# Algorithm for Inputting Fixed Point Numbers in E-notation

Edosomwan Joseph H. E., Member IAENG

**Abstract-The purpose of this paper is to forward an algorithm that is useful to the assembly language programmer to enable him/her input numbers that are in fixed point or e-notation form. This algorithm shall enhance the maximization of the potentials of the Numerical Data Processor – the co-processor that performs floating point arithmetic.**

**Index Terms-Algorithm, Assembly language, Programmer, Fixed-point, Floating-point, co-processor**

## I INTRODUCTION

Programming in assembly language requires the use of explicit input/output routines. Be it manipulation of integer numbers, character data, fixed point numbers or floating point numbers, the programmer must either write and make use of his own input/output routines using the DOS service functions (for IBM microcomputer) or the ones pre-written and assembled into object codes by someone else.

Writing of an efficient input/output routine is like writing a normal program. And achieving the intended result depends largely on how efficient is this algorithm.

The algorithm below converts a number supplied in either fixed point form such as 357.896 or scientific e-notation form like 3.5796E02 to its ASC11 string code equivalent number.

This article assumed that the ASC11 string representing the number is terminated by a dollar ($) sign.

## II THE ALGORITHM AND THE NUMERICAL DATA PROCESSOR

The numerical data processor (hereafter referred to as NDP) is a co-processor dedicated to the manipulation of very large or very small floating point or fixed point numbers. It uses the ten-bytes (eighty binary digits) format to represent number[2]. With the ten-byte notation, the NDP uses the long format, so the range of numbers represented is:

$4.19 \times 10^{-307}$ to $1.67 \times 10^{308}$ (4)

[1] Gives an algorithm with which to convert fixed point numbers to its ASCII string form like 1.67 x 10, $^{308}$ it has to be converted to ASCII form or in e-notation form (1.67E308) before the intended result can be achieved. This is what the algorithm intended to address.

## III THE ALGORITHM

Below is the algorithm for inputting fixed and/e-notation numbers:

1. Initialise appropriate variables

    VALUE = 0.0;

    MINUS = FALSE;

    EXMINUS = FALSE;

    E_FOUND = FALSE;

    THE_DET = -1;

2. Point to first ASCII character;

3. Repeat the following until ‘e’ or ‘E’ or ‘$’ is found.

    If (character = ‘e’) then

    Replace character with ‘E’

        E_ FOUND = TRUE;

            *Exit loop;*

        *Elseif (character = ‘E’) then*

    *E_FOUND = TRUE*

        *Exit loop;*

    *Elseif (character = ‘$’) then*

        *Exit loop;*

    *Else*

        *Point at next ASCII character;`*

    *Endif;*

*End repeat;*

*4. Point at first ASCII character;*

*5. if (character = (‘.’) then*

Snr. Lecturer, Department of Computer Science, College of Education, Ekiadolor-Benin. P. .M. .B. 1144, Benin City, Edo State, Nigeria.. (maryjoe872002@yahoo.com)

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

*MINUS = TRUE;*

*Point at next ASCII character;*

*Endif*

*6. Increase pointer until point (.) is found;*

*7. if (E_FOUND = TRUE) then*

*While (character not 'E') do*

*Increment THE-DET;*

*Point at next ASCII character;*

*Endwhile;*

*Else*

*While (character not '$') do*

*Increment THE _DET;*

*Point at next ASCII character;*

*Endwhile*

*Endif*

8. If (E_FOUND = TRUE) then

Move pointer until 'E' is found;

Point at next ASCII character;

If (character = '-') then

EXMINUS = TRUE;

Point at next ASCII character;

Endif

Initialize EXPONENT to zero

(i.e. EXPONENT = 0)

While (character not '$')

Multiply EXPONENT BY 10;

Add digit equivalent of character to EXPONENT;

Point at next ASCII character;

Endwhile;

If (EXMINUS = TRUE) then

Negate EXPONENT;

Endif;

Endif;

9. If (E_FOUND = TRUE) then

Subtract THE _DET from EXPONENT and store result as EXPONENT;

10. Point at first ASCII character;

11. If (MINUS = TRUE) then

Advance pointer to next ASCII character;

Endif;

12. If (E_FOUND = TRUE) then

While (character not 'E')

If (character = '.') then

Point to next ASCII character;

Endif;

This algorithm below converts characters to its digit equivalent and vice versa:

Multiply VALUE by 10.0;

Add the equivalent fixed point number to VALUE;

Point at next ASCII character;

Endwhile;

Else

While (character not ''0

If (character = '.') than

Point at next ASCII character;

Endif;

Convert character to equivalent digit, and

Convert digit to equivalent fixed point number;

Multiply VALUE by 10.0;

Add equivalent fixed point number to VALUE;

Point at next ASCII character;

Endwhile;

Endif;

13. If (E_FOUND = TRUE) then

If (EXPONENT less 0)then

Negate EXPONENT;

Multiply VALUE by 0.1

(EXPONENT) times;

Elseif (EXPONENT ›0) then

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Multiply VALUE by 10.0

(EXPONENT) times;

Else

VALUE = VALUE;

Endif;

Else

If (THE_DET I Greater than 0) then

Multiply VALUE by 0.1

(THE_DET) times;

Endif;

Endif;

14. If (MINUS = TRUE) then

Negate VALUE;

Endif;

Stop[3]

## IV EXPLANATION OF THE ALGORITHM

In the above algorithm, certain variables are used and initialized for a start. These variables and what they depict are:-

1. VALUE represents the final number after it is converted.

2. THE DET is a variable used to represent fractional digits numbers physically supplied along with the original number.

3. MINUS is a Boolean variable whose value will be TRUE if the original number is negative (numbers less than one).

4 EXMINUS is another Boolean variable whose value will be TRUE if the sign of the exponent (numbers in E-notation) is negative (exponent less than one) otherwise its value remains FALSE (numbers in fixed point form).

5 E-FOUND is the last of the Boolean variables and its values will be TRUE if 'e' or 'E' is in the original number ( number in e-notation) else the value remains FALSE (number in fixed point form).

6 EXPONENT represents the exponent (i. e. for numbers in E-notation).

Step 2 of the algorithm sets pointer to the beginning of the ASCII string (first ASCII character).

In step 3, the ASCII string is scanned through and if a small letter 'e' is found (numbers in E-notation), the small letter is replaced with big letter E and E-FOUND is set to TRUE. Also if an upper case 'E' is found, E-FOUND is set TRUE and this conditions lead to exit from the loop. However, if a dollar sign (which marks the end of the string) is encountered, then the number is not in e-notation.

Step 4 takes us back to the beginning of the string.

In step 5, the first character of the ASCII string is checked, if it is the negative (-) sign, then it is a real number less than zero and accordingly, the Boolean variable MINUS is set to TRUE. The value of the variable remains FALSE if the first character is positive.

Step 6, causes the pointer to point to character point (.)

In step 7, if E_FOUND is TRUE, then the variable THE_DET is incremented by one (1) as many times as the number of ASCII characters is encountered, before the character 'E', otherwise it is incremented by one (1) as many times as the number of ASCII characters encountered before '$'.

Step 8, sets the Boolean variable EXMINUS to TRUE if the first byte after 'E' (numbers in E-notation) is a negative (-) sign otherwise its initial value of FALSE remains. Afterward, it obtains the exponent by multiplying the variable EXPONENT by 10 and adding the value equivalent of each ASCII character to EXPONENT for every ASCII character encountered before dollar ($) sign. Finally if EXMINUS was TRUE the eventual EXPONENT is negated.

Step 9 serves to obtain the difference between the exponent and number of fractional digits for real numbers in E-notation. This is necessary in determining how many times to the right or left the decimal point will be moved by way of multiplication by 10.0 or 0.1 to obtain the final number.

Step 10 takes the pointer to the first character of the ASCII string again.

Step 11.Advances pointer to the second ASCII character if the first was the negative (-) sign.

In step 12, the initial number is built and stored as VALUE. If the original number was supplied in e-notation, then for every ASCII character encountered before 'E' its equivalent digit is obtained and converted to an equivalent fixed point number. Then VALUE which was initially set to 0.0 is multiplied by 10.0 and the fixed point number equivalent of the encountered character is added to it (VALUE) after the multiplication by 10.0. However, if the number was supplied in fixed point form then the character that terminates the process above will be the dollar ($) sign and not 'E'.

Step 13.The value obtained in step 12 will be of the form 357896.0 if the number supplied were 357.896 or 3.57896E02. Therefore for a number supplied in fixed point form, the initial values obtained have to be multiplied by 0.1 as many times as the number of fractional digits to fix the decimal point at its right position. If the number were supplied in E-notation, the difference between the exponent and the number of fractional digits will have to be obtained. If this difference is negative, then the initial value will be multiplied by 0.1 as many times as the

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

positive value of the difference; and if the difference is positive, the initial value is multiplied by 10.0 as many times as the differences to fix the decimal point at its proposition. This is what step 13 achieves.

However, if the difference between the exponent and fractional digits is zero, thus for a numbers in E-notation form and there are no fractional digits (number in fixed point form), then VALUE remains the same.

Step 14 negates VALUE to obtain the final equivalent number if the number was negative else the final number equivalent to the ASCII string remains what is now and stored in position of VALUE.

Step 15 ends the algorithm.

## V CONCLUSION

This algorithm is not meant to compete with any other in terms of complexity and efficiency except that it seeks to complement Detmer's work[1] and to enable assembly language programmers to be able to input very small and very large numbers such as are manipulated by the NDP.

However, it has the weak point of not being able to detect illegal characters in a number[4].

## REFERENCES

[1] R. C. Detmer, "Fundamentals of Assembly Programming Using the IBM PC and Compatibles." D.C. Health and Company, Massachusetts. 1990, Pp. 530.

[2] Sanchez, "Assembly Language Tools and Techniques for the IBM Microcomputers". Prentice-hall; New Jersey: 1990, pp 447.

[3] K. R. Robert, Data Structure and Programming Design: Prentice Hall 3rd edition New Jersey. 1981

[4] J. H. Edosomwan, Sorting Algorithm: Analysis and comparison of Performance, Msc Project thesis at the University of Port Harcourt, Feb. 2004

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)