disassembler Final Project" to A1 for printing
    JSR TrapTask13

    LEA begMSG2, A1     *load "The following program was created by Amy Meyers, Mariah Files and
Mustafa Majeed" to A1 for printing
    JSR TrapTask13

    LEA begMSG3, A1     *load 1st task message
    JSR TrapTask13

    JSR read_file       *go to read_file mehthod. Read the cfg file

    LEA MSG6,A1         *load 2nd task message
    JSR TrapTask13

main *//main loop for program
    CMP.L   A3,A2       *compare cur address with ending address
    BGT     end         *go to end if A2(cur address) is greater than A3 (ending address)
    LEA     GBuf,A4     *load address to beginning of good string buf
    LEA     BBuf,A5     *load address to beginning of bad string buf
    JSR     fillBuff    *fill both buffers with correct address information
    JSR     dis         *start disassembling the machine code residing at A2 current address

*******************************************************************************
* Method Name: dis
* Description: this method is used to read the machine code that is at A2 current
*               address and update good/bad string buffers to contain the correct
*               information based on the read instruction
* Preconditions: A2 must contain current address with machine code. good string
*                   buffer address must be in A4 and bad string buffer address
*                   must be in A5.
* Postconditions: good or bad string buffers will be updated to contain the
*                   correct information based on the read instruction at the
*                   current memory location in A2
*******************************************************************************
dis *{
    *//begin disassembling instruction at current address in A2
    BRA disassemble
    *_____________________________________________________________

    *//call this method when running into bad instruction after adding
    *//the bad instruction that was read to A5
```

```
    *JSR print_bad

    *//call this method after filling A4 with proper message when finished
    *//reading good machine code from current address in A2
    *JSR print_good
*}

*****************************************************************************
* Method Name: print_bad
* Description: this method is used to load bad string buffer to A1 and print it to
*              to console and store it in output file in same directory as this
*              X68 file.
* Preconditions: BBuf must be properly set and NULL terminated to ensure that it is
*                  output correctly
* Postconditions: BBuf will be displayed to console and stored in output file and
*                  A2 will be moved to next word of memory and branch back to main
*                  loop
*****************************************************************************
print_bad *{
    JSR     writeInst   * write bad instruction to BBuff
    MOVE.B  #$00,(A5)+  * NULL terminate BBuff
    LEA     BBuf,A1     * load for output
    JSR     TrapTask13
    MOVE.W  (A2)+,D7    * Move to next instruction in memory
    BRA     main        * go back to main loop

writeInst
    *//callee save
    MOVEM.L  Dreg_list,-(SP)
    *_________________

    MOVE.W (A2),D4      * Load D4 with instruction from A2 current Address
    JSR wordHextoAscii  * Get 1st char
    MOVE.W (A2),D4      * Load D4 with instruction from A2 current Address (again)
    LSL.W #4,D4         * make shift to be able to extract 2nd char
    JSR wordHextoAscii  * retrieve it
    MOVE.W (A2),D4      * Load D4 with instruction from A2 current Address (again)
    LSL.W #8,D4         * make shift to position 3rd char
    JSR wordHextoAscii  * retrieve it
    MOVE.W (A2),D4      * Load D4 with instruction from A2 current Address (again)
    LSL.W #4,D4         * make shift to position last
    LSL.W #8,D4         * make shift to position last
    JSR wordHextoAscii  * retrieve it

    *//callee save tests
    MOVE.L #$00FFFF33,D2
    MOVE.L #$00FFFF33,D5
    MOVE.L #$00FFFF33,D6
    MOVE.L #$00FFFF33,D7

    *//callee restore
    MOVEM.L  (SP)+,Dreg_list
    *_________________
    RTS

wordHextoAscii
    LSR.W #4,D4
    LSR.W #8,D4
    CMP.B #$0A,D4   * check if 0-9
    BGE letter      * its A-F
    ADDI.B #$30,D4  * get correct Ascii value for number
    MOVE.B D4,(A5)+ * put in BBuff
    RTS             * back to writeAdd

letter
    ADDI.B #$37,D4  * get correct Ascii value
    MOVE.B D4,(A5)+ * put in BBuff
    RTS             * back to writeAdd
*}

*****************************************************************************
```

```
* Method Name: print_good
* Description: this method is used to load good string buffer to A1 and print it to
*               to console and store it in output file in same directory as this
*               X68 file.
* Preconditions: GBuf must be properly set and NULL terminated to ensure that it is
*                   output correclty
* Postconditions: GBuf will be displayed to console and stored in output file and
*                   A2 will be moved to next word of memory and branch back to main
*                   loop
*********************************************************************************
print_good *{
    MOVE.B  #$00,(A4)+  * NULL terminate GBuf
    LEA     GBuf,A1     * load for output
    JSR     TrapTask13
    CMPI.B  #1,D1
    BEQ     skipNLong
    MOVE.W  (A2)+,D7    * Move to next instruction in memory
    BRA     main        * go back to main loop
*}

*//this will end the program... USE WHEN CURRENT ADDRESS IS GREATER THAN ENDING ADDRESS
end
    STOP #$00003000

skipNLong
    MOVEQ   #0,D1
    MOVE.W  (A2)+,D7
    MOVE.L  (A2)+,D7
    BRA     main



*********************************************************************************
* Method Name: read_file
* Description: this method is used to read a Cconfig.cfg" file for a starting and
*               an ending address for the disassembler program to begin working
*               from and to
* Preconditions:  "Config.cfg" must be placed in the same directory as this
*                   main.X68 file
* Postconditions: A2 will be pointing to the starting address, A3 will be
*                   pointing to ending address
*********************************************************************************
read_file
    LEA     inputFileName,A1    *load file name of desired file that contains info
    MOVE.B  #51,D0
    TRAP    #15

    MOVE.L  #fileSize,D2        * attemp to read 80 bytes
    LEA     inputFileBuf,A1     * load address of where file contents will be stored
    MOVE.B  #53,D0
    TRAP    #15                 * get contents

    CMPI.L  #$00,D2
    BEQ     emptyFile

    *//check if 2 32 bit mem addresses were read
    *//if
    CMPI.L  #20,D2
    BNE     LFonly
    *//else
    MOVEA.L A1,A0               * move address where file contents are stored to A0
    LEA     startingBuf,A1      * load startingBuf address to A1

loop        *//will read each char in file until CR is read
    CMP.B   #$0D,(A0)           * is this a CR char?
    BEQ     endOfStart          * end loop
    MOVE.B  (A0)+,(A1)+         * if not, copy this char to startingBuf address
    BRA     loop                * check next char

endofStart  *//loop exit condition
    MOVE.B  #0,(A1)+            * NULL terminate startingBuf
```

```
    LEA     MSG4,A1             * load "Starting address read = " message for output
    JSR     TrapTask13
    LEA     startingBuf,A1
    JSR     TrapTask13          * load read starting address to output

    JSR     AsciiToHex          * change ascii read address to hex value
    MOVE.L  D7,A2               * move hex value for address to A2 register to track current
address
    LEA     endingBuf,A1        * load endingBuf address to A1
    MOVE.B  (A0)+,D7            * discard the CR char
    MOVE.B  (A0)+,D7            * discard the LF char

loopEnd     *// read remaining char until a CR is read
    CMP.B   #$0D,(A0)           * is this a CR char?
    BEQ     endOfMethod         * end loop
    MOVE.B  (A0)+,(A1)+         * if not, copy this char to endingBuf address
    BRA     loopEnd             * check next char

endOfMethod *//loop exit condition
    MOVE.B  #0,(A1)+            * NULL terminate endingBuf
    LEA     MSG5,A1             * load "Ending address read = " message for output
    JSR     TrapTask13
    LEA     endingBuf,A1        * load read endingBuf address to output
    JSR     TrapTask13
    JSR     AsciiToHex          * change ascii read address to hex value
    MOVE.L  D7,A3               * move hex value for address to A3 reg to track ending address

    CMP.L   A3,A2               * check if start add < end add
    BGE     invalidSE           * go to print err message
    JSR     odd
    *//number of instructions to disassemble (WE MIGHT NOT EVEN NEED THIS)
    *SUB.L   A2,D7               * subtract starting address from ending address to find out the
scope of the memory locations
    *DIVU.   #$2,D7              * divide by 2 to check how many instructions will need
disassembling
    *MOVE.L  D7,D6               * store the result in D6--------------(we might not even need
this)

    RTS

*************************************************************************
* Method Name: LFonly
* Description: this method is used to attempt to read a file that is using MAC
*               text file line endings... ONLY LF char
* Preconditions:  "Config.cfg" must be placed in the same directory as this
*                  main.X68 file
* Postconditions: A2 will be pointing to the starting address, A3 will be
*                  pointing to ending address
*************************************************************************
LFonly
    CMPI.L  #12,D2
    BNE     inputError

    MOVEA.L A1,A0               * move address where file contents are stored to A0
    LEA     startingBuf,A1      * load startingBuf address to A1

loop1        *//will read each char in file until CR is read
    CMP.B   #$0A,(A0)           * is this a CR char?
    BEQ     endOfStart1         * end loop
    MOVE.B  (A0)+,(A1)+         * if not, copy this char to startingBuf address
    BRA     loop1               * check next char

endofStart1  *//loop exit condition
    MOVE.B  #0,(A1)+            * NULL terminate startingBuf
    LEA     MSG4,A1             * load "Starting address read = " message for output
    JSR     TrapTask13
    LEA     startingBuf,A1
    JSR     TrapTask13          * load read starting address to output

    JSR     AsciiToHex          * change ascii read address to hex value
```

```
    MOVE.L  D7,A2               * move hex value for address to A2 register to track current
address
    LEA     endingBuf,A1        * load endingBuf address to A1
    MOVE.B  (A0)+,D7            * discard the LF char

loopEnd1     *// read remaining char until a CR is read
    CMP.B   #$0A,(A0)           * is this a CR char?
    BEQ     endOfMethod1        * end loop
    MOVE.B  (A0)+,(A1)+         * if not, copy this char to endingBuf address
    BRA     loopEnd1            * check next char

endOfMethod1 *//loop exit condition
    MOVE.B  #0,(A1)+            * NULL terminate endingBuf
    LEA     MSG5,A1             * load "Ending address read = " message for output
    JSR     TrapTask13
    LEA     endingBuf,A1        * load read endingBuf address to output
    JSR     TrapTask13
    JSR     AsciiToHex          * change ascii read address to hex value
    MOVE.L  D7,A3               * move hex value for address to A3 reg to track ending address

    CMP.L   A3,A2               * check if start add < end add
    BGE     invalidSE           * go to print err message
    JSR     odd
    RTS
*******************************************************************************
* Method Name: inputError
* Description: this method is used to let the user know that the "Config.cfg" file
*               is not properly formatted and will end the program.
* Preconditions:  "Config.cfg" must be placed in the same directory as this
*                   main.X68 file
* Postconditions: The program will end because config.cnf file is not formatted
*                   correctly
*******************************************************************************
inputError
    LEA     inFileERR,A1    * load error formatting message
    JSR     trapTask13      * print and save
    SIMHALT                 * end progran

*******************************************************************************
* Method Name: invalidSE
* Description: this method is used to let the user know that the "Config.cfg" file
*               contains a starting Address that is equal to or larger than ending
*                address
* Preconditions:  "Config.cfg" must be placed in the same directory as this
*                   main.X68 file
* Postconditions: The program will end because config.cnf file is not formatted
*                   correctly
*******************************************************************************
invalidSE
    LEA     inval,A1        * load error formatting message
    JSR     trapTask13      * print and save
    SIMHALT                 * end progran

*******************************************************************************
* Method Name: emptyFile
* Description: this method is used to let the user know that the "Config.cfg" file
*               is empty
* Preconditions:  "Config.cfg" must be placed in the same directory as this
*                   main.X68 file
* Postconditions: The program will end because config.cnf file is empty
*******************************************************************************
emptyFile
    LEA     emptyF,A1
    JSR     trapTask13      * print and save
    SIMHALT                 * end progran

*******************************************************************************
* Method Name: odd
* Description: this method is used to check if any of the read addresses from the
*               Config.cfg file are odd
* Preconditions:  "Config.cfg" must be placed in the same directory as this
```

```
*                   main.X68 file
* Postconditions: The program will end because config.cnf file is empty
*******************************************************************************
odd
    MOVE.L  A2,D4
    MOVE.B  #$02,D3
    DIVU    D3,D4
    SWAP    D4
    CMPI.B  #$01,D4
    BEQ     oddStart

    MOVE.L  A3,D4
    MOVE.B  #$02,D3
    DIVU    D3,D4
    SWAP    D4
    CMPI.B  #$01,D4
    BEQ     oddEnd

    RTS

oddStart
    LEA     oddS,A1
    JSR     TrapTask13
    SIMHALT

oddEnd
    LEA     oddE,A1
    JSR     TrapTask13
    SIMHALT
*******************************************************************************
* Method Name: fillBuff
* Description: this method will fill the good and bad string buffers to the correct
*               output for the beginning of the instruction
*
* Preconditions:  A4 reg contains good buffer and A5 contains bad buffer
* Postconditions: good buffer will contain the current address and bad buffer will
*                   conatin current address and DATA flag. THE BUFFERS ARE WILL
*                   NOT BE NULL TERMINATED AFTER THIS METHOD CALL. !YOU MUST ENSURE
*                   THAT THEY ARE NULL TERMINATED USING ANOTHER METHOD!
*******************************************************************************
fillBuff:
    *//Callee save reg
    MOVEM.L  Dreg_list,-(SP)
    *--------------------
    MOVE.B  #'A',(A4)+  *// fill the beginning of the buffers
    MOVE.B  #'A',(A5)+
    MOVE.B  #'t',(A4)+
    MOVE.B  #'t',(A5)+
    MOVE.B  #':',(A4)+
    MOVE.B  #':',(A5)+
    MOVE.B  #' ',(A4)+
    MOVE.B  #' ',(A5)+
    MOVE.B  #$30,(A4)+
    MOVE.B  #$30,(A4)+
    MOVE.B  #$30,(A5)+
    MOVE.B  #$30,(A5)+

    *//most sig char
    MOVE.L  A2,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//next char
    MOVE.L  A2,D5
    LSL.L   #4,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//next char
    MOVE.L  A2,D5
    LSL.L   #8,D5
```

```
    LSL.L   #8,D5
    JSR     hexToAscii

    *//next char
    MOVE.L  A2,D5
    LSL.L   #4,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//next char
    MOVE.L  A2,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//last char
    MOVE.L  A2,D5
    LSL.L   #4,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    MOVE.B  #' ',(A4)+
    MOVE.B  #' ',(A4)+
    MOVE.B  #' ',(A4)+
    MOVE.B  #' ',(A5)+
    MOVE.B  #' ',(A5)+
    MOVE.B  #' ',(A5)+
    MOVE.B  #'D',(A5)+
    MOVE.B  #'A',(A5)+
    MOVE.B  #'T',(A5)+
    MOVE.B  #'A',(A5)+
    MOVE.B  #' ',(A5)+
    MOVE.B  #' ',(A5)+
    MOVE.B  #' ',(A5)+
    MOVE.B  #'$',(A5)+

    *//Callee restore reg
    MOVEM.L  (SP)+,Dreg_list
    *---------------------

    RTS
*******************************************************************************
* Method Name: hexToAscii
* Description: this method will convert a hex value to an Ascii char value for
*               help with printing current address or instruction to both the
*               good and bad display buffers
*
* Preconditions:  D5 must contain the byte you want to translate in the least sig
*                   byte position
* Postconditions: A4 and A5 buffers will both contain the chars that correspond to
*                   the values in D5 at the time of the call to this method
*******************************************************************************
hexToAscii
    LSR.L   #4,D5
    LSR.L   #8,D5
    LSR.L   #8,D5
    LSR.L   #8,D5

    CMP.B   #$0A,D5
    BLT     asciiNumber
    ADDI.B  #$37,D5      * Add to value to get A-F char
    MOVE.B  D5,(A4)+    * copy to good buff
    MOVE.B  D5,(A5)+    * copy to bad buff
    RTS

*//use to translate 0-9 values
asciiNumber
```

```
    ADDI.B  #$30,D5      * Add to get 0-9 char
    MOVE.B  D5,(A4)+    * copy to good buff
    MOVE.B  D5,(A5)+    * copy to bad buff
    RTS

*-----------------------------------------------------------
* OPCODE RECOGNITION
* Written by :Mariah Files
* Date       :3/7/19
* Description:Recognizes supported opcodes.
*
* NOTE: may need to distinguish between MOVEM and LEA if
* the first part of the next chunk of code starts with a 1
*
* Assumes that A2 is pointing at a longword sized
* machine code in binary. (I/O)
*
* Assumes that the machine code starts with the opcode
* and has trailing zeros if necessary to fill up
* longword size.
*
* Replace print functions with JTS EA
*
* EA needs to distinguish between:
* ASL/ASR, LSL/LSR, ROL/ROR
* CMP and EOR
* DIVS and OR
* MOVE and MOVEA (only in .W and .L sizes)
* ADD and ADDA
*
* DATA REGISTER USES
*
* D0: trap tasks
* D1: print numbers
* D2: whole machine code
* D3: carry set T/F
* D4: portion of machine code
* D5: desired opcode
* D6: opcode size
* D7: counter
*
* ADDRESS REGISTER USES
*
* A2: machine code pointer
* A7: stack pointer
*-----------------------------------------------------------

*-----------------------------------------------
* EQUATES SECTION
* Contains the binary codes to be used for
* comparison to determine which opcode it is.
* Ordered by size of opcode (nibble, 5 bit,
* byte, 10 bit, word)
*-----------------------------------------------
CR      EQU    $0D
LF                  EQU     $0A
stack       EQU     $000A0000               * locate the stack if necessary

* NIBBLE SIZE OPCODES
ADD_CODE    EQU     %1101
AsLsRo_CODE EQU     %1110
Bcc_CODE    EQU     %0110 *BCS, BGE, BLT, BVC
BCLR_CODE   EQU     %0000
CmpEor_CODE EQU     %1011
DivsOr_CODE EQU     %1000
LEA_CODE    EQU     %0100
MOVEB_CODE  EQU     %0001
MOVEW_CODE  EQU     %0011
MOVEL_CODE  EQU     %0010
MULS_CODE   EQU     %1100
SUB_CODE    EQU     %1001
```

```
SUBQ_CODE   EQU     %0101

* 5 BIT SIZE OPCODE
MOVEM_CODE  EQU     %01001

* BYTE SIZE OPCODES
BRA_CODE    EQU     %01100000
BCS_CODE    EQU     %01100101
BGE_CODE    EQU     %01101100
BLT_CODE    EQU     %01101101
BVC_CODE    EQU     %01101000

* 10 BIT OPCODE
JSR_CODE    EQU     %0100111010
CMPIB_CODE  EQU     %0000110000
CMPIW_CODE  EQU     %0000110001
CMPIL_CODE  EQU     %0000110010
NEGB_CODE   EQU     %0100010000
NEGW_CODE   EQU     %0100010001
NEGL_CODE   EQU     %0100010010
ORIB_CODE   EQU     %0000000000
ORIW_CODE   EQU     %0000000001
ORIL_CODE   EQU     %0000000010

* WORD SIZE OPCODE
RTS_CODE    EQU     %0100111001110101
NOP_CODE    EQU     %0100111001110001

* OPMODES
BYTE1_CODE  EQU     %00000000
WORD1_CODE  EQU     %00000001
LONG1_CODE  EQU     %00000010
BYTE2_CODE  EQU     %00000100
WORD2_CODE  EQU     %00000101
LONG2_CODE  EQU     %00000110

ADDAW_CODE  EQU     %00000011               * ADDA.W
ADDAL_CODE  EQU     %00000111               * ADDA.L

* TEST VALUES
ADDB1_T     EQU     %1101101000110001
ADDW1_T     EQU     %1101101001110001
ADDL1_T     EQU     %1101101010110001
ADDB2_T     EQU     %1101101100110001
ADDW2_T     EQU     %1101101101110001
ADDL2_T     EQU     %1101101110110001
ADDAW_T     EQU     %1101101011110001
ADDAL_T     EQU     %1101101111110001

ASLB_T      EQU     %1110101100100001
ASLW_T      EQU     %1110101101100001
ASLL_T      EQU     %1110101110100001
ASL_T       EQU     %1110101111100001
ASRB_T      EQU     %1110101000100001
ASRW_T      EQU     %1110101001100001
ASRL_T      EQU     %1110101010100001
ASR_T       EQU     %1110101011100001

LSLB_T      EQU     %1110101100101001
LSLW_T      EQU     %1110101101101001
LSLL_T      EQU     %1110101110101001
LSL_T       EQU     %1110101111101001
LSRB_T      EQU     %1110101000101001
LSRW_T      EQU     %1110101001101001
LSRL_T      EQU     %1110101010101001
LSR_T       EQU     %1110101011101001

BCS_T       EQU     %0110010100000000
BGE_T       EQU     %0110110000000000
BLT_T       EQU     %0110110100000000
BVC_T       EQU     %0110100000000000
```

```
BCLR_T      EQU     %0000101001110001

CMPB_T      EQU     %1011101000110001
CMPW_T      EQU     %1011101001110001
CMPL_T      EQU     %1011101010110001
EORB_T      EQU     %1011101100110001
EORW_T      EQU     %1011101101110001
EORL_T      EQU     %1011101110110001

DIVSW_T     EQU     %1000101111110001
ORB1_T      EQU     %1000101000110001
ORW1_T      EQU     %1000101001110001
ORL1_T      EQU     %1000101010110001
ORB2_T      EQU     %1000101100110001
ORW2_T      EQU     %1000101101110001
ORL2_T      EQU     %1000101110110001

LEA_T       EQU     %0100001001110001

MOVEB_T     EQU     %0001101011110001
MOVEW_T     EQU     %0011101011110001
MOVEL_T     EQU     %0010101011110001
MOVEAW_T    EQU     %0011101001110001
MOVEAL_T    EQU     %0010101001110001


MULS_T      EQU     %1100101001110001
SUBB_T      EQU     %1001101000110001
SUBW_T      EQU     %1001101001110001
SUBL_T      EQU     %1001101010110001

SUBQB_T     EQU     %0101101000110001
SUBQW_T     EQU     %0101101001110001
SUBQL_T     EQU     %0101101010110001

MOVEMW_T    EQU     %0100110100111000
MOVEML_T    EQU     %0100110101111000
BRA_T       EQU     %0110000010100111
JSR_T       EQU     %0100111010100111
CMPIB_T     EQU     %0000110000100111
CMPIW_T     EQU     %0000110001100111
CMPIL_T     EQU     %0000110010100111
NEGB_T      EQU     %0100010000100111
NEGW_T      EQU     %0100010001100111
NEGL_T      EQU     %0100010010100111
ORIB_T      EQU     %0000000000100111
ORIW_T      EQU     %0000000001100111
ORIL_T      EQU     %0000000010100111
RTS_T       EQU     %0100111001110101
NOP_T       EQU     %0100111001110001

*****************************************************
* Method Name:  disassemble
* Description:  Determines which opcode is indicated
*               by the machine code. If the opcode
*               is not supported, it prints the
*               bad buffer. If the opcode is
*               supported, it adds the opcode portion
*               to the good buffer and jumps to the
*               EA portion.
* Preconditions:A2 is pointing to the machine code
*               represented as a binary number.
*
*****************************************************
disassemble:
            MOVE.L  #0,D4
            MOVE.L  (A2),D4                 * Put machine code in D4
            MOVE.L  #16,D6                  * end loop
            MOVE.L  #0,D7
LOOP_OP     ASL.L   #1,D4                   * Shift machine code
```

```
            BCS     LONG_SIZED              * If carry was set at any time, it is a long sized
word
            ADDI.L  #1,D7
            CMP.L   D6,D7
            BEQ     WORD_SIZED
            BRA     LOOP_OP

WORD_SIZED  ADDA.W #2,A2                   * Increment past leading 0's
            MOVE.L  #0,D5
            CLR.L   D6
            CLR.L   D7
            CLR.L   D4
            BRA     START_OPS

            *MOVE.W  #SUBQL_T,(A2)             * Hardcoded test machine code
LONG_SIZED  MOVE.L  #0,D5                  * Initialize D5
            BRA     START_OPS

*****************************************************
* RECOGNIZE WORD SIZE OPCODES
*****************************************************
START_OPS   MOVE.L  (A2),D2                * Initialize D2 with machine code
            MOVE.L  #0,D4                  * Initialize D4 for opcode
            MOVE.L  #16,D6                 * Initialize D6 with current opcode size to check
            MOVE.L  #0,D7                  * Initialize D7 for number of bits shifted
            BRA     SHIFT

WORD_OPS    CMPI.W  #RTS_CODE,(A2)         * Is it RTS?
            BEQ     PRINT_RTS              * If so, print RTS

            CMPI.W  #NOP_CODE,(A2)         * Is it NOP?
            BEQ     PRINT_NOP              * If so, print NOP

*****************************************************
* RECOGNIZE TEN BIT OPCODES
*****************************************************
            MOVE.L  (A2),D2                * Initialize D2 with machine code
            MOVE.L  #0,D4                  * Initialize D4 for opcode
            MOVE.L  #10,D6                 * Initialize D6 with current opcode size to check
            MOVE.L  #0,D7                  * Initialize D7 for number of bits shifted

            BRA     SHIFT

TENBIT_OPS  MOVE.L  #JSR_CODE,D5           * JSR
            CMP.L   D5,D4
            BEQ     PRINT_JSR

            MOVE.L  #CMPIB_CODE,D5         * CMPI.B
            CMP.L   D5,D4
            BEQ     PRINT_CMPIB

            MOVE.L  #CMPIW_CODE,D5         * CMPI.W
            CMP.L   D5,D4
            BEQ     PRINT_CMPIW

            MOVE.L  #CMPIL_CODE,D5         * CMPI.L
            CMP.L   D5,D4
            BEQ     PRINT_CMPIL

            MOVE.L  #NEGB_CODE,D5          * NEG.B
            CMP.L   D5,D4
            BEQ     PRINT_NEGB

            MOVE.L  #NEGW_CODE,D5          * NEG.W
            CMP.L   D5,D4
            BEQ     PRINT_NEGW

            MOVE.L  #NEGL_CODE,D5          * NEG.L
            CMP.L   D5,D4
            BEQ     PRINT_NEGL
```

```
            MOVE.L  #ORIB_CODE,D5           * ORI.B
            CMP.L   D5,D4
            BEQ     PRINT_ORIB

            MOVE.L  #ORIW_CODE,D5           * ORI.W
            CMP.L   D5,D4
            BEQ     PRINT_ORIW

            MOVE.L  #ORIL_CODE,D5           * ORI.L
            CMP.L   D5,D4
            BEQ     PRINT_ORIL

*****************************************************
* RECOGNIZE BYTE SIZE OPCODES
*****************************************************

            MOVE.L  (A2),D2                 * Initialize D2 with machine code
            MOVE.L  #0,D4                   * Initialize D4 for opcode
            MOVE.L  #8,D6                   * Initialize D6 with current opcode size to check
            MOVE.L  #0,D7                   * Initialize D7 for number of bits shifted

            BRA     SHIFT

BYTE_OPS    MOVE.L  #BRA_CODE,D5            * BRA
            CMP.L   D5,D4
            BEQ     PRINT_BRA

            MOVE.L  #BCS_CODE,D5            * BCS
            CMP.L   D5,D4
            BEQ     PRINT_BCS

            MOVE.L  #BGE_CODE,D5            * BGE
            CMP.L   D5,D4
            BEQ     PRINT_BGE

            MOVE.L  #BLT_CODE,D5            * BLT
            CMP.L   D5,D4
            BEQ     PRINT_BLT

            MOVE.L  #BVC_CODE,D5            * BVC
            CMP.L   D5,D4
            BEQ     PRINT_BVC

*****************************************************
* RECOGNIZE FIVE BIT OPCODES
*****************************************************

            MOVE.L  (A2),D2                 * Initialize D2 with machine code
            MOVE.L  #0,D4                   * Initialize D4 for opcode
            MOVE.L  #5,D6                   * Initialize D6 with current opcode size to check
            MOVE.L  #0,D7                   * Initialize D7 for number of bits shifted

            BRA     SHIFT

FIVEBIT_OPS MOVE.L  #MOVEM_CODE,D5          * MOVEM
            CMP.L   D5,D4
            BEQ     PRINT_MOVEM

*****************************************************
* RECOGNIZE NIBBLE SIZE OPCODES
*****************************************************

            MOVE.L  (A2),D2                 * Initialize D2 with machine code
            MOVE.L  #0,D4                   * Initialize D4 for opcode
            MOVE.L  #4,D6                   * Initialize D6 with current opcode size to check
            MOVE.L  #0,D7                   * Initialize D7 for number of bits shifted

            BRA     SHIFT

NIBBLE_OPS  MOVE.L  #ADD_CODE,D5            * ADD
            CMP.L   D5,D4
```

```
            BEQ     PRINT_ADD

            MOVE.L  #AsLsRo_CODE,D5         * AS*/LS*/RO*
            CMP.L   D5,D4
            BEQ     AsLsRo_OPM

            MOVE.L  #BCLR_CODE,D5           * BCLR
            CMP.L   D5,D4
            BEQ     PRINT_BCLR

            MOVE.L  #CmpEor_CODE,D5         * CMP or EOR
            CMP.L   D5,D4
            BEQ     CmpEor_OPM

            MOVE.L  #DivsOr_CODE,D5         * DIVS or OR
            CMP.L   D5,D4
            BEQ     DivsOr_OPM

            MOVE.L  #LEA_CODE,D5            * LEA or MOVEM
            CMP.L   D5,D4
            BEQ     PRINT_LEA

            MOVE.L  #MOVEB_CODE,D5          * MOVE.B
            CMP.L   D5,D4
            BEQ     PRINT_MOVEB

            MOVE.L  #MOVEW_CODE,D5          * MOVE.W
            CMP.L   D5,D4
            BEQ     PRINT_MOVE

            MOVE.L  #MOVEL_CODE,D5          * MOVE.L
            CMP.L   D5,D4
            BEQ     PRINT_MOVE

            MOVE.L  #MULS_CODE,D5           * MULS
            CMP.L   D5,D4
            BEQ     PRINT_MULS

            MOVE.L  #SUB_CODE,D5            * SUB
            CMP.L   D5,D4
            BEQ     PRINT_SUB

            MOVE.L  #SUBQ_CODE,D5           * SUBQ
            CMP.L   D5,D4
            BEQ     PRINT_SUBQ

            JSR     print_bad               * IF IT GETS HERE, NO OPCODE WAS RECOGNIZED

*****************************************************
* Method Name:      SHIFT
* Description:      Shifts machine code left by 1.
*                   If carry flag was set by this op,
*                   add a 1 to the current opcode.
*                   If carry flag was not set (0),
*                   add a 0 to the current opcode.
* Precondition:     D6 is initialized with opcode size.
*                   Long size machine code is in D2.
*                   D4 and D7 are initialized to 0.
* Postcondition:    D4 contains current opcode.
*                   D2 has had opcode shifted out,
*                   leaving the remainder behind.
*****************************************************

GET_BIT     CMP.B   D6,D7                   * Do we have the bits stored that we need?
            BEQ     SIZE_OPS                * If so, then check to see what size opcode
SHIFT       ASL.L   #1,D2                   * Shift machine code by 1
            BCC     ADD_0
            ASL.L   #1,D4                   * If not, shift opcode left by 1
            ADDI.L  #1,D4                   * Add 1 to opcode
            ADDI.B  #1,D7                   * Increment counter
            BRA     GET_BIT                 * Loop back to get next bit
```

```
ADD_0       ASL.L   #1,D4                   * Shift D4 left by 1
            ADDI.L  #1,D7                   * Increment counter
            BRA     GET_BIT                 * Loop back to get next bit

SIZE_OPS    MOVE.L  #0,D3 * Reset D3 so it's only set for MOVE ops
            CMPI.B  #16,D6
            BEQ     WORD_OPS
            CMPI.B  #10,D6
            BEQ     TENBIT_OPS
            CMPI.B  #8,D6
            BEQ     BYTE_OPS
            CMPI.B  #5,D6
            BEQ     FIVEBIT_OPS
            CMPI.B  #4,D6
            BEQ     NIBBLE_OPS
            CMPI.B  #3,D6
            BEQ     getThree_D
            CMPI.B  #2,D6
            BEQ     getTwo_D

*******************************************************
* Method Name:      getThree
* Description:      Gets the first three bits of the
*                   machine code in D2 and stores it
*                   in D4.
* Precondition:     D6 is initialized with 3.
*                   Machine code is in D2.
*                   D4 and D7 are initialized to 0.
* Postcondition:    D4 contains interested bits.
*                   D2 has whatever portion of the
*                   machine code is left after shift.
*******************************************************
getThree    MOVE.L  #0,D4                   * Initialize D4 to hold the three bits retrieved
            MOVE.L  #3,D6                   * Initialize D6 with 3
            MOVE.L  #0,D7                   * Initialize D7 for number of bits shifted
            BRA     SHIFT                   * Shift out three bits and store in D4
getThree_D  RTS

getTwo      MOVE.L  #0,D4                   * Initialize D4 to hold the three bits retrieved
            MOVE.L  #2,D6                   * Initialize D6 with 2
            MOVE.L  #0,D7                   * Initialize D7 for number of bits shifted
            BRA     SHIFT                   * Shift out three bits and store in D4
getTwo_D    RTS

*******************************************************
* ADD TO BUFFER
* Move the opcode portion to the good buffer,
* then branch to EA to decipher the rest.
*******************************************************

PRINT_NOP   MOVE.B  #'N',(A4)+              * NOP added to good buffer
            MOVE.B  #'O',(A4)+
            MOVE.B  #'P',(A4)+
            BRA     print_good

PRINT_RTS   MOVE.B  #'R',(A4)+              * RTS added to good buffer
            MOVE.B  #'T',(A4)+
            MOVE.B  #'S',(A4)+
            BRA     print_good

PRINT_ADD   MOVE.B  #'A',(A4)+              * ADD added to good buffer
            MOVE.B  #'D',(A4)+
            MOVE.B  #'D',(A4)+
            BRA     ADDA_OPM

PRINT_AS    MOVE.B  #'A',(A4)+              * AS added to good buffer
            MOVE.B  #'S',(A4)+
            BRA     AsLsRo_DIR

PRINT_LS    MOVE.B  #'L',(A4)+              * LS added to good buffer
```

```
            MOVE.B  #'S',(A4)+
            BRA     AsLsRo_DIR

PRINT_RO    MOVE.B  #'R',(A4)+              * RO added to good buffer
            MOVE.B  #'O',(A4)+
            BRA     AsLsRo_DIR

PRINT_BCS   MOVE.B  #'B',(A4)+              * BCS added to good buffer
            MOVE.B  #'C',(A4)+
            MOVE.B  #'S',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            BRA     print_good

PRINT_BGE   MOVE.B  #'B',(A4)+              * BGE added to good buffer
            MOVE.B  #'G',(A4)+
            MOVE.B  #'E',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            BRA     print_good

PRINT_BLT   MOVE.B  #'B',(A4)+              * BGE added to good buffer
            MOVE.B  #'L',(A4)+
            MOVE.B  #'T',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            BRA     print_good

PRINT_BVC   MOVE.B  #'B',(A4)+              * BVC added to good buffer
            MOVE.B  #'V',(A4)+
            MOVE.B  #'C',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            BRA     print_good

PRINT_CMP   MOVE.B  #'C',(A4)+              * CMP added to good buffer
            MOVE.B  #'M',(A4)+
            MOVE.B  #'P',(A4)+
            BRA     CMP_SIZE

PRINT_EOR   MOVE.B  #'E',(A4)+              * EOR added to good buffer
            MOVE.B  #'O',(A4)+
            MOVE.B  #'R',(A4)+
            BRA     EOR_SIZE

PRINT_DIVSW MOVE.B  #'D',(A4)+              * DIVS.W added to good buffer
            MOVE.B  #'I',(A4)+
            MOVE.B  #'V',(A4)+
            MOVE.B  #'S',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'W',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea             * call EA method
            BRA     print_good

PRINT_OR    MOVE.B  #'O',(A4)+              * OR added to good buffer
            MOVE.B  #'R',(A4)+
            BRA     OR_SIZE

PRINT_LEA   MOVE.B  #'L',(A4)+              * LEA added to good buffer
            MOVE.B  #'E',(A4)+
            MOVE.B  #'A',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea             * call EA method
            BRA     print_good

PRINT_MOVEB MOVE.B  #'M',(A4)+              * MOVE.B added to good buffer
            MOVE.B  #'O',(A4)+
            MOVE.B  #'V',(A4)+
            MOVE.B  #'E',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'B',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            MOVE.L  #1,D3
            JSR     effectivea
```

```
            BRA     print_good

PRINT_MOVE  MOVE.B  #'M',(A4)+              * MOVE added to good buffer
            MOVE.B  #'O',(A4)+
            MOVE.B  #'V',(A4)+
            MOVE.B  #'E',(A4)+
            BRA     MOVE_OPM

PRINT_MULS  MOVE.B  #'M',(A4)+              * MULS.W added to good buffer
            MOVE.B  #'U',(A4)+
            MOVE.B  #'L',(A4)+
            MOVE.B  #'S',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'W',(A4)+
            MOVE.B  #' ',(A4)+
            JSR     effectivea              * SPACE AFTER
            BRA     print_good
                        * call EA method

PRINT_SUB   MOVE.B  #'S',(A4)+              * SUB added to good buffer
            MOVE.B  #'U',(A4)+
            MOVE.B  #'B',(A4)+
            BRA     SUB_OPM

PRINT_SUBQ  MOVE.B  #'S',(A4)+              * SUBQ added to good buffer
            MOVE.B  #'U',(A4)+
            MOVE.B  #'B',(A4)+
            MOVE.B  #'Q',(A4)+
            BRA     SUBQ_OPM

PRINT_BRA   MOVE.B  #'B',(A4)+              * BRA  added to good buffer
            MOVE.B  #'R',(A4)+
            MOVE.B  #'A',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            BRA     print_good              * print good buffer

PRINT_CMPIB MOVE.B  #'C',(A4)+              * CMPI.B added to good buffer
            MOVE.B  #'M',(A4)+
            MOVE.B  #'P',(A4)+
            MOVE.B  #'I',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'B',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good
                        * call EA method

PRINT_CMPIW MOVE.B  #'C',(A4)+              * CMPI.W  added to good buffer
            MOVE.B  #'M',(A4)+
            MOVE.B  #'P',(A4)+
            MOVE.B  #'I',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'W',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good
                         * call EA method

PRINT_CMPIL MOVE.B  #'C',(A4)+              * CMPI.L  added to good buffer
            MOVE.B  #'M',(A4)+
            MOVE.B  #'P',(A4)+
            MOVE.B  #'I',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'L',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good
                        * call EA method

PRINT_ORIB  MOVE.B  #'O',(A4)+              * ORI.B added to good buffer
            MOVE.B  #'R',(A4)+
```

```
            MOVE.B  #'I',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'B',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good
                         * call EA method

PRINT_ORIW  MOVE.B  #'O',(A4)+              * ORI.W added to good buffer
            MOVE.B  #'R',(A4)+
            MOVE.B  #'I',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'W',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good
                         * call EA method

PRINT_ORIL  MOVE.B  #'O',(A4)+              * ORI.L added to good buffer
            MOVE.B  #'R',(A4)+
            MOVE.B  #'I',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'L',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good
                          * call EA method

PRINT_NEGB  MOVE.B  #'N',(A4)+              * NEG.B added to good buffer
            MOVE.B  #'E',(A4)+
            MOVE.B  #'G',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'B',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good

                          * call EA method

PRINT_NEGW  MOVE.B  #'N',(A4)+              * NEG.W added to good buffer
            MOVE.B  #'E',(A4)+
            MOVE.B  #'G',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'W',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good

                          * call EA method

PRINT_NEGL  MOVE.B  #'N',(A4)+              * NEG.L added to good buffer
            MOVE.B  #'E',(A4)+
            MOVE.B  #'G',(A4)+
            MOVE.B  #'.',(A4)+
            MOVE.B  #'L',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR      effectivea
            BRA     print_good

                         * call EA method

PRINT_JSR   MOVE.B  #'J',(A4)+              * JSR added to good buffer
            MOVE.B  #'S',(A4)+
            MOVE.B  #'R',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            MOVE.B  #' ',(A4)+
            MOVE.B  #' ',(A4)+
            BRA     JSR_ABS_L

PRINT_MOVEM MOVE.B  #'M',(A4)+              * MOVEM added to good buffer
            MOVE.B  #'O',(A4)+
```

```
            MOVE.B  #'V',(A4)+
            MOVE.B  #'E',(A4)+
            MOVE.B  #'M',(A4)+
            BRA     MOVEM_OPM

PRINT_BCLR  MOVE.B  #'B',(A4)+              * BCLR
            MOVE.B  #'C',(A4)+
            MOVE.B  #'L',(A4)+
            MOVE.B  #'R',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good

PRINT_B     MOVE.B  #'.',(A4)+              * .B added to good buffer
            MOVE.B  #'B',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good

PRINT_W     MOVE.B  #'.',(A4)+              * .W added to good buffer
            MOVE.B  #'W',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good

PRINT_L     MOVE.B  #'.',(A4)+              * .L added to good buffer
            MOVE.B  #'L',(A4)+
            MOVE.B  #' ',(A4)+              * SPACE AFTER
            JSR     effectivea
            BRA     print_good

APP_A       MOVE.B  #'A',(A4)+              * A added to good buffer
            RTS

APP_R       MOVE.B  #'R',(A4)+              * R added to good buffer
            RTS

APP_L       MOVE.B  #'L',(A4)+              * L added to good buffer
            RTS

            BRA     print_bad               * Opcode not recognized

*****************************************************
* DISTINGUISH OPMODES
*****************************************************
ADDA_OPM    ASL.L   #3,D2                   * Discard register bits
            JSR     getThree
            JSR     SIZE_CHK_1
            JSR     SIZE_CHK_2
            JSR     APP_A
            CMPI.B  #ADDAW_CODE,D4          * Is it ADDA.W?
            BEQ     PRINT_W
            CMPI.B  #ADDAL_CODE,D4          * Is it ADDA.L?
            BEQ     PRINT_L

AsLsRo_OPM  ASL.L   #7,D2                   * Discard irrelevant bits
            JSR     getTwo
            CMPI.B  #0,D4                   * Is it AS?
            BEQ     PRINT_AS
            CMPI.B  #%00000001,D4           * Is it LS?
            BEQ     PRINT_LS
            CMPI.B  #%00000011,D4           * Is it RO?
            BEQ     PRINT_RO

AsLsRo_DIR  MOVE.L  (A2),D2                 * Reload machine code
            ASL.L   #8,D2                   * Skip first 8 bits
            SCS     D3                      * Populate D3 with last bit shifted out
            JSR     DIR_CHK

CmpEor_OPM  ASL.L   #4,D2                   * Discard register bits
            BCS     EOR_OPM
```

```
            BCC     CMP_OPM

EOR_OPM     JSR     PRINT_EOR
EOR_SIZE    JSR     TWO_SIZE

CMP_OPM     JSR     PRINT_CMP
CMP_SIZE    JSR     TWO_SIZE

DivsOr_OPM  ASL.L   #3,D2                   * Discard register bits
            JSR     getThree
            CMPI.B  #%00000111,D4           * Is it a DIVS.W?
            BEQ     PRINT_DIVSW             * If so, print DIVS.W
            BNE     PRINT_OR                * If not, print OR
OR_SIZE     JSR     SIZE_CHK_1              * Then do both size checks
            JSR     SIZE_CHK_2


PRINT_MOVEA JSR     APP_A
            BRA     MOVE_SIZE
MOVE_OPM    ASL.L   #3,D2                   * Advance to destination mode
            JSR     getThree
            MOVE.L  #1,D3
            CMPI.B  #%00000001,D4           * Is it a MOVEA?
            BEQ     PRINT_MOVEA
            BRA     MOVE_SIZE
MOVE_SIZE   MOVE.L  (A2),D2                 * Reload machine code
            ASL.L   #2,D2                   * Discard opcode
            JSR     getTwo
            MOVE.L  #1,D3
            CMPI.B  #BYTE1_CODE,D4
            BEQ     PRINT_B
            CMPI.B  #%00000011,D4
            BEQ     PRINT_W
            CMPI.B  #LONG1_CODE,D4
            BEQ     PRINT_L

MOVEM_OPM   ASL.L   #5,D2
            MOVE.L  #1,D3
            BCS     PRINT_L
            BCC     PRINT_W

SUB_OPM     ASL.L   #3,D2
            JSR     getThree
            JSR     SIZE_CHK_1

SUBQ_OPM    ASL.L   #4,D2
            JSR     getTwo
            JSR     SIZE_CHK_1

*****************************************************
* Method name:      DIR_CHK
* Description:      Checks the direction and two
*                   successive bits for size.
* Preconditions:    D3 has the carry bit saved that
*                   was set by shifting out the
*                   direction bit from the machine code.
*****************************************************

DIR_CHK     CMPI.B  #0,D3                   * Is it a shift R?
            BEQ     RIGHT_SHIFT
            BNE     LEFT_SHIFT
RIGHT_SHIFT JSR     APP_R                   * If so, append an R
            BRA     TWO_SIZE
LEFT_SHIFT  JSR     APP_L                   * If not, append an L
            BRA     TWO_SIZE
TWO_SIZE    JSR     getTwo
            JSR     SIZE_CHK_1
            JSR     SIZE_CHK_2
            RTS

*****************************************************
```

```
* Method name:      SIZE_CHK_1
* Description:      Appends .B if size is 000
*                   .W if size is 001, .L if size
*                   is 010
* Preconditions:    D4 stores the size bits
* Postcondition:    Correct size is printed, or
*                   RTS if no size match
**************************************************

SIZE_CHK_1  CMPI.B  #BYTE1_CODE,D4          * Is it .B?
            BEQ     PRINT_B
            CMPI.B  #WORD1_CODE,D4          * Is it .W?
            BEQ     PRINT_W
            CMPI.B  #LONG1_CODE,D4          * Is it .L?
            BEQ     PRINT_L

            RTS

**************************************************
* Method name:      SIZE_CHK_2
* Description:      Appends .B if size is 100
*                   .W if size is 101, .L if size
*                   is 110
* Preconditions:    D4 stores the size bits
* Postcondition:    Correct size is printed, or
*                   RTS if no size match
**************************************************

SIZE_CHK_2  CMPI.B  #BYTE2_CODE,D4          * Is it .B?
            BEQ     PRINT_B
            CMPI.B  #WORD2_CODE,D4          * Is it .W?
            BEQ     PRINT_W
            CMPI.B  #LONG2_CODE,D4          * Is it .L?
            BEQ     PRINT_L
            RTS

*-----------------------------------------------------------------------------------------
* EFFECTIVE ADDRESSING
* Determines EA Type, EA to 'good' buffer, increments, and saves in D4
* Checks D3=1 to see if it's a MOVE opcode (1 = yes, move, 0 = not move)
* Looks for EA information in XXX register/xxx address

*A2 Starting Address
*A3 Ending Address
*A4 Good Buffer (the string we will display if this is a recognized 68k instruction)
*A5 Bad Buffer (the string we will display if its not recognized)

*D6 Remaining word count
*D7 Current instruction

* Required Modes:
* Data Register Direct: Dn, 000
* Address Register Direct: An, 001
* Address Register Indirect: (An)
* Immediate Addressing: #<xxx>, Mode field: 111, Reg. field: 100
* Address Register Indirect with Post incrementing: (An)+, 011
* Address Register Indirect with Pre decrementing: -(An), 100
* Absolute Long Address: xxx.L, Mode field: 111, Reg. field: 000
* Absolute Word Address: xxx.W, Mode field: 111, Reg. field: 000
*-----------------------------------------------------------------------------------------

effectivea  *//Callee save reg
            MOVEM.L  Dreg_list,-(SP)
            MOVE.L #0,D2 * Set registers to empty
            MOVE.L #0,D5


            MOVE.W  (A2),D5
            LSL.W   #8,D5
            LSR.W   #8,D5
            LSL.B   #2,D5   *Bye bye most significant 2 bits
```

```
            LSR.B   #5,D5   *Bits 3 to 5

            MOVE.W  (A2),D2
            LSL.W   #8,D2
            LSR.W   #8,D2
            LSL.B   #5,D2
            LSR.B   #5,D2   *Bits 0 to 2
            CMP.B   #1,D3  *Checks D3=1
            BEQ     eamove
            CMP.B   #0,D3  *Checks D3=0
            BEQ     eall    *Branches to other EA instructions

            *//Callee restore reg
            MOVEM.L  (SP)+,Dreg_list
            RTS

eall
            MULU    #6,D5
            LEA     mode,A0 *Loads EA mode table
            JSR     hexToAscii

            MOVEM.L  (SP)+,Dreg_list
            RTS

*-----------------------------------------------
* Effective Addressing MOVE
*
* Source to buffer
* Destination to buffer
*-----------------------------------------------
eamove

            MULU    #6,D5
            LEA     mode,A0 *Loads EA mode table
            JSR     00(A0,D5)
            MOVE.B  #',',(A4)+

            MOVE.W  (A2),D2
            LSL.W   #4,D2
            LSR.W   #5,D2
            LSR.W   #8,D2

            MOVE.W  (A2),D5
            LSL.W   #7,D5
            LSR.W   #5,D5
            LSR.W   #8,D5

            MULU    #6,D5
            LEA     mode,A0
            JSR     hexToAscii

            *//Callee restore reg
            MOVEM.L  (SP)+,Dreg_list
            RTS

*-----------------------------------------------
* Effective Addressing MODEs
*
* EA Mode Table
* Put each EA mode into buffer
*-----------------------------------------------
mode    JMP mode000     *Dn
        JMP mode001     *An
        JMP mode010     *(An)
        JMP mode011     *(An)+
        JMP mode100     *-(An)
        JMP mode111     * .W or .L

*-----------------------------------------------
* EA mode 000
* Register Direct
```

```
* Data, Dn
*-------------------------------------------------
mode000     MOVE.B  #'D',(A4)+  *D is for Data Register
            MULU    #6,D2
            LEA     register_mode,A1 *Loads the register table
            JSR     hexToAscii
            RTS

*-------------------------------------------------
* EA mode 001
* Register Direct
* Address, An
*-------------------------------------------------
mode001     MOVE.B  #'A',(A4)+
            MULU    #6,D2
            LEA     register_mode,A1 *Loads the register table
            JSR     hexToAscii
            RTS

*-------------------------------------------------
* EA mode 010
* Register Indirect
* Address, (An)
*-------------------------------------------------
mode010     MOVE.B  #'(',(A4)+
            MOVE.B  #'A',(A4)+
            MULU    #6,D2
            LEA     register_mode,A1 *Loads the register table
            JSR     hexToAscii
            MOVE.B  #')',(A4)+
            RTS

*-------------------------------------------------
* EA mode 011
* Register Indirect
* Address with Postincrement, (An)+
*-------------------------------------------------
mode011     MOVE.B  #'(',(A4)+
            MOVE.B  #'A',(A4)+
            MULU    #6,D2
            LEA     register_mode,A1 *Loads the register table
            JSR     hexToAscii
            MOVE.B  #')',(A4)+
            MOVE.B  #'+',(A4)+
            RTS

*-------------------------------------------------
* EA mode 100
* Register Indirect
* Address with Predecrement, -(An)
*-------------------------------------------------
mode100     MOVE.B  #'-',(A4)+
            MOVE.B  #'(',(A4)+
            MOVE.B  #'A',(A4)+
            MULU    #6,D2
            LEA     register_mode,A1 *Loads the register table
            JSR     hexToAscii
            MOVE.B  #')',(A4)+
            RTS

*-------------------------------------------------
* EA mode 111
* Absolute Data Addressing
* Short, (xxx).W
* Long, (xxx).L
*
* Immediate, #<xxx>
*-------------------------------------------------
mode111     MULU    #6,D2
            LEA     absolute_mode,A1 *Loads the register table
            JSR     hexToAscii
```

```
            RTS

*-----------------------------------------------
* Effective Addressing Register MODEs
* Registers 0, 1, 2, 3, 4, & 7
*
* Each mode below puts the register number in the good buff
* Increments pointer
*-----------------------------------------------
register_mode    JMP registermode000
                 JMP registermode001
                 JMP registermode010
                 JMP registermode011
                 JMP registermode100
                 JMP registermode111

*-----------------------------------------------
* EA register mode 000
*-----------------------------------------------
registermode000 MOVE.B  #'0',(A4)+
                RTS

*-----------------------------------------------
* EA register mode 001
*-----------------------------------------------
registermode001 MOVE.B  #'1',(A4)+
                RTS

*-----------------------------------------------
* EA register mode 010
*-----------------------------------------------
registermode010 MOVE.B  #'2',(A4)+
                RTS

*-----------------------------------------------
* EA register mode 011
*-----------------------------------------------
registermode011 MOVE.B  #'3',(A4)+
                RTS

*-----------------------------------------------
* EA register mode 100
*-----------------------------------------------
registermode100 MOVE.B  #'4',(A4)+
                RTS

*-----------------------------------------------
* EA register mode 111
*-----------------------------------------------
registermode111 MOVE.B  #'7',(A4)+
                RTS

*-----------------------------------------------
* Effective Addressing Adsolute MODEs
* Absolute mode tables will jump to mode
* Then loads absolute addressing into the buffer for each
* Absolute Long Address: xxx.L, Mode field: 111, Reg. field: 000
* Absolute Word Address: xxx.W, Mode field: 111, Reg. field: 000
*-----------------------------------------------
absolute_mode    JMP absolutemode000
                 JMP absolutemode001
                 JMP absolutemode010
                 JMP absolutemode011
                 JMP absolutemode100
                 JMP absolutemode111


*-----------------------------------------------
* EA absolute mode 000
* Short, (xxx).W
* Loads into buff
```

```
* Plus a lot of shifting!

*---------This May be Wrong---------
*-------------------------------------------------
absolutemode000 MOVE.B  #'$',(A4)+
                MOVEA.L A2,A6
                MOVE.W  (A6)+,D5
                MOVE.L  (A6),D5
                CMPI.L  #1,D3   * Checks D6=0
                BEQ     print_bad

                SUBI.L  #1,D3  * Decrement
                LSR.W   #8,D5   * Shift 8 right
                LSR.W   #4,D5   * Shift 4 right
                JSR     hexToAscii

                MOVE.L  (A6),D5
                LSL.W   #4,D5   * Shift 4 left
                LSR.W   #8,D5   * Shift 8 back
                LSR.W   #4,D5   * Shift 4 back
                JSR     hexToAscii



                MOVE.L  (A6),D5
                LSL.W   #8,D5
                LSR.W   #8,D5
                LSR.W   #4,D5
                JSR     hexToAscii

                MOVE.L  (A6),D5
                LSL.W   #8,D5   * Shift 8 left
                LSL.W   #4,D5   * Shift 4 left
                LSR.W   #8,D5   * Shift 8 back
                LSR.W   #4,D5   * Shift 4 back
                JSR     hexToAscii

                MOVE.L  #1,D1

                RTS

*-------------------------------------------------
* EA absolute mode 001
* Long, (xxx).L
* Loads into buff
* Same as above, but for xxx.L
*-------------------------------------------------
absolutemode001 MOVE.B  #'$',(A4)+
                MOVE.W  (A2),D0
                CMPI.L  #1,D3   * Checks D6=0
                BEQ     print_bad

                SUBI.L  #2,D3   * Decrement
                LSR.W   #8,D0   * Shift 8 right
                LSR.W   #4,D0   * Shift 4 right
                JSR     hexToAscii

                MOVE.W  (A2),D0
                LSL.W   #4,D0   * Shift 4 left
                LSR.W   #8,D0   * Shift 8 back
                LSR.W   #4,D0   * Shift 4 back
                JSR     hexToAscii



                MOVE.W  (A2),D0
                LSL.W   #8,D0
                LSR.W   #8,D0
                LSR.W   #4,D0
                JSR     hexToAscii
```

```
                MOVE.W  (A2),D0
                LSL.W   #8,D0   * Shift 8 left
                LSL.W   #4,D0   * Shift 4 left
                LSR.W   #8,D0   * Shift 8 back
                LSR.W   #4,D0   * Shift 4 back
                JSR     hexToAscii

                MOVE.L  (A2)+,D5
                MOVE.L  #1,D1


                ADD.L   #2,A5  * Increments for the next address
                RTS


*-----------------------------------------------
* EA absolute mode 010
* unrecognized
*-----------------------------------------------
absolutemode010
                RTS *temp


*-----------------------------------------------
* EA absolute mode 011
* unrecognized
*-----------------------------------------------
absolutemode011
                RTS *temp


*-----------------------------------------------
* EA absolute mode 100
* Decodes Immediate Address
*-----------------------------------------------
absolutemode100 MOVE.B  #'#',(A4)+
                CMPI.L  #0,D6
                BEQ     print_bad

                JSR     hexToAscii

                SUBI.L  #1,D6
                ADD.L   #2,A5
                RTS


*-----------------------------------------------
* EA absolute mode 111
* unrecognized
*-----------------------------------------------
absolutemode111
                RTS *temp

*-----------------------------------------------
* EA For JSR
* unrecognized
* Mustafa Added
*-----------------------------------------------
JSR_ABS_L
    MOVE.W  (A2)+,D5

    MOVE.B  #$30,(A4)+
    MOVE.B  #$30,(A4)+

    *//most sig char
    MOVE.L  (A2),D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//next char
    MOVE.L  (A2),D5
```

```
    LSL.L   #4,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//next char
    MOVE.L  (A2),D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//next char
    MOVE.L  (A2),D5
    LSL.L   #4,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//next char
    MOVE.L  (A2),D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *//last char
    MOVE.L  (A2),D5
    LSL.L   #4,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    LSL.L   #8,D5
    JSR     hexToAscii

    *MOVE.L  (A2)+,D5
    MOVE.L  #1,D1
    BRA     print_good




*********************************END Amy's************************************

********* GIVEN ****************************
*******************************************************************************
* Method Name: TrapTask13
* Description: Creates a file if none exists, and appends bytes to that file
*   while also echoing the written bytes to the screen.  You shouldn't need to
*   change this code.
*
* Calling Convention: Callee-Saved
*
* Preconditions & Method Input:
*   A1 points to the null-terminated buffer to write (newline will be added for you)
*
* Postconditions & Output:
*   ALL files that were previously open will be CLOSED (FileIDs will be invalid)
*   See 'Output.txt' in directory for the results, also piped to the console
*
*
*  A2 holds a pointer to null terminated string to write (input)
*  A3 points to the null-terminated file name
*  D3 holds the number of bytes already in the file to write
*
*  D5 holds number of bytes to write
*******************************************************************************
toSave REG D0-D5/A2-A3
TrapTask13:
    ********************************************************************
    * Method initialization, regsiter spilling, parameter saving, etc.
    ********************************************************************
    MOVEM.L toSave, -(SP)   ;  Callee-Saved, so save and restore
```

```
    MOVEA.L A1, A2 ; save this buffer to write
    LEA outFilename, A3  ; save this for later, too

    move #50,d0
    trap #15 ; close all files, suggested to begin any IO
    ****************************************************************
    * End Method Init
    ****************************************************************

    ********************************************************************************************
    * Calculate the number of bytes to write by searching for the null in the target buffer A0
    ********************************************************************************************
    CLR.L D5 *D5 is now the number of bytes to write
nullLoop:
    MOVE.B (A1)+, D0
    CMPI.B #0,D0  * compare to null
    BEQ findNullLoopDone
    ADDI.W #1, D5
    BRA nullLoop

findNullLoopDone:
    MOVEA.L A3, A1 * reset A1 so it points to the file to write to (to open, next)

    ;check if file exists, and open with task 51 if so, otherwise 52
    ;(precondition here is A1 points to the null-terminated filename )
    MOVE.B #51, D0      ;open file (task 51 is existing, 52 is new)
    trap #15

    if.w D0 <NE> #0 then.s         ; if file error (404, not found)
        MOVE.B #52, D0             ; open new file (52 is new)
        trap #15
    endi


********************************************************************************************
********
    * Seek to END of FILE by counting the number of bytes, closing, reopening, then seeking.
    *    (first, count number of bytes already in the file to obtain seek position)

********************************************************************************************
********
    Clr.L D3   ;TODO: reg save, D3 is now our count of bytes read
    MOVE.L #1, D2 ; read one byte at a time
    LEA byteRead, A1

countLoop:
    MOVE.B #53, D0 ; try to read one byte (TODO: move out of loop)
    trap #15

    CMPI.W #1,D0  ;1 == EOF
    BEQ countDone
    ADDI #1, D3
    BRA countLoop

countDone:
    * close this file
     move #56,d0
     trap #15

     * reopen the target file
     MOVE.L A3,A1
     MOVE #51, D0
     trap #15

    * seek to right position, then continue with writing
    MOVE.L D3, D2 ; move the number of bytes found in the file to D2
    MOVE #55, D0  ; position file task
    trap #15

    ********************************************************************************
```

```
    * Actually write the buffer to the file, after caculating the number of bytes
    *  to write and after seeking to the right location in the file for append
    ********************************************************************************

    MOVE.L D5, D2 ; restore this for the actually writing the buffer
    ; assumes A0 hasnt changed since handed to this method
    MOVEA.L A2, A1 ; load the address of the buffer we want to write to disk
    ; assumes file ID is still stored in D1.L
    MOVE.B #54, D0 ; subtask 54 is write to open file (append, or?), assumes D2 holds # of bytes
    trap #15

    ; add a newline to the file output
    LEA NEWLINE, A1
    MOVE.B #54, D0
    MOVE.B #2,D2  ; kills # of bytes to write from input param
    trap #15

    ; finally, close only this file
    MOVE.B #56, D0 ; close file task
    trap #15

    ; report to screen
    MOVEA.L A2, A1 ; load the address of the buffer we want to write to disk & screen
    MOVE.B #13, D0
    trap #15

    ; restore context
    MOVEM.L (SP)+, toSave

    RTS


*-------------------------------------------------------------------------------------
* Method Name: AsciiToHex
* Written by : Berger, Modified by Nash
* Date       : 3/1/2019
* Description: Converts chars '0'-'9' and 'a'-'f' to 0-9,a-F
*              Transforms/unpacks 8 chars (8b each) pointed to by A1 into
*              its (4b each) equivalent hex value
*
*  Preconditions & Input
*       A1 (input) points to a memory buffer holding 8 ascii chars (not null-terminated)
*       This function calls another function (strip_ascii)
*
*  Postconditions & Output
*       D7 (output) holds the converted value
*       Caller-Saved : D0 is temp, D6 is a loop var
*-------------------------------------------------------------------------------------
AsciiToHexRegList REG D0,D6
AsciiToHex
    MOVEM.L asciiToHexRegList, -(SP)  *save context
    CLR.L D7 * clear our return value
    MOVE.L #8, D6 ; and set up our loop counter

chrLoop
    MOVE.B (A1)+,D0 * Get the first byte
    jsr strip_ascii * Get rid of the ascii code
    OR.W D0,D7 * Load the bits into D7

    subI.B #1,D6  *decrement our loop variable
    BEQ chrDone   *skip shifting if we are done

    ASL.L #4,D7 * shift left 4 bits to prepare for next byte
    BRA chrLoop

chrDone
    MOVEM.L (SP)+,asciiToHexRegList
    RTS


*****************************************************************
```

```
* SUBROUTINE: strip_ascii
* remove the ascii code from the digits 0-9,a-f, or A-F
* Input Parameters: <D0> = ascii code
*
* Return parameters: D0.B = number 0...F, returned as 00...0F
* Registers used internally: D0
* Assumptions: D0 contains $30-$39, $41-$46 or $61-66
*
*****************************************************************
strip_ascii
      CMP.B #$39,D0 * Is it in range of 0-9?
      BLE sub30 * Its a number
      CMP.B #$46,D0 * Is is A...F?
      BLE sub37 * Its A...F
      SUB.B #$57,D0 * Its a...f
      BRA ret_sa * Go back
sub37 SUB.B #$37,D0 * Strip 37
      BRA ret_sa * Go back
sub30 SUB.B #$30,D0 * Strip 30
ret_sa RTS * Go back


* Required variables and constants go here for your Disassembler
CR                  EQU $0D
LF                  EQU $0A
NEWLINE             DC.B CR,LF,0
begMSG1             DC.B 'Welcome to CSS 422 Disassembler Final Project',CR,LF,0
begMSG2             DC.B 'The following program was created by Amy Meyers, Mariah
Files',CR,LF,'and Mustafa Majeed',CR,LF,0
inputFileName       DC.B 'Config.cfg',0
outFilename         DC.B 'Output.txt',0
begMSG3             DC.B 'Reading a starting address and and an ending address from :
',CR,LF,'Config.cfg',CR,LF,0
MSG4                DC.B 'Starting address read = ',0
MSG5                DC.B 'Ending address read = ',0
MSG6                DC.B CR,LF,'Disassembled code is as follows : ',CR,LF,0
inFileERR           DC.B 'INPUT FILE NOT PROPERLY FORMATTED',CR,LF,'PLEASE TRY AGAIN',CR,LF,0
inval               DC.B 'ERR::: STARTING ADDRESS IS LARGER THAN ENDING ADDRESS',CR,LF,'PLEASE
FORMAT FILE CORRECTLY',CR,LF,0
emptyF              DC.B 'ERR::: THE FILE THAT WAS READ WAS EMPTY, PLEASE TRY AGAIN',CR,LF,0
oddS                DC.B 'ERR::: STARTING ADDRESS IS ODD, PLEAST TRY AGAIN',CR,LF,0
oddE                DC.B 'ERR::: ENDING ADDRESS IS ODD, PLEAST TRY AGAIN',CR,LF,0
inputFileBuf        DS.B 80
fileSize            DC.B 80
GBuf                DS.B 80
BBuf                DS.B 80
startingBuf         DS.B 10
endingBuf           DS.B 10
byteRead            DS.B 1

   END    START        ; last line of source
```