APPLIED RESEARCH ON IMPLEMENTATION AND USE OF LIST PROCESSING LANGUAGES

by

Salzmann, Roy M. (PI)
Flaherty, Thomas A.
Ponton, Marlene E.

of

CHARLES W. ADAMS ASSOCIATES, INC.
575 Technology Square
Cambridge, Massachusetts

Contract No. AF19(628)-5026
Project No. 4641
Task No. 464102

FINAL REPORT
Period covered: January 1965-January 1966
Date of report: May 1966

Prepared for

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS

Notice No. — Best Available Copy
ABSTRACT

This final report on Contract No. AF19(628)-5026 contains a summary of the five major tasks performed during the one-year duration of the contract. The work performed consisted of: 1) a variable-precision floating-point package for the solution of problems requiring very high precision, 2) extension of and improvements to the software system developed under an earlier contract, 3) assistance in the implementation and validation of a LISP Compiler, 4) development of a program for powerful manipulation of symbolic text (TECO), and 5) specification of a set of generalized display routines for visual communication with the computer. All work done related to the M-460 research computer at AFCRL.
CONTENTS

I. Summary of Objectives 1

II. Summary of Tasks Performed 2

Variable-Precision Floating-Point Package 2
M-460 Assembly Language System 4
LISP Implementation 5
TECO 5
Generalized Display Package 6

III. Personnel and Publications 8

Personnel 8
Publications 8

Appendix A

VARIABLE PRECISION FLOATING-POINT PACKAGE

I. Introduction A-1

II. Program Characteristics A-2

Number Representation A-2
Summary of Routines A-2
Calling Sequence Convention A-3

III. Subroutine Descriptions A-5

FLAD A-6
FLSB A-9
FLTMUL A-11
FLDV A-14
FLTCOM A-16
FLTFIX A-19
FXTFLT A-21
NUMIN A-23
NUMOUT A-25
FLTMOV A-27
IV. Constants

V. Flow Charts
I. SUMMARY OF OBJECTIVES

A. Investigate and evaluate techniques for the implementation of processors for list-processing languages. Consider possible extensions to, in particular, the LISP language, and investigate the corresponding problems of implementation.

B. Study techniques, including use of visual display devices for facilitating convenient interaction between an on-line user and a computer system. Extend the techniques for on-line debugging and program modification already in use in the AFCRL LISP system.

C. Apply the results of these studies to the implementation of a "second-generation" version of the on-line LISP list-processing system currently in operation on the AN/USQ-17 computer at AFCRL. Develop techniques for the convenient modification and updating of this programming system.
II. SUMMARY OF TASKS PERFORMED

Variable-Precision Floating-Point Package

Some problem areas in the Data Sciences Laboratory require the use of floating-point operations with very high precision. Although routines were available for floating-point operations on the M-460, the 45-bit mantissa available with these routines did not provide sufficient precision. Hence a new set of routines allowing indefinite precision was designed and constructed.

The range of numbers dealt with by these routines was not increased; 14 bits for the exponent plus a single sign bit was still utilized. The precision is indefinitely extensible, however, in increments of 30 bits beyond the minimum 45-bit mantissa. A total of 14 routines comprise the package, including routines for addition, subtraction, multiplication, division, comparison, conversion to and from fixed-point form, moving, input and output. Other functions available in the package, such as shifting, normalizing and conversion from signed magnitude to 1's complement form, are intended primarily for use by the more user-oriented routines mentioned above; but they may also be used by the programmer for more sophisticated operations.

Each routine determines the precision required for that operation by a number in one of the index registers (b6) on
entry to the routine. This number represents the number of full computer words (of 30 bits each) necessary to contain the mantissa beyond the minimum 45 bits. Other index registers contain other parameters appropriate to the particular routines.

The division operation is the only exception to the variable-precision nature of the package. Since the problem areas for which the routines were constructed did not require division and the implementation problems were considerable, a single-precision (45-bit) routine was prepared to be compatible with the rest of the package, i.e., coded in the RAP language with similar calling sequences, word formats and general conventions.

The input-output routines do not communicate directly with peripheral devices but, rather, with a character buffer which in turn is utilized by the general I/O routines in the M-460 software such as "typel" and "flexinl". Completely general format is permitted allowing for decimal exponential notation, integer, fixed-point, etc.

A separate group of routines was also prepared for single-precision floating-point numbers. This was felt to be desirable because of the considerable saving in speed possible with single-precision floating-point arithmetic by using separate routines rather than the single-word case of the variable-precision
adams associates

routines. These routines have calling sequences identical to their variable-precision counterparts so that a user program could be checked out with the single-precision routines, then expanded by merely changing the names of the routines called.

Updated and expanded documentation of this package is given in Appendix A to this report.

M-460 Assembly Language System

As a result of work done under a previous contract, many basic software routines were available for providing such necessary functions as editing symbolic source programs, assembling, loading and linking separately assembled subprograms, debugging in symbolic language using the on-line flexewriter, using magnetic tape for memory dumps, etc. The system developed to control all of these functions had many inadequacies which required substantial work during the course of this contract in order to have available a smoothly-running, efficient and well-integrated system.

In addition to innumerable small tasks performed on a day-by-day basis to improve operation of the entire software package, such functions were performed as preparing library tapes containing certain system subroutines for use by applications programs independently of the system, relocating portions of
the system to various areas of core memory, making a minimum form of the FILER to be used independently of the remainder of the system, and adding some primitive TIC-like functions to the FILER. Several other features were added to the FILER to facilitate the simultaneous use of all magnetic tape units for speedier operation and allow for the use of multiple copies of a file with a given name on one reel of tape. The latter features and associated command characters are described in Appendix B to Quarterly Status Report No. 2.

**LISP Implementation**

Assistance was given to members of the AFCRL research staff in the implementation of the LISP compiler on the M-460. This work included editing and compiling portions of the compiler written in the LISP language using the previous version of the compiler, checking out the results of compilation, etc.

**TECO**

A program named TECO (Tape Editor and Corrector) was developed using specifications for similar programs written for the PDP-1 and PDP-6. This program will permit the use of the Type 340 display device attached to the M-460 for program debugging and updating dealing with generalized text strings. Of
particular interest is the ability to operate flexibly on LISP statements with visual verification of the manipulations performed.

TECO uses an on-line command language (which permits macro definitions, conditionals, etc.) to control input-output as well as text operations. The macro language allows the most sophisticated search, match and substitution operations as well as simple typographical corrections to text.

Although the existing input-output routines in the M-460 software system have been used to the maximum extent possible, it was necessary to develop new routines for use of the display device. This work was performed in conjunction with the generalized display routines described below.

**Generalized Display Package**

The Type 340 display device interfaced to the M-460 enables powerful graphical techniques to be used for on-line problem solving. Both line-drawing (using incremental vector hardware) and alphanumeric (using character generation hardware) facilities may be used to great advantage in the preparation and checkout of list-oriented programs.

Sinc the specific techniques to be used will be determined largely by experimentation, it is desirable to have a
flexible system within which many graphical techniques can be investigated. Specifications were prepared for a set of primitive routines to perform the necessary functions of display initiation and regeneration, display buffer maintenance, object identification, simple object generation (points, lines, circles, characters, etc.), light-pen identification, and utilization of hardware features of intensity control, character angle control, etc. These routines will be implemented under a separate contract.
III. PERSONNEL AND PUBLICATIONS

Personnel

The following personnel of Adams Associates have been the principal participants in the work done under this contract:

Roy M. Salzman       Supervisory Analyst
Donald E. Ellis      Programmer Analyst
John S. Hermistone    Programmer Analyst
Thomas A. Flaherty    Senior Programmer
Marlene E. Ponton     Senior Programmer

Publications

Other than the Quarterly Status Reports and this final report, no publications have been required by or produced under the terms of this contract.
APPENDIX A

VARIABLE PRECISION FLOATING-POINT PACKAGE

I. INTRODUCTION

The original floating-point arithmetic package for the Univac 460 allowed for a 14-bit exponent, 45-bit mantissa and a single sign bit. Since precision permitted by the size of the mantissa was not adequate for some problems, a variable-precision package was developed to provide basically the same arithmetic functions but with no limit on the length of the mantissa. The only function not available in the variable-precision version is floating-point division. This operation is available in the package as a single-precision routine with similar conventions since no need could be seen for variable precision in this operation. Input and output routines have also been added to facilitate communication between the package and the user routines.

The scale of the numbers which can be handled by this new package is the same as with the previous set of routines since the exponent and the basic mode of representation are unchanged.
II. PROGRAM CHARACTERISTICS

Number Representation

The floating-point binary number format used is a 1-l4-n format with one common sign bit indicating the sign of the total numerical quantity, a 14-bit exponent with a bias of 20000, and an n-bit mantissa where n=30(m)+15; m=1,2,3,---. Thus each floating-point number requires m+1 30-bit storage registers. The mode of representation is 1's complement rather than the more usual signed magnitude representation. Using this mode, a negative floating-point number is a complete 1's complement of its positive counterpart. The number format may be diagrammed as follows:

```
<table>
<thead>
<tr>
<th>Word 1</th>
<th>Word 2</th>
<th>Word 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0=</td>
<td>20001</td>
<td>40000</td>
</tr>
<tr>
<td></td>
<td>000000000</td>
<td>000000000</td>
</tr>
<tr>
<td>Exponent</td>
<td>mantissa</td>
<td></td>
</tr>
</tbody>
</table>

-1.0=   57776 37777    7777777777    7777777777
| Exponent | mantissa |        |
| with     |         |        |
| sign bit |         |        |
```

Summary of Routines

- FLAD - floating-point addition
- FLSB - floating-point subtraction
- FLTMUL - floating-point multiplication
- FLTDV - floating-point division
FLTCOM - floating-point comparison
FLTFIX - floating-point to fixed-point conversion
FXTFLT - fixed-point to floating-point conversion
NUMIN - input a floating-point number from a packed buffer
NUMOUT - output a floating-point number to a packed buffer
FLTMOV - move a floating-point number from one area to another
CTOSM - convert from signed magnitude to 1's complement
RSHIFT - right shift a floating-point number and adjust exponent
LSHIFT - left shift a floating-point number and adjust exponent
FLTNORM - normalize a floating-point number
GET1 - input a single character from a packed buffer
PUT1 - output a single character to a packed buffer

The first nine of the above routines would normally be used by the application programmer to perform floating-point arithmetic and input-output operations. The last seven routines may also be used, but their primary purpose is to perform appropriate utility functions for the first nine.

**Calling Sequence Convention**

The basic input arguments to the primary (first nine) sub-routines are contained in the A and Q registers upon entry to the routine. Normally, A contains the address of the first operand, Q the address of the second operand, and C(FLTN) the number of words required for the mantissa of the operands. In the case of fixed-point to floating-point conversion, b7 contains the binary-point position of the fixed-point quantity.
Output of the arithmetic function routines (FLAD, FLSB, FLTMUL, FLTDV) is left in registers AF through AF+C(FLTN).
The C(FLTN) is never altered by the subroutines and therefore need not be set up before every call. It should be set initially and at such time as the length of the data changes.
III. SUBROUTINE DESCRIPTIONS

The 16 subroutines referred to above are described in detail on the following pages.
Function

To add two floating-point numbers of variable length (format 1-14-m) and store the normalized sum in a common storage area.

Calling Sequence

lda x
ldq y
rj flad
(normal return)

Input

x - address of first word of X
y - address of first word of Y
n - number of words - 1 stored in FLTN

Output

Z(1)-Z(n+1) - floating-point sum of X and Y stored in AF(1)-AF(n+1).

Subroutines Used

CTOSM, FLTNORM, FLTMOV, RSHIFT

Storage Areas Read

X(1) - X(n+1), Y(1) - Y(n+1), FLTN

Storage Areas Written

AF(1)-AF(n+1), SIGN, SIGN1, SIGN SWITCH, FLTNUM(1)-FLTNUM(n+1), FLTEXP, AFSIGN
Method

1. Convert addends to signed magnitude format.

2. Calculate a shift factor $k$
   
   $t = |\exp(x)| - |\exp(y)|$
   
   $k = |t|$

3. Shift smaller addend right $k$ places in order to line up binary point.

4. Set

   $x = a_1 2^{-15} + a_2 2^{-30} + \ldots + a_m 2^{-15m}$
   
   $y = b_1 2^{-15} + b_2 2^{-30} + \ldots + b_m 2^{-15m}$

   where
   
   $m = 1$ to $(2n+1)$

5. Add (for range of $m$)

   $W_m = b_m 2^{-15} + a_m 2^{-15m}$
   
   set $W_0 = 0$

6. Combine $W$'s to form sum $z$

   for $m = (2n-1)$ to $1$; $r = overflow$
   
   $\frac{z_{m+1}}{2} + \frac{r_{m+1}}{2} = W_{m-1} 2^{-15} + W_m + \frac{r_{m+1}}{2}$

   for $m = 2n+1$
   
   $z_{n+1} + r_{n+1} = W_{2(n-1)} 2^{-15} + W_{2n+1}$

7. If $r_1$ is equal to zero, go to step 8.
   If $r_1$ is not equal to zero, shift entire sum right and adjust exponent accordingly.

8. Insert exponent

   $z_1 + \exp(z) \cdot 2^{-15} \rightarrow z_1$
9. Convert addends back to 1's complement.


**Error Conditions**

If the exponent of the result $z$ exceeds 37777, control is transferred to "flterror." A STOP 4 halt will result and the address at which the error occurred will appear in the A-register.
Function

To subtract one variable-length floating-point number (format 1-14-m) from another of the same length and store the normalized difference in a common storage area.

Calling Sequence

\begin{verbatim}
lda  x
ldq  y
rj   flsb
(normal return)
\end{verbatim}

Input

\begin{itemize}
\item x - address of first word of X
\item y - address of first word of Y
\item n - number of words - 1 stored in FLTN
\end{itemize}

Output

\begin{itemize}
\item Z(1)-Z(n+1) - floating-point difference of X and Y stored in AF(1)-AF(n+1).
\end{itemize}

Subroutines Used

FLAD

Storage Area Read

\begin{itemize}
\item X(1) - X(n+1), Y(1) - Y(n+1), FLTN
\end{itemize}

Storage Area Written

\begin{itemize}
\item AF(1)-AF(n+1)
\end{itemize}
Method

1. Complement floating-point number y. Save original sign.

2. Go to add x and y (see FLAD for method).

3. Restore y to original sign.

Error Conditions

None except those provided by FLAD.
Function

To multiply two floating-point numbers of variable length (format 1-14-m) and store the normalized result in a common storage area.

Calling Sequence

```
ida x
ldq y
rj fltmul
(normal return)
```

Input

- x - address of first word of X
- y - address of first word of Y
- n - number of words - 1 stored in FLTN

Output

- Z(1)-Z(n+1) - floating-point product of X and Y stored in AF(1)-AF(n+1)

Subroutines Used

- CTOSM, FLTNORM

Storage Areas Read

- X(1) - X(n+1), Y(1) - Y(n+1), FLTN

Storage Areas Written

- AF(1)-AF(n+1), FLTNUM(1) - FLTNUM(2n+1), SIGN
Method

1. Let $x = a_1 2^0 + a_2 2^{-15} + a_3 2^{-30} + \ldots + a_t 2^{-15} (2t+3)$
   
   $t \neq 0, 1$
   $t = 2^{-(n+1)}$

   Let $y = b_1 2^0 + b_2 2^{-15} + b_3 2^{-30} + \ldots + b_s 2^{-15} (s-1)$
   
   $s \neq 0, 1$
   $s = 2^{-2(n+1)}$

2. Then for $i = 1$ to $s$
   
   $j = 1$ to $t$

   calculate

   $2b_i \left| \frac{a_j - 1}{2} \right| + b_i = C_k$

   $C_k = d_1 2^0 + d_2 2^{-15} + d_3 2^{-30}$

3. If $j = 1$ or 2
   
   set: $f_{i+j-1} = d_1 2^0 + f_{i+j-1}$

   $f_{i+j} = d_2 2^{-15} + f_{i+j}$

   $f_{i+j} = d_3 2^{-30} + f_{i+j+1}$

4. If $j \neq 1$ or 2
   
   set: $f_{i+j} = d_1 2^0 + f_{i+j}$

   $f_{i+j+1} = d_2 2^{-15} + f_{i+j+1}$

   $f_{i+j+2} = d_3 2^{-30} + f_{i+j+2}$
5. Sum intermediate results

\[ Z_{n+1} + r_1 + f_{2n+1}2^{-15} + f_{2n} \]

\[ Z_n + r_2 = f_{2n-1}2^{-16} + f_{2n-2} + r_1 \]

\[ \vdots \]

\[ Z_1 = f_1 + r_{n+3} \]

Notes

1. Length of input is limited only by amount of storage reserved for a working area. Presently the maximum word length is 10.

2. To avoid an error, both words should occupy the same amount of storage. If one input value is shorter, it should be filled with zeros.

Error Conditions

None.
FLDV

Function

To divide a pair of floating-point numbers each of length 2 (format 1-14-45) and store the normalized result in a common storage area (AF,AF+1).

Calling Sequence

lda x
ldq y
rjp fldv
(normal return)

Input

x - address of first word of X
y - address of first word of Y

Output

Z(1),Z(2) - floating-point quotient of X/Y stored in AF,AF+1

Subroutines Used

CTOSM

Storage Areas Read

X, X+1, Y, Y+1

Storage Areas Written

AF,AF+1, DVHPL, DVHPR1, DVHP2, DVHPR2, FLTNUM, FLTNUM+1
Method

1. Change representation to signed magnitude.
2. Compute exponent by subtraction.
3. If $X = 0$ go to step 9
4. If $Y = 0$ go to step 10
5. Separate divisor, $Y$, into two signed words $Y_1, Y_2$
   \[ Y = Y_1 + Y_2 \]
6. Divide $X$ value by $Y_1$
   \[ X/Y_1 = X_1 + \text{remainder} \]
7. Divide remainder by $Y_1$
   \[ \frac{\text{Rem}}{Y_1} = X_2 \]
8. Get correction
   \[ \frac{2(X_1)Y_2}{Y_1} \]
9. Set result
   \[ Z = X_1 + X_2 + \text{bias} - \frac{2(X_1)Y_2}{Y_1} \]
10. Store result in $AF, AF+1$, two's complement. Return to user.
11. Set $AF, AF+1$ to 0 and exit.
12. Set $AF, AF+1 = \pm \alpha$ depending on $X$; exit.
Function

To compare arithmetically two variable-length floating-point numbers (format 1-14-m). Control is returned to one of the three locations following the call depending on the relationship between the numbers.

Calling Sequence

lda x
ldq y
rj fltcom
(return 1 if x < y)
(return 2 if x = y)
(return 3 if x > y)

Input

x - address of first word of X
y - address of first word of Y
n - number of words - 1 stored in FLTIN

Output

None

Subroutines Used

FLSB, LSHIFT

Storage Areas Read

X(1)-X(n+1), Y(1)-Y(n+1), FLTIN

Storage Areas Written

FLTNOT
Method

1. Check X and Y for like signs.

2. If signs are not alike, choose positive value as the greater and transfer control to proper return.

3. If signs are alike and positive, calculate K.
   
   \[ K = \exp(x) - \exp(y) \]
   
   a. if \( K < 0 \) then \( X < Y \)
   b. if \( K > 0 \) then \( X > Y \)
   c. if \( K = 0 \) then go to FLSB

   subtract:
   
   \[ X - Y = Z \]
   
   if \( Z > 0 \) then \( X > Y \)
   if \( Z < 0 \) then \( X < Y \)
   if \( Z = 0 \) then \( X = Y \)
   d. transfer control to proper return

4. If signs are alike and negative, complement both values. Then calculate K.
   
   \[ K = \exp(x) - \exp(y) \]
   
   a. if \( K < 0 \) then \( X > Y \)
   b. if \( K > 0 \) then \( X < Y \)
   c. if \( K = 0 \) then go to FLSB

   subtract
   
   \[ X - Y = Z \]
   
   if \( Z < 0 \) then \( X > Y \)
   if \( Z > 0 \) then \( X < Y \)
   if \( Z = 0 \) then \( X = Y \)
d. Restore original signs of input

e. Transfer control to proper return

Error Conditions

None.
Function

To convert a variable-length floating-point number (format 1-14-m) to a fixed-point number and store the binary point 8.

Calling Sequence

    lda x
    ldq y
    rj fltfix
    (normal return)
    b7 contains the binary point (8) on return

Input

x - address of first word of floating-point number X to be converted
y - address of first word of area into which fixed-point number Y should be stored
n - number of words - 1 stored in FLTN

Output

8 - binary point of number generated
Y(1)-Y(n+1) fixed-point X

Subroutines Used

LSHIFT, FLTMOV, CTOSM

Storage Areas Read

X(1) - X(n+1), FLTN
Storage Areas Written

SIGN, FLTNUM, Y(1) - Y(n+1)

Method

1. Set $X = |X|$; save original sign.

2. Move input to $f$ (working storage - FLTNUM)
   for $m = 1 - (n+1)$
   \[ f_{m+1} = X_m \]

3. Set $f = f \cdot 2^{-14}$ by shifting left 14 places. This left justifies the input in the area $f(2)$ to $f(n+2)$ with one bit remaining for sign.

4. Set $Y_m = f_{m+1}$ for $m = 1$ to $(n+1)$.

5. Restore original sign of value.

Error Conditions

None.
Function

To convert a variable-length fixed-point number to a floating-point number in format 1-14-m. Output will be normalized.

Calling Sequence

ida x
ldq y
ldb ß,7
rj fxtflt
(normal return)

Input

x - address of first word of fixed-point number X to be converted
y - address of first word of converted number Y
n - number of words - 1 stored in FLTN
ß - binary point of input

Output

Y(1)-Y(n+1) - floating-point representation of input X.

Subroutines Used

FLTNORM, FLTMOV, CTOSM

Storage Areas Read

X(1)-X(n+1), FLTN

Storage Areas Written

Y(1)-Y(n+1)
Method

1. Let \( X = |X|, \) save original sign.

2. Move input to working storage \( f. \)
   
   for \( m = 1 \) to \( (n+1) \)
   
   \[
   f_{m+1} = X_m
   \]

3. Set \( \exp(f) = \text{bias} + 15 \)
   
   \[
   n = n + 1
   \]

4. Insert exponent into first word of \( f\)-area
   
   \[
   f(1) = \exp(f) \cdot 2^{-16}
   \]

5. Normalize \( f\)-area.

6. Restore original sign.

7. Transfer \( f\)-area to \( Y\)-area.

Error Conditions

None.
NUMIN

Function

To convert a floating-point number represented by a flex-code array to binary format floating-point number of specified length. Numin will accept any format

12
12.0
12.
1.2E+1
1.2E1
120E-1

-100 < E < 100

A 77 code will delimit the array.

Calling Sequence

lda x
ldg buffer
rj numin
(error return)
(normal return)

Input

buffer = starting address of flex-code array
x = starting address of binary floating-point number
n = number of words - 1 stored in FLTIN

Output

X(1)-X(n+1)

Subroutines Used

GET1, FLTNORM, FLTMOV, FLTMUL, FLAD, POWER, CTOSM
Storage Areas Read

FLTIN

Storage Areas Written

$X(1)-X(n+1)$
NUMOUT

Function

To convert a binary floating-point number to a flex-code array with integral and fractional portions specified and an exponent. A 77 will delimit the output buffer.

Calling Sequence

ida buffer
ldq x
rj numout
uandl k,l
(error return)
(normal return)

Input

buffer = starting address of flex-code array
x = starting address binary array
k = number of decimal places before decimal point
l = number of decimal place after decimal point
n = number of words -1 of input stored in FLTN

The length of the buffer LIM is determined by the input

\[ LIM = \frac{K + L + 15}{5} \]

Output

A flex-code buffer representing the floating-point number, stored in C(LIM) locations beginning at BUFFER and delimited by 77.

Subroutines Used

CTOSM, FLTMUL, FLTMOV, FLTCOM, PUT1, FLTNORM
NUMOUT - con't

Storage Area Read

\[ X(1) - X(n+1), \text{ FLTIN} \]

Storage Area Written

\[ \text{BUFFER(1)} - \text{BUFFER(LIM)} \]
Function
To transfer any string of words from one area to another.

Calling Sequence
lda x
ldq y
rj fltmov
(normal return)

Input
x - starting address of area to be transferred
y - starting address of area to be filled
n - number of words to be moved - 1 stored in FLTN

Output
Y(1)-Y(n+1)

Subroutines Used
None

Storage Areas Read
X(1)-X(n+1), FLTN

Storage Areas Written
Y(1)-Y(n+1)

Error Conditions
None
CTOSM

Function
To convert a 1's complement floating-point number to signed magnitude, or vice versa.

Calling Sequence
ida x
rj ctosm
(normal return)

Input
x - address of first word of floating-point number X
n - number of words - 1 stored in FLT

Output
X(1)-X(n+1) converted floating-point number.

Subroutines Used
None

Storage Areas Read
X(1)-X(n+1), FLT

Storage Areas Written
X(1)-X(n+1)

Method
If X > 0 transfer control to user
2. If $X < 0$ set $X = -X$.
3. Invert sign bit.

**Error Conditions**

None.
RSHIFT

Function

to shift a variable-length floating-point number (format l-14-m) n places right (end off) and adjust the exponent accordingly.

Calling Sequence

lda  x
ldq  k
rj  rshift
(normal return)

Input

x - address of first word of floating-point number X to be shifted
n - number of words - 1 stored in FLTN
k - number of places to shift

Output

X(1)-X(n+1) shifted input in same area

Subroutines Used

CTOSM

Storage Areas Read

X(1)-X(n+1), FLTN

Storage Areas Written

RTEMP, RTEML, RTEML2, IR2, RSH7
Method

1. Set $K = k$.
2. If $k > 30$ set $k' = 30$ and $k = k - 30$
   If $k \leq 30$ set $k' = k$ and $k = 0$
3. Let
   \[ X = a_1 2^{-15} + a_2 2^{-30} + \ldots + a_{n+1} 2^{-15(n+1)} \]
   \[ b_1 = a_1 2^{-15} \]
   \[ b_e = a_2 2^{-30} \]
   \[ \vdots \]
   \[ b_m = a_{n+1} 2^{-15(n+1)} \]
   \[ r = \text{overflow} \]
   for $m = n+1$
   \[ C_m + r_m = b_m \cdot 2^{-k'} \]
   for $m = n$ to $0$
   \[ C_m + r_m = b_m \cdot 2^{-k'} + r_{m+1} \]
4. Check $k = 0$. If $k = 0$ replace original input with $C$.
   Set \( \text{exp} = \text{exp} + K \).
   If $k \neq 0$ return to step 3.

Error Conditions

None.
LSHIFT

Function

To shift a floating-point number of variable length (format l-14-m) a specified number of places left and adjust the exponent accordingly.

Calling Sequence

lda x
ldq k
rj lshift
(normal return)

Input

x - address of first word of floating point number X to be shifted
n - number of words - 1 stored in FLTN
k - shift factor

Output

X(1)-X(n+1) - shifted word replaces original input

Subroutines Used

CTOSM

Storage Areas Read

X(1)-X(n+1), FLTN

Storage Areas Written

LSTEM, LSEXP, LSTEM2, LSEND
Method

Same as RSHIFT routine except that K is positive rather than negative.

Error Conditions

If at the end of shifting there is a remainder of non-zero, control will be transferred to FLTERROR. The address at which the error occurred will be shown in the A-register and a STOP # will occur.
Function

To normalize a floating-point number of variable length (format l-14-m). The first significant bit will be 0 if number is negative and 1 if number is positive. The exponent will be adjusted accordingly.

Calling Sequence

`lda x`
`rj fltnorm`
(normal return)

Input

x - address of first word of floating-point number X
n - number of words - 1 stored in FLT

Output

X(1)-X(n+1) a normalized floating-point number stored in area of input.

Subroutines Used

LSHIFT, CTOSM

Storage Areas Read

X(1)-X(n+1), FLT

Storage Areas Written

X(1)-X(n+1)
Method

1. Examine number starting with first word.
2. Inspect one bit at a time until reaching the first significant bit.
3. Index k at each check to determine a shift factor.
4. Go to LSHIFT with k as input.
GET1

Function

To unpack a specified buffer of 6-bit flex-code characters starting with the leftmost character and leave the character right justified in the A register in bci mode.

Calling Sequence

\[(\text{normal return})\]

Input

GPOINT = starting address of buffer
GCHAR = character (0-4) to start reading (left to right)

Output

A - next 6-bit bci character
GPOINT, GCHAR - updated character position

Subroutines Used

None

Storage Areas Read

GPOINT, GCHAR TBCIFLX (table)

Storage Areas Written

GPOINT, GCHAR
PUT1

Function

To pack a buffer, from left to right, with 6-bit characters converted to flex-code from bci.

Calling Sequence

lda $k
jr putl
(normal return)

Input

POINT = starting address of buffer
PCiAR = character (0-4) to start packing (left to right)
$K$ = six-bit flex-code character

Output

A packed buffer of bci code
POINT, PCiAR - updated character position

Subroutines Used

None

Storage Areas Read

TBICFLX, PCiAR, POINT

Storage Areas Written

PCiAR, POINT
IV. CONSTANTS

<table>
<thead>
<tr>
<th>Label</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>con4</td>
<td>40000.0000</td>
<td>Used to set and complement the sign bit</td>
</tr>
<tr>
<td>con8</td>
<td>0000077777</td>
<td>Used in subtract routine</td>
</tr>
<tr>
<td>con7</td>
<td>00001.0000</td>
<td>Used to check overflow</td>
</tr>
<tr>
<td>inst1</td>
<td>rj flad</td>
<td>Instruction switch used in 'flsb'</td>
</tr>
<tr>
<td>inst2</td>
<td>jl flsh</td>
<td>Instruction switch used in 'flsb'</td>
</tr>
</tbody>
</table>

V. FLOW CHARTS

Flow charts for the subroutines described in the preceding section appear on the remaining pages of this appendix.
FLAD
Sheet 1 of 3
$f(t) = e^{at}$

- If $t < 0$, then $s(t) = 0$.
- If $t > 0$, then $s(t) = 1$.

For $t = 0$, $s(t) = 0$.

The diagram shows a flowchart with decision points and arithmetic operations. It includes:

- A decision point for $t > 0$ and $t < 0$.
- Arithmetic operations including addition and exponentiation.
- A decision for $s(t) = 0$ or $s(t) = 1$.

The flowchart also includes a section marked "FLAD" and "Sheet 2 of 3."
x = \( a_1 2^0 + a_2 2^{-15} + \ldots + a_t 2^{-15(t+3)} \)
= \( A_1 + A_2 + \ldots + A_{t+1} \)

\[ Y = b_1 2^0 + b_2 2^{-15} + \ldots + b_s 2^{-15(s-1)} \]
= \( B_1 + B_2 + \ldots + B_{s+n+1} \)
adam's associates

D

\text{yend = } N + S

\text{aend = } N + R

\text{stend = } 2(N+1)

\text{clear working storage and output area.}

\text{t = 0}
\text{E = t}

\text{clear switch}
\text{G = 0}

C = 2[A(1) - 14 \cdot \beta(1)]

\text{d}_1^0 + \text{d}_2^{15} = C

\text{F(1) = } d_1^0
\text{F(2) = } d_2^{15}

D

E

\text{G = 0}

\text{NO}

\text{YES}

\text{t = t + 1}

\text{t = stend}

\text{NO}

\text{YES}

\text{t = t + 1}

\text{G = G + 1}

\text{R = aend}

\text{NO}

\text{YES}

\text{R = R + 1}
\text{L = L + 1}

G

K

\text{Sheet 3}

\text{Sheet 3}

\text{Sheet 3}

F

G

H

\text{Sheet 3}

\text{Sheet 3}

\text{Sheet 3}

L

Sheet 2 of 6
FLIMUL

Sheet 3 of 6
\[ G = G + 1 \]
\[ R = \text{a} \text{end} \]
\[ E = E + 1 \quad t = E \quad G = 0 \]
\[ R = [x] \quad \ell = 1 \]
\[ \text{Load: } A(1) \cdot 2^{-14} \]

\[ \text{Load: } A(\ell) - 1 \]
\[ \frac{1}{2} \]
\[ \text{Mul: } 2B(m) \]
\[ \text{Add: } B(m) \]
\[ \text{Store } C \]
\[ d_1^0 + d_2^1 + d_3^30 = C \]
\[ R(t) = d_4^2 + F(t) \]
\[ F(k) = d_5^2 + F(x) \]
\[ F(k2) = d_6^2 + F(x2) \]

\[ R \text{ Sheet 5} \]
CTOSM
change representation of no. to signed mag

FLTMOV
Move input to w.s. minus exp

binary point = exp - bias + 1

LSHIFT
shift word 14 left

CTOSM
Change representation to 1's comp

Move
binary point
to index register 7

Exit

FLTFIX
Entry

Save index registers

FLTHOV
Move input to output+1

CTOSU
Change representation to signed mag

Clear sign bit and save

Set up exp in output
exp = 20017
no.words = 1 + no.words

FLTNORM
Normalize word

Restore sign

Restore index registers

CTOSU
Change representation back to 1's comp

Exit

FXTFLT
Entry

Save index registers

Shift last word end off

Shift=60-shift

Left shift from last word to 1st

Restore index registers

Exit

RSHIFTA
Sheet 2 of 2
ENTRY

CTOSM
Change representation to signed mag

A

SHIFT > 30

YES

K = shift - 30

SHIFT = 30

LSHIFTA
Shift left
S = 30

SHIFT = K

A

LSHIFTA
Shift < 30

CTOSM
Change representation

Exit

LSHIFT
Sheet 1 of 2
ENTRY

SAVE END ADDRESS INDEX REGISTERS

EXP = EXP-SHIFT

SHIFT 1ST WORD OVERFLOW OCCUR?

YES: FLTERRO

STARTING AT WORD 0 SHIFT ENTIRE WORD

INCLUDE EXP

RESTORE INDEX REGISTERS

EXIT

LSHIFTA
Sheet 2 of 2
Let: \( x = a_1 2^{-15} + a_2 2^{-30} + a_3 2^{-45} + \ldots + a_{n+1} 2^{-15(n+1)} \)
\( b_1 = a_1 2^{-15} \)
\( b_2 = a_2 2^{-30} \)
\( \vdots \)
\( b_{n+1} = a_{n+1} 2^{-15(n+1)} \)
FLTNORM
Sheet 2 of 2
C-10M

Entry

Save index registers and input parameters

A

X(1-N) = 0
s = 0
se = 0
d = 0

B

GET1
Input char(s),
to low order 6 bits of A

C

C = 77

Yes

End of message

G

Sheet 2

No

C = num

Yes

Part of input number

K

Sheet 3

No

C = ±

Yes

Sign of input number

L

Sheet 3

No

C = .

Yes

End of integer

M

Sheet 3

No

C = E

Yes

Exponent follows

E

Sheet 2

No

D

Set for error return

I

Sheet 3

=C = Input character
X = Generated number
N = Number of words occupied by X
n = Number of decimal places
s = Sign of number
se = Sign of exponent
e = Exponent
d = 0 if no decimal point
d = -0 if decimal point

NUMIN

Sheet 1 of 3
\[ x = x \cdot 10^E \]

\[ \text{Sign} = + \]

If \( \text{Sign} = + \), then:

- \( x = 10x + C' \)

If \( \text{Sign} = - \), then:

- Complement number
- Set sign bit negative
- CTOSM

\[ n = n + 1 \]

\[ \text{Convert C to FLTPT C'} \]

\[ \text{FLTNORM} \]

\[ n = 0 \]

\[ d = -0 \]

End of integer start count of fractional places

Set \( s = \pm \)

B Sheet 1

B Sheet 1

Exit

NUMIN
Sheet 3 of 3
This final report on Contract No. AF19(628)-5026 contains a summary of the five major tasks performed during the one-year duration of the contract. The work performed consisted of: 1) a variable-precision floating-point package for the solution of problems requiring very high precision, 2) extension of and improvements to the software system developed under an earlier contract, 3) assistance in the implementation and validation of a LISP Compiler, 4) development of a program for powerful manipulation of symbolic text (TECO), and 5) specification of a set of generalized display routines for visual communication with the computer. All work done related to the M-460 research computer at AFCRL.
Comari report number 8b, be unique to this report. selected so that no security classification subproject number, military department identification, such as project number, the report was written. The applicable number of the contract or grant number which ends with an indication 8a.

The total page count 13. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBERS REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and originating the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers (whether by the originator or by the sponsor), also enter this number(s).

10. AVAILABILITY LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classifications, etc., standard agreements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U.S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through DDG."

(4) "U.S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through DDG."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through DDG."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring (paying for) the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as sensitive, classified, (F), or (C).

There is no limitation on the length of the abstract. However, the suggested length is from 125 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as an index entry for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.