FORTRAN

COMMERCIAL

SUBROUTINE

PACKAGE

093-000107-00

Ordering No. 093-000107

Printed in the United States of America
Rev. 00, September 1974

NOTICE

Original Release September 1974

INTRODUCTION

While the typical FORTRAN program generally requires lengthy internal processing time using minimal input and output, the reverse is normally the case in the commercial and business environment. In these applications extensive manipulation of data is usually necessary and program input utilizes entire files of data rather than a few items of information. Additionally the output of commercial and business programs often results in extensively edited and formally structured output.

The Data General FORTRAN Commercial Subroutine Package is a set of subroutines and function subprograms expressly written to help the programmer overcome many of the inherent problems of using FORTRAN for many business and commercial applications (such as inventory control, payroll calculations, information updating and retrieval, etc.). For example, one of the features of the package enables the programmer to read data from an input device without the need of having to know the format of the data before it is read. This is a distinct advantage in some business problems, since it is not always practical to know the format of a record in advance (e.g., updating of a personnel file where input would consist of two or more different types of cards).

Several of the subroutines permit the manipulation of characters and character strings for the editing of output data into meaningful combinations of alphabetic, numeric, and special characters. Two of the subroutines also allow for comparison of data against other data for use in sorting of information prior to writing to a disk file or other output media.

Arithmetic operations (add, subtract, multiply, and divide) can be performed using variable-length decimal data fields and double-word integers. Because these subroutines operate with whole numbers, they overcome some of the problems encountered with extended precision values and exact representation of fractional numbers.

Other subroutines allow for the conversion of data into formats which permit faster computation and manipulation of information. The capability of packing data for more efficient use of storage media is also one of the important features of the package, since in many instances it may be necessary to compact the information because of the significant volume of data to be stored.

Finally there are special utility subroutines which permit data to be input from terminal devices and convert numeric data to real numbers. The transfer speed of data between memory and input/output devices is also substantially increased by using the subroutines in the package rather than the standard FORTRAN I/O subroutines.

The following Data General publications will assist users of the FORTRAN Commercial Subroutine Package:

093-000053	FORTRAN IV User's Manual
093-000068	FORTRAN IV Runtime Manual
093-000085	FORTRAN V User's Manual
093-000096	FORTRAN V Runtime Manual
093-000075	RDOS User's Manual
093-000087	BATCH User's Manual

TABLE OF CONTENTS

CHAPTER 1

GENERAL

INTRODUCTION

This manual consists of the documentation for the individual subroutines and function subprograms contained in the Data General FORTRAN Commercial Subroutine Package. These subroutines and function subprograms are categorized under four major groupings, according to their general use:

1. Data conversion subroutines for use in reformatting data for execution by other subroutines, and for packing and unpacking of both alphabetic and numeric information for more efficient use of storage media (e.g., tape and disk).

2. Data manipulation and comparison subroutines which enable the programmer to edit data for output, comparison of character strings, data moving, zone interrogation, and automatic right-justification of a data field within an array.

3. Add, subtract, multiply, and divide subroutines which offer the ability of operating on both variable-length decimal data and 32-bit signed binary integer numeric fields. Additionally, there are two subroutines in this subset which can be used for variable-length decimal data and 32-bit signed binary integer data comparison.

4. Input/Output subroutines are included which can read from, or write to, the system console, or print a line on the system printer. The QREAD subroutine enables the programmer to read a card from the system card reader without having to know, in advance, the format of the data to be read.

Of equal importance to the reader is APPENDIX B, which includes a series of five sample programs which may prove useful to the programmer, since the majority of subroutines and functions subprograms are utilized within the sample programs themselves.

SUBROUTINE DESCRIPTIONS

All subroutines are uniformly documented, and appear in alphanumeric sequence under one of the four major groupings (Chapters 3-6).

SUBROUTINE DESCRIPTIONS (Continued)

Each subroutine is described as follows:

1. SUBROUTINE NAME - The name of the subroutine or function subprogram as it is called in the statement format.

2. PURPOSE - A brief general description of the use of the subroutine.

3. STATEMENT FORMAT - The arguments in the CALL statement must agree in order, number, and type with the corresponding arguments in the subroutine. A number may be passed to a subroutine either as an integer constant, expression or variable.

4. ARGUMENT DESCRIPTIONS - Describe each argument used in the FORTRAN CALL statement. Special note must be taken that any array used as an argument in a FORTRAN Commercial Subroutine CALL statement or function subroutine statement must have been defined in a prior DIMENSION statement.

5. GENERAL USAGE NOTES - Describe the special considerations to be taken in using the subroutine.

6. PROGRAM ERROR DESCRIPTIONS - Outline any special error conditions which may occur during execution of the subroutine.

7. EXAMPLE - Shows the use of the subroutine.

DATA FORMATS

In the majority of cases, the subroutines operate on data in A1 format (one ASCII character per word). There are, however, subroutines which operate on data in other formats. For this reason, a special chapter (Chapter 2), has been included which describes all the formats used with the subroutines and function subprograms. Additionally, this chapter gives a general overview of integer, real, and double precision data.

USING THE COMMERCIAL SUBROUTINE PACKAGE WITH DATA GENERAL FORTRAN

The following paragraphs briefly describe features of Data General FORTRAN which may be of special interest to users of the Commercial Subroutine Package. For more detailed description of Data General FORTRAN, refer to the appropriate FORTRAN manual.

Arithmetic

Standard Data General FORTRAN arithmetic provides for 16-bit integers (± 32,767), single precision floating point (32 bits), and double precision floating point numbers (64 bits). Full mixed mode is permitted in expressions and assignment statements. A COMPILER DOUBLE PRECISION statement is provided which automatically treats all real variables as double precision.

Input/Output

Operating under the RDOS Operating System, the FORTRAN programmer has access to up to 63 files and/or devices simultaneously. Files and devices are opened explicitly and implicitly by the FORTRAN program, thereby avoiding complicated set-up procedures.

In addition to standard FORTRAN I/O, the programmer may create, seek, and access files randomly and may perform block I/O. These facilities, in addition to binary I/O, are extremely useful in applications having heavy use of disk files.

Data General FORTRAN also provides two statements which facilitate console I/O. The TYPE and ACCEPT allow the user to output and input data to the system console in a free formatted mode.

Multitasking

Multitasking provides an advanced method of having many processes or tasks executing within the processor asynchronously.

Assume, for example, a user wishes to collect data from a number of terminals, process the data, and prepare output reports. Using the multitasking facility, the program can accept data from a terminal (a low-speed device) while processing data from other terminals. The data processing tasks will be executing while the data collection process is awaiting input. This multitasking facility results in high throughput of data.

Program Segmentation

Program segmentation allows the operation of programs that are too lengthy to run in available core space. These programs, written in executable segments, are stored in core image format and are brought into the user area at execution time. Each segment calls the next segment until the complete program has been executed.

Program Segmentation (Continued)

With program segmentation, programs of virtually unlimited size can be run on the system.

There are three methods of program segmentation, namely, chaining, swapping, and user overlays. Chaining will replace the calling program with another program, completely overwriting the calling program. Swapping occurs when the calling program calls another program from disk into core; the calling program is swapped out to disk to await a call which will bring it back into core. One or more overlays are stored in an overlay file and are brought into core only when necessary for execution. Overlays overwrite each other but do not overwrite any portion of the main program. The overwriting of overlays occurs only when one overlay is finished executing and another overlay is brought into core; the second overlay will overwrite the first, and so on.

CHAPTER 2

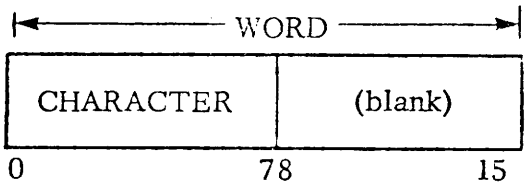
DATA FORMATS

In general the subroutines in the FORTRAN Commercial Subroutine Package operate on data in A1 format. There are, however, some subroutines that must employ other data formats for their execution.

The following is a description of the data formats which are used with the FORTRAN Commercial Subroutine Package. Chapter 3 details the conversion routines which convert data between A1 format and the other formats.

A1 FORMAT

One character is stored in each 16-bit word. Bits 0-7 contain the character; bits 8-15 are set to blank (32_{10} or 40_8).

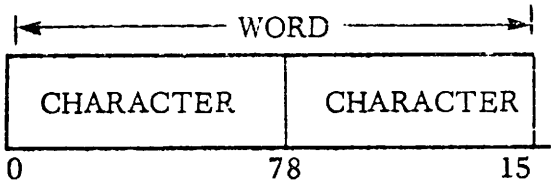


Numeric quantities may be represented in A1 Format by use of a field of decimal characters. Conversion routines are provided to convert these characters into D1 or Double Word Format for arithmetic manipulation. A negative quantity is represented by use of an 11-zone punch over the last decimal character in the field. This has the effect of transforming the low-order decimal character into another ASCII character, indicated in the table below:

DIGIT	CHARACTER
0	—
1	J
2	K
3	L
4	M
5	N
6	O
7	P
8	Q
9	R

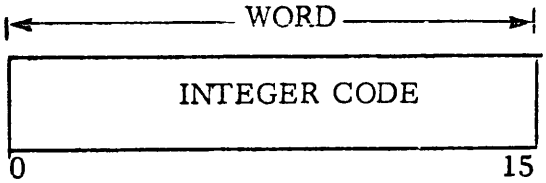
A2 FORMAT

Two characters are stored in each 16-bit word. Bits 0-7 contain the first character; bits 8-15 contain the second character.



A3 FORMAT

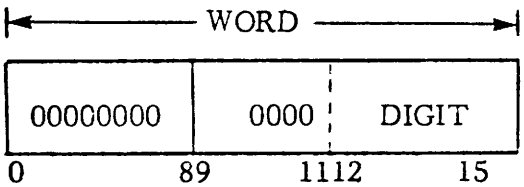
Three characters are stored in each 16-bit word. The characters are converted to a special integer code using the formula in the A1A3 data conversion subroutine.



NOTE: There are restrictions to using this format for packing data. Only 40 characters of the entire ASCII Character Set can be used. A discussion of these restrictions is contained in A1A3 data conversion subroutine description in Chapter 3.

D1 FORMAT

One decimal digit is stored in each 16-bit word. Bits 0-11 are set to zero, and bits 12-15 will contain the digit. If the numeric field is negative, the right-most digit is set as described in the NOTE on the following page.



D1 FORMAT (Continued)

NOTE: There is a special consideration for the programmer in setting up negative constants using the D1 format. The computer cannot represent a negative zero (-0). Because of this restriction a negative zero (-0) is carried internally as -1, a negative one (-1) as -2, etc. (See table below).

If the field is negative and the right-most digit is:	The internal representation of the right-most digit in D1 format will be:
0	-1
1	-2
2	-3
3	-4
4	-5
5	-6
6	-7
7	-8
8	-9
9	-10

EXAMPLES:

1. +12345 is stored as 12345 (one digit per word)
2. -12345 is stored as 1234-6 (one digit per word)

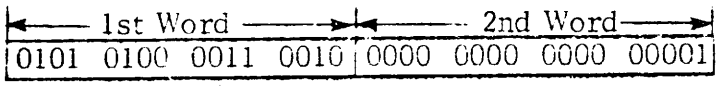
D4 FORMAT

Four decimal digits are stored in each 16-bit word; each digit is represented by 4 bits. The sign of the field is placed separately in the last word in the D4 field. Any unused portion of a word is filled with 1 bits (Hexadecimal F) to the right of the digit.

D4 FORMAT (Continued)

EXAMPLES:

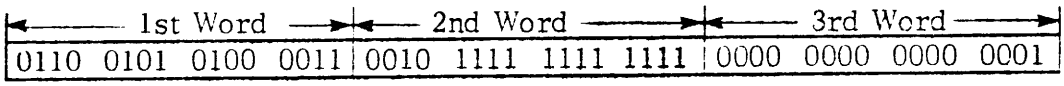
+54321=



(represents)

5 4 3 2 +1

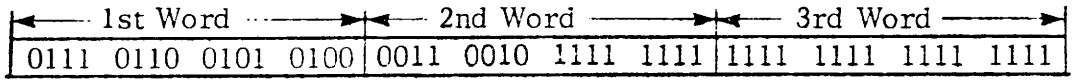
+654321=



(represents)

6 5 4 3 2 F F F +1

-7654321=

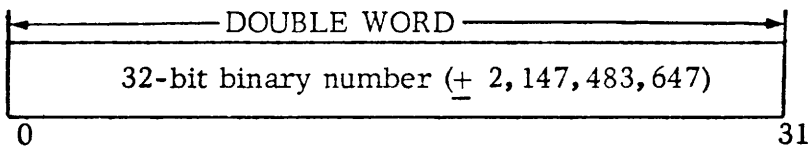


(represents)

7 6 5 4 3 2 F F -1

DOUBLE WORD FORMAT

The Double Word Format contains a 32-bit binary number (maximum ± 2,147,483,647)



DOUBLE WORD REFERENCING

A Double Word quantity is contained in two 16-bit words and referenced by the address of the leftmost word. Therefore, if a Double Word is contained in words three and four of integer array I, it is referenced as I (3).

Additionally, a double word integer may be referenced as a real or double precision variable or array element. Examples of referencing a double word integer are:

```
CALL  DWADD(IARY(3),IARY(5))
```

ADDS the double word contained in array elements 3 and 4 to the double word contained in words 5 and 6.

```
CALL DWADD(VAR1,VAR2)
```

ADDS the double word contained in real variable VAR1 to the double word contained in real variable VAR2.

INTEGER DATA

An integer constant is a signed or unsigned whole number written without a decimal point.

An integer variable is usually implicitly typed, i.e., if the first character of the symbolic name is I, J, K, L, M, or N, the symbolic name represents and integer variable unless otherwise specified. Examples of integer constants and variables are:

Constants	Variables
-125	ITEM
0	JOBNO
+4525	LUCKY
377K	MASKBYTE

INTEGER DATA (Continued)

As shown, integer constants can be specified in octal format by writing the number followed by the letter K. Some additional examples are:

Octal Constant	Decimal Value
10K	8
777K	511
-1K	-1

An integer datum is stored in one word (16 bits). The range of integer value is -X -32,767 to 32,767 exclusive.

REAL DATA

A real constant is signed or unsigned and consists of one of the following:

1) One or more decimal digits written with a decimal point.

2) One or more decimal digits written with or without a decimal point, following by a decimal exponent written as the letter E followed by a signed or unsigned integer constant. When the decimal point is omitted, it is always assumed to be immediately to the right of the right-most digit. The exponent value may be explicitly 0; the exponent field may not be blank.

A real variable is usually implicitly typed. If the first character of the symbolic name is not I, J, K, L, M, or N the symbolic name represents a real variable unless otherwise specified.

Constants	Constant Value	Variables
0.0	0.0	ALPHA
.000056789	.000056789	B25
-15.E-04	+.0015	EXIT
-005E2	-500	C

A real datum is stored in two 16-bit words.

DOUBLE PRECISION DATA

A double precision constant is signed or unsigned and consists of the following:

> A sequence of decimal digits written with or without a decimal point, followed by a decimal exponent written as the letter D followed by a signed or unsigned integer constant. When the decimal point of a double precision constant is omitted, it is always assumed to be immediately to the right of the right-most digit. The exponent value may be explicitly 0; the exponent field may not be blank.

A double precision variable must be explicitly specified as such in a DOUBLE PRECISON type statement. *

Constants	Constant Value
-21987654321D0	-21987654321
5.0D-3	.005
.203D0	.203

Variable Type Statement

```
DOUBLE PRECISON D, E, F2
```

* If the first statement of the FORTRAN program is: "COMPILER DOUBLE PRECISON" each real variable or constant will be forced to type DOUBLE PRECISON.

CHAPTER 3

DATA CONVERSION SUBROUTINES

This Chapter describes the two major types of data conversion subroutines available in the FORTRAN Commercial Subroutine Package:

1. Routines which reformat data for execution by other subroutines. For example, the decimal arithmetic subroutines (QADD, QDIV, QMUL, and QSUB) operate on numeric data in D1 (decimal) format only. Since data can be stored in several types of formats, the programmer can, by using the data conversion subroutines described in this Chapter, convert numeric data from one format to another.

2 The second type of conversion routine permits the packing and unpacking of alphabetic and numeric data for more efficient use of storage media.

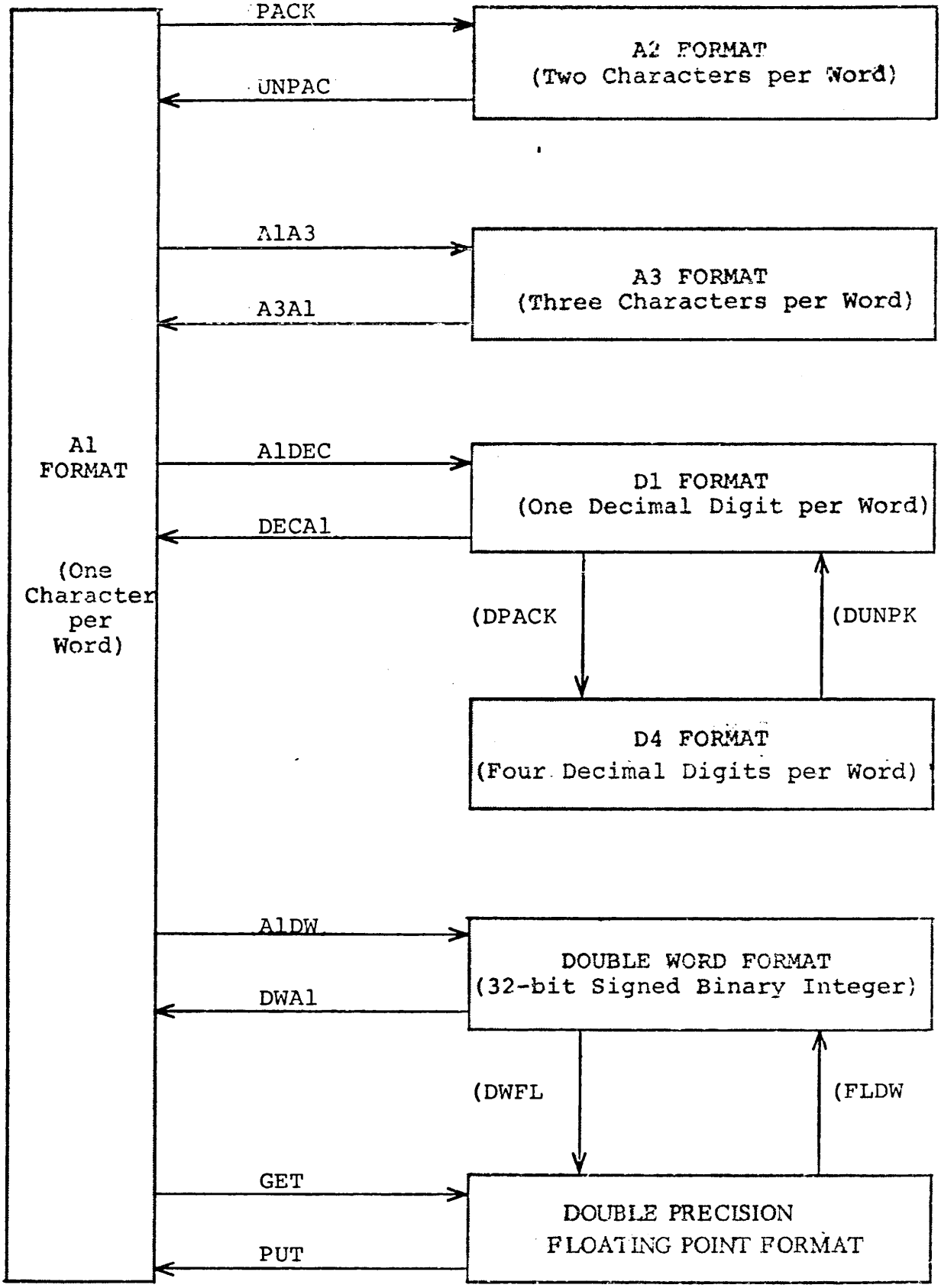
The following is a brief description of the function of each subroutine described in this Chapter, listed in alphanumeric sequence by Subroutine Name.

Subroutine Name	Purpose
A1A3	Data in A1 format (one character per word, left-justified) is converted to A3 format (three characters per word).
A1DEC	Numeric data in A1 format (one digit per word, left-justified) is converted to D1 (decimal) format (one digit per word, right-justified).
A1DW	Numeric data (maximum of $\pm$ 2,147,483,647) in A1 format (one character per word, left-justified) is converted into a double word binary integer.
A3A1	Data in A3 format (three characters per word) is converted to A1 format (one character per word, left-justified).
DECA1	Data in D1 (decimal) format (one digit per word, right-justified) is converted to A1 format (one character per word, left-justified).
DPACK	Numeric data in D1 (decimal) format (one digit per word, right-justified) is converted to D4 format (four digits per word).

DATA CONVERSION SUBROUTINES (Continued)

Subroutine Name	Purpose
DUNPK	Numeric data in D4 format (four digits per word) is converted to D1 format (one digit per word, right-justified).
DWA1	Numeric data represented as a double word binary integer is converted to A1 format (one character per word, left-justified).
DWFL	Converts a signed double word integer into a double precision floating point number.
FLDW	Converts the whole portion of a double precision floating point number into Double Word integer format.
GET	Permits numeric data to be decoded after it has been read, which allows input records to be entered in an unknown sequence. Converts data in A1 format (one digit per word, left-justified) to real numbers.
PACK	Data in A1 format (one character per word, left-justified) is converted to A2 format (two characters per word).
PUT	Converts the whole portion of a Double Precision variable to an ASCII string in A1 format (one character per word, left-justified). The number is half-adjusted and truncated as specified by the programmer.
UNPAC	Data in A2 format (two characters per word) is converted to A1 format (one character per word, left-justified).

SUMMARY OF DATA FORMAT CONVERSION PATHS



SUBROUTINE NAME: A1A3

PURPOSE: This subroutine converts data in a one-dimensional integer array from A1 format (one character per word, left-justified), placing the converted data into a second array in A3 format (three characters per word). The data in the first array remains unchanged during execution of the subroutine.

STATEMENT FORMAT: CALL A1A3(ICON, IFRST, ILST, MCVTD, MFRST, NTBL)

ARGUMENT DESCRIPTIONS

ICON - The name of the one-dimensional integer array that contains the data to be converted. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine, the data to be converted must be in A1 format (one character per word, left-justified). During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression or variable that identifies the position of the first character (left-most position) in the ICON array to be converted.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the ICON array to be converted.

MCVTD - The name of the one-dimensional integer array that will contain the converted data. This array must have been defined in a prior DIMENSION statement. After execution of the subroutine the converted data will be in A3 format (three characters per word).

MFRST - An integer constant, expression, or variable that identifies the position of the first converted three-character element (left-most position) in the MCVTD array.

NTBL - The name of a one-dimensional integer array to be used as a conversion table. This array must have been defined in prior DIMENSION statement. The contents of this array are not altered by the subroutine.

GENERAL USAGE RULES:

It is the programmer's responsibility to create the NTBL array. The table must be composed of 40 characters. Any 40 characters may be used, placed in any desired sequence within the array; the NTBL array must contain a blank. It is also advisable to place those characters most frequently used in the conversion at the beginning of the array.

Sample of NTBL content:

```
NTBL = △0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ,.&
       ↑  ↑  ↑  ↑  ↑  ↑  ↑  ↑  ↑
(Relative
Position)  0  4  9  14 19 24 29 34 39
```

The subroutine uses the following method for conversion:

1. The relative position of each character in a three character group is found in NTBL. (Relative positions in the NTBL array are from 0-39).

2. The subroutine then uses the following formula to code each three-character word:

   A3 word = (R1-20)*1600+(R2*40)+R3

   Where:

   R1=relative position of the first character in NTBL,
   R2=relative position of the second character in NTBL,
   R3=relative position of the third character in NTBL.

The programmer can create and input NTBL by using one of the following methods shown below:

```
DIMENSION NTBL(40)
READ(2,1)NTBL
FORMAT(40A1)
    .
   (or)
DIMENSION NTBL(40)
CALL QREAD(NTBL,1,40)        ;INPUT FROM CARD READER
    .
   (or)
```

GENERAL USAGE RULES: (Continued)

```
DIMENSION NTBL(40)
CALL KEYBD(NTBL,1,40)          ;INPUT FROM KEYBOARD
```

The NTBL array can also be data initialized. In order to initialize NTBL, it must be named in a common block and must be initialized in A1 format (one character per word, left-justified).

Examples:

```
COMMON/WORKA/NTBL(40)
DATA NTBL/"ΔΔ","1Δ","2Δ",".....(etc)....."/
    .
    .
(or)

COMMON/WORKA/NTBL(40)
DATA NTBL/"ΔΔ1Δ2Δ3Δ.....(etc)..........."/
```

Conversion speed improvements can be accomplished by careful layout of NTBL with the most frequently used characters placed at the beginning of the NTBL array.

PROGRAM ERROR DESCRIPTIONS:

If, during conversion, a character is encountered in the ICON array which has not been specified in the NTBL array, the subroutine will treat the character in the ICON array as a blank (Δ).

EXAMPLE

```
      DIMENSION  ICON(21),MCVTD(7),NTBL(40)
          .
          .
          .
      CALL  A1A3(ICON,1,21,MCVTD,1,NTBL)
```

1. CONTENTS OF ICON, MCVTD, AND NTBL BEFORE EXECUTION OF THE SUBROUTINE:

```
          ICON  = THIS△IS△A△SAMPLE△1234
                  ↑    ↑    ↑    ↑    ↑
    (WORD
    POSITION )    1    5    10   15   20

         MCVTD  = ABCDEFG
                  ↑    ↑
    (WORD
    POSITION)     1    5

          NTBL  =  △0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ,.&
                   ↑    ↑    ↑    ↑    ↑    ↑    ↑    ↑    ↑
    (WORD
    POSITION)      1    5    10   15   20   25   30   35   40
```

2. CONTENTS OF ICON, MCVTD, AND NTBL AFTER EXECUTION OF THE SUBROUTINE:

```
          ICON  = UNCHANGED

         MCVTD  = +16739+14419+14411-30829 +5862 -7998-27035
                    ↑     ↑     ↑     ↑     ↑     ↑     ↑
    (WORD
    POSITION)       1     2     3     4     5     6     7

    REPRESENTS      THI   S△I   S△A   △SA   MPL   E△1   234

          NTBL  =  UNCHANGED
```

SUBROUTINE NAME: A1DEC

PURPOSE: This subroutine converts numeric data in a one-dimensional integer array from A1 format (one character per word, left-justified), to D1 format (one digit per word, right-justified), placing the converted data into the original array.

STATEMENT FORMAT: CALL A1DEC(ICON,IFRST,ILST,NINV)

ARGUMENT DESCTIPTIONS:

ICON - The name of the one-dimensional integer array that contains the numeric data to be converted. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data to be converted must be in A1 format (one character per word, left-justified). After execution of the subroutine the converted data will be contained in this array in D1 format (one digit per word, right-justified).

IFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the ICON array to be converted.

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the ICON array to be converted.

NINV - An integer variable that can be tested after conversion to determine if any invalid characters (non-blanks or non-numerics) were encountered (except for ILST, which can contain a sign).

GENERAL USAGE RULES:

The task of initializing, testing, and resetting of the NINV indicator is the responsibility of the programmer.

PROGRAM ERROR DESCRIPTIONS:

The first non-blank (blanks are changed to zero) or non-numeric character encountered during conversion will force the NINV indicator to be set to the position of the invalid character and conversion of the field will continue. If additional invalid characters are encountered the NINV indicator will be set to the position of that character (NINV will always contain the position of the last invalid character even though other invalid characters may exist to the left of the field being converted).

Note that negative quantities are represented by an 11 zone punch over the last character in the field.

The NINV indicator will also be set if a negative number appears in any position except the last.

EXAMPLE

```
DIMENSION  ICON(15)
     .
     .
     .
NINV=0
CALL A1DEC(ICON,3,11,NINV)
```

1. CONTENTS OF ICON AND NINV BEFORE EXECUTION OF THE SUBROUTINE:

```
        ICON  = X△Y△0△2△1△6△3△1△0△4△4△A△B△C△D△
                ↑       ↑       ↑       ↑
(WORD
POSITION )      1       5       10      15

        NINV  = 0
```

2. CONTENTS OF ICON AND NINV AFTER EXECUTION OF THE SUBROUTINE:

```
        ICON  = X△Y△0002010603010004044△B△C△D△
                ↑       ↑       ↑       ↑
(WORD
POSITION)       1       5       10      15

        NINV  = 0  (NO NON-BLANK OR NON-NUMERIC CHARACTERS)
```

SUBROUTINE NAME: A1DW

PURPOSE: This subroutine converts a numeric field in a one-dimensional integer array from A1 format (one character per word, left-justified), into Double Word format (32-bit signed binary integer).

STATEMENT FORMAT:

CALL A1DW (ICON, IFRST, ILST, DWORD, NCHK)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the numeric field to be converted. This array must have been defined in a prior DIMENSION statement. Before execution of this subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the ICON array to be converted.

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the ICON array to be converted.

DWORD - - The double word that will contain the converted numeric field. After execution of the subroutine the converted field will be in Double Word format (32-bit signed binary integer).

NCHK - An integer variable that can be tested after conversion to determine in any invalid characters were encountered during the conversion process.

GENERAL USAGE RULES:

1. The task of initializing, testing, and resetting of the NCHK indicator is the responsibility of the programmer.

2. A double word integer occupies two contiguous words of storage and may be referenced as a real variable, real array element, or two words of an integer array (addressed by the first word [left-most position]).

PROGRAM ERROR DESCRIPTIONS.

If the field being converted contains any non-numeric character other than blank, the conversion will be terminated and the NCHK indicator will be set to the position of the invalid character. Note that negative quantities are represented by an 11 zone punch over the last character in the field.

EXAMPLE

```
      DIMENSION ICON(9),IWORD(10)
           .
           .
           .
      CALL A1DW(ICON,1,6,IWORD(3),NCHK)
```

1. CONTENTS OF ICON, MCVTD, AND NCHK BEFORE EXECUTION OF THE SUBROUTINE:

```
          ICON = 123456789
                 ↑    ↑
    (WORD
    POSITION )   1    5

          IWORD = 0000000000
                  ↑    ↑
    (WORD
    POSITION)    1    5     10

          NCHK  = 0
```

2. CONTENTS OF ICON, IWORD, AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

```
          ICON = 123456789 (UNCHANGED)

          IWORD = 00(3 AND 4 CONTAIN BINARY INTEGER 123456)000000
                  ↑                                      ↑     ↑
    (WORD
    POSITION)    1                                      5     10

          NCHK = 0 (UNCHANGED)
```

SUBROUTINE NAME: A3A1

PURPOSE: This subroutine converts data in a one-dimensional integer array from A3 format (three characters per word), placing the converted data into a second array in A1 format (one character per word, left-justified). The data in the first array remains unchanged during execution of the subroutine.

NOTE: The Data General FORTRAN Commercial Subroutine Package A3 format is not the same as the standard FORTRAN format. Data in FORTRAN Commerical Subroutine Package A3 format has been converted from A1 format using the CSP A1A3 subroutine.

STATEMENT FORMAT:

```
CALL A3A1(ICON,IFRST,ILST,MCVTD,MFRST,NTBL)
```

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the data to be converted. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data to be converted must be in A3 format (three characters per word). During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first three-character element (left-most position) in the ICON array to be converted.

ILST - An integer constant, expression, or variable that identifies the position of the last three-character element (right-most position) in the ICON array to be converted.

MCVTD - The name of the one-dimensional integer array that will contain the converted data. This array must have been defined in a prior DIMENSION statement. After execution of the subroutine the converted data will be in A1 format (one character per word, left-justified).

ARGUMENT DESCRIPTIONS: (Continued)

MFRST - An integer constant, expression, or variable that identifies the position of the first converted character (left-most position) in the MCVTD array.

NTBL - The name of a one dimensional integer array to be used as a conversion table. This array must have been defined in a prior DIMENSION statement. The contents of this array are not altered by the subroutine.

NOTE: This array must be identical to the array used in converting the data in ICON from the original A1 format to A3 format.

GENERAL USAGE RULES:

It is the programmer's responsibility to create the NTBL array. The table must be identical to the NTBL array used in converting the data in ICON from its original A1 format (see the explanation of the CSP A1A3 subroutine for further information).

Sample of NTBL content:

NTBL = △0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ,.&

(Relative Position)	0	4	9	14	19	24	29	34	39

The subroutine uses the following method for converting a one-word three-character element from ICON into three words in MCVTD:

CHAR1 = A3 ELEMENT/1600 + 20

IR A3 ELEMENT IS POSITIVE

(A3 ELEMENT 32000)/1600

IR A3 ELEMENT IS NEGATIVE

CHAR2 = (A3 ELEMENT - (CHAR1-20)*1600)/40

CHAR3 = A3 ELEMENT - (CHAR1-20)*1600

- CHAR2 *40

The result of each computation is used to find the character value in the NTBL array.

PROGRAM ERROR DESCRIPTION

If the ILST argument is less than the IFRST argument the subroutine will convert only the first character of the three-character element, placing that character in three separate words.

EXAMPLE

```
      DIMENSION  ICON(7),MCVTD(21),NTBL(40)
           .
           .
           .
      CALL  A3A1(ICON,1,7,MCVTD,1,NTBL)
```

1. CONTENTS OF ICON, MCVTD, AND NTBL BEFORE EXECUTION OF THE SUBROUTINE:

```
          ICON = +16739+14419+14411-30829 +5862 -7998-27035
                    ↑     ↑     ↑     ↑     ↑     ↑     ↑
  (WORD
  POSITION)         1     2     3     4     5     6     7

 REPRESENTS       THI   S I   S A   SA   MPL   E 1   234

          MCVTD = ABCDEFGHIJKLMNOPQRSTU
                  ↑    ↑     ↑     ↑     ↑
  (WORD
  POSITION)       1    5     10    15    20

          NTBL = △0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ,.$
                 ↑    ↑     ↑     ↑     ↑     ↑     ↑     ↑     ↑
  (WORD
  POSITION)      1    5     10    15    20    25    30    35    40
```

2. CONTENTS OF ICON, MCVTD, AND NTBL AFTER EXECUTION OF THE SUBROUTINE:

```
          ICON = UNCHANGED

          MCVTD = THIS△IS△A△SAMPLE△1234
                  ↑    ↑     ↑     ↑     ↑
  (WORD
  POSITION)       1    5     10    15    20

          NTBL = UNCHANGED
```

SUBROUTINE NAME: DECA1

PURPOSE: This subroutine converts numeric data in a one-dimensional integer array from D1 format (one digit per word, right-justified), to A1 format (one character per word, left-justified), placing the converted data into the original array.

STATEMENT FORMAT:

```
CALL DECA1(ICON,IFRST,ILST,NINV)
```

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the numeric data to be converted. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data to be converted must be in D1 format (one digit per word, right-justified). After execution of the subroutine the converted data will be contained in this array in A1 format (one character per word, left-justified).

IFRST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the ICON array to be converted).

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the ICON array to be converted.

NINV - An integer variable that can be tested after conversion to determine if any invalid characters were found during conversion.

GENERAL USAGE RULES:

The task of initializing, testing, and resetting of the NINV indicator is the responsibility of the programmer.

PROGRAM ERROR DESCRIPTIONS:

The first non-numeric character or negative number encountered during conversion (except for the content of the position designated by ILST, which can be a negative number), will force the NINV indicator to be set to the position of the invalid character, and conversion will continue. If any additional invalid characters are found, the NINV indicator will be set to that position (NINV will always contain the position of the last invalid character even though other invalid characters might have been found during the conversion process).

EXAMPLE
=======

```
DIMENSION  ICON(15)
     .
     .
NINV=0
CALL DECA1(ICON,3,11,NINV)
```

1. CONTENTS OF ICON AND NINV BEFORE EXECUTION OF THE SUBROUTINE:

```
        ICON  = X△Y△0002010603010004044A△B△C△D△
                ↑       ↑          ↑          ↑
(WORD
POSITION )      1       5          10         15

        NINV  = 0
```

2. CONTENTS OF ICON AND NINV AFTER EXECUTION OF THE SUBROUTINE:

```
        ICON  = X△Y△0△2△1△6△3△1△0△4△4△A△B△C△D△
                ↑       ↑          ↑          ↑
(WORD
POSITION)       1       5          10         15

        NINV  = 0  (NO INVALID CHARACTERS)
```

SUBROUTINE NAME: DPACK

PURPOSE: This subroutine converts numeric data in a one-dimensional integer array from D1 format (one digit per word, right-justified), packing the converted numeric data into a second array in D4 format (four digits per word). The data in the first array remains unchanged during execution of the subroutine.

STATEMENT FORMAT:

CALL DPACK(ICON,IFRST,ILST,MCVTD,MFRST)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the field of digits to be packed. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the digits to be converted must be in D1 format (one digit per word, right-justified). During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the ICON array to be packed.

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the ICON array to be packed.

MCVTD - The name of the one-dimensional integer array that will contain the packed digits. This array must have been defined in a prior DIMENSION statement. After execution of the subroutine the packed digits will be in D4 format (four digits per word).

MFRST - An integer constant, expression, or variable that identifies the position of the first packed four-digit element (left-most position) in the MCVTD array.

GENERAL USAGE RULES:

1. The subroutine assumes that the contents of the word designated by the ILST argument (in the ICONV array) contains the sign of the field to be packed, and will place this digit unpacked into the last word in the MCVTD array.

2. If the ILST argument is less than or equal to IFRST only one digit will be packed (and will be treated as a sign).

   The following table is supplied as an aid in determining the length of the receiving array:

LENGTH		LENGTH		LENGTH	
WORDS BEFORE PACKING	WORDS AFTER PACKING	WORDS BEFORE PACKING	WORDS AFTER PACKING	WORDS BEFORE PACKING	WORDS AFTER PACKING
2	2	18	6	34	10
3	2	19	6	35	10
4	2	20	6	36	10
5	2	21	6	37	10
6	3	22	7	38	11
7	3	23	7	39	11
8	3	24	7	40	11
9	3	25	7	41	11
10	4	26	8	42	12
11	4	27	8	43	12
12	4	28	8	44	12
13	4	29	8	45	12
14	5	30	9	46	13
15	5	31	9	46	13
16	5	32	9	48	13
17	5	33	9	49	13

3. The length of the MCVTD array is determined by the subroutine using the following formula:

$$\text{MCVTD length} = \frac{\text{ILST-IFRST+7}}{4}$$

(Answer rounded down)

EXAMPLE:

```
CALL DPACK(ICON, 1, 6, MCVTD, 1)
```

```
BEFORE:                    AFTER:

ICON = 123456              MCVTD = 12345FFF0006
          ↑   ↑                      ↑   ↑   ↑
(Word     1   5            (Word     1   2   3
position)                  position)
```

EXAMPLE

```
DIMENSION  ICON(13),MCVTD(6)
     .
     .
     .
CALL DPACK(ICON,1,12,MCVTD,1)
```

1. CONTENTS OF ICON AND MCVTD BEFORE EXECUTION OF THE SUBROUTINE:

```
       ICON = 2163104400321
              ↑   ↑    ↑
(WORD
POSITION )    1   5    10

      MCVTD = ABCDEF
              ↑   ↑
(WORD
POSITION)     1   5
```

2. CONTENTS OF ICON AND MCVTD AFTER EXECUTION OF THE SUBROUTINE:

```
       ICON = UNCHANGED

      MCVTD = 2163104400320001E△F△
              ↑   ↑   ↑   ↑   ↑ ↑
(WORD
POSITION)     1   2   3   4   5 6
```

SUBROUTINE NAME: DUNPK

PURPOSE: This subroutine converts numeric data in a one-dimensional integer array from D4 format (four digits per word), unpacking the converted numeric data into a second array in D1 format (one digit per word, right-justified). The data in the first array remains unchanged during execution of the subroutine.

STATEMENT FORMAT:

CALL DUNPK(ICON,IFRST,ILST,MCVTD,MFRST)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the field of digits to be unpacked. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the digits to be unpacked must be in D4 format (four digits per word). During execution of the subroutine the data remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first four-digit element (left-most position) in the ICON array to be unpacked.

ILST - An integer constant, expression, or variable that identifies the position of the last four-digit element (right-most position) in the ICON array to be unpacked.

MCVTD - the name of the one-dimensional integer array that will contain the unpacked digits. This array must have been defined in a prior DIMENSION statement. After execution of the subroutine the unpacked digits will be in D1 format (one digit per word, right-justified).

MFRST - An integer constant, expression, or variable that identifies the position of the first unpacked digit (left-most position) in the MCVTD array.

GENERAL USAGE RULES:

1. The length of the MCVTD array is determined by the subroutine using the following formula:

   MCVTD length = 4 x (ILST-IFRST) +1

   NOTE: Since the DPACK subroutine fills in some of the four-bit fields with one bits (designated by "F"), the exact length of MCVTD may be up to a maximum of three words shorter than the answer to the calculation (see the CSP DPACK subroutine for a further explanation).

2. The contents of the word designated by the ILST argument is treated as the sign of the field and is moved into the MCVTD array without change.

EXAMPLE
=======

```
DIMENSION  ICON(13),MCVTD(16)
     .
     .
     .
CALL DUNPK(ICON,1,4,MCVTD,1)
```

1. CONTENTS OF ICON AND MCVTD BEFORE EXECUTION OF THE SUBROUTINE:

```
       ICON = 2163104400320001ABCDEFGHI
              ↑   ↑   ↑   ↑   ↑    ↑
(WORD
POSITION )    1   2   3   4   5    10

       MCVTD = ABCDEFGHIJKLMNOP
               ↑   ↑    ↑    ↑
(WORD
POSITION)     1   5    10   15
```

2. CONTENTS OF ICON AND MCVTD AFTER EXECUTION OF THE SUBROUTINE:

```
       ICON = UNCHANGED

       MCVTD = 216310440032INOP
               ↑   ↑    ↑    ↑
(WORD
POSITION)     1   5    10   15
```

SUBROUTINE NAME: DWA1

PURPOSE. This subroutine converts a 32-bit signed binary integer in Double Word format to A1 format (one character per word, left-justified).

STATEMENT FORMAT:

CALL DWA1 (DWORD, MCVTD, MFRST, MLST)

ARGUMENT DESCRIPTIONS:

DWORD - The 32-bit signed binary integer in Double Word format that is to be converted to A1 format. During execution of the subroutine this data remains unchanged.

MCVTD - The name of the one-dimensional integer array that will contain the converted 32-bit signed binary integer. This array must have been defined in a prior dimension statement. After execution of the subroutine the data in this array will be in A1 format.

MFRST - An integer constant, expression, or variable that identifies the position of the first converted digit (left-most position) in the MCVTD array.

MLST - An integer constant, expression, or variable that identifies the position of the last converted digit (right-most position) in the MCVTD array.

GENERAL USAGE RULES:

A double word integer occupies two contiguous words of storage and may be referenced as a real variable, real array element, or two words of an integer array (addressed by the first word [left-most position]).

EXAMPLE

```
      DIMENSION IWORD(10),MCVTD(10)
           .
           .
           .
      CALL DWA1(IWORD,MCVTD,1,10)
```

1. CONTENTS OF IWORD AND MCVTD BEFORE EXECUTION OF THE SUBROUTINE:

```
          IWORD  = (1 AND 2 CONTAIN BINARY INTEGER 123456)00000000
                    ↑                                     ↑     ↑
    (WORD
    POSITION )      1                                     5    10

          MCVTD  = ABCDEFGHIJ
                    ↑   ↑    ↑
    (WORD
    POSITION)       1   5    10
```

2. CONTENTS OF IWORD AND MCVTD AFTER EXECUTION OF THE SUBROUTINE:

```
          IWORD  = UNCHANGED

          MCVTD  = 0000123456
                    ↑   ↑    ↑
    (WORD
    POSITION)       1   5    10
```

NOTE THAT THE CONVERTED FIELD IS RIGHT-JUSTIFIED AND FILLED TO THE LEFT OF THE MOST SIGNIFICANT DIGIT WITH ZEROS.

SUBROUTINE NAME: DWFL

PURPOSE: This subroutine converts a 32-bit signed binary integer into a double precision floating point number.

STATEMENT FORMAT:

CALL DWFL (DWORD, DPVAR)

ARGUMENT DESCRIPTIONS:

DWORD - The 32-bit signed binary integer that is to be converted to a double precision floating point number. Before execution of the subroutine this data must be in Double Word format. During execution of the subroutine this data remains unchanged.

DPVAR - The double precision floating point number after conversion of DWORD.

GENERAL USAGE RULES:

The double precision variable name must appear in a double precision specification statement.

EXAMPLE

```
DIMENSION IA(2)
DOUBLE PRECISION D1
CALL DWFL (IA,D1)
```

1. BEFORE EXECUTION OF THE SUBROUTINE:

   IA = 32-BIT SIGNED BINARY INTEGER 109873

2. AFTER EXECUTION OF THE SUBROUTINE:

   D1 = 109873.00

SUBROUTINE NAME: FLDW

PURPOSE: This subroutine converts the whole portion of a double precision floating point number into a 32-bit signed binary integer.

STATEMENT FORMAT:

CALL FLDW (DPVAR, DWORD)

ARGUMENT DESCRIPTIONS:

DPVAR - A double precison constant variable or expression whose whole portion is to be converted to a 32-bit signed binary integer.

DWORD - After conversion this will contain the whole portion of DPVAR as a 32-bit signed binary integer.

GENERAL USAGE RULES:

1. The double precision variable name must appear in a double precision specification statement.

2. A double word integer occupies two contiguous words of storage and may be referenced as a real variable, real array element, or two words of an integer array (addressed by the first word [left-most position]).

EXAMPLE
=======

```
DIMENSION IA(2)
DOUBLE PRECISION D1
CALL FLDW (D1,IA)
```

1. BEFORE EXECUTION OF THE SUBROUTINE:

   D1 = 109873.D0

   IA = 00

2. AFTER EXECUTION OF THE SUBROUTINE:

   IA = 32-BIT SIGNED BINARY INTEGER 109873

SUBROUTINE NAME: GET

PURPOSE: This subroutine is a function subprogram that extracts a floating point double precision number from an array containing data in A1 format.

STATEMENT FORMAT:

GET (ICON,IFRST,ILST,DECP)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the character string to be converted. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data to be converted must be in A1 format (one character per word, left-justified). During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the ICON array to be converted.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the ICON array to be converted.

DECP - A real constant, expression or variable used in one of two ways. If decimal places are required in the number, DECP is equal to 10**(-P) where P is the number of decimal places. If a scale factor is required, DECP is equal to 10**(P) where P is the number of zeros.

GENERAL USAGE RULES

When this function is used the name GET must appear in a Double Precision Specification statement.

EXAMPLE

```
      DIMENSION IA(10)
      DOUBLE PRECISION A,B,GET
```

BEFORE EXECUTION OF THE SUBROUTINE:

IA = 0123456789

A = GET(IA,1,5,1.0)

B = GET(IA,5,7,.01)

AFTER EXECUTION OF THE SUBROUTINE:

IA = UNCHANGED

A = 1234.0 DO

B = 4.56 DO

SUBROUTINE NAME: PACK

PURPOSE: This subroutine converts data in a one-dimensional integer array from A1 format (one-character per word, left-justified). Packing the data into a second array in A2 format (two characters per word). The data in the first array remains unchanged during execution of the subroutine.

STATEMENT FORMAT:

```
CALL PACK(ICON,IFRST,ILST,MCVTD,MFRST)
```

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the data to be packed. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data to be packed must be in A1 format (one character per word, left-justified). During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the ICON array to be packed.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the ICON array to be packed.

MCVTD - The name of the one-dimensional integer array that will contain the packed data. This array must have been defined in a prior DIMENSION statement. After execution of the subroutine the packed data will be in A2 format (two characters per word).

MFRST - An integer constant, expression, or variable that identifies the position of the first packed two-character element (left-most position) in the MCVTD array.

GENERAL USAGE RULES:

1. The length of the MCVTD array is determined by the subroutine using the following formula:

$$\text{length of MCVTD} = \frac{\text{ILST} - \text{IFRST} + 2}{2} \quad \text{(Rounded down)}$$

   ILST must always be a multiple of two since packing of characters is done in pairs. If an odd number is used in the ILST argument the character at ILST +1 will be converted as the second character in the pair.

2. If the ILST argument is less than or equal to IFRST the subroutine will pack the first two characters starting with the contents of field designated in the IFRST argument.

EXAMPLE
=======

```
DIMENSION  ICON(26),MCVTD(13)
     .
     .
     .
CALL  PACK(ICON,1,26,MCVTD,1)
```

1. CONTENTS OF ICON AND MCVTD BEFORE EXECUTION OF THE SUBROUTINE:

```
      ICON  = THIS△IS△A△SAMPLE△SENTENCE△
              ↑   ↑    ↑    ↑     ↑    ↑
(WORD
POSITION )    1   5    10   15    20   25

      MCVTD = ABCDEFGHIJKLM
              ↑   ↑    ↑
(WORD
POSITION)     1   5    10
```

2. CONTENTS OF ICON AND MCVTD AFTER EXECUTION OF THE SUBROUTINE:

```
      ICON  = UNCHANGED

      MCVTD = THIS△IS△A△SAMPLE△SENTENCE
              ↑       ↑          ↑
(WORD
POSITION)     1       5          10
```

SUBROUTINE NAME: PUT

PURPOSE: This subroutine converts the whole portion of a double precision variable to an ASCII character string in A1 format (one character per word, left-justified). The number is half-adjusted and truncated as specified. If the quantity to be converted is a negative number, an 11-zone punch is placed over the low order position (right-most position) in the array.

STATEMENT FORMAT:

```
CALL PUT(ICON,IFRST,ILST,DPVAL,HADJ,TRUNC)
```

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that will contain the result of the conversion. This array must have been defined in a prior DIMENSION statement. After execution of the subroutine the whole portion of the real number will be in A1 format (one character per word, left-justified).

IFRST - An integer constant, expression, or variable that identifies the first position (left-most position) in the ICON array to be filled with the result of the conversion.

ILST - An integer constant, expression, or variable that identifies the last position (right-most position) in the ICON array to be filled with the result of the conversion.

DPVAL - A double precision constant, expression, or variable. This is the actual number whose whole portion is to be converted to A1 format. This name must appear in a Double precision statement.

HADJ - A real constant, expression, or variable that is added to DPVAL to be used as a factor for half-adjustment of DPVAL.

TRUNC - An integer constant, expression, or variable that indicates the number of digits to be truncated from the right-hand portion of DPVAL.

GENERAL USAGE RULES:

1. If ILAST is equal to or less than IFRST, only one digit will be placed in the ICON array.

2. If ICON is not large enough to hold the whole portion of DPVAL, only the low-order digits are placed in the ICON array.

3. The HADJ argument must be used for every PUT subroutin used, and should never be less than .5(since this will preven fraction inaccuracies). The following illustrates the use of HADJ.

   If the number to be put is 567.00, the binary representation of this number in storage could be 566.999.

   a. If HADJ is 0, then the number PUT will be 566.

   b. If HADJ is .5, then the number PUT will be 567, since the binary representation will be adjusted to 567.499.

   The value of the HADJ argument should be 5 in the decimal position one to the right of the low-order digit of the number in VAR to be PUT.

4. The HADJ and TRUNC arguments are used as a pair, and should be used as shown below:

HADJ		TRUNC
.5	and	0
5.	and	1
50.	and	2
500.	and	3
5000.	and	4
50000.	and	5

EXAMPLE 1

```
DIMENSION IA(25)
DOUBLE PRECISION A
CALL PUT (IA,1,12,A,5.0,1)
```

BEFORE EXECUTION OF THE SUBROUTINE:

IA = ABCDEFGHIJKLMNOPQRST

A = 9876543.

AFTER EXECUTION OF THE SUBROUTINE:

IA = 000000987654MNOPQRST

A = 9876543.

NOTE THAT IN THIS EXAMPLE 5.0 WAS ADDED TO A FOR ROUNDING, AND THEN THE LAST POSITION WAS TRUNCATED.

EXAMPLE 2

```
DIMENSION  IA(25)
DOUBLE PRECISION A
CALL PUT (IA,1,12,.5,0)
```

BEFORE EXECUTION OF THE SUBROUTINE:

IA = ABCDEFGHIJKLMNOPQRST

A = 987654.6

AFTER EXECUTION OF THE SUBROUTINE:

IA = 000000987655MNOPQRST

A = 987654.6

NOTE THAT IN THIS EXAMPLE .5 WAS ADDED TO THE NUMBER, WHICH ROUNDED THE UNITS POSITION TO 5. NO TRUNCATION WAS PERFORMED.

SUBROUTINE NAME: UNPAC

PURPOSE: This subroutine converts data in a one-dimensional integer array from A2 format (two characters per word), unpacking the data into a second array in A1 format (one character per word, left-justified). The data in the first array remains unchanged during execution of the subroutine.

STATEMENT FORMAT:

```
CALL UNPAC(ICON,IFRST,ILST,MCVTD,MFRST)
```

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the data to be unpacked. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data to be unpacked must be in A2 format (two characters per word). During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the ICON array to be unpacked.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the ICON array to be unpacked.

MCVTD - The name of the one-dimensional integer array that will contain the unpacked data. This array must have been defined in a prior DIMENSION statement. After execution of the subroutine the unpacked data will be in A1 format (one character per word, left-justified).

MFRST - An integer constant, expression, or variable that identifies the position of the first unpacked character (left-most position) in the MCVTD array.

GENERAL USAGE RULES:

1. The length of the MCVTD array is determined by the subroutine using the following formula:

   MCVTD length = 2(ILST-IFRST +1)

2. If the ILST argument is less than or equal to IFRST only the first two-character element will be unpacked, placing the characters into two separate words

   EXAMPLE: CALL UNPAC(ICON, 1, 1, MCVTD, 1)

Before:			After:	
ICON =	THIS△IS		MCVTD =	T△H△
	↑	↑		↑ ↑
(Word Position)	1	5		1 2

EXAMPLE

```
DIMENSION  ICON(13),MCVTD(26)
    .
    .
    .
CALL UNPAC(ICON,1,13,MCVTD,1)
```

1. CONTENTS OF ICON AND MCVTD BEFORE EXECUTION OF THE SUBROUTINE:

```
      ICON  = THIS△IS△A△SAMPLE△SENTENCE△
              ↑       ↑          ↑
(WORD
POSITION )    1       5          10

      MCVTD = ABCDEFGHIJKLMNOPQRSTUVWXYZ
              ↑   ↑    ↑    ↑    ↑    ↑
(WORD
POSITION)     1   5    10   15   20   25
```

2. CONTENTS OF ICON AND MCVTD AFTER EXECUTION OF THE SUBROUTINE:

```
      ICON  = UNCHANGED

      MCVTD = THIS△IS△A△SAMPLE△SENTENCE△
              ↑   ↑    ↑    ↑    ↑    ↑
(WORD
POSITION)     1   5    10   15   20   25
```

CHAPTER 4

DATA MANIPULATION AND COMPARISON SUBROUTINES

This Chapter examines the subroutines which provide means for the logical testing and manipulation of data. Among these special subroutines are:

EDIT	This subroutine is used in the preparation of printed reports to give them a high degree of legibility. The operation consists of merging a field of variable data with a field of constants to produce a meaningful combination (e.g., the digits 24515 could be edited to form $245.15, ***245.15, $ 245.15CR, etc.).
FILL	This subroutine fills a character string with a program-specified character.
NCOMP	This is a function subprogram that can compare two variable-length character data fields and, depending on the result of the comparison, sets an indicator. The indicator will show whether the result of the comparison was greater, less, or equal.
NZONE	This subroutine examines the zone portion of a character and returns a code that identifies the zone. If a zone change is required, it will also modify the zone.
QMOVE	This subroutine moves a character or a string of characters from one array into another.
RJUST	This subroutine will right justify a string of characters in a field.

SUBROUTINE NAME: EDIT

PURPOSE: This subroutine edits data from a one-dimensional integer array (called the source field), placing the edited data into another one-dimensional integer array (mask field). The data in the source field remains unchanged after execution of the subroutine.

STATEMENT FORMAT:

CALL EDIT(ICON,IFRST,ILST,MASK,MFRST,MLST)

ARGUMENT DESCRIPTIONS

ICON - The name of the one-dimensional integer array that contains the data to be edited. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data must be in A1 format (one character per word, left-justified). During execution of the subroutine, the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the ICON array to be edited.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the ICON array to be edited.

MASK - The name of the one-dimensional integer array that contains the edit mask (mask field). This array must have been defined in a prior DIMENSION statement. After execution of the subroutine this array will contain the edited data, in A1 format (one character per word, left-justified).

MFRST - An integer constant, expression, or variable that identifies the position of the first character of the edit mask (left-most position) in the mask array.

MLST - An integer constant, expression, or variable that identifies the position of the last character of the edit mask (right-most position) in the mask array.

GENERAL USAGE RULES:

The programmer can create and input the mask array by using one of the following methods shown below:

```
DIMENSION MASK (NN)                ; NN=LENGTH OF MASK ARRAY
READ(2,1)MASK
FORMAT(80A1)
     .
```

(or)

```
DIMENSION MASK(NN)
CALL QREAD(MASK,1,NN)
     .
```

(or)

```
DIMENSION MASK(NN)
CALL KEYBD(MASK,1,NN)
     .
```

The mask array can also be data initialized. In order to initialize mask it must be named in a common block and must be initialized in A1 format (one character per word, left-justified).

EXAMPLES:

```
COMMON/WORKA/MASK(10)
DATA MASK/"△△", "$△", "C△"R".....(etc)....."/
     .
     .
```

(or)

```
COMMON/WORKA/MASK(10)
DATA MASK/"△△$△C△R△.....(etc)....."/
```

GENERAL USAGE RULES: (Continued)

Since the mask field is destroyed after each use, the mask field is usually moved into an output array prior to each CALL EDIT statement. The editing operation then takes place in the output array prior to output.

EXAMPLE:

```
DATA NTBL/"ΔΔ  $Δ  CΔ  RΔ..."/          DATA MASK/'$--,--*,--CR'
                                  (OR)
CALL QMOVE (NTBL,1,10,IOUTAREA)     CALL UNPAC (MASK,1,N,IOUT,12)
```

The table on the following pages identifies the codes which can be used in the mask array and their functions:

CODE	PURPOSE
Δ (Blank)	A blank character in the mask field is replaced by a character from the source field.

Mask	Source	After Execution:
ΔΔΔΔΔΔΔ	1234567	1234567
ΔΔΔΔΔΔΔ	0000001	0000001
ΔΔΔΔΔΔΔ	ΔΔΔΔΔΔ1	ΔΔΔΔΔΔ1

0 (Zero)	This is the zero suppress function. Leading zeros will be suppressed to the left of the leading digit. Zeros will be replaced by blanks. This indicator can only be used once in a mask.

Mask	Source	After Execution:
ΔΔΔΔΔΔΔ0	0003333	ΔΔΔΔ3333

* (Asterisk)	Leading zeros will be suppressed and replaced by asterisks. This indicator can only be used once in a mask.

Mask	Source	After Execution:
ΔΔΔΔΔΔΔ*	0000123	*****123

GENERAL USAGE RULES: (Continued)

CODE	PURPOSE
$ (Dollar)	Leading zeros will be suppressed and replaced by a dollar sign to the left of the first significant digit. Can only be used once in a mask.

Mask	Source	After Execution:
ΔΔΔΔΔΔ$	000123	ΔΔΔ$123

CODE	PURPOSE
, (Comma)	Will remain in the mask field where placed. If zero suppress is requested commas will also be suppressed to the left of the zero suppression character and replaced with blanks.

Mask	Source	After Execution:
ΔΔΔ,ΔΔΔ	123456	123,456
ΔΔΔ,ΔΔΔ0	000001	ΔΔΔΔΔΔΔ1
ΔΔΔ,ΔΔΔ*	000001	*******1

CODE	PURPOSE
. (Decimal Point)	Will remain in the mask field where placed. If zero suppress is requested the decimal point will also be suppressed to the left of the zero suppression character and replaced with blanks.

Mask	Source	After Execution:
ΔΔΔΔ.ΔΔ	123456	1234.56
ΔΔΔΔ.ΔΔ0	000001	ΔΔΔΔΔΔΔ 1
ΔΔΔΔ.ΔΔ*	000001	*******1

GENERAL USAGE RULES: (Continued)

CODE	PURPOSE
CR	Used to indicate a credit or negative field. If CR appears in the right-most position of the mask the CR characters will be blanked out if the last character in the source field does not contain an 11 punch or a negative sign. The characters in the source field will appear if an 11 punch or a negative sign is found in the last position.

Mask	Source	After Execution:
ΔΔΔ,ΔΔ$.ΔΔCR	0012345	ΔΔΔ$123.45ΔΔ
ΔΔΔ,ΔΔ$.ΔΔCR	001234N	ΔΔΔ$123.45CR

- (Minus)	A minus sign in the right-most position of the mask will have the same effect as the CR code except that a minus sign will be placed in the mask field.

Mask	Source	After Execution:
ΔΔΔΔ,ΔΔ$.ΔΔ-	01234N	ΔΔΔΔ$123.21-

NOTES:

1. The EDIT mask must contain as many blanks (Δ) as there are characters in the source field. If it does not, the field will be filled with * , as shown below.

   MASK = 1ΔΔΔ1
   SOURCE = ROBERT
   AFTER EDIT = *****

2. If a zero suppress character is used in the mask field ($, 0 or *), the first character in the mask must be a blank (Δ) . Only one zero suppress character may be used in a mask.

GENERAL USAGE RULES: (Continued)

NOTES: (Continued)

3. All other characters placed in a mask field will not be replaced by characters from the source field. The incoming characters will be moved around these characters.

EXAMPLE: (1) MASK = ΔΔΔ -ΔΔ- ΔΔΔΔ
SOURCE = 003221108
AFTER EDIT = 003-22-1108

EXAMPLE: (2) MASK = ΔΔΔ, ΔΔ$. ΔΔ CR
SOURCE = 123456
AFTER EDIT = Δ$1,234.56 ΔΔ

EXAMPLE: (3) MASK = ΔΔΔ, ΔΔ*. ΔΔ
SOURCE = 0123456
AFTER EDIT = **1,234.56

SUBROUTINE NAME: FILL

PURPOSE: This subroutine fills a specified field with a programmer-designated character.

STATEMENT FORMAT:

CALL FILL(ICON, IFRST, ILST, NFILL)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the field to be filled. This array must have been defined in a prior DIMENSION statement.

IFRST - An integer constant, expression, or variable that identifies the first position (left-most position) in the ICON array to be filled.

ILST - An integer constant. expression or variable that identifies the last position (right-most position) in the ICON array to be filled.

NFILL - An integer constant, expression or variable that represents one of the following:

1. An ASCII character or its decimal equivalent to be used as a fill character (See Appendix A)

2. Any integer value.

EXAMPLE

```
DIMENSION  ICON(15)
     .
     .
     .
CALL  FILL(ICON,3,11,8224)
   (OR)
CALL  FILL(ICON,3,11,"△△")
```

1. CONTENTS OF ICON BEFORE EXECUTION OF THE SUBROUTINE:

```
        ICON  = ABCDEFGHIJKLMNO
                ↑   ↑    ↑    ↑
(WORD
POSITION )      1   5    10   15
```

2. CONTENTS OF ICON AFTER EXECUTION OF THE SUBROUTINE:

```
        ICON  = AB△△△△△△△△△△LMNO
                ↑   ↑    ↑    ↑
(WORD
POSITION)       1   5    10   15
```

SUBROUTINE NAME: NCOMP

PURPOSE: This subroutine is an integer function subprogram that compares two variable-length data fields and the result of the comparison sets NCOMP to minus (-), plus (+), or zero (0). The contents of both fields being compared are not changed by the subroutine.

STATEMENT FORMAT:

NCOMP (IONE,IFRST,ILST,MTWO,MFRST)

ARGUMENT DESCRIPTIONS:

IONE - The name of the one-dimensional integer array that contains the first variable-length field to be compared. This array must have been defined in a prior DIMENSION statement. During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the IONE array to be compared.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the IONE array to be compared.

MTWO - The name of the one-dimensional integer array that contains the second variable-length field to be compared. This array must have been defined in a prior DIMENSION statement. During execution of the subroutine the data in this array remains unchanged.

MFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the MTWO array to be compared.

GENERAL USAGE RULES:

1. The comparison will be based upon the ASCII collating sequence contained in APPENDIX A.

The following table shows the value of NCOMP depending upon the relationship of the IONE and MTWO fields.

NCOMP		
+	if	IONE > MTWO
0	if	IONE = MTWO
-	if	IONE < MTWO

EXAMPLE

```
DIMENSION  IONE(8),MTWO(8)
     .
     .
     .
IF (NCOMP(IONE,1,8,MTWO,1))1,2,3
```

1. CONTENTS OF IONE AND MTWO BEFORE EXECUTION OF THE SUBROUTINE:

   IONE = ABCD1044

   MTWO = ABCD1044

2. CONTENTS OF IONE AND MTWO AFTER EXECUTION OF THE SUBROUTINE:

   IONE = UNCHANGED

   MTWO = UNCHANGED

   NCOMP = 0

SUBROUTINE NAME: NZONE

PURPOSE: This subroutine serves two purposes:

1. Examines the zone of a character and returns a code that identifies the zone.

2. Allows the modification of the zone, if required.

STATEMENT FORMAT:

CALL NZONE(IONE,IFRST,NZN,NRES)

ARGUMENT DESCRIPTIONS:

IONE - The name of the one-dimensional integer array that contains the character to be examined. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the character to be examined must be in A1 format (one character per word, left-justified).

IFRST - An integer constant, expression, or variable that identifies the position of the character in the IONE array to be examined.

NZN - An integer constant, expression, or variable that identifies the code to be used in modifying the zone of the character, if required.

NZN	Zone will be changed to:
1	12 zone
2	11 zone
3	0 zone
4	no zone
greater than 4	no zone change

ARGUMENT DESCRIPTIONS: (Continued)

NRES - After execution of the subroutine this indicator will contain the code indicating the zone of the character before the zone was altered (if a change in zone was requested).

NRES	Zone of the Character Indicated:
1	was a character from A-I
2	was a character from J-R
3	was a character from S-Z
4	was a digit from 0-9
5	was another character

GENERAL USAGE RULES:

In card input (Hollerith Code), the zone portion of a character is used to indicate that a digit or numeric field is either positive or negative. The presence or absence of a specific zone may also be used to indicate a special code, (e.g., transaction type).

Hollerith Card Code Conventions are:

no zero punch	-	a positive numeric value
"12" punch	-	a positive value or a letter
"11" punch	-	a negative value or a letter

For example, a negative 5 would be indicated by an "11" overpunch in the source card column (i.e. an "N"), a positive 5 by a "12" overpunch in the same column, (i.e. an "E").

GENERAL USAGE RULES: (Continued)

Customarily only the low order digit of a numeric value may contain an overpunch, however many commerical users use overpunches in other positions in lieu of setting aside separate card columns for transaction codes. NZONE permits these overpunches to be tested, altered, or tested and altered.

Example:

```
Dimension IA(10)
Call NZONE (IA, 5, NZN, NRES)
```

BEFORE		AFTER	
IA(5)	NZN=	IA(5) =	NRES=
A	4	1	1
K	3	S	Z
S	2	K	3
1	1	A	4
X	5	X	3

EXAMPLE

```
      DIMENSION  IONE(10)
          .
          .
          .
      CALL NZONE(IONE,4,2,NRES)
```

BEFORE EXECUTION:

NRES = 0

AFTER EXECUTION:

NRES = 1

SUBROUTINE NAME: QMOVE

PURPOSE: This subroutine moves data from a one-dimensional integer array into another one-dimensional integer array. The data in the first array remains unchanged by the execution of the subroutine.

STATEMENT FORMAT:

CALL QMOVE(IONE,IFRST,ILST,MTWO,MFRST)

ARGUMENT DESCRIPTIONS:

IONE - The name of the one-dimensional integer array that contains the data to be moved. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data in this array may be in any format. During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the IONE array to be moved.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the IONE array to be moved.

MTWO - The name of the one-dimensional integer array that will contain the data moved from IONE. This array must have been defined in a prior DIMENSION statement.

MFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) moved from the IONE array.

EXAMPLE

```
      DIMENSION  IONE(44),MTWO(55)
                 .
      ILST=44
      MFRST=6
      CALL QMOVE(IONE,1,ILST,MTWO,MFRST)
```

1. CONTENTS OF IONE AND MTWO BEFORE EXECUTION OF THE SUBROUTINE:

```
    IONE = THIS△IS△A△SAMPLE△SENTENCE△1234△BEFORE△MOVING
           ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑
(WORD
POSITION)  1   5   10  15  20  25  30  35  40


    MTWO = ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABC
           ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑
(WORD
POSITION)  1   5   10  15  20  25  30  35  40  45  50  55
```

2. CONTENTS OF IONE AND MTWO AFTER EXECUTION OF THE SUBROUTINE:

```
    IONE = UNCHANGED

    MTWO = ABCDETHIS△IS△A△SAMPLE△SENTENCE△1234△BEFORE△MOVINGWXYZABC
           ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑
(WORD
POSITION)  1   5   10  15  20  25  30  35  40  45  50  55
```

SUBROUTINE NAME: RJUST

PURPOSE: This subroutine right-justifies a field in an array, placing blanks to the left of the field.

NOTE: This subroutine is normally used in conjunction with the KEYBD subroutine (See example).

STATEMENT FORMAT:

```
CALL RJUST ( ICON, IFRST, ILST)
```

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the field to be right-justified. This array must have been defined in a prior DIMENSION statement. Before execution of this subroutine the data to be right justified must be in A1 format. After execution of the subroutine this array will contain the justified field with leading blanks. Any imbedded blanks are also removed.

IFRST - An integer constant, expression, or variable that identifies the first position of the field in the ICON array to be right-justified.

ILST - An integer constant, expression, or variable that identifies the last position of the field in the ICON array to be right-justified.

EXAMPLE
=======

```
DIMENSION ICON(21)
CALL KEYBD(ICON,1,21)
     .
     .
CALL RJUST(ICON,1,21)
```

INPUT MESSAGE FROM KEYBOARD = 12345(CARRIAGE RETURN)

1. CONTENTS OF ICON BEFORE EXECUTION OF THE RJUST SUBROUTINE:

```
ICON = 12345ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
       ↑   ↑     ↑     ↑     ↑
(WORD
POSITION)  1   5     10    15    20
```

2. CONTENTS OF ICON AFTER EXECUTION OF THE RJUST SUBROUTINE:

```
ICON = ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ 12345
       ↑   ↑     ↑     ↑     ↑
(WORD
POSITION)  1   5     10    15    20
```

CHAPTER 5

ARITHMETIC SUBROUTINES

The FORTRAN Commercial Subroutine Package offers the programmer two types of arithmetic subroutines: Decimal and 32-bit Signed Binary Integer.

The decimal arithmetic subset operates on whole numbers using variable -length fields, thereby overcoming some of the problems encountered with extended precision values and exact representation of fractional numbers. The 32-bit signed binary integer arithmetic subroutines give the user the computational ability to operate on numeric fields to a maximum of $\pm$ 2,147,483,647.

The following is a brief description of each subroutine in the this subset:

DWADD	Sums two 32-bit signed binary integers.
DWCMP	This is a function subprogram that compares two 32-bit signed binary integers.
DWDIV	Divides a 32-bit signed binary integer by another 32-bit signed binary integer.
DWMPY	Multiplies a 32-bit signed binary integer by another 32-bit signed binary integer.
DWSUB	Calculates the difference between two 32-bit signed binary integers.
ICOMP	This is a function subprogram that compares two variable-length decimal data fields.
NSIGN	This subroutine examines the sign of a digit and returns a code identifying the sign; it will also change the sign, if required.
QADD	Sums two variable-length decimal data fields.
QDIV	Divides a variable-length decimal data field by another variable-length decimal data field.
QMPY	Multiplies a variable-length decimal data field by another variable-length decimal data field.
QSUB	Calculates the difference between two variable-length decimal data fields.

WHOLE Truncates the fractional portion of a double precision floating point variable or expression.

SUBROUTINE NAME: DWADD

PURPOSE: This subroutine adds two 32-bit signed binary integers, placing the sum in the second 32-bit signed binary integer.

STATEMENT FORMAT:

CALL DWADD (DWORD1, DWORD2, NCHK)

ARGUMENT DESCRIPTIONS:

DWORD1 - The 32-bit signed binary integer that is to be added to the second 32-bit signed binary integer. Before execution of the subroutine this data must be in Double Word format. During execution of the subroutine this data remains unchanged.

DWORD2 - The 32-bit signed binary integer that is to be added with DWORD1. Before execution of the subroutine of the subroutine this data must also be in Double Word format. After exectuion of the subroutine this 32-bit signed binary integer will be the sum.

NCHK - An integer variable that can be tested after execution of the subroutine to determine if overflow has occurred.

GENERAL USAGE RULES:

1. The task of initializing, testing, and resetting of the NCHK indicator is the responsibility of the programmer.

2. A double word integer occupies two contiguous words of storage and may be referenced as a real variable, real array element, or two words of an integer array (addressed by the first word [left-most position]).

EXAMPLE

```
DIMENSION  IA(4)
     .
     .
NCHK=0
CALL DWADD(IA,IA(3),NCHK)
```

1. CONTENTS OF IA AND NCHK BEFORE EXECUTION OF THE SUBROUTINE:

   IA = WORDS 1 AND 2 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 109413.
   WORDS 3 AND 4 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 215741.

   NCHK = 0

2. CONTENTS OF IA AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

   IA = WORDS 1 AND 2 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 109413 (UNCHANGED).
   WORDS 3 AND 4 CONTAIN THE 32-BIT SIGNED BINARY INTEGER 325154 (SUM).

   NCHK = 0 (UNCHANGED - NO OVERFLOW)

SUBROUTINE NAME: DWCMP

PURPOSE: This subroutine is a function subprogram that compares two 32-bit signed binary integers. The result of the comparison sets DWCMP to negative, positive, or zero.

STATEMENT FORMAT:

DWCMP (DWCRD1, DWORD2)

ARGUMENT DESCRIPTIONS:

DWORD1 - The first 32-bit signed binary integer to be compared. Before execution of the subroutine this data must be in Double Word format. During execution of the of the subroutine this data remains unchanged.

DWORD2 - The second 32-bit signed binary integer to be compared. Before execution of the subroutine this data must also be in Double Word format. During execution of the subroutine this data remains unchanged.

GENERAL USAGE RULES:

1. The result of the comparison sets DWCMP to one of the following:

   \+ If DWORD1 is greater than DWORD2
   0 If DWORD1 is equal to DWORD2
   \- If DWORD1 is less than DWORD2

2. A double word integer occupies two contiguous words of storage and may be referenced as a real variable, real array element, or two words of an integer array (addressed by the first word [left-most position]).

EXAMPLE

```
      DIMENSION IA(10)
      READ BINARY(5)IA
      IF (DWCMP(IA(4),IA(6)))10,20,30
30    TYPE "GREATER"
      STOP
20    TYPE "EQUAL"
      STOP
10    TYPE "LESS"
      STOP
      END
```

SUBROUTINE NAME: DWDIV

PURPOSE: This subroutine divides a 32-bit signed binary integer (dividend) by another 32-bit signed binary integer (divisor), placing the quotient into the dividend and the remainder will be a separate 32-bit signed binary integer.

STATEMENT FORMAT:

CALL DWDIV (DWORD1, DWORD2, DWORD3, NCHK)

ARGUMENT DESCRIPTIONS:

DWORD1 - The 32-bit signed binary that is to be used as the divisor. Before execution of the subroutine this data must be in Double Word format. During execution of the subroutine this data remains unchanged.

DWORD2 - The 32-bit signed binary integer that is to be used as the dividend. Before execution of the subroutine this data must be in Double Word format. After the execution of the subroutine this 32-bit signed binary integer will be the quotient, in Double Word format.

DWORD3 - After execution of the subroutine this 32-bit signed binary integer will be the remainder, in Double Word format.

NCHK - An integer variable that can be tested after execution of the subroutine to determine if division by zero was attempted.

GENERAL USAGE RULES:

1. The task of initializing, testing, and resetting of the NCHK indicator is the responsibility of the programmer.

2. A double word integer occupies two contiguous words of storage and may be referenced as a real variable, real array element, or two words of an integer array (addressed by the first word [left-most position]).

EXAMPLE

```
DIMENSION  IA(6)
     .
     .
NCHK=0
CALL DWDIV(IA,IA(3),IA(5),NCHK)
```

1. CONTENTS OF IA AND NCHK BEFORE EXECUTION OF THE SUBROUTINE:

   IA = WORDS 1 AND 2 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 256.
   WORDS 3 AND 4 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 131079.

   NCHK = 0

2. CONTENTS OF IA AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

   IA = WORDS 1 AND 2 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 256 (UNCHANGED).
   WORDS 3 AND 4 CONTAIN THE 32-BIT SIGNED BINARY INTEGER 512 (QUOTIENT).
   WORDS 5 AND 6 CONTAIN THE 32-BIT SIGNED BINARY INTEGER 7 (REMAINDER).

   NCHK = 0 (UNCHANGED - NO DIVISION BY ZERO)

SUBROUTINE NAME: DWMPY

PURPOSE: This subroutine multiplies a 32-bit signed binary integer (multiplicand) by another 32-bit signed binary integer (multiplier). After execution of the subroutine the product replaces the multiplicand.

STATEMENT FORMAT:

CALL DWMPY (DWORD1, DWORD2, NCHK)

ARGUMENT DESCRIPTIONS:

DWORD1 - The 32-bit signed binary integer that is to be used as the multiplier. Before execution of the subroutine this data must be in Double Word format. During execution of the subroutine this data remains unchanged.

DWORD2 - The 32-bit signed binary integer that is to be used as the multiplicand. Before execution of the subroutine this data must also be in Double Word format. After execution of the subroutine this 32-bit signed binary integer will be the product.

NCHK - An integer variable that can be tested after execution of the subroutine to determine if DWORD2 is not long enough to hold the product.

GENERAL USAGE RULES:

1. The task of initializing, testing, and resetting the NCHK indicator is the responsibility of the programmer.

2. A double word integer occupies two contiguous words of storage and may be referenced as a real variable, real array element, or two words of an integer array (addressed by the first word [left-most position]).

EXAMPLE

```
      DIMENSION  IA(4)
          .
          .
      NCHK=0
      CALL DWMPY(IA,IA(3),NCHK)
```

1. CONTENTS OF IA AND NCHK BEFORE EXECUTION OF THE SUBROUTINE:

   IA = WORDS 1 AND 2 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 256.
   WORDS 3 AND 4 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER -512.

   NCHK = 0

2. CONTENTS OF IA AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

   IA = WORDS 1 AND 2 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 256 (UNCHANGED).
   WORDS 3 AND 4 CONTAIN THE 32-BIT SIGNED BINARY INTEGER -131072 (PRODUCT).

   NCHK = 0 (UNCHANGED)- IA(3) IS LONG ENOUGH TO HOLD THE PRODUCT.

SUBROUTINE NAME: DWSUB

PURPOSE: This subroutine calculates the difference between two 32-bit signed binary integers.

STATEMENT FORMAT:

CALL DWSUB (DWORD1, DWORD2, NCHK)

ARGUMENT DESCRIPTIONS:

DWORD1 - The 32-bit signed binary integer that is to be subtracted from the second 32-bit signed binary integer. Before execution of the subroutine this data must be in Double Word format. During execution of the subroutine this data remains unchanged.

DWORD2 - The 32-bit signed binary integer that DWORD1 is to be subtracted from. Before execution of the subroutine this data must also be in Double Word fromat. After execution of the subroutine this 32-bit signed binary integer will be the difference.

NCHK - An integer variable that can be tested after execution of the subroutine to determine if overflow has occurred.

GENERAL USAGE RULES:

1. The task of initializing, testing, and resetting of the NCHK indicator is the responsibility of the programmer.

2. A double word integer occupies two contiguous words of storage and may be referenced as a real variable, real array element, or two words of an integer array (addressed by the first word [left-most position]).

EXAMPLE

```
DIMENSION  IA(4)
     .
     .
NCHK=0
CALL DWSUB(IA,IA(3),NCHK)
```

1. CONTENTS OF IA AND NCHK BEFORE EXECUTION OF THE SUBROUTINE:

   IA = WORDS 1 AND 2 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 123456.
   WORDS 3 AND 4 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 999999.

   NCHK = 0

2. CONTENTS OF IA AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

   IA = WORDS 1 AND 2 CONTAIN THE 32-BIT SIGNED BINARY INTEGER NUMBER 123456 (UNCHANGED).
   WORDS 3 AND 4 CONTAIN THE 32-BIT SIGNED BINARY INTEGER 876543 (DIFFERENCE).

   NCHK = 0 (UNCHANGED - NO OVERFLOW)

SUBROUTINE NAME: ICOMP

PURPOSE: This subroutine is a function subprogram that compares two variable-length numeric data fields and the result of the comparison sets ICOMP to minus (-), plus (+), or zero (0). The contents of both fields being compared are not changed by the function subprogram.

STATEMENT FORMAT:

ICOMP (IONE, IFRST, ILST, MTWO, MFRST, MLST)

ARGUMENT DESCRIPTIONS:

IONE - The name of the one-dimensional integer array that contains the first variable-length numeric field to be compared. This array must have been defined in a prior DIMENSION statement. The data in this array must be in D1 format (one digit per word, right-justified). During execution of the function subprogram the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the IONE array to be compared.

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the IONE array to be compared.

MTWO - The name of the one-dimensional integer array that contains the second variable-length numeric field to be compared. This array must have been defined in a prior DIMENSION statement. The data in this array must also be in D1 format. During execution of this function subprogram data in this array remains unchanged.

MFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the MTWO array to be compared.

MLST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the MTWO array to be compared.

GENERAL USAGE RULES:

The subroutine assumes that the fields to be compared are right-justified.

The following table shows the value of ICOMP depending upon the relationship of the IONE and MTWO fields.

ICOMP		
+	if	IONE > MTWO
0	if	IONE = MTWO
-	if	IONE < MTWO

PROGRAM ERROR DESCRIPTION:

1. The result of the comparison cannot be predicted if either ILST is less than IFRST, or MLST is less than MFRST.

2. The IONE field length must not be greater than the MTWO field length.

EXAMPLE

```
DIMENSION  IONE(8),MTWO(8)
     .
     .
     .
IF (ICOMP(IONE,1,8,MTWO,1,8))1,2,1
```

1. CONTENTS OF IONE AND MTWO BEFORE EXECUTION OF THE SUBROUTINE:

IONE = 21631044

MTWO = 21631022

2. CONTENTS OF IONE AND MTWO AFTER EXECUTION OF THE SUBROUTINE:

IONE = UNCHANGED

MTWO = UNCHANGED

ICOMP = +

SUBROUTINE NAME: NSIGN

PURPOSE: This subroutine serves two purposes:

1. Examines the sign of a digit and returns a code that identifies the sign of the digit.

2. Allows the modification of the sign.

STATEMENT FORMAT:

CALL NSIGN(IONE,IFRST,NCODE,NRES)

ARGUMENT DESCRIPTIONS:

IONE - The name of the one-dimensional integer array that contains the digit to be examined. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the digit to be examined must be in D1 format (one digit per word, right-justified).

IFRST - An integer constant, expression, or variable that identifies the position of the digit in the IONE array to be examined.

NCODE - An integer constant, expression, or variable that identifies the code to be used in modifying the sign of the digit, if required.

NCODE	Sign will be:
+1	positive
0	sign is changed to the opposite of the old sign
-1	negative
NRES	no change in sign will be made

NRES - After execution of the subroutine this indicator will contain a code that will identify the sign of the digit before it was changed (if a change was specified).

ARGUMENT DESCRIPTIONS: (Continued)

NRES (Continued)

NRES	Sign of digit was:
+1	positive
-1	negative

EXAMPLE

```
      DIMENSION  IONE(5)
           .
           .
           .
      CALL NSIGN(IONE,5,0,NRES)
```

BEFORE EXECUTION:	AFTER EXECUTION:
IONE(5) = 5	IONE(5) = -5
NRES = 0	NRES = +1

SUBROUTINE NAME: QADD

PURPOSE: This subroutine adds the contents of two variable-length decimal data fields, placing the sum in the second array. The contents of the first array remains unchanged during execution of the subroutine.

STATEMENT FORMAT:

CALL QADD(IADD,IFRST,ILST,MSUM,MFRST,MLST,NOFL)

ARGUMENT DESCRIPTIONS:

IADD - The name of the one-dimensional integer array that contains the variable-length decimal data field that will be added to the contents of the second array. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data must be in D1 format (one digit per word, right-justified). During execution of the subroutine, the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the IADD array to be added.

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the IADD array to be added.

MSUM - The name of the one-dimensional integer array that contains the variable-length decimal data field that will be added with the contents of the numeric data in IADD. After execution of the subroutine this array will contain the sum, in D1 format (one digit per word, right-justified).

MFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the MSUM array to be added.

ARGUMENT DESCRIPTIONS: (Continued)

MLST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the MSUM array to be added.

NOFL - An integer variable that can be tested after execution of the subroutine to determine if arithmetic overflow occurred.

GENERAL USAGE RULES:

1. Both arrays must be in D1 format prior to execution of the subroutine. Numeric data in A1 format can be converted to D1 format using the A1DEC subroutine; conversion from D4 to D1 format is accomplished using the DUNPK subroutine.

2. The task of initializing, testing, and resetting of the NOFL indicator is the responsibility of the programmer.

PROGRAM ERROR DESCRIPTIONS:

1. If the length of the IADD field is longer than the length of the MSUM field the NOFL indicator is set to MLST. The ADD operation is not performed and the contents of IADD and MSUM remain unchanged.

2. If the length of MSUM is not long enough to hold the SUM the NOFL indicator is set to MLST. The MSUM field is then filled with nines (9).

EXAMPLE

```
      DIMENSION  IADD(6),MSUM(10)
          .
          .
      NOFL=0
      CALL  QADD(IADD,1,6,MSUM,1,10,NOFL)
```

1. CONTENTS OF IADD, MSUM AND NOFL BEFORE EXECUTION OF THE SUBROUTINE:

```
          IADD  = 018537
                  ↑   ↑
(WORD
POSITION)         1   5

          MSUM  = 0037598303
                  ↑   ↑   ↑
(WORD
POSITION)         1   5   10

          NOFL  = 0
```

2. CONTENTS OF IADD, MSUM AND NOFL AFTER EXECUTION OF THE SUBROUTINE:

```
          IADD  = UNCHANGED

          MSUM  = 0037616840
                  ↑   ↑   ↑
(WORD
POSITION)         1   5   10

          NOFL  = 0  (NO OVERFLOW)
```

SUBROUTINE NAME: QDIV

PURPOSE: This subroutine divides the contents of a variable-length decimal data field (dividend) by the contents of a second variable-length decimal data field (divisor). After execution of the subroutine the quotient and remainder replaces the dividend; the field used as the divisor remains unchanged.

STATEMENT FORMAT:

CALL QDIV(IDIVR,IFRST,ILST,MDIVD,MFRST,MLST,NZER)

ARGUMENT DESCRIPTIONS:

IDIVR - The name of the one-dimensional integer array that contains the divisor (the variable-length data field that will be divided into the second array). This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data must be in D1 format (one digit per word, right-justified). During execution of the subroutine the data in this array will remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the IDIVR array to be used as the divisor.

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the IDIVR array to be used as the divisor.

MDIVD - The name of the one-dimensional integer array that contains the dividend (the variable-length data field that will be divided by the contents of the numeric data in IDIVR). After execution of the subroutine this array will also contain the quotient and remainder, in D1 format (one digit per word, right-justified).

MFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the MDIVD array to be divided into.

ARGUMENT DESCRIPTIONS: (Continued)

MLST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the MDIVD array to be divided into.

NZER - An integer variable that is tested after execution of the subroutine to determine if division by zero was attempted, or it the length of MDIVD was not long enough to hold the quotient and remainder.

GENERAL USAGE RULES:

1. Both arrays must be in D1 format prior to execution of the subroutine. Numeric data in A1 format can be converted to D1 format using the A1DEC subroutine; conversion from D4 to D1 format is accomplished using the DUNPK subroutine.

2. The task of initializing, testing, and resetting of the NZER indicator is the responsibility of the programmer.

3. The quotient and remainder are found in the MDIVD array:

   Location of first digit or quotient = MFRST - (ILST - IFRST + 1)
   Location of last digit or quotient = MLST - (ILST - IFRST +1)
   Location of first digit or remainder = MLST - (ILST - IFRST)
   Location of last digit or remainder = MLST

NOTE:

MFRST (the first position of MDIVD array) must allow for expansion, and must be at least ILST - IFRST +1 positions from the start of the IDIVR array (e.g, in the example IDIVR is 8 positions in length [8 -1 + 1 = 8]. Therefore MFRST must be equal to at least 9).

4. The subroutine operates with whole numbers only.

PROGRAM ERROR DESCRIPTIONS:

1. If the length of MDIVR is not long enough to hold the quotient and remainder the NCHK indicator is set to MLST and the execution of the subroutine is terminated. The contents of IDIVR and MDIVD remain unchanged.

2. The contents of MDIVD will be filled with zeros and NZER set to last if division by zero is attempted.

EXAMPLE

```
DIMENSION IDIVR(8),MDIVD(21)
    .
    .
NZER=0
CALL  QDIV(IDIVR,1,8,MDIVD,9,21,NZER)
```

1. CONTENTS OF IDIVR, MDIVD AND NZER BEFORE EXECUTION OF THE SUBROUTINE:

```
        IDIVR  = 00376871
                 ↑   ↑
(WORD
POSITION )       1   5


       MDIVD  =  ABCDEFGH0000021631044
                 ↑   ↑    ↑    ↑    ↑
(WORD
POSITION)        1   5    10   15   20


        NZER  =  0
```

2. CONTENTS OF IDIVR, MDIVD AND NZER AFTER EXECUTION OF THE SUBROUTINE:

```
        IDIVR  = UNCHANGED


        MDIVD  = 000000000000057  00278749
                 ↑   ↑    ↑      ↑    ↑
(WORD
POSITION)        1   5    10     15   20

                 QUOTIENT        REMAINDER


         NZER  = 0  (NDIVD LONG ENOUGH TO HOLD QUOTIENT
                     AND REMAINDER)
```

SUBROUTINE NAME: QMPY

PURPOSE: This subroutine multiplies the contents of a variable-length decimal data field (multiplicand) by the contents of a second variable-length decimal data field (multiplier). After execution of the subroutine the product replaces the multiplicand; the field used as the multiplier remains unchanged.

STATEMENT FORMAT:

```
CALL QMPT(IMPYR,IFRST,ILST,MPRDT,MFRST,MLST,NCHK)
```

ARGUMENT DESCRIPTIONS:

IMPYR - The name of the one-dimensional integer array that contains the multiplier (variable-length numeric field that will be used to multiply the variable-length decimal data field in the second array). This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data must be in D1 format (one digit per word, right-justified). During execution of the subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the IMPYR array to be used as the multiplier.

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the IMPYR array to be used as the multiplier.

MPRDT - The name of the one-dimensional integer array that contains the multiplicand (variable-length data field that will be multiplied by the numeric data in IMPYR). After execution of the subroutine this array will contain the product, in D1 format (one digit per word, right-justified).

MFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the MPRDT array to be multiplied.

ARGUMENT DESCRIPTIONS: (Continued)

MLST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the MPRDT array to be multiplied.

NCHK - An integer variable that can be tested after execution of the subroutine to determine if the length of MPRDT is long enough to hold the product.

GENERAL USAGE RULES:

1. Both arrays must be in D1 format prior to execution of the subroutine. Numeric data in A1 format can be converted to D1 format using the A1DEC subroutine; conversion from D4 to D1 format is accomplished using the DUNPK subroutine.

2. The task of initializing, testing, and resetting of the NOFL indicator is the responsibility of the programmer.

3. The subroutine operates with whole numbers only.

4. The length of MPRDT must be long enough to hold the product of the multiplication. The MPRDT array must be long enough to hold the product; MFRST must be at least ILST-IFRST +1.

5. The product will be found in the MPRDT array beginning at MFRST -(ILST-IFRST +1) and will end at MLAST.

PROGRAM ERROR DESCRIPTIONS:

If the length of MPRDT is not long enough to hold the product the NCHK indicator is set to MLST and the execution of the subroutine is terminated. The contents of IMPYR and MPRDT remain unchanged.

EXAMPLE

```
      DIMENSION IMPYR(6),MPRDT (17)
          .
          .
      NCHK=0
      CALL  OMPY(IMPYR,1,6,MPRDT,7,17,NCHK)
```

1. CONTENTS OF IMPYR, MPRDT AND NCHK BEFORE EXECUTION OF THE SUBROUTINE:

```
          IMPYR  = 003221
                   ↑   ↑
     (WORD
     POSITION )    1   5

          MPRDT  =  WXYZAB00021631044
                   ↑   ↑    ↑    ↑
      (WORD
      POSITION)    1   5    10   15

           NCHK  =  0
```

2. CONTENTS OF IMPYR, MPRDT AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

```
          IMPYR  = UNCHANGED

          MPRDT  = 000000693735927 24
                   ↑   ↑    ↑    ↑
     (WORD
     POSITION)    1   5    10   15

           NCHK  = 0  (MPRDT LONG ENOUGH TO HOLD PRODUCT)
```

SUBROUTINE NAME: QSUB

PURPOSE: This subroutine calculates the difference between two variable-length decimal data fields. The difference is placed in the second array. The contents of the first array remains unchanged during the execution of the subroutine.

STATEMENT FORMAT:

```
CALL  QSUB(ISUB,IFRST,ILST,MDIF,MFRST,MLST,NCHK)
```

ARGUMENT DESCRIPTIONS:

ISUB - The name of the one-dimensional integer array that contains the variable-length decimal data field that will be subtracted from the field in the second array. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data must be in D1 format (one digit per word right-justified). After execution of the subroutine the data in this array will remain unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the ISUB array to be subtracted

ILST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the ISUB array to be subtracted.

MDIF - The name of the one-dimensional integer array that contains the variable-length decimal data field that ISUB will be subtracted from. After execution of the subroutine this array will also contain the difference, in D1 format (one digit per word, right-justified).

MFRST - An integer constant, expression, or variable that identifies the position of the first digit (left-most position) in the MDIF array to be subtracted from.

ARGUMENT DESCRIPTIONS: (Continued)

MLST - An integer constant, expression, or variable that identifies the position of the last digit (right-most position) in the MDIF array to be subtracted from.

NCHK - An integer variable that is tested after execution of the subroutine to determine if arithmetic overflow has occurred.

GENERAL USAGE RULES:

1. Both arrays must be in D1 format prior to execution of the subroutine. Numeric data in A1 format can be converted to D1 format using the A1DEC subroutine; conversion from D4 to D1 format is accomplished using the DUNPK subroutine.

2. The task of initializing, testing, and resetting of the NCHK indicator is the responsibility of the programmer.

PROGRAM ERROR DESCRIPTIONS:

1. If the length of the ISUB field is longer than the length of the MDIF field the NCHK indicator is set to MLST. The operation is not performed and the contents of ISUB and MDIF remain unchanged.

2. If the length of MDIF is not long enough to hold the difference the NOFL indicator is set to MLST.

EXAMPLE

```
      DIMENSION  ISUB(6),MSUM(10)
           .
           .
      NOFL=0
      CALL  QSUB(ISUB,1,6,MDIF,1,10,NCHK)
```

1. CONTENTS OF ISUB, MDIF AND NCHK BEFORE EXECUTION OF THE SUBROUTINE

```
          ISUB  = 018537
                  ↑   ↑
(WORD
POSITION )        1   5

          MDIF  =  0037598303
                   ↑   ↑    ↑
(WORD
POSITION)         1   5    10

          NCHK  =  0
```

2. CONTENTS OF ISUB, MDIF AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

```
          ISUB  = UNCHANGED

          MSUM  = 0037579766
                  ↑   ↑    ↑
(WORD
POSITION)         1   5    10

          NCHK  = 0  (NO OVERFLOW)
```

SUBROUTINE NAME: WHOLE

PURPOSE: This subroutine truncates the fractional portion of a double precision floating point variable or expression.

STATEMENT FORMAT:

WHOLE(DOUBLE)

ARGUMENT DESCRIPTIONS:

DOUBLE - A double precision floating point variable or expression.

EXAMPLE

```
DOUBLE PRECISION A,B,WHOLE
B=7.51DO
A=WHOLE(B)

RESULT   A=7.00DO
```

GENERAL USAGE RULES:

1. If the argument uses a variable, then the function name must be declared in a double precision specification statement.

2. If the argument is a constant it should be expressed as a double precision quantity by using a D exponent.

CHAPTER 6

INPUT/OUTPUT SUBROUTINES

This chapter describes four special I/O subroutines:

KEYBD	This subroutine reads up to a maximum of 80 characters from the system console device.
PRINT	This subroutine prints one line of data from an integer array onto the system line printer.
QREAD	This subroutine reads characters from a card reader and places the input into an array in A1 format. This feature enables the programmer to read data from an input device without the need of having to know the format of the data before it is read.
TYPER	This subroutine prints one line on the system console device.

These subroutines access the indicated peripherals through pre-assigned channel numbers, as indicated below:

Channel	By Default Opened To:
6	Plotter
9	Card Reader
10	Console Output
11	Console Input
12	Line Printer

These assignments are made upon the first reference to the channel without explicitly opening of the channel (i.e., CALL OPEN). The user may, however, explicitly open these channels to other devices. For example, the user may create an input file on disk and access the data by a call to QREAD by explicitly opening Channel 9 to the file name (which is faster than Formatted I/O).

SUBROUTINE NAME: KEYBD

PURPOSE: This subroutine reads up to a maximum of 80 characters from the system console device.

STATEMENT FORMAT:

CALL KEYBD (ICON, IFRST, ILST)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that will contain the input information. This array must have been defined in a prior DIMENSION statement. The input characters will be stored in A1 format.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) that will contain the input information.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) that will contain the input information.

GENERAL USAGE RULES:

1. Only valid ASCII characters may appear in the input field.

2. Output is to FORTRAN channel 10 which is by default opened to the system console ($TTI in background or $TTI1 in foreground), however the user may explicitly open to any device or file.

3. When a call to KEYBD is made, an asterisk is printed on the keyboard as a prompt.

4. Input will be placed in the ICON array left-justified and filled to the right with blanks (see example).

EXAMPLE

```
DIMENSION  ICON(30)
     .
     .
     .
CALL KEYBD(ICON,1,30)
```

1. CONTENTS OF ICON BEFORE EXECUTION OF THE SUBROUTINE:

```
      ICON = ABCDEFGHIJKLMNOPQRSTUVWXYABCDE
             ↑    ↑    ↑    ↑    ↑    ↑    ↑
(WORD
POSITION)    1    5    10   15   20   25   30
```

2. CONTENTS OF ICON AFTER EXECUTION OF THE SUBROUTINE:

```
      ICON = THISΔISΔTHEΔINPUTΔMESSAGEΔΔΔΔΔ
             ↑    ↑    ↑    ↑    ↑    ↑    ↑
(WORD
POSITION)    1    5    10   15   20   25   30
```

SUBROUTINE NAME: PRINT

PURPOSE: Prints one line of data contained in one-dimensional integer array onto the line printer.

STATEMENT FORMAT:

CALL PRINT (ICON, IFRST, ILST, NCHK)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one dimensional integer array that contains the characters to be printed. This array must have been defined in a prior DIMENSION statement. Before execution of the subroutine the data to be printed must be in A1 format (one character per word, left-justified.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the ICON array to be printed.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the ICON array to be printed.

NCHK - An integer variable that can be tested after execution of the subroutine to determine if printing was successful. The following codes are returned to the user:

0 = Intermediate Error
1 = Successful Print
2 = System Action in Progress
3 = RDOS System Error +3

For a comprehensive discussion of these error codes, see "FORTRAN IV User's Manual." 093-000053.

GENERAL USAGE RULES:

1. Only ASCII characters may appear in the field.

2. The data is written in line mode with a carriage return appended to the end of the line (not more than 132 characters may be written at one time).

3. Output is to FORTRAN channel 12, which is by default associated with the system line printer ($LPT). However, the user may explicitely open channel 12 to any other device or file.

EXAMPLE

```
DIMENSION ICON(23)
     .
     .
NCHK=0
CALL PRINT(ICON,1,23,NCHK)
```

1. CONTENTS OF ICON AND NCHK BEFORE EXECUTION OF THE SUBROUTINE:

```
     ICON  = THISΔMESSAGEΔISΔPRINTED
                ↑    ↑    ↑    ↑    ↑
(WORD
POSITION)       1    5    10   15   20

     NCHK  = 0
```

2. CONTENTS OF ICON AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

ICON = UNCHANGED

NCHK = 1 (SUCCESSFUL WRITE)

SUBROUTINE NAME: QREAD

PURPOSE: This subroutine reads characters from a card reader and places them into an array in A1 format.

STATEMENT FORMAT:

CALL QREAD (ICON, IFRST, ILST, NCHK)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that will contain the input characters. This array must have been defined in a prior DIMENSION statement.

IFRST - An integer constant, expression, or variable that identifies the position of the first input character (left-most position).

ILST - An integer constant, expression, or variable that identifies the position of the last input character (right-most position).

NCHK - An integer variable that can be tested after execution of the subroutine to determine if reading was successful. The following codes are returned to the user:

0	=	Intermediate Error
1	=	Successful Read
2	=	System Action in Progress
3	=	RDOS System Error + 3

For a comprehensive discussion of these error codes. See "FORTRAN IV User's Manual." 093-000053.

GENERAL USAGE RULES:

1. Only ASCII characters may appear in the field.

2. Reading will terminate normally after transmitting one line (terminated by form feed, or null). Reading will terminate abnormally after transmission of 132 (decimal) characters without detecting a carriage return, form feed, or null as the 133rd character and NCHK will be set appropriately.

3. Input is to FORTRAN channel 9, which is by default associated with the system card reader ($CDR). However, the user may explicitly open channel 9 to any other device or file.

EXAMPLE
=======

```
DIMENSION  ICON(25)
     .
     .
     .
CALL QREAD(ICON,1,25,NCHK)
```

1. CONTENTS OF ICON AND NCHK BEFORE EXECUTION OF THE SUBROUTINE:

```
      ICON = ABCDEFGHIJKLMNOPQRSTUVWXY
             ↑    ↑    ↑    ↑    ↑    ↑
(WORD
POSITION)    1    5    10   15   20   25

      NCHK = 0
```

2. CONTENTS OF ICON AND NCHK AFTER EXECUTION OF THE SUBROUTINE:

```
      ICON = THISΔISΔAΔSAMPLEΔSENTENCE
             ↑    ↑    ↑    ↑    ↑    ↑
(WORD
POSITION)    1    5    10   15   20   25

      NCHK = 1  (SUCCESSFUL READ)
```

SUBROUTINE NAME: TYPER

PURPOSE: This subroutine prints one line on the system console device.

STATEMENT FORMAT:

CALL TYPER (ICON, IFRST, ILST)

ARGUMENT DESCRIPTIONS:

ICON - The name of the one-dimensional integer array that contains the characters to be printed. This array must have been defined in a prior dimension statement. Before execution of the subroutine the characters to be printed must be in A1 format (one character per word, left-justified). During execution of this subroutine the data in this array remains unchanged.

IFRST - An integer constant, expression, or variable that identifies the position of the first character (left-most position) in the ICON array to be printed.

ILST - An integer constant, expression, or variable that identifies the position of the last character (right-most position) in the ICON array to be printed.

GENERAL USAGE RULES:

1. Only valid ASCII characters may appear in the field.

2. The data is written in line mode with a carriage return appended to the end of the line.

3. Output is to FORTRAN channel 10 which is by default opened to the system console ($TTO in background or $TTO1 in foreground), however the user may explicitly open to any device or file.

EXAMPLE

```
DIMENSION ICON(21)
    .
    .
    .
CALL TYPER(ICON,1,21)
```

1. CONTENTS OF ICON BEFORE EXECUTION OF THE SUBROUTINE:

```
   ICON = THIS△MESSAGE△IS△TYPED
          ↑   ↑    ↑    ↑   ↑
(WORD
POSITION) 1   5    10   15  20
```

2. CONTENTS OF ICON AFTER EXECUTION OF THE SUBROUTINE:

```
   ICON = UNCHANGED
```

APPENDIX A

TABLE OF ASCII CHARACTERS IN A1 FORMAT AND DECIMAL EQUIVALENTS

ASCII	DECIMAL	ASCII	DECIMAL	ASCII	DECIMAL
"Δ	= 8224	"?	= 16160	"\	= 23584
"!	= 8480	"@	= 16416	"]	= 23840
""	= 8736	"A	= 16672	"↑	= 24096
"#	= 8992	"B	= 16928	"←	= 24352
"$	= 9248	"C	= 17184	"\	= 24608
"%	= 9504	"D	= 17440	(LOWER CASE)	
"&	= 9760	"E	= 17696	"A	= 24864
"'	= 10016	"F	= 17952	"B	= 25120
"(	= 10272	"G	= 18208	"C	= 25376
")	= 10528	"H	= 18464	"D	= 25632
"*	= 10784	"I	= 18720	"E	= 25888
"+	= 11040	"J	= 18976	"F	= 26144
",	= 11296	"K	= 19232	"G	= 26400
"-	= 11552	"L	= 19488	"H	= 26656
".	= 11808	"M	= 19744	"I	= 26912
"/	= 12064	"N	= 20000	"J	= 27168
"0	= 12320	"O	= 20256	"K	= 27424
"1	= 12576	"P	= 20512	"L	= 27680
"2	= 12832	"Q	= 20768	"M	= 27936
"3	= 13088	"R	= 21024	"N	= 28192
"4	= 13344	"S	= 21280	"O	= 28448
"5	= 13600	"T	= 21536	"P	= 28704
"6	= 13856	"U	= 21792	"Q	= 28960
"7	= 14112	"V	= 22048	"R	= 29216
"8	= 14368	"W	= 22304	"S	= 29472
"9	= 14624	"X	= 22560	"T	= 29728
":	= 14880	"Y	= 22816	"U	= 29984
";	= 15136	"Z	= 23072	"V	= 30240
"<	= 15392	"[	= 23328	"W	= 30496
"=	= 15648			"X	= 30752
">	= 15904			"X	= 31008
				"Z	= 31264

APPENDIX B

The following sample programs are included to demonstrate how the Commercial Subroutine Package can be used with FORTRAN to handle the requirements of commercial data processing.

Programs 1 and 2 collect and edit data from card input, and this data in turn is used to update a master file from which reports are written.

Programs 3-5 illustrate how data can be collected from terminals interactively. Note in these examples how files are created and subsequently updated.

```
C***
C***
C*************************************************************************
C***
C     FORTRAN COMMERCIAL SUBROUTINE PACKAGE EXAMPLE PROGRAM #1
C     ========================================================
C***
C     THE PURPOSE OF THIS EXAMPLE IS TO SHOW THE EASE OF
C     MANIPULATING DATA USING THE COMMERCIAL SUBROUTINE
C     PACKAGE (CSP) IN CONJUNCTION WITH THE STANDARD DATA
C     GENERAL SUPPLIED REAL TIME DISK OPERATING SYSTEM (RDOS)
C***
C     THE PROGRAM WILL ILLUSTRATE THE INITIAL BUILDING
C     OF A DATA BASE AND/OR THE LATER ADDITION OF
C     DATA RECORDS TO THE DATA BASE.
C***
C     THE INPUT TO THE PROGRAM IS ASSUMED TO BE A DECK OF
C     CARDS THAT REPRESENTS THE INVENTORY CARDS TO BE ADDED
C     TO THE DATA BASE. THIS PROGRAM WILL APPEND THESE RECORDS
C     TO THE END OF THE CURRENT INVENTORY FILE.
C***
C     THE PROGRAM SEQUENTIALLY READS IN THE CARDS, LISTS
C     THE CARDS TO THE LINE PRINTER, RE-ARRANGES AND COMPACTS
C     THE DATA INTO THE INVENTORY FILE RECORD FORMAT, AND
C     FINALLY OUTPUTS THE RECORD TO THE FILE.
C***
C     THE FORMAT OF THE DATA CARDS ARE AS FOLLOWS:-
C***
C     CARD COLUMNS      ITEM DESCRIPTION
C     ============      ================
C***
C          1-4          STOCK NUMBER
C          7-30         DESCRIPTION
C         33-39         STOCK LOCATION
C         42-43         UNIT QUANTITY CODE
C         46-50         UNIT QUANTITY PRICE
C         53-54         QUANTITY ON HAND
C***
C     THE INVENTORY FILE HAS THE FOLLOWING ORGANIZATION:-
C***
C        RECORD ITEM DESCRIPTION          WORDS    FORMAT
C        =======================          =====    ======
C***
C     STOCK NUMBER (XXXX)                   1-2      2A2
C     DESCRIPTION (24 CHARACTERS)           3-10     8A3
C     STOCK LOCATION (XXXX-XXX)            11-12      Dw
C     UNIT QUANTITY CODE (EA,DZ,CT)          13       A2
C     UNIT QUANTITY PRICE ($XXXXX.XX)      14-15      Dw
C     QUANTITY ON HAND (XXXX)                16      SP
C***
C     TOTAL RECORD SIZE IS 16 - 16 BIT WORDS
C***
C*************************************************************************
```

```
C***
C***
C*****************************************************************************
C***
C     DATA STORAGE ALLOCATION AREA
C***
C*****************************************************************************
C***
      INTEGER RAREA,PRICE,CSPER
C***
      DIMENSION IDESC(24),LOCAT(7),PRICE(5),ITIME(3),IDATE(3)
C***
      COMMON /RECRD/ RAREA(16),NTBL(40)
      COMMON /CHLS/ INPUT,LIST,IVEN
C***
C     INITIALIZE THE INPUT/OUTPUT CHANNEL NUMBERS
C***
      DATA INPUT,LIST,IVEN /0,1,2/
C***
C     INITIALIZE THE 40 CHARACTER TABLE FOR 'A1A3' AND 'A3A1'
C***
      DATA NTBL/"  1 2 3 4 5 6 7 8 9 0 A B C D E F G H I J ",
     *          "K L M N O P Q R S T U V W X Y Z , . / "/
C***
C     INITIALIZE THE CSP ERROR FLAG 'CSPER' TO ZERO.
C     IF IT CHANGES FROM THIS VALUE DURING PROGRAM EXCUTION
C     AN ERROR IS INDICATED.
C***
      CSPER=0
```

```
; C*****************************************************************
; C***
; C     LOG OPERATION DESCRIPTION ON THE CONSOLE LISTING DEVICE
; C***
; C*****************************************************************
; C***
;       TYPE ! !
;       TYPE ! *** INVENTORY UPDATE PROGRAM ***!
;       TYPE ! !
; C***
; C*****************************************************************
; C***
; C     OPEN AND INITIALIZE ALL FILES AND DEVICES
; C***
; C*****************************************************************
; C***
; C     OPEN THE CARD READER FOR READING IN THE CARD DECK.
; C***
;       CALL OPEN (INPUT,"$CDR",0,IER)
; C***
; X     IF (IER.EQ.1) GOTO 10           ; CHECK IF OPEN SUCCESSFUL
; X     TYPE !OPEN $CDR ERROR =!,IER    ; NO--TERMINATE PROCESSING
; X     STOP CDR ERROR
; X 10  CONTINUE
; C***
; C     OPEN THE LISTING FILE !$LPT!
; C***
;       CALL OPEN (LIST,"$LPT",0,IER)
; C***
; X     IF (IER.EQ.1) GOTO 15           ; CHECK IF OPEN SUCCESSFUL
; X     TYPE ! OPEN ERROR =!,IER        ; NO--TERMINATE PROCESSING
; X     STOP $LPT ERROR
; C***
; C     OPEN THE INVENTORY FILE FOR APPENDING ONLY BY THIS PROGRAM
; C     THIS IS INDICATED BY MODE 3 IN THE FOLLOWING OPEN CALL.
; C***
; 15    CALL APPEND (IVEN,"INVENTORY",3,IER,32)
; C***
;       IF (IER.EQ.13) GOTO 20          ; CHECK IF FILE EXISTS
; C***
;       IF (IER.EQ.1) GOTO 25           ; CHECK IF OPEN SUCCESSFUL
; C***
; X     TYPE ! OPEN ERROR =!,IER        ; NO--TERMINATE PROCESSING
; X     STOP INVENTORY ERROR
; C***
; C     ATTEMPTED TO OPEN A NON-EXISTENT FILE
; C     CREATE A RANDOM FILE !INVENTORY! AND GO OPEN IT
; C***
; 20    CALL CFILW ("INVENTORY",2,IER)
; C***
;       IF (IER.EQ.1) GOTO 15           ; TRY TO OPEN IT NOW
; C***
; X     TYPE ! FILE CREATE ERROR =!,IER ; ERROR WHILE CREATING FILE
; X     STOP CREATE !INVENTORY! ERROR
; 25    CONTINUE
```

```
C***
C***
C*****************************************************************
C***
C     PUT OUT HEADING INFORMATION INCLUDING TIME AND DATE
C***
C*****************************************************************
C***
      CALL TIME (ITIME,IER)                    ; GET TIME OF DAY
      CALL DATE (IDATE,IER)                    ; GET DATE VALUE
C***
      WRITE (LIST,28) ITIME,IDATE
C***
28    FORMAT (//" INVENTORY FILE UPDATE"
     *          T34,I2'':'I2'':'I2,I5'/'I2'/'I2/
     *          " ==================",//
     *          " NEW RECORDS ADDED TO INVENTORY FILE"//)
C***
C***
C*****************************************************************
C***
C     READ IN A CARD FROM THE CARD READER
C***
C*****************************************************************
C***
30    CONTINUE
      READ (INPUT,35,END=300,ERR=400) RAREA(1),RAREA(2),IDESC,
     *                LOCAT,RAREA(13),PRICE,RAREA(16)
C***
35    FORMAT (2A2,2X,24A1,2X,7A1,2X,A2,2X,5A1,I4)
C***
C     WRITE THE NEW RECORD TO THE LISTING FILE
C***
      WRITE (LIST,36) RAREA(1),RAREA(2),IDESC,LOCAT,
     *                          RAREA(13),PRICE,RAREA(16)
C***
36    FORMAT (1X,2A2,2X,24A1,2X,7A1,2X,A2,2X,5A1,I4)
C***
C*****************************************************************
C***
C     PACK THE DESCRIPTION READ INTO 'IDESC' IN A1 FORMAT
C     INTO THE 'RAREA' IN A3 FORMAT.
C***
C*****************************************************************
C***
      CALL A1A3 (IDESC,1,24,RAREA,3,NTBL)
```

```
C
C***
C*************************************************************************
C***
C     CONVERT THE STOCK LOCATION INFORMATION READ INTO 'LOCAT'
C     IN A1 FORMAT TO A DOUBLE WORD VALUE IN 'RAREA'.
C***
C*************************************************************************
C***
      CALL A1DW (LOCAT,1,7,RAREA(11),CSPER)
C***
X     IF (CSPER.EQ.0) GOTO 40          ; CHECK IF OVERFLOW ERROR
X     TYPE ' A1DW CONVERSION ERROR =',CSPER  ; YES--TERMINATE
X     STOP LOCATION ERROR
X 40  CONTINUE
C***
C*************************************************************************
C***
C     CONVERT THE UNIT QUANTITY PRICE INFORMATION READ
C     INTO 'PRICE' IN A1 FORMAT TO THE 'RAREA' AREA
C     IN DOUBLE WORD FORMAT.
C***
C*************************************************************************
C***
      CALL A1DW (PRICE,1,5,RAREA(14),CSPER)
C***
X     IF (CSPER.EQ.0) GOTO 50          ; CHECK IF OVERFLOW ERROR
X     TYPE ' A1DW CONVERSION ERROR =',CSPER
X     STOP PRICE ERROR
X 50  CONTINUE
C***
C*************************************************************************
C***
C     APPEND THE NEW RECORD OUT TO THE END OF THE INVENTORY FILE
C***
C*************************************************************************
C***
      WRITE BINARY (IVEN) RAREA
C***
C***
C*************************************************************************
C***
C     INCREMENT RECORD COUNT AND CONTINUE PROCESSING CARDS
C***
C*************************************************************************
C***
      IRCRD=IRCRD+1
      GOTO 30
```

```
;
; C***
; C***
; C*************************************************************************
; C***
; C      END OF CARD DECK FOUND
; C      TYPE OUT NUMBER OF CARDS PROCESSED THIS UPDATE
; C***
; C*************************************************************************
; C***
; 300    CONTINUE
; C***
;        TYPE ' RECORDS PROCESSED THIS UPDATE =',IRCRD,'<15>'
; C***
;        WRITE (LIST,310) IRCRD
; C***
; 310    FORMAT (//" RECORDS PROCESSED DURING THIS UPDATE =",I8)
; C***
;        CALL RESET                ; CLOSE ALL OPEN FILES/DEVICES
; C***
;        STOP END OF INVENTORY UPDATE PROGRAM
; C***
; C***
; C***
; C*************************************************************************
; C***
; C      AN ERROR WAS ENCOUNTERED IN READING AN INPUT CARD
; C***
; C*************************************************************************
; C***
; C***
; 400    CONTINUE
;        TYPE ' ERROR FOUND WHILE READING CARDS '
;        IRCRD=IRCRD-1
;        TYPE ' PROCESSING STOPPED AFTER ',IRCRD,' RECORDS'
; C***
;        CALL RESET                ; CLOSE ALL OPEN FILES
; C***
;        STOP CARD READ ERROR
; C***
;        END
```

PROGRAM # 1 - INPUT CARDS

```
CC.     1    5    10   15   20   25   30   35   40   45   50   55   60
        ....+....+....+....+....+....+....+....+....+....+....+....+....
        8264  NOVA 840 W/16K,  W/ MMPU  2715237  EA  16530  01
        8265  NOVA 840 W/16K, W/O MMPU  2715247  EA  13230  02
        8290  NOVA 840 W/24K,  W/ MMPU  2715420  EA  19730  03
        8291  NOVA 840 W/32K,  W/ MMPU  2715267  EA  22930  04
        8292  NOVA 840 W/40K,  W/ MMPU  2715601  EA  26130  05
        8293  NOVA 840 W/48K,  W/ MMPU  2716234  EA  29330  06
        8294  NOVA 840 W/64K,  W/ MMPU  2716320  EA  35730  07
        8295  NOVA 840 W/80K,  W/ MMPU  2717086  EA  45130  08
        8296  NOVA 840 W/96K,  W/ MMPU  2717742  EA  51530  09
        8297  NOVA 840 W/128K, W/ MMPU  2717748  EA  64330  10
        8298  NOVA 840 W/24K, W/O MMPU  2719382  EA  16430  11
        8299  NOVA 840 W/32K, W/O MMPU  2720416  EA  19630  12
```

PROGRAM # 1 - CONSOLE LOG

```
R
CSP1

 *** INVENTORY UPDATE PROGRAM ***

 RECORDS PROCESSED THIS UPDATE =      12

STOP END OF INVENTORY UPDATE PROGRAM
R
```

PROGRAM # 1 - PRINTER OUTPUT (Verification of Input)

INVENTORY FILE UPDATE 16:14:46 6/16/74

NEW RECORDS ADDED TO INVENTORY FILE

8264	NOVA 840 W/16K, W/ MMPU	2715237	EA	16530	1
8265	NOVA 840 W/16K, W/O MMPU	2715247	EA	13230	2
8290	NOVA 840 W/24K, W/ MMPU	2715420	EA	19730	3
8291	NOVA 840 W/32K, W/ MMPU	2715267	EA	22930	4
8292	NOVA 840 W/40K, W/ MMPU	2715601	EA	26130	5
8293	NOVA 840 W/48K, W/ MMPU	2716234	EA	29330	6
8294	NOVA 840 W/65K, W/ MMPU	2716320	EA	35730	7
8295	NOVA 840 W/80K, W/ MMPU	2717086	EA	45130	8
8296	NOVA 840 W/96K, W/ MMPU	2717742	EA	51530	9
8297	NOVA 840 W/128K, W/ MMPU	2717748	EA	64330	10
8298	NOVA 840 W/24K, W/O MMPU	2719382	EA	16430	11
8299	NOVA 840 W/32K, W/O MMPU	2720416	EA	19630	12

RECORDS PROCESSED DURING THIS UPDATE = 12

```
C***
C***
C*************************************************************************
C***
C     FORTRAN COMMERCIAL SUBROUTINE PACKAGE EXAMPLE PROGRAM #2
C     ========================================================
C***
C     THE PURPOSE OF THIS EXAMPLE IS TO SHOW THE EASE OF
C     USING THE COMMERCIAL SUBROUTINE PACKAGE (CSP) IN CONJUNCTION
C     WITH THE STANDARD DATA GENERAL SUPPLIED REAL TIME DISK
C     OPERATING SYSTEM (RDOS) IN BUSINESS AND COMMERCIAL
C     APPLICATIONS.
C***
C     THE PROGRAM WILL PRINT AN INVENTORY SUMMARY OF ALL NOVA 640
C     COMPUTERS FOUND IN STOCK, IT ILLUSTRATES SOME OF THE MANY
C     WAYS OF HANDLING DATA WITH RDOS AND THE FORTRAN CSP PACKAGE.
C***
C*************************************************************************
C***
C     THE FILE TO BE ACCESSED IS AN INVENTORY FILE WITH THE
C     FOLLOWING ORGANIZATION:-
C***
C        RECORD ITEM DESCRIPTION       WORDS    FORMAT
C        ======================        =====    ======
C***
C     STOCK NUMBER (XXXX)                1-2     2A2
C     DESCRIPTION (24 CHARACTERS)        3-10    8A3
C     STOCK LOCATION (XXXX-XXX)         11-12     DW
C     UNIT QUANTITY CODE (EA,DZ,CT)       13      A2
C     UNIT QUANTITY PRICE ($XXXXX.XX)   14-15     DW
C     QUANTITY ON HAND (XXXX)             16      SP
C***
C     TOTAL RECORD SIZE IS 16 = 16 BIT WORDS
C***
C*************************************************************************
```

```
C***
C***
C*************************************************************************
C***
C     DATA STORAGE ALLOCATION AREA
C***
C*************************************************************************
C***
      INTEGER RAREA,PRICE,CSPER,WORK,TVAL
C***
      DIMENSION IDESC(24),LOCAT(7),NUMBR(2),ITITL(40)
      DIMENSION ITIME(3),IDATE(3)
      DIMENSION PRICE(2),IPRIC(5),INUM(4),IVAL(6),WORK(8),TVAL(8)
C***
      COMMON /CHLS/ LIST,IVEN
      COMMON /RECRD/ RAREA(16),NTBL(40),TOTAL
      COMMON /MASK/ MASK1(8),MASK2(10)
C***
      DATA LIST,IVEN /0,1/ TOTAL/0.0/
      DATA NTBL /"  1 2 3 4 5 6 7 8 9 0 A B C D E F G H I J ",
     *           "K L M N O P Q R S T U V W X Y Z , . / "/
      DATA MASK1/'        ,     0 '/MASK2/'    ,       ,     $ '/
C***
      CSPER=0                   ! INITIALIZE CSPER ERROR FLAG
C***
C***
C***
C*************************************************************************
C***
C     INITIALIZE FILES AND GET STARTING/ENDING RECORD NUMBERS
C***
C*************************************************************************
C***
      TYPE ' '
      TYPE ' *** INVENTORY ANALYSIS PROGRAM ***'
      TYPE ' '
      TYPE ' ENTER A 40 CHARACTER TITLE FOR THIS RUN'
      TYPE ' '
C***
C     READ IN 40 CHARACTERS FROM THE CONSOLE KEYBOARD
C***
      CALL KEYBD (ITITL,1,40)
      TYPE ' '
C***
5     ACCEPT ' STARTING RECORD NUMBER = ',ISTRT
      ACCEPT ' ENDING RECORD NUMBER   = ',IEND
C***
      IF (IEND.LT.ISTRT) GOTO 5
      TYPE ' '
C***
      CALL OPEN (LIST,"$LPT",0,IER)
      CALL OPEN (IVEN,"INVENTORY",0,IER,32)
```

```
C***
C***
C*************************************************************************
C***
C     PUT OUT REPORT HEADING
C***
C*************************************************************************
C***
      CALL TIME (ITIME,IER)              ; GET CUURENT TIME OF DAY
      CALL DATE (IDATE,IER)              ; GET TODAY'S DATE
C***
      WRITE (LIST,10) ITITL,ITIME(1),ITIME(2),IDATE
C***
10    FORMAT (///,T10,40A1,T54,I2':'I2,I4'/'I2'/'I2/
     *        T10,40('=')//
     *        " STOCK   STOCK        ITEM DESCRIPTION",
     *        T43,"UNIT   QTY    UNIT   VALUE"/
     *        " LOCATION NUMBER",T43,"CODE ON HAND   $        $" /
     *        " --------- ------      ---------------",
     *        T43,"---- -------  ----   -----"
     *  /)
C***
C*************************************************************************
C***
C     SEQUENTIALLY PROCESS THE INVENTORY FILE FROM THE STARTING
C     RECORD 'ISTRT' TO THE ENDING RECORD 'IEND' BY READING
C     THE RECORD FROM THE DISK FILE 'INVENTORY' , EXPANDING THE
C     RECORD DATA , FORMATTING IT FOR OUTPUT , AND OUTPUTTING
C     IT TO THE LINE PRINTER - '$LPT'.
C***
C*************************************************************************
C***
      DO 1000 I=ISTRT,IEND
C***
C     READ IN THE 'ITH' RECORD FROM THE DISK FILE
C***
      CALL RDRW (IVEN,I,RAREA,1,IER)
C***
      IF (IER.EQ.9) GOTO 2000            ; END OF FILE ENCOUNTERED
C***
X     IF (IER.EQ.1) GOTO 20              ; INSURE RECORD READ OKAY
X     TYPE ' RECORD READ ERROR =',IER
X     GOTO 9000
X 20  CONTINUE
C***
```

```
} C***
} C***
} C*************************************************************************
} C***
} C      EXPAND THE STOCK LOCATION VALUE
} C              RECORDED IN WORDS 11 AND 12 IN DOUBLE WORD FORMAT
} C***
} C*************************************************************************
} C***
}       CALL DWA1 (RAREA(11),LOCAT,1,7)
} C***
} C*************************************************************************
} C***
} C      EXPAND THE DESCRIPTIION
} C              RECORDED IN WORDS 3 TO 10 IN A3 FORMAT
} C***
} C*************************************************************************
} C***
}       CALL A3A1 (RAREA,3,10,IDESC,1,NTBL)
} C***
} C***
} C*************************************************************************
} C***
} C      MOVE THE UNIT QUANTITY CODE
} C              RECORDED IN WORD 13 IN A2 FORMAT
} C***
} C*************************************************************************
} C***
}       ICODE=RAREA(13)
} C***
} C*************************************************************************
} C***
} C      GET THE QUANTITY ON HAND
} C              RECORDED IN WORD 16 IN SINGLE WORD FORMAT
} C***
} C*************************************************************************
} C***
}       NUMBR(1)=0                ; SET HIGH ORDER POSITION ZERO
}       NUMBR(2)=RAREA(16)        ; PUT QUANTITY IN LOW ORDER POSITION
} C***
}       CALL DWA1 (NUMBR,INUM,1,4)
} C***
} C*************************************************************************
} C***
} C      GET THE UNIT QUANTITY PRICE AND CONVERT FOR OUTPUT
} C              RECORDED IN WORDS 14 AND 15 IN DOUBLE WORD FORMAT
} C***
} C*************************************************************************
} C***
}       PRICE(1)=RAREA(14)
}       PRICE(2)=RAREA(15)
} C***
}       CALL DWA1 (PRICE,IPRIC,1,5)
```

```
C***
C***
C***
C*************************************************************************
C***
C     CALCULATE THE PRODUCT VALUE
C***
C     VALUE = (QUANTITY ON HAND) * (UNIT COST)
C***
C*************************************************************************
C***
      CALL DWMPY (NUMBR,PRICE,CSPER)
C***
X     IF (CSPER.EQ.0) GOTO 60
X     TYPE ' DWMPY ERROR =',CSPER
X     GOTO 9000
X 50  CONTINUE
C***
      CALL DWA1 (PRICE,IVAL,1,6)        ; CONVERT TO A1 FORMAT
C***
C     ADD THE VALUE ON HAND TO THE TOTAL INVENTORY VALUE
C***
      CALL DWADD (PRICE,TOTAL,CSPER)    ; ADD TWO VALUES
C***
X     IF (CSPER.EQ.0) GOTO 70           ; INSURE ADDITION OKAY
X     TYPE ' DWADD ERROR =',CSPER
X     GOTO 9000
X 70  CONTINUE
C***
C*************************************************************************
C***
C     FORMAT AND PRINT THE LINE ITEM ON THE LIST FILE '$LPT'
C***
C*************************************************************************
C***
      CALL QMOVE (MASK1,1,8,WORK,1)             ; GET VALUE MASK
C***
      CALL EDIT (IVAL,1,6,WORK,1,8)             ; EDIT VALUE
C***
      WRITE (LIST,200) LOCAT,RAREA(1),RAREA(2),IDESC,
     *                 ICODE,INUM,IPRIC,WORK
C***
200   FORMAT (' ',4A1'-'3A1,2X,2A2,2X,24A1,2X,A2,4X,4A1,
     *        '  '2A1','3A1,8A1)
C***
1000  CONTINUE
```

```
; C***
; C***
; C*************************************************************************
; C***
; C      FORMAT AND PRINT THE SUMMARY INFORMATION
; C***
; C*************************************************************************
; C***
; 2000  CONTINUE
;       CALL DWA1 (TOTAL,TVAL,1,8)       ; CONVERT VALUE TO A1 FORMAT
; C***
;       CALL EDIT (TVAL,1,8,MASK2,1,10) ; EDIT TOTAL FOR OUTPUTTING
; C***
;       WRITE (LIST,2100) MASK2          ; OUTPUT SUMMARY INFORMATION
; C***
; 2100  FORMAT (///T25'TOTAL VALUE OF INVENTORY ON HAND = ',10A1)
; C***
; C***
; C*************************************************************************
; C***
; C      CLOSE ALL OPEN FILES AND DEVICES
; C***
; C*************************************************************************
; C***
; X9000 CONTINUE
;       CALL CLOSE (IVEN,IER)
;       CALL CLOSE (ILPT,IER)
; C***
; C*************************************************************************
; C***
; C      PROGRAM IS FINISHED, RETURN TO RDOS
; C***
; C*************************************************************************
; C***
;       TYPE ' '
;       STOP INVENTORY ANALYSIS COMPLETED
;       END
```

PROGRAM #2 - CONSOLE LOG

```
R
CSP2

 *** INVENTORY ANALYSIS PROGRAM ***

 ENTER A 40 CHARACTER TITLE FOR THIS RUN

* NOVA 840 INVENTORY POSITION

 STARTING RECORD NUMBER = 0
 ENDING RECORD NUMBER   = 11


STOP INVENTORY ANALYSIS COMPLETED
R
```

PROGRAM #2 - PRINTER OUTPUT

NOVA 840 INVENTORY POSITION 16:15 5/16/74

STOCK LOCATION	STOCK NUMBER	ITEM DESCRIPTION	UNIT CODE	QTY ON HAND	UNIT $	VALUE $
2715-237	8264	NOVA 840 W/16K, W/ MMPU	EA	0001	16,530	16,530
2715-247	8265	NOVA 840 W/16K, W/O MMPU	EA	0002	13,230	26,460
2715-420	8290	NOVA 840 W/24K, W/ MMPU	EA	0003	19,730	59,190
2715-267	8291	NOVA 840 W/32K, W/ MMPU	EA	0004	22,930	91,720
2715-601	8292	NOVA 840 W/40K, W/ MMPU	EA	0005	26,130	130,650
2716-234	8293	NOVA 840 W/48K, W/ MMPU	EA	0006	29,330	175,980
2716-320	8294	NOVA 840 W/65K, W/ MMPU	EA	0007	35,730	250,110
2717-086	8295	NOVA 840 W/80K, W/ MMPU	EA	0008	45,130	361,040
2717-742	8296	NOVA 840 W/96K, W/ MMPU	EA	0009	51,530	463,770
2717-748	8297	NOVA 840 W/128K, W/ MMPU	EA	0010	64,330	643,300
2719-382	8298	NOVA 840 W/24K, W/O MMPU	EA	0011	16,430	180,730
2720-416	8299	NOVA 840 W/32K, W/O MMPU	EA	0012	19,630	235,560

TOTAL VALUE OF INVENTORY ON HAND = $2,635,040

```
C***
C***
C**************************************************************************
C***
C      FORTRAN COMMERCIAL SUBROUTINE PACKAGE EXAMPLE PROGRAM #3
C      ========================================================
C***
C      THE PURPOSE OF THIS PROGRAM IS TO SHOW THE USE OF THE
C      FORTRAN CSP SUBROUTINES IN AN INTERACTIVE APPLICATION
C      WHERE A USER BUILDS UP A FILE OF DATA ABOUT A GROUP
C      OF CUSTOMERS.
C***
C***
C      THE PROGRAM QUERIES THE OPERATOR FOR THE CUSTOMER NAME,
C      ADDRESS INCLUDING THE STREET,CITY,STATE,AND ZIP CODE AND
C      A CURRENT BALANCE. THE OPERATOR SIGNALLS THE PROGRAM
C      THAT THE OPERATION HAS BEEN COMPLETED BY ENTERING THE
C      FOLLOWING MESSAGE 'END OF FILE ' FOR A CUSTOMER NAME.
C***
C      DATA IS FACKED INTO THE FILE IN A2 FORMAT AND IS WRITTEN
C      TO THE FILE IN SEQUENTIAL FASHION IN 35 WORD RECORDS.
C***
C**************************************************************************
C***
C      THE MASTER FILE 'MASTERFILE' HAS THE FOLLOWING
C      RECORD FORMAT :-
C***
C              POSITION          ITEM DESCRIPTION
C              ========          ================
C***
C                 1-20           CUSTOMER NAME
C                21-40           STREET ADDRESS
C                41-60           CITY, STATE, ZIP CODE
C                61-70           BALANCE
C***
C      THE LAST RECORD IN THE MASTER FILE CONTAINS THE FOLLOWING
C      MESSAGE ,'END OF FILE ', IN POSITIONS 1-12 TO INDICATE
C      THE END OF THE FILE .
C***
C**************************************************************************
```

```
C***
C***
C***
C***
C**************************************************************************
C***
C     DATA STORAGE AREA ALLOCATION AND INITIALIZATION
C***
C**************************************************************************
C***
      DIMENSION IN(70),INS(35)
      COMMON /LBL/IEF(12)
      DATA IEF/'E N D   O F   F I L E   '/
C***
      TYPE ' '
      TYPE ' *** MASTERFILE BUILD/ANALYSIS PROGRAM ***'
      TYPE ' '
C***
1000  TYPE ' '
      ACCEPT ' OPTION (1=NEW,2=LIST,3=END) ',IOPT
C***
      IF (IOPT.LT.1.OR.IOPT.GT.3) GOTO 1000
C***
      GOTO (2000,3000,4000),IOPT
```

```
; C***
; C*****************************************************************
; C***
; C***
; C       COMMUNICATE WITH THE OPERATOR FOR INPUT TO BUILD UP
; C       THE MASTERFILE. THE FILE IS COMPLETED BY THE OPERATOR
; C       ENTERING 'END OF FILE ' IN RESPONCE TO THE NAME QUERY.
; C***
; C*****************************************************************
; C***
; 2000    CALL OPEN (1,'MASTERFILE',0,IER)
; C***
; 2050    TYPE ' '
;         TYPE ' NAME 10A2'
;         CALL KEYBD (IN,1,20)
; C***
;         IF (NCOMP(IN,1,12,IEE,1)) 15,40,15
; 15      CONTINUE
; C***
;         TYPE ' '
;         TYPE ' ADDRESS 10A2'
;         CALL KEYBD (IN,21,40)
; C***
;         TYPE ' '
;         TYPE ' CITY,STATE,ZIP 10A2'
;         CALL KEYBD (IN,41,60)
; C***
;         TYPE ' '
;         TYPE ' BALANCE (CENTS)'
;         CALL KEYBD (IN,61,70)
; C***
;         CALL RJUST (IN,61,70)
; C***
;         CALL PACK (IN,1,70,INS,1)
;         WRITE BINARY (1) INS
;         GOTO 2050
; C***
; 40      CALL FILL (IN,13,70,'  ')
;         CALL PACK (IN,1,70,INS,1)
;         WRITE BINARY (1) INS
;         CALL CLOSE (1,IER)
;         GOTO 1000
```

```
; C***
; C***
; C*****************************************************************
; C***
; C     LIST THE CONTENTS OF THE 'MASTERFILE' TO THE PRINTER.
; C***
; C*****************************************************************
; C***
; 3000  CONTINUE
;       CALL OPEN (0,'$LPT',0,IER)
;       CALL OPEN (1,'MASTERFILE',0,IER)
;       WRITE (0,3001)
; 3001  FORMAT (//T10'"MASTERFILE" FILE CONTENTS'/
;      *                T10'==========================='/)
; 3005  READ BINARY (1) INS
; C***
;       CALL UNPAC (INS,1,6,IN,1)
; C***
;       IF (NCOMP(IN,1,11,IFF,1)) 3010,3020,3010
; C***
; 3010  CONTINUE
;       WRITE (0,3011) INS
; 3011  FORMAT (T10,10A2/T12,10A2/T14,10A2//T12'BALANCE = ',5A2//)
;       GOTO 3005
; C***
; 3020  CONTINUE
;       WRITE (0,3021)
; 3021  FORMAT (//T10'*** END OF FILE ***')
;       CALL CLOSE (0,IER)
;       CALL CLOSE (1,IER)
;       GOTO 1000
; C***
; C*****************************************************************
; C***
; C     END OF ANALYSIS PROGRAM
; C***
; C*****************************************************************
; C***
; 4000  TYPE ' '
;       STOP  MASTERFILE BUILD/ANALYSIS COMPLETED
;       END
```

PROGRAM #3 -- CONSOLE LOG AND OPERATOR INPUT

```
R
CSP3

 *** MASTERFILE BUILD/ANALYSIS PROGRAM ***


 OPTION (1=NEW,2=LIST,3=END) 1

 NAME 10A2
* ELIZABETH KAHN

 ADDRESS 10A2
* 293 HOWLAND RD.

 CITY,STATE,SIP 10A2
* MARLBORO, MASS 01752

 BALANCE (CENTS)
* 2323

 NAME 10A2
* RALPH

 NAME 10A2
* DAVES MARKET

 ADDRESS 10A2
* 1997 WASHINGTON ST.

 CITY,STATE,SIP 10A2
* NEWTOWN, MASS 02158

 BALANCE (CENTS)
* 7819
 NAME 10A2
* RUNNING MOTORS

 ADDRESS 10A2
* 10 WATER STREET

 CITY,STATE,SIP 10A2
* PLYMOUTH, ROCK 02296

 BALANCE (CENTS)
* 26068

 NAME 10A2
* END OF FILE

 OPTION (1=NEW,2=LIST,3=END) 2

 OPTION (1=NEW,2=LIST,3=END) 3

STOP  MASTERFILE BUILD/ANALYSIS COMPLETED
R
```

PROGRAM #3 - LISTING OF INPUT PRODUCED BY PROGRAM

```
"MASTERFILE" FILE CONTENTS
**************************

ELIZABETH KAHN
  293 HOWLAND RD.
    MARLBORO, MASS 01752

  BALANCE =       2323


RALPH E. MATHEWSON
  193 RIVERBANK DR.
    MAYNARD, MASS 01754

  BALANCE =       1616


VIOLET A. SAWYER
  93 EAST MAIN ST.
    HUDSON, MASS 01749

  BALANCE =       4824


DAVES MARKET
  1997 WASHINGTON ST.
    NEWTOWN, MASS 02158

  BALANCE =       7819


RUNNING MOTORS
  10 WATER STREET
    PLYMOUTH, ROCK 02296

  BALANCE =      26068



*** END OF FILE ***
```

```
C***
C***
C*************************************************************************
C***
C       FORTRAN COMMERCIAL SUBROUTINE PACKAGE EXAMPLE PROGRAM #4
C       ********************************************************
C***
C       THE PURPOSE OF THIS PROGRAM IS SIMILAR TO PROGRAM #3
C       EXCEPT THAT IT ILLUSTRATES THE USE OF SOME ADDITIONAL
C       CONVERSION CAPABILITIES. IT ALLOWS
C       THE OPEATOR TO ENTER INTEGER OR FLOATING POINT DATA FOR
C       THE AMOUNT AND QUANTITY VALUES.
C***
C       THE OPERATOR SIGNALS THE END OF THE FILE BUILD-UP PROCESS
C       BY ENTERING 'END OF FILE ' FOR THE CUSTOMER NAME.
C       IF THE OPERATOR ENTERS A LEGAL CUSTOMER NAME, A HEADER OR
C       CUSTOMER RECORD IS PUT OUT TO THE 'DAILY' TRANSACTION FILE.
C       THE PROGRAM THEN ASKS THE OPERATOR FOR AN ITEM DESCRIPTION.
C       IF 'END OF DATA ' IS ENTERED INSTEAD OF AN ITEM DESCRIPTION,
C       THE PROGRAM REQUESTS THE NEXT CUSTOMER NAME, AFTER A
C       LEGAL ITEM DESCRIPTION HAS BEEN ENTTERED, THE PROGRAM ASKS
C       FOR THE QUANTITY, AND INDIVIDUAL ITEM PRICE, CALCULATES THE
C       TOTAL AMOUNT (QUANTITY*ITEM PRICE) , AND OUTPUTS THE VALUE
C       TO THE FILE.
C***
C***
C*************************************************************************
C***
C       THE TRANSACTION FILE 'DAILY' HAS ONE OF THE
C       FOLLOWING RECORD FORMATS :-
C***
C               A) CUSTOMER NAME RECORD
C                  --------------------
C***
C                1-20   CUSTOMER NAME
C               21-30   BLANK
C               31-32   ZERO
C***
C               B) CUSTOMER ITEM RECORD
C                  --------------------
C***
C                1-20   ITEM DESCRIPTION
C               21-28   TOTAL AMOUNT
C               29-32   QUANTITY
C***
C               C) END OF DATA RECORD
C                  -----------------
C***
C                1-12   "END OF FILE "
C               13-30   BLANK
C               31-32   ZERO
C***
C*************************************************************************
```

```
C***
C***
C***
C*************************************************************************
C***
C       DATA STORAGE AREA ALLOCATION AND INITIALIZATION
C***
C*************************************************************************
C***
        DOUBLE PRECISION AMT,QTY
C***
        DIMENSION IN(32),INS(16)
C***
        COMMON /LBL/IEF(12),IED(12)
C***

        DATA IED/'E N D   O F   D A T A   '/
        DATA IEF/'E N D   O F   F I L E   '/
C***
        TYPE ' '
        TYPE ' *** DAILY FILE BUILD/ANALYSIS PROGRAM ***'
        TYPE ' '
C***
1000    TYPE ' '
        ACCEPT ' OPTION (1=NEW,2=LIST,3=END) ',IOPT
C***
        IF (IOPT.LT.1.OR.IOPT.GT.3) GOTO 1000
C***
        GOTO (2000,3000,4000),IOPT
```

```
C***
C*************************************************************************
C***
C       COMMUNICATE WITH THE OPERATOR TO BUILD AN INITIAL 'DAILY'
C       FILE OF TRANSACTIONS FOR TODAY .
C***
C*************************************************************************
C***
2000    CALL OPEN (1,'DAILY',0,IER)
2050    TYPE ' '
        TYPE ' NAME 10A2'
        CALL KEYBD (IN,1,20)
C***
        IF (NCOMP(IN,1,12,IEF,1)) 2100,2500,2100
C***
2100    CONTINUE
        CALL FILL (IN,21,30,'  ')
        CALL PACK (IN,1,30,INS,1)
        INS(16)=0
        WRITE BINARY (1) INS
C***
2200    CONTINUE
        TYPE ' '
        TYPE ' ITEM DESCRIPTION 10A2'
        CALL KEYBD (IN,1,20)
C***
        IF (NCOMP(IN,1,11,IED,1)) 2300,2050,2300
C***
2300    CONTINUE
        TYPE ' '
2350    ACCEPT ' QUANTITY -9999 => 9999 ',QTY
        IF (QTY.LT.-9999.D0.OR.QTY.GT.9999.D0) GOTO 2350
        CALL PUT (IN,29,32,QTY,0.5,0)
C***
        TYPE ' '
2400    ACCEPT ' ITEM PRICE ',AMT
        IF (AMT.LT.0.D0) GOTO 2400
C***
C*************************************************************************
C***
C       CALCULATE THE  TOTAL AMOUNT
C***
C       AMOUNT=ITEM PRICE(AMT) * QUANTITY(QTY)
C***
C*************************************************************************
C***
        AMT=AMT*QTY*100.0D0
C***
        CALL PUT (IN,21,28,AMT,0.5,0)
C***
        CALL PACK (IN,1,32,INS,1)
        WRITE BINARY (1) INS
        GOTO 2200
C***
2500    CALL FILL (IN,12,30,'  ')
        CALL PACK (IN,1,30,INS,1)
        INS(16)=0
        WRITE BINARY (1) INS
        CALL CLOSE (1,IER)
        GOTO 1000
```

```
C***
C***
C***
C*************************************************************************
C***
C       LIST THE CONTENTS OF THE !DAILY! FILE TO THE PRINTER
C***
C*************************************************************************
C***
3000    CONTINUE
        CALL OPEN (0,!@LPT!,0,IER)
        CALL OPEN (1,!DAILY!,0,IER)
C***
        WRITE (0,3001)
3001    FORMAT (//T10!"DAILY" FILE CONTENTS!/
     *           T10!====================!/)
C***
3005    READ BINARY (1) INS
C***
        CALL UNPAC (INS,1,6,IN,1)
C***
        IF (INS(16)) 3010,3020,3010
C***
3010    CONTINUE
        WRITE (0,3011) INS(15),INS(16),(INS(I),I=1,10),(INS(I),I=11,14
3011    FORMAT (T10,2A2!   !10A2!   !4A2)
        GOTO 3005
C***
3020    CONTINUE
        IF (NCOMP(IN,1,11,IEF,1)) 3021,3030,3021
3021    WRITE (0,3022) (INS(I),I=1,10)
3022    FORMAT (//T5,10A2/)
        GOTO 3005
C***
3030    WRITE (0,3031)
3031    FORMAT (//T10!*** END OF FILE ***!)
        CALL CLOSE (0,IER)
        CALL CLOSE (1,IER)
        GOTO 1000
C***
C*************************************************************************
C***
C       END OF BUILD/ANAYSIS PROGRAM
C***
C*************************************************************************
C***
4000    TYPE ! !
        STOP DAILY BUILD/ANALYSIS COMPLETED
        END
```

```
R
CSP4

 *** DAILY FILE BUILD/ANALYSIS PROGRAM ***

 OPTION (1=NEW,2=LIST,3=END) 1

 NAME 10A2
* RUNNING MOTORS

 ITEM DESCRIPTION 10A2
* AIR CLEANERS - CASES

 QUANTITY -9999 -> 9999
* 20

 ITEM PRICE
* 10.0

 ITEM DESCRIPTION 10A2
* GREASE - BARRELS

 QUANTITY -9999 -> 9999
* 6

 ITEM PRICE
* 27.21

 ITEM DESCRIPTION 10A2
* TIRES - 850 X 15

 QUANTITY -9999 -> 9999
* 50

 ITEM PRICE
* 20.24



 ITEM DESCRIPTION 10A2
* END OF DATA

 NAME 10A2
* VIOLET A. SAWYER

 ITEM DESCRIPTION 10A2
* TOMATO SOUP - CANS

 QUANTITY -9999 -> 9999
* 4

 ITEM PRICE


 ITEM DESCRIPTION 10A2
* END OF  DATA
```

```
 NAME 10A2
* DAVES MARKET

 ITEM DESCRIPTION 10A2
* POTATOES - BAGS

 QUANTITY -9999 -> 9999
* 25

 ITEM PRICE
* 1.10




 ITEM DESCRIPTION 10A2
* HAM ( RETURNED )

 QUANTITY -9999 -> 9999
* -12

 ITEM PRICE
* 3.06

 ITEM DESCRIPTION 10A2
* END OF DATA


 NAME 10A2
* END OF FILE

 OPTION (1=NEW,2=LIST,3=END) 2

 OPTION (1=NEW,2=LIST,3=END) 3

STOP DAILY BUILD/ANALYSIS COMPLETED
R
```

"DAILY" FILE CONTENTS
****

RUNNING MOTORS

20	AIR CLEANERS - CASES	20003
6	GREASE - BARRELS	16524
50	TIRES - 850 X 15	101200
2	TIRES - SPARE	12345
12	GASOLINE CAPS	1212
1	SCRAPPERS	22

VIOLET A. SAWYER

4	TOMATO SOUP - CANS	92
12	CORNED BEEF - CANS	3375
4	ROAST BEEF - POUND	972
2	GINGER ALE - CASES	176

DAVES MARKET

25	POTATOES - BAGS	2750
100	TOMATOES - LOOSE	2400
50	CARROTS - BUNCHES	1123
100	BREAD - LOAF	4500
10	MILK - QUARTS	455
40	MILK - HALF GALS	3200
12	HAM (RETURNED)	367K

*** END OF FILE ***

```
C***
C***
C*************************************************************************
C***
C     FORTRAN COMMERCIAL SUBROUTINE PACKAGE EXAMPLE PROGRAM #5
C     ********************************************************
C***
C     THE PURPOSE OF THIS PROGRAM IS TO CREATE INVOICES FROM
C     A DAILY TRANSACTION FILE THAT WAS BUILT DURING THE DAY.
C     EACH CUSTOMER MUST HAVE A MASTER RECORD IN A FILE CALLED
C     'MASTERFILE'. TRANSACTIONS ARE FOUND IN A FILE
C     CALLED 'DAILY' CREATED BY EXAMPLE PROGRAM # 4.
C***
C     THE PROGRAM READS THE FIRST RECORD FROM 'DAILY', COMPARES
C     IT WITH ENTRIES IN THE MASTER FILE UNTIL IT EITHER
C     FINDS A MATCH OR THE END OF THE FILE IS REACHED WHICH
C     INDICATED BY THE NAME FIELD CONTAINING 'END OF FILE '.
C***
C     IF A MATCH IS FOUND, THE CUSTOMER NAME,ADDRESS,AND
C     PREVIOUS BALANCE ARE PRINTED. THE 'DAILY' FILE IS
C     THEN ACCESSED FOR INDIVIDUAL TRANSACTION ENTRIES WHICH
C     ARE PRINTED WITH THEIR DOLLAR AMOUNT AND QUANTITY TO
C     THE LIST FILE. THIS CONTINUES UNTIL EITHER THE NEXT
C     CUSTOMER NAME RECORD OR 'END OF DATA ' RECORD IS
C     ENCOUNTERED. WHEN THIS OCCURS THE TOTAL FOR THE
C     INVOICE IS PRINTED AND THE NEXT CUSTOMER IS PROCESSED
C     OR THE PROGRAM IS TERMINATED.
C***
C     IF A MATCH IS NOT FOUND, THE CUSTOMER NAME IS LOGGED
C     OUT TO THE CONSOLE PRINTER AND THE PROGRAM MOVES
C     THROUGH THE TRANSACTION FILE 'DAILY' TO THE NEXT
C     CUSTOMER NAME RECORD OR THE 'END OF DATA ' RECORD.
C***
C*************************************************************************
```

```
C***
C***
C*************************************************************************
C***
C     THE MASTER FILE 'MASTERFILE' HAS THE FOLLOWING
C     RECORD FORMAT :-
C***
C              POSITION          ITEM DESCRIPTION
C              ********          ****************
C***
C                1-20            CUSTOMER NAME
C               21-40            STREET ADDRESS
C               41-60            CITY, STATE, ZIP CODE
C               61-70            BALANCE
C***
C     THE LAST RECORD IN THE MASTER FILE CONTAINS THE
C     FOLLOWING 'END OF FILE ' IN POSITIONS 1-12 TO INDICATE
C     THE END OF THE FILE .
C***
C     THE TRANSACTION FILE 'DAILY' HAS ONE OF THE
C     FOLLOWING RECORD FORMATS :-
C***
C              A) CUSTOMER NAME RECORD
C                 --------------------
C***
C               1-20   CUSTOMER NAME
C              21-30   BLANK
C              31-32   ZERO
C***
C              B) CUSTOMER ITEM RECORD
C                 --------------------
C***
C               1-20   ITEM DESCRIPTION
C              21-28   TOTAL AMOUNT
C              29-32   QUANTITY
C***
C              C) END OF DATA RECORD
C                 -----------------
C***
C               1-12   "END OF FILE "
C              13-30   BLANK
C              31-32   ZERO
C***
C*************************************************************************
```

```
C***
C***
C*****************************************************************
C***
C     DATA STORAGE ALLOCATION AREA
C***
C*****************************************************************
C***
      INTEGER CSPER,FILER
C***
      DIMENSION ITOT(5),ISUM(8),IPRNT(80)
      DIMENSION INRCD(16),IHEAD(35),IDATA(35)
C***
      COMMON /TERM/ IEOF(6),IEOD(6),MASK(14),IPRVB(8),NWBAL(6)
      COMMON /CHLS/ MASTR,ITRAN,INVCE
      COMMON /RECRD/ IRCRD
C***
      DATA IEOF/'END OF FILE '/IEOD/'END OF DATA '/
      DATA MASK/'          ,     $ .     C R '/
      DATA MASTR,ITRAN,INVCE/0,1,2/IRCRD/0/
C***
      DATA IPRVB/'PREVIOUS BALANCE'/
      DATA NWBAL/'NEW BALANCE '/
C***
C     INITIALIZE THE CSP ERROR FLAG 'CSPER' TO ZERO.
C     IF IT CHANGES FROM THIS VALUE DURING PROGRAM EXCUTION
C     AN ERROR IS INDICATED.
C***
      CSPER=0
```

```
C***
C***
C*************************************************************************
C***
C     LOG OPERATION DESCRIPTION ON THE CONSOLE LISTING DEVICE
C***
C*************************************************************************
C***
      TYPE ' '
      TYPE ' *** INVOICE PROGRAM ***'
      TYPE ' '
C***
C*************************************************************************
C***
C     OPEN AND INITIALIZE ALL FILES AND DEVICES
C***
C*************************************************************************
C***
C     OPEN THE LINE PRINTER AS THE INVOICE FILE
C***
      CALL OPEN (INVCE,"$LPT",0,FILER)
C***
      IF (FILER.EQ.1) GOTO 5          ; CHECK IF OPEN SUCCESSFUL
      TYPE 'OPEN ERROR =',FILER       ; NO--TERMINATE PROCESSING
      STOP INVOICE ERROR
5     CONTINUE
C***
C     OPEN THE MASTER FILE 'MASTERFILE' WITH 70 CHAR. RECORDS
C***
      CALL OPEN (MASTR,"MASTERFILE",0,FILER,70)
C***
      IF (FILER.EQ.1) GOTO 10         ; CHECK IF OPEN SUCCESSFUL
      TYPE ' OPEN ERROR =',FILER      ; NO--TERMINATE PROCESSING
      STOP 'MASTERFILE' ERROR
C***
C     OPEN THE TRANSACTION FILE 'DAILY' WITH 32 CHAR. RECORDS
C***
10    CALL OPEN (ITRAN,'DAILY',0,FILER,32)
C***
      IF (FILER.EQ.1) GOTO 15         ; CHECK IF OPEN SUCCESSFUL
C***
      TYPE ' OPEN ERROR =',FILER      ; NO--TERMINATE PROCESSING
      STOP TRANSACTION FILE ERROR
C***
15    CONTINUE
      CALL TIME (ITIME,TIMER)         ; GET THE TIME OF DAY
      CALL DATE (IDATE,TIMER)         ; GET TODAY'S DATE
```

```
C***
C***
C*************************************************************************
C***
C     GET CUSTOMER NAME RECORD FROM TRANSACTION FILE
C***
C*************************************************************************
C***
      NRCRD=0
      NTRAN=0
C***
25    CALL RDRW (ITRAN,NRCRD,INRCD,1,LFLER)
C***
      IF (LFLER.EQ.1) GOTO 30          ; RECORD READ OKAY ?
C***
      IF (LFLER.EQ.9) GOTO 1000        ; NO--END OF FILE !!
C***
      TYPE ' TRANSACTION FILE ERROR =',LFLER ; ERROR--PROCESS IT
      GOTO 9000
C***
30    IF (INRCD(16).NE.0) GOTO 100     ; MUST BE CUSTOMER ITEM RECORD
C***
      IF (NCOMP(INRCD,1,6,IEOF,1).NE.0) GOTO 31 ; END OF DATA RECORD
C***
      NTRAN=-1                         ; INDICATE END OF DATA
C***
31    IF (NTRAN.EQ.0) GOTO 34
C***
C     PUT OUT NEW BALANCE ON INVOICE AND UPDATE FILE
C***
      CALL FILL (IPRNT,1,80,'  ')
C***
      CALL UNPAC (NWBAL,1,6,IPRNT,23)
C***
      CALL GMOVE (MASK,1,14,IPRNT,47)
C***
      CALL DECA1 (ISUM,1,10,CSPER)
C***
      CALL EDIT (ISUM,1,10,IPRNT,47,60)
C***
      WRITE (INVCE,33) IPRNT
33    FORMAT (/1X,80A1//)
C***
      CALL PACK (ISUM,1,10,IHEAD,31)
C***
      CALL WRTRW (MASTR,INAME,IHEAD,1,FILER)
C***
      IF (NTRAN.EQ.-1) GOTO 200
```

```
C***
C*************************************************************************
C***
C     LOOK UP CUSTOMER NAME IN MASTER FILE 'MASTERFILE'
C***
C*************************************************************************
C***
34    INAME=0
35    CALL RDRW (MASTR,INAME,IHEAD,1,FILER)
C***
      IF (FILER.EQ.1) GOTO 40          ; CHECK RECORD READ OKAY
      TYPE ' MASTER FILE READ ERROR =',FILER
      GOTO 9000
C***
40    IF (NCOMP(IHEAD,1,10,INRCD,1).EQ.0) GOTO 50 ;LEGAL CUSTOMER ?
C***
      IF (NCOMP(IHEAD,1,6,IEOF,1).NE.0) GOTO 45  ; NO-END OF FILE?
C***
      TYPE 'NO CUSTOMER NAME MATCH '
      GOTO 9000
C***
C     NO MATCH FOUND YET - SO KEEP LOOKING THROUGH THE FILE
C***
45    INAME=INAME+1
      GOTO 35
```

```
C***
C*************************************************************************
C***
C     LEGAL CUSTOMER NAME FOUND IN TRANSACTION FILE
C***
C*************************************************************************
C***
50    CONTINUE
      WRITE (INVCE,55) (IHEAD(I),I=1,30)
55    FORMAT (//2X,10A2/3X,10A2/4X,10A2//)
C***
      WRITE (INVCE,56)
56    FORMAT (T17,'QTY',T30,'NAME',T55'AMT'/)
C***
C     UNPACK THE PREVIOUS BALANCE FROM A2 FORMAT TO DECIMAL
C***
      CALL UNPAC (IHEAD,31,35,ISUM,1)
C***
C     FILL THE PRINT AREA WITH SPACES
C***
      CALL FILL (IPRNT,1,80,' ')
C***
C     MOVE 'PREVIOUS BALANCE ' PACKED IN A2 FORMAT
C     TO THE PRINT AREA BUFFER 'IPRNT'
C***
      CALL UNPAC (IPRVB,1,8,IPRNT,23)
C***
C     MOVE EDIT MASK TO PRINT AREA
C***
      CALL QMOVE (MASK,1,14,IPRNT,47)
C***
C     EDIT IN THE PREVIOUS BALANCE VALUE
C***
      CALL EDIT (ISUM,1,10,IPRNT,47,60)
C***
C     PRINT THE BUFFER FULL TO INVOICE OUTPUT DEVICE
C***
      WRITE (INVCE,70) IPRNT
70    FORMAT (1X,80A1)
C***
      CALL A1DEC (ISUM,1,10,CSPER)
C***
      NRCRD=NRCRD+1
      GOTO 25
```

```
C***
C***
C*************************************************************************
C***
C      CUSTOMER ITEM RECORD
C***
C*************************************************************************
C***
100    CONTINUE
       CALL FILL (IPRNT,1,80,' ')
C***
       CALL UNPAC (INRCD,1,16,IDATA,1)
C***
       CALL QMOVE (IDATA,1,20,IPRNT,23)
C***
       CALL QMOVE (MASK,1,14,IPRNT,47)
C***
       CALL EDIT (IDATA,21,28,IPRNT,47,60)
C***
       CALL A1DEC (IDATA,21,28,CSPER)
C***
       CALL QADD (IDATA,21,28,ISUM,1,10,CSPER)
C***
       CALL QMOVE (MASK,1,4,IPRNT,13)
       CALL QMOVE (MASK,1,3,IPRNT,17)
C***
       CALL EDIT (IDATA,29,32,IPRNT,13,18)
C***
       WRITE (INVCE,70) IPRNT
C***
       NRCRD=NRCRD+1
       NTRAN=NTRAN+1
       GOTO 25
C***
C***
C*************************************************************************
C***
C      END OF THE DAILY FILE UPDATE PROGRAM
C***
C*************************************************************************
C***
200    CONTINUE
       TYPE ' END OF DAILY FILE'
       TYPE ' '
       GOTO 9000
C***
C*************************************************************************
C***
C      ERROR WHILE PROCESSING THE DAILY OR MASTERFILE
C***
C*************************************************************************
C***
1000   TYPE ' END OF FILE'
       TYPE ' '
9000   STOP INVOICE PROGRAM EXECUTION COMPLETED
       END
```

PROGRAM #5 - CONSOLE LOG

```
R
CSP5

 *** INVOICE PROGRAM **

 END OF DAILY FILE
$TOP INVOICE PROGRAM
R
```

PROGRAM #5 - OUTPUT REPORT

RUNNING MOTORS
10 WATER STREET
PLYMOUTH, ROCK 02296

QTY	NAME	AMT
	PREVIOUS BALANCE	$268.68
0020	AIR CLEANERS - CASES	$200.00
0006	GREASE - BARRELS	$165.26
0050	TIRES - 850 X 15	$1,012.00
0002	TIRES - SPARE	$123.46
0012	GASOLINE CAPS	$12.12
0001	SCRAPPERS	$.22
	NEW BALANCE	$1,773.74

VIOLET A. SAWYER
93 EAST MAIN ST.
HUDSON, MASS 01749

QTY	NAME	AMT
	PREVIOUS BALANCE	$48.24
0004	TOMATO SOUP - CANS	$.92
0012	CORNED BEEF - CANS	$33.72
0004	ROAST BEEF - POUND	$9.72
0002	GINGER ALE - CASES	$1.76
	NEW BALANCE	$94.36

DAVES MARKET
1997 WASHINGTON ST.
NEWTOWN, MASS 02158

QTY	NAME	AMT
	PREVIOUS BALANCE	$78.19
0025	POTATOES - BAGS	$27.50
0100	TOMATOES - LOOSE	$24.00
0050	CARROTS - BUNCHES	$11.00
0100	BREAD - LOAF	$45.00
0010	MILK - QUARTS	$4.50
0046	MILK - HALF GALS	$32.00
0012	HAM (RETURNED)	$36.72CR
	NEW BALANCE	$185.47

APPENDIX C

STATEMENT FORMAT REFERENCE TABLE

STATEMENT FORMAT	DATA FORMAT BEFORE	AFTER
CALL A1A3(ICON,IFRST,ILST,MCVTD,MFRST,NTBL)	A1	A3
CALL A1DEC(ICON,IFRST,ILST,NINV)	A1	D1
CALL A1DW(ICON,IFRST,ILST,DWORD,NCHK)	A1	DWD
CALL A3A1(ICON,IFRST,ILST,MCVTD,MFRST,NTBL)	A3	A1
CALL DECA1(ICON,IFRST,ILST,NINV)	D1	A1
CALL DPACK(ICON,IFRST,ILST,MCVTD,MFRST)	D1	D4
CALL DUNPK(ICON,IFRST,ILST,MCVTD,MFRST)	D4	D1
CALL DWADD(DWORD1,DWORD2,NCHK)	DWD	DWD
CALL DWA1(DWORD1,MCVTD,MFRST,MLST)	DWD	A1
DWCMP(DWORD1,DWORD2)	DWD	DWD
CALL DWDIV(DWORD1,DWORD2,DWORD3,NCHK)	DWD	DWD
CALL DWFL(DWORD,DPVAL)	DWD	DFL
CALL DWMPY(DWORD1,DWORD2,NCHK)	DWD	DWD
CALL DWSUB(DWORD1,DWORD2,NCHK)	DWD	DWD
CALL EDIT(ICON,IFRST,ILST,MASK,MFRST,MLST)	A1	A1
CALL FILL(ICON,IFRST,ILST,NFILL)	ANY	ANY
CALL FLDW(DPVAR,DWORD)	DFL	DWD
GET(ICON,IFRST,ILST,DECP)	A1	DFL
ICOMP(IONE,IFRST,ILST,MTWO,MFRST,MLST)	D1	D1
CALL KEYBD(ICON,IFRST,ILST)	ANY	A1
NCOMP(IONE,IFRST,ILST,MTWO,MFRST)	A1	A1
CALL NSIGN(IONE,IFRST,NCODE,NRES)	D1	D1
CALL NZONE(IONE,IFRST,NZN,NRES)	A1	A1
CALL PACK(ICON,IFRST,ILST,MCVTD,MFRST)	A1	A2
CALL PRINT(ICON,IFRST,ILST,NCHK)	A1	A1
CALL PUT(ICON,IFRST,ILST,DPVAL,HADJ,TRUNC)	DFL	A1
CALL QADD(IADD,IFRST,ILST,MSUM,MFRST,MLST,NOFL)	D1	D1
CALL QDIV(IDIVR,IFRST,ILST,MDIVD,MFRST,MLST,NZER)	D1	D1
CALL QMOVE(IONE,IFRST,ILST,MTWO,MFRST)	ANY	ANY
CALL QMPY(IMPYR,IFRST,ILST,MPRDT,MFRST,MLST,NCHK)	D1	D1
CALL QREAD(ICON,IFRST,ILST,NCHK)	ANY	ANY
CALL QSUB(ISUB,IFRST,ILST,MDIF,MFRST,MLST,NCHK)	D1	D1
CALL RJUST(ICON,IFRST,ILST)	A1	A1
CALL TYPER(ICON,IFRST,ILST)	A1	A1
CALL UNPAC(ICON,IFRST,ILST,MCVTD,MFRST)	A2	A1
WHOLE(WHOLE,DOUBLE)	DFL	DFL

DFL = DOUBLE PRECISION FLOATING POINT
DWD = DOUBLE WORD SIGNED INTEGER