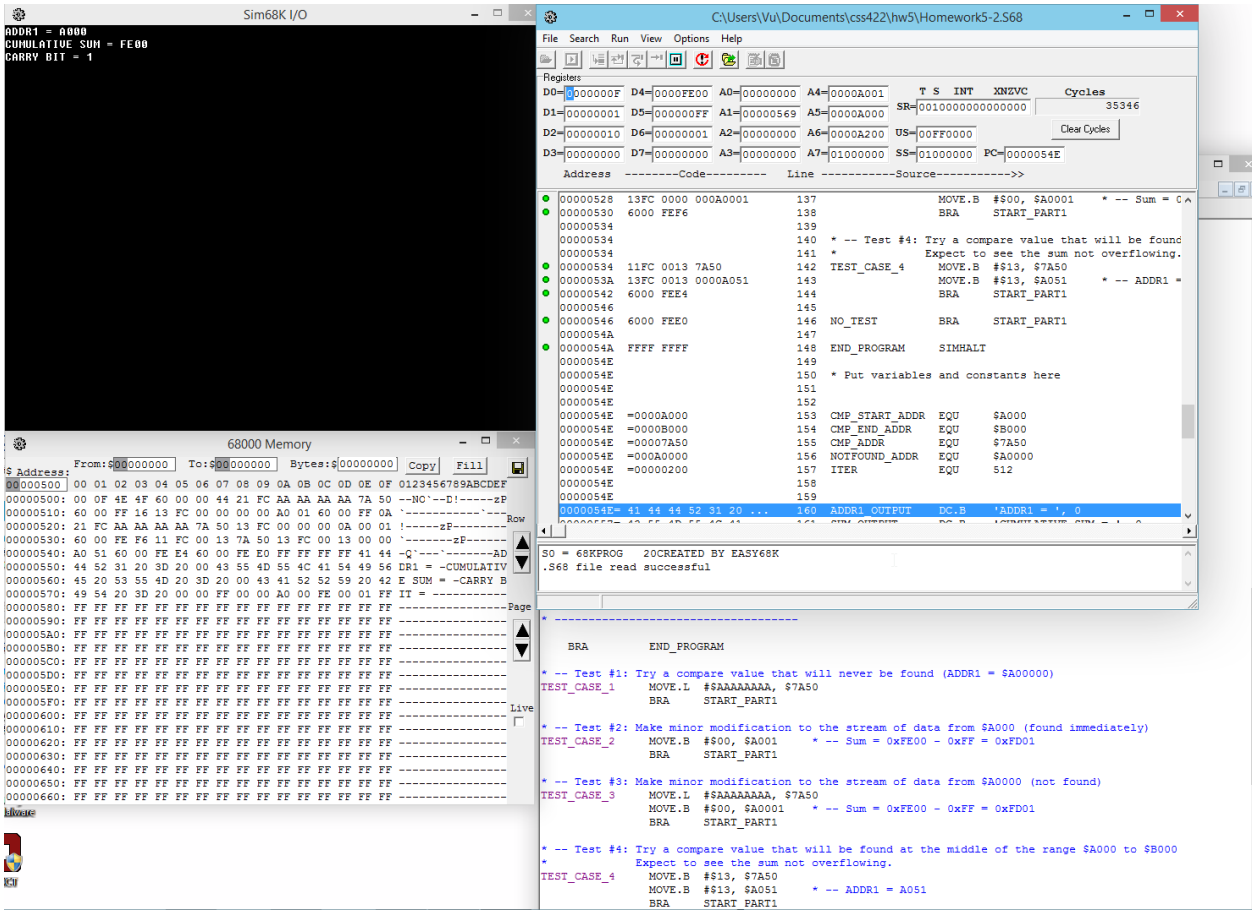
Homework 5

Vu Dinh

Part 1. Bit Shifting, Bit Rotating, and CCR bits

1. ASL.B #2, #$C1A8D372:
   Result: C1A8D3**C8**
   CCR: 11011
2. ASL.L #5, #$C1A8D372:
   Result: **351A6E40**
   CCR: 00010
3. LSR.B #4, #$C1A8D372:
   Result: C1A8D3**07**
   CCR: 00000
4. ROR.W #2, #$C1A8D372:
   Result: C1A8**B4DC**
   CCR: 01001
5. ROL.L #7, #$C1A8D372:
   Result: **D469B960**
   CCR: 01000

Part 2. Pattern matching and cumulative sum



See source (full resolution) at: http://i.imgur.com/zHAiin5.png

Source code:

```
*-----------------------------------------------------------------------
* Title      :      Homework 5 part 2 // Pattern finding and Cumulative sum
* Written by :      Vu Dinh
* Date       :      Nov 28 2014
* Comment    :      There are a lot of vague specifications of the program.
*                   I've created some test cases and stated expected
results.
*-----------------------------------------------------------------------



    ORG    $400
START:                  ; first instruction of program

* ------------------------------------
* --------- PART 3: TESTING ----------
* ------------------------------------

TEST_CASE   MOVE.B  #$0, D7     * -- test case number
            CMP.B   #$1, D7
            BEQ     TEST_CASE_1
            CMP.B   #$2, D7
            BEQ     TEST_CASE_2
            CMP.B   #$3, D7
            BEQ     TEST_CASE_3
            CMP.B   #$4, D7
            BEQ     TEST_CASE_4
DEFAULT     BRA     NO_TEST
```

```
* ------------------------------------
* --- PART 1: PATTERN MATCHING -------
* ------------------------------------
* | Tested for the following cases:  |
* | >> Pattern found immediately at  |
* |    $A000                         |
* | >> Pattern not found             |
* | >> Pattern found between $A000   |
* |    and $B000                     |
* ------------------------------------


START_PART1 CLR     D1

            MOVEA.L     #CMP_START_ADDR, A4         * -- starting search
location
LOOP        CMPA.L      #CMP_END_ADDR, A4           * -- ending search
location
            BEQ         NOT_FOUND
            MOVEA.L     A4, A5                      * -- keep a copy before
incrementing A4

COMPARE     MOVE.B      (A4)+, D4                   * -- currently
processed data
            CMP.B       CMP_ADDR, D4
            BEQ         FOUND
ENDLOOP     BRA         LOOP
```

```
FOUND       MOVE.L      A5, ADDR1                   * -- match is found
addr1 = matched location
            BRA         CSUM

NOT_FOUND   MOVE.L      #NOTFOUND_ADDR, ADDR1        * -- match not found,
addr1 = special value
            BRA         CSUM



* ------------------------------------
* --- PART 2: CUMULATIVE SUM ---------
* ------------------------------------
* | Tested for the following cases:  |
* | >> Change one summand to $00     |
* ------------------------------------

CSUM    CLR         D4
        MOVEA.L     #$0, A3                 * -- resetting A3
        MOVE.W      #ITER, D2               * -- D2 = loop counter
        MOVEA.L     ADDR1, A6

SUM_LOOP    CMP.L       #0, D2
            BEQ         SUM_FINISH
            MOVE.B      (A6)+, D5           * -- data is extracted as BYTE
            ADD.W       D5, D4              * -- but summed as WORD
            BCS         SET_CARRY           * -- if the carry bit is set,
record it
```

```
LOOPBACK    SUB.L       #1, D2
            BRA         SUM_LOOP

SET_CARRY   MOVE.B      #1, D6              * -- record carry bit
            BRA         LOOPBACK            * -- going back into loop

SUM_FINISH  MOVE.W  D4, ADDSUM              * -- move the sum to requested
location
            MOVE.B  D6, CARRYBIT            * -- move the carry bit to the
requested location
            BRA     PRINTRESULT

PRINTRESULT LEA     ADDR1_OUTPUT, A1
            MOVE.B  #14, D0
            TRAP    #15
            MOVE.L  ADDR1, D1           * -- printing ADDR1
            MOVE.B  #16,D2
            MOVE.B  #15,D0
            TRAP    #15

            LEA     EMPTY_LINE, A1      * -- printing empty line
            MOVE.B  #13,D0
            TRAP    #15

            LEA     SUM_OUTPUT, A1
            MOVE.B  #14, D0
            TRAP    #15
```

```
            MOVE.L  D4, D1              * -- printing the sum (stored
temporarily in D4)
            MOVE.B  #16,D2              * -- the sum is printed in hex
format
            MOVE.B  #15,D0
            TRAP    #15

            LEA     EMPTY_LINE, A1      * -- empty line
            MOVE.B  #13,D0
            TRAP    #15

            LEA     CARRY_OUTPUT, A1
            MOVE.B  #14, D0
            TRAP    #15
            MOVE.L  D6, D1              * -- the carry bit information
(overwrites D1)
            MOVE.B  #16,D2
            MOVE.B  #15,D0
            TRAP    #15



* -----------------------------------
* --------- PART 3: TESTING ----------
* -----------------------------------

    BRA         END_PROGRAM

* -- Test #1: Try a compare value that will never be found (ADDR1 =
$A00000)
```

```
TEST_CASE_1     MOVE.L  #$AAAAAAAA, $7A50
                BRA     START_PART1

* -- Test #2: Make minor modification to the stream of data from $A000
(found immediately)
TEST_CASE_2     MOVE.B  #$00, $A001     * -- Sum = 0xFE00 - 0xFF = 0xFD01
                BRA     START_PART1

* -- Test #3: Make minor modification to the stream of data from $A0000
(not found)
TEST_CASE_3     MOVE.L  #$AAAAAAAA, $7A50
                MOVE.B  #$00, $A0001    * -- Sum = 0xFE00 - 0xFF = 0xFD01
                BRA     START_PART1

* -- Test #4: Try a compare value that will be found at the middle of the
range $A000 to $B000
*             Expect to see the sum not overflowing.
TEST_CASE_4     MOVE.B  #$13, $7A50
                MOVE.B  #$13, $A051     * -- ADDR1 = A051
                BRA     START_PART1

NO_TEST         BRA     START_PART1

END_PROGRAM     SIMHALT

* Put variables and constants here
```

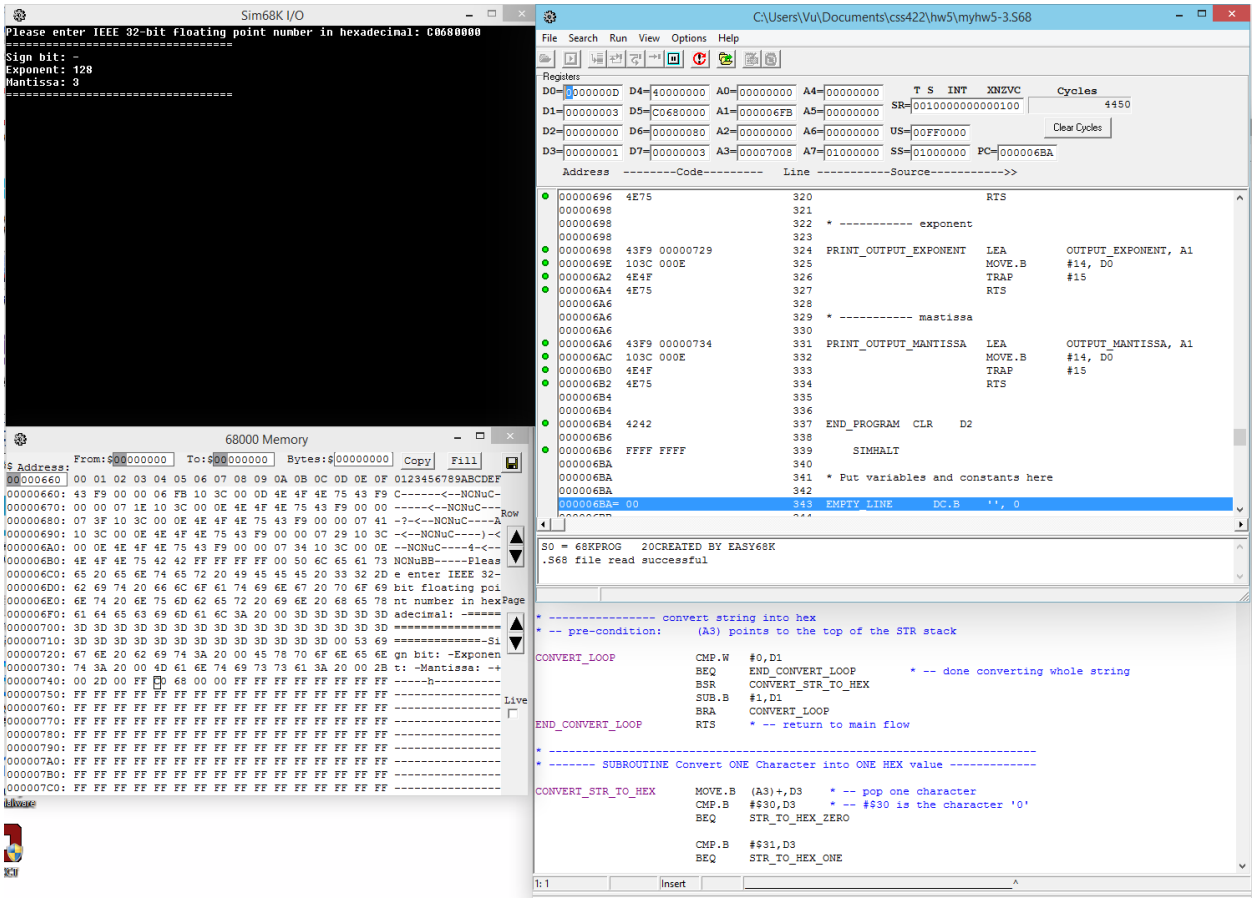
```
CMP_START_ADDR  EQU     $A000
CMP_END_ADDR    EQU     $B000
CMP_ADDR        EQU     $7A50
NOTFOUND_ADDR   EQU     $A0000
ITER            EQU     512


ADDR1_OUTPUT    DC.B    'ADDR1 = ', 0
SUM_OUTPUT      DC.B    'CUMULATIVE SUM = ', 0
CARRY_OUTPUT    DC.B    'CARRY BIT = ', 0
EMPTY_LINE      DC.B    '', 0

ADDR1           DS.L    $1
ADDSUM          DS.W    $1
CARRYBIT        DS.B    $1

    END    START        ; last line of source
```

Part 3. Floating point number decoding



See source (full resolution) at: http://i.imgur.com/RVTitg4.png

Source code:

```
*-----------------------------------------------------------------------
--
* Title      :      Homework 5 part 2 // Pattern finding and Cumulative sum
* Written by :      Vu Dinh
* Date       :      Nov 28 2014
* Comment    :      Not thoroughly tested.
*-----------------------------------------------------------------------
--
    ORG    $400
START:                  ; first instruction of program


    BSR     PRINT_INPUT_PROMPT

*************************************************************************
*********

* ------------------------------------------------------
* ---------------------- SUBROUTINE --------------------
* ------------------------------------------------------
* | I borrowed this String --> Hex conversion subroutine |
* | from the disassembler project.                       |
* |     post-condition: The converted hex location is    |
* |                     stored in D4.                    |
* ------------------------------------------------------


CONVERT_START_LOCATION  CLR     D1
```

```
                        MOVEA.L #$7000,A1
                        MOVE.B  #2,D0   * --- read NULL-terminated string
                        TRAP    #15     * --- read string into (A1)
                        MOVEA.L A1,A3   * --- make a copy to preserve
original input
END_CONVERT_START_LOCATION  BSR     CONVERT_LOOP
                            BRA     CONVERT_DONE

* ---------------- convert string into hex
* -- pre-condition:     (A3) points to the top of the STR stack

CONVERT_LOOP            CMP.W   #0,D1
                        BEQ     END_CONVERT_LOOP        * -- done
converting whole string
                        BSR     CONVERT_STR_TO_HEX
                        SUB.B   #1,D1
                        BRA     CONVERT_LOOP
END_CONVERT_LOOP        RTS     * -- return to main flow

* ------------------------------------------------------------------------
* ------- SUBROUTINE Convert ONE Character into ONE HEX value -------------

CONVERT_STR_TO_HEX      MOVE.B  (A3)+,D3    * -- pop one character
                        CMP.B   #$30,D3     * -- #$30 is the character '0'
                        BEQ     STR_TO_HEX_ZERO

                        CMP.B   #$31,D3
                        BEQ     STR_TO_HEX_ONE
```

```
                    CMP.B   #$32,D3
                    BEQ     STR_TO_HEX_TWO

                    CMP.B   #$33,D3
                    BEQ     STR_TO_HEX_THREE

                    CMP.B   #$34,D3
                    BEQ     STR_TO_HEX_FOUR

                    CMP.B   #$35,D3
                    BEQ     STR_TO_HEX_FIVE

                    CMP.B   #$36,D3
                    BEQ     STR_TO_HEX_SIX

                    CMP.B   #$37,D3
                    BEQ     STR_TO_HEX_SEVEN

                    CMP.B   #$38,D3
                    BEQ     STR_TO_HEX_EIGHT

                    CMP.B   #$39,D3            * -- #$39 is the
character '9'
                    BEQ     STR_TO_HEX_NINE

                    CMP.B   #$41,D3            * -- #$41 is the
character 'A'
```

```
                        BEQ     STR_TO_HEX_A
                        CMP.B   #$61,D3             * -- #$61 is the
character 'a'
                        BEQ     STR_TO_HEX_A

                        CMP.B   #$42,D3
                        BEQ     STR_TO_HEX_B
                        CMP.B   #$62,D3
                        BEQ     STR_TO_HEX_B

                        CMP.B   #$43,D3
                        BEQ     STR_TO_HEX_C
                        CMP.B   #$63,D3
                        BEQ     STR_TO_HEX_C

                        CMP.B   #$44,D3
                        BEQ     STR_TO_HEX_D
                        CMP.B   #$64,D3
                        BEQ     STR_TO_HEX_D

                        CMP.B   #$45,D3
                        BEQ     STR_TO_HEX_E
                        CMP.B   #$65,D3
                        BEQ     STR_TO_HEX_E

                        CMP.B   #$46,D3
                        BEQ     STR_TO_HEX_F
                        CMP.B   #$66,D3
```

```
                        BEQ     STR_TO_HEX_F

                        BRA     INVALID_CHARACTER

* --------------- Conversion definitions ------------

INVALID_CHARACTER       NOP             * -- skip invalid character
                        RTS
STR_TO_HEX_ZERO         MOVE.L  #$0,D3   * -- push HEX 0 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_ONE          MOVE.L  #$1,D3   * -- push HEX 1 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_TWO          MOVE.L  #$2,D3   * -- push HEX 2 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_THREE        MOVE.L  #$3,D3   * -- push HEX 3 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_FOUR         MOVE.L  #$4,D3   * -- push HEX 4 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
```

```
                        RTS
STR_TO_HEX_FIVE         MOVE.L  #$5,D3   * -- push HEX 5 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_SIX          MOVE.L  #$6,D3   * -- push HEX 6 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_SEVEN        MOVE.L  #$7,D3   * -- push HEX 7 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_EIGHT        MOVE.L  #$8,D3   * -- push HEX 8 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_NINE         MOVE.L  #$9,D3   * -- push HEX 9 into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_A            MOVE.L  #$A,D3   * -- push HEX A into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_B            MOVE.L  #$B,D3   * -- push HEX B into HEX stack
                        BSR     SHIFT_START_ADDR
```

```
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_C            MOVE.L  #$C,D3   * -- push HEX C into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_D            MOVE.L  #$D,D3   * -- push HEX D into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_E            MOVE.L  #$E,D3   * -- push HEX E into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS
STR_TO_HEX_F            MOVE.L  #$F,D3   * -- push HEX F into HEX stack
                        BSR     SHIFT_START_ADDR
                        ADD.L   D3,D4
                        RTS

SHIFT_START_ADDR        CLR     D7
                        MOVE.W  D1,D7
                        SUB.W   #1,D7
                        ASL     #2,D7   * -- D7 = (D1 - 1) * 4
                        ASL.L   D7,D3
END_SHIFT_START_ADDR    RTS
```

```
* ------------------------------------------------------
* ----------------- END OF SUBROUTINE --------------------
* ------------------------------------------------------
* | I borrowed this String --> Hex conversion subroutine |
* | from the project.                                    |
* |     post-condition: The converted hex location is    |
* |                     stored in D4.                    |
* ------------------------------------------------------


*************************************************************************
********

CONVERT_DONE    MOVE.L  D4, INPUT_ADDRESS   * -- store the addr at the end
of the program
                MOVE.L  D4, D5              * -- make a copy
                CMP.L   #$0, D4
                BLT     NEGATIVE
                BRA     EXTRACT_EXP

NEGATIVE        MOVE.B  #1, D3              * -- Store the sign bit in D3
                SUB.L   #$80000000, D4
                BRA     EXTRACT_EXP

* -------------------------- Finished processing the sign bit

EXTRACT_EXP     MOVE.B  #23, D0     * -- shifting the IEEE number by 23
bits
```

```
                                    * -- to the right will expose the
exponent
                                    * -- Sign-extension is a problem, but
it is
                                    * -- taken care of in the NEGATIVE
branch if
                                    * -- the number IS negative.
                ASR.L   D0, D4
STORE_EXP       MOVE.L  D4, D6      * -- Store the exponent in D6
                BRA     EXTRACT_MANT

* --------------------------- Finished processing the exponent

EXTRACT_MANT    MOVE.L  D5, D7      * -- Move the copy to D7
                ASL.L   D0, D4      * -- Shift the exponent 23-bits to the
left
                SUB.L   D4, D7      * -- Original value - exponent value =
mantissa

COUNT_MASTISSA  CLR     D1
ROTATE_LOOP     CMP.B   #0, D0
                BEQ     ROTATE_DONE
                ROR.L   #1,D7
                BCS     INCREMENT_MANT  * -- if the bit 1 is rotated out, C
= 1
CONT_ROTATE     SUB.B   #1, D0
                BRA     ROTATE_LOOP

INCREMENT_MANT  ADD.B   #1, D1      * -- D1 (later D7) stores the count of
the 1's in the mantissa
```

```
                BRA     CONT_ROTATE

ROTATE_DONE     BRA     PRINT_RESULT

* -------------------------- Finished processing the mantissa



************************************************************************
************************ OUTPUT OUTPUT OUTPUT ************************
************************************************************************
************************ OUTPUT OUTPUT OUTPUT ************************
************************************************************************
************************ OUTPUT OUTPUT OUTPUT ************************
************************************************************************

PRINT_RESULT    BSR     PRINT_OUTPUT_SEPARATOR
                MOVE.L  D1, D7  * -- make a copy of the mantissa results

* ---------------------------------------- sign bit
PRIME_SIGNBIT   BSR     PRINT_OUTPUT_SIGNBIT
                CMP.B   #1, D3
                BEQ     SIGNBIT_NEG
                BRA     SIGNBIT_POS

SIGNBIT_NEG     BSR     PRINT_SIGNBIT_NEGATIVE
                BRA     PRIME_EXPONENT
```

```
SIGNBIT_POS     BSR     PRINT_SIGNBIT_POSITIVE
                BRA     PRIME_EXPONENT

* ---------------------------------------- exponent
PRIME_EXPONENT  BSR     PRINT_EMPTY_LINE
                BSR     PRINT_OUTPUT_EXPONENT
                MOVE.L  D6, D1  * -- the exponent to be printed
                MOVE.B  #10, D2 * -- print in base 10
                MOVE.B  #15, D0 * -- trap task #15
                TRAP    #15
                BRA     PRIME_MANTISSA

* ---------------------------------------- mantissa  (now stored in D7)
PRIME_MANTISSA  BSR     PRINT_EMPTY_LINE
                BSR     PRINT_OUTPUT_MANTISSA
                MOVE.L  D7, D1  * -- the exponent to be printed
                MOVE.B  #10, D2 * -- print in base 10
                MOVE.B  #15, D0 * -- trap task #15
                TRAP    #15

CLEANING_UP     BSR     PRINT_EMPTY_LINE
                BSR     PRINT_OUTPUT_SEPARATOR
                BRA     END_PROGRAM



* ------------------------------------------------
* ----------- PROGRAM FLOW SUBROUTINES ----------------
```

```
* ------------------------------------------------

PRINT_EMPTY_LINE    LEA     EMPTY_LINE, A1
                    MOVE.B  #13, D0
                    TRAP    #15
                    RTS

PRINT_INPUT_PROMPT  LEA     INPUT_PROMPT, A1    * -- displaying input
message
                    MOVE.B  #14, D0
                    TRAP    #15
                    RTS

PRINT_OUTPUT_SEPARATOR  LEA         OUTPUT_SEPARATOR, A1
                        MOVE.B      #13, D0
                        TRAP        #15
                        RTS

* ----------- sign bit

PRINT_OUTPUT_SIGNBIT    LEA         OUTPUT_SIGNBIT, A1
                        MOVE.B      #14, D0
                        TRAP        #15
                        RTS

PRINT_SIGNBIT_POSITIVE  LEA         OUTPUT_SIGNBIT_POSITIVE, A1
                        MOVE.B      #14, D0
                        TRAP        #15
```

```
                        RTS

PRINT_SIGNBIT_NEGATIVE  LEA         OUTPUT_SIGNBIT_NEGATIVE, A1
                        MOVE.B      #14, D0
                        TRAP        #15
                        RTS

* ----------- exponent

PRINT_OUTPUT_EXPONENT   LEA         OUTPUT_EXPONENT, A1
                        MOVE.B      #14, D0
                        TRAP        #15
                        RTS

* ----------- mastissa

PRINT_OUTPUT_MANTISSA   LEA         OUTPUT_MANTISSA, A1
                        MOVE.B      #14, D0
                        TRAP        #15
                        RTS


END_PROGRAM  CLR    D2

    SIMHALT

* Put variables and constants here
```

```
EMPTY_LINE      DC.B    '', 0

INPUT_PROMPT    DC.B    'Please enter IEEE 32-bit floating point number in
hexadecimal: ', 0

OUTPUT_SEPARATOR    DC.B    '==================================', 0
OUTPUT_SIGNBIT      DC.B    'Sign bit: ', 0
OUTPUT_EXPONENT     DC.B    'Exponent: ', 0
OUTPUT_MANTISSA     DC.B    'Mantissa: ', 0

OUTPUT_SIGNBIT_POSITIVE     DC.B    '+', 0
OUTPUT_SIGNBIT_NEGATIVE     DC.B    '-', 0

INPUT_ADDRESS   DS.L    $1

    END    START        ; last line of source
```