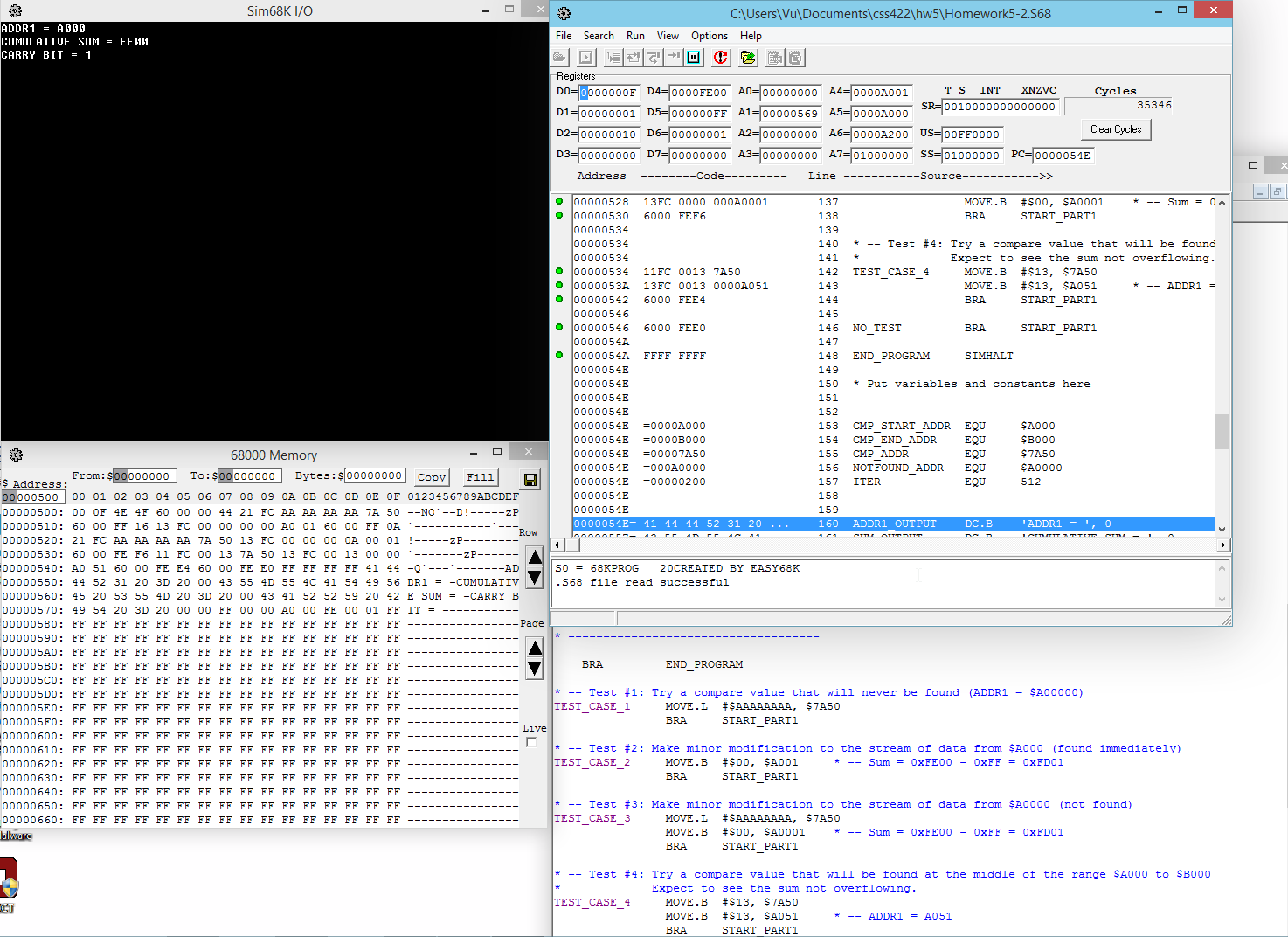
Homework 5

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

#### 

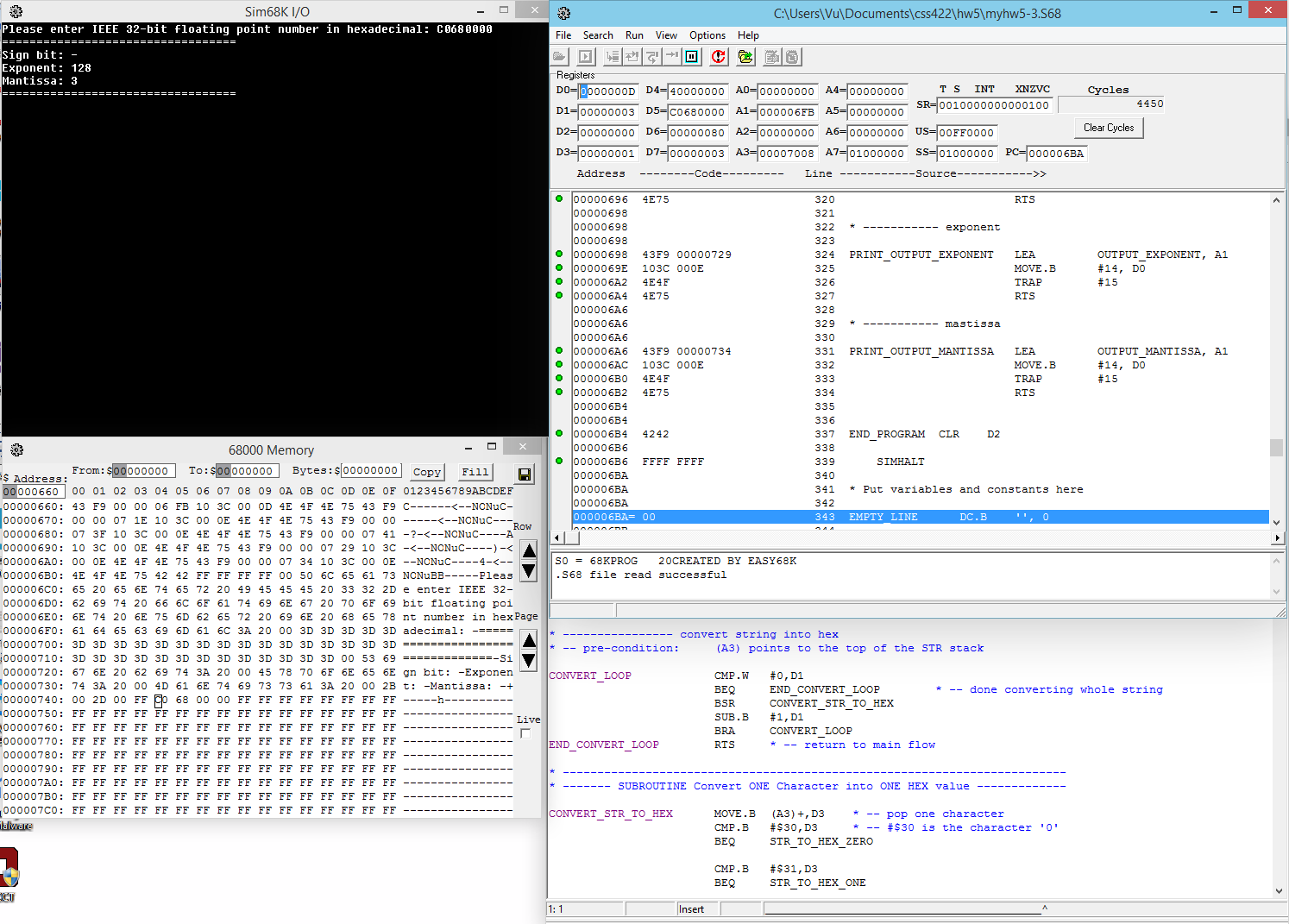
#### 

#### 

#### 

#### 

# 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

#### 

#### 