

Floating Point Conversion

Your program, i.e. F(x), will use a special format of floating point numbers. A "floating point" representation of a number writes the number as a mantissa and a signed exponent according to

$$x = \text{mantissa} * 2^{\text{exponent}}$$

where * denotes multiplication, ^ denotes exponentiation, and the mantissa is restricted to the range [0.5,1]. This means that the mantissa is always greater than or equal to 1/2 but less than 1. To help you with understanding this concept consider the representation of x=3 in this notation.

$$x = 3(\text{base } 10) = 11.000(\text{base } 2) = 0.11(\text{base } 2) \times 2^{+2} = 0.C000(\text{base } 16) \times 2^{+2}$$

Similarly,

$$x = -3(\text{base } 10) = -0.C000(\text{base } 16) \times 2^{+2}$$

$$x = 3.1(\text{base } 10) = 0.C666(\text{base } 16) \times 2^{+2}$$

$$x = 1/8(\text{base } 10) = 0.8000(\text{base } 16) \times 2^{-2}$$

$$x = -4096(\text{base } 10) = -0.8000(\text{base } 16) \times 2^{+12} = -0.C000(\text{base } 16) \times 2^{+0C}$$

(Note that the exponent is now written as +0C(base 16))

$$x = -4095(\text{base } 10) = -0.FFE0(\text{base } 16) \times 2^{+11} = -0.C000(\text{base } 16) \times 2^{+0B}$$

(Note that the exponent is now written as +0B(base 16))

Your program will input a word length number x in signed magnitude notation, convert the number x into "floating point" as defined above and store it in memory according to the format shown below.

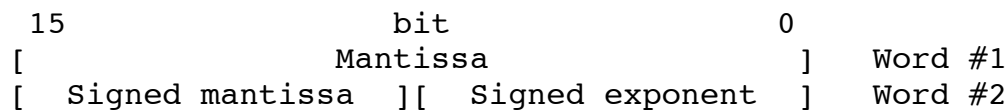


Figure 4. EEAP 282 Floating Point Number Representation

Note that this is a two word format. The sign actually only needs to be a single bit but, for convenience, we have defined it to be a byte. To help you understand this format several examples are shown in Figure 5 below.

$$x=3.0$$

[\$C000]	Word #1		
[\$00][\$02]	Word #2

$$x=3.1$$

[\$C666]	Word #1		
[\$00][\$02]	Word #2

```

x=-3
[      $C000      ]   Word #1
[  $FF  ][  $02  ]   Word #2

x=1/8
[      $8000      ]   Word #1
[  $00  ][  $FE  ]   Word #2

x=-4096
[      $8000      ]   Word #1
[  $FF  ][  $0C  ]   Word #2

x=3.1
[      $4095      ]   Word #1
[  $FF  ][  $0B  ]   Word #2

```

Figure 5 - Examples of the EEAP 282 floating point number format

Floating Point Conversion Subroutine

You will implement a procedure for converting a fixed point sine representation into a floating point representation as a subroutine which can be called from anywhere in your program.

Your subroutine should:

- utilize a register as an input. You will receive a small amount of extra credit for using HEXIN for input.
- include use of the MOVEM instruction to save ALL registers at the beginning of your subroutine and restore them at the end of the subroutine.
- pass the input word to the subroutine ON THE STACK. Although other methods of parameter passing to a subroutine are allowed, only this one will be accepted for purposes of grading.
- return a floating point long word (two words) ON THE STACK.

Your main program should:

- display the original, i.e. input data. NOTE: This can be entered into a data register PRIOR to executing your program.
- call your subroutine to do floating point conversion (with the parameters being passed ON THE STACK).
- display your floating point results.

Note that it is the programmer's responsibility to:

- set the stack pointer to the correct return address when the stack is used for parameter passing.
- set the stack pointer to an appropriate initial value. TYPICALLY THE DEBUGGER DOES NOT DO THIS AUTOMATICALLY FOR YOU!

FLOATING POINT TANGENT AND DIVISION SUBROUTINES

This part of the laboratory will develop a floating point divide routine. You will use the floating point format used for the first part of this lab. Furthermore, you will monitor your division routine for possible numeric overflows during the calculation process.

Questions your program and write-up should address are:

1. What are the limits of our floating point notation?

SOLUTION FOR PART I:

A floating point conversion PROGRAM is shown below.

- * Program which converts a signed magnitude value
- * into a floating point value.
- * Programmer defines the position of the binary point within the longword.
- * INPUTS:
- * D1 = signed magnitude longword
- * D0 = place of binary point (0-31); to right of specified digit
- * OUTPUTS:
- * D1 = floating point longword with the following format:
- * Bits 16-31 = binary value mantissa such that $0.5 \leq x < 1$
- * Bits 8-15 = sign byte (\$00 is positive, \$FF is negative)
- * Bits 0-7 = power of two in two's complement format.

	MOVEQ	#0,D2	CLEAR D2 FOR USE
	BTST	#31,D1	TEST THE INPUT SIGN BIT
	SNE	D2	SET LOW BYTE OF D2 TO ALL 1'S IF D1 NEGATIVE
	LSL.W	#8,D2	MOVE D2 UP ONE BYTE
	CMPI.W	#0,D1	CHECK INPUT FOR ZERO
	BNE	MANTLOOP	IF ZERO, THEN
ZEROMANT	MOVEQ	#0,D0	SET OUTPUT EXPONENT TO ZERO
	MOVEQ	#0,D1	SET OUTPUT MANTISSA TO ZERO (SIGN BIT MAY BE SET)
	BRA.S	CONSTRUCT	FORGET THE REST OF THE ROUTINE
MANTLOOP	ADDQ.B	#1,D0	OTHERWISE INCLUDE THE BINARY POINT MARKER
	LSL.L	#1,D1	SHIFT UP
	BPL	MANTLOOP	SHIFT UNTIL NEGATIVE
	SUBI.B	#32,D0	D0-32
	NEG.B	D0	NEGATE D0 TO GET EXPONENT
CONSTRUCT	MOVE.W	D2,D1	MOVE IN SIGN BYTE
	MOVE.B	D0,D1	MOVE IN EXPONENT