DECUS
PROGRAM LIBRARY

DECUS NO.	8-199
TITLE	ACESSING DATA ARRAYS AND TELETYPE TEXT INPUT/OUTPUT
AUTHOR	David G. Frutchey
COMPANY	Beckman Instruments, Inc. Fullerton, California
DATE	May 2, 1969
SOURCE LANGUAGE	PAL III

ACESSING DATA ARAYS AND TELETYPE TEXT INPUT/OUTPUT

DECUS Program Library Write-up

DECUS No. 8-199

ABSTRACT

Two subroutine packages are presented which, when used with any of the standard Floating Point Packages, provide the user with a powerful, yet concise programming methodology. Data storage or retrieval with floating to fixed, or fixed to floating point conversion can be performed on terms analogous to single variable, subscripted FORTRAN array terms such as "ARRAY (a*J+b)." A concise facility for text output (6? characters), character input, line spacing and tabulating is also demonstrated. These subroutine packages were written for the PDP-8 family of computers and were intended for use with the PAL-III language.

EFFICIENT STORAGE UTILIZATION

The programming of a digital computer requires a knowledge of both procedural and arithmetic techniques. Higher level languages such as FORTRAN and FOCAL are recognized as being a significant aid to the programmer particularly for procedural operations and through the relative ease in which the language can be learned. Also, the completeness of these languages can be illustrated by the simple, direct way in which the procedural tasks of input/output, branching logic, data conversion (integer storage/floating point storage), looping to repeat procedures, etc. can be done. Among these procedural tasks is the ability to process single-subscripted-variable data array terms of the form ARRAY (a*J+b). The terminology used here is as defined in the American Standard FORTRAN in that the symbol, ARRAY, is a name given the beginning of successively stored data values and the subscript enclosed in parentheses with integer values of 1, 2, 3, ... refers, respectively, to the first, second, third, ... data array number.

Analogous procedural programming requires more effort in the assembly language of the computer than of the higher level languages available. However, quite often, reverting to the use of assembly language can permit significant gains in the critical availability of computer core storage. In a small machine, such as the PDP-8 family of computers, core storage is extremely limited and the desire to achieve improved core storage utilization is often an application program necessity.

What are some of the programming concepts which adversely affect core utilization? The following are a few:

1. Direct storage of higher level language in a form much more lengthy than the assembly language equivalent code.

2. Use of an interpretive system for all coding which in equivalent assembly form would not always require the use of an interpreter program.

3. Lengthy preparations for calling subroutines or lengthy calling sequence requirements to subroutines.

4. Duplication of coding in that an internally available routine is not used.

5. Failure to share work region requirements; that is, core storage which is temporarily used and otherwise available.

6. Overemphasis of the coding consideration of the seldom used (if ever) requirements, rather than permitting omissions with an attendant savings in core; or by providing alternately coded packages so that the user may have a choice.

The appearance of programming systems which attempt to preserve the procedural niceties of the higher level languages are prevalent today; however, many of these systems suffer from some of these same adverse conditions and produce little improvement in core utilization. The adverse concepts listed above may serve as a guide when evaluating both the system presented herein and other such systems.

TWO NEW PACKAGES

Two subroutine packages were developed during a biomedical project in which minimizing the core storage requirements was essential. To this end, the packages entitled "Teletype Text Input/Output Package" and the "Array Accessing Subroutine Package" were written. The usage and features of these routines enable a user to code several procedural tasks such as teletype page formatting, output and input, and processing data arrays. Also illustrated in the examples given are means by which other procedural tasks such as looping and decision branching could be coded. These two packages require the use of the Floating Point Package[1] and are partially integrated in their use and performance. The overall purpose in developing these packages was to ENABLE MINIMUM CORE STORAGE REQUIREMENTS. At the same time, the peripheral benefits of being able to think or plan a program in FORTRAN are not altogether lost. In fact, the examples show FORTRAN equivalents which are a suggestion that FORTRAN coding could readily precede the use of

these routines. A second peripheral benefit is in the increase obtained in computational computer speed by these routines when compared to the present FORTRAN-produced code.

THE TELETYPE TEXT I/O PACKAGE

This package consists of three routines; namely, subroutines TAB, TEXT and ECHO. The subroutines are "called" in the PAL-III assembly language through the use of the Jump to Subroutine (JMS) mnemonic. As in the Floating Point Package, a TLS must be used at least once prior to using routines TAB and TEXT.

The Teletype Text I/O Package is available in two forms which are a Stand-Alone version and a version to be used when the Floating Point Package is resident which will be referred to as the FPP version. The Stand-Alone version occupies 63 decimal storage locations. The FPP version requires 55 decimal storage locations. The FPP version has the distinct advantage of using the Floating Point Package for all teletype commands. This feature is useful in that if interrupt I/O is implemented, modification to the Floating Point Package only need be made.

Subroutine TAB

It performs tabulation across a page by typing blanks, or performs multiple line spacing down a page by typing carriage return (CR), line feed (LF) combinations. The routine reacts to the accumulator c(Ac) contents as follows:

If c(Ac)	Output
=0	One CR and one LF
=n positive,	One CR and n+1 LF
=n negative,	|n|-1 blanks

No calling sequence parameters are expected by subroutine TAB other than the c(Ac), and the routine always returns with c(Ac)=0.

Subroutine TEXT

This subroutine performs the output of a message stored as six-bit (stripped) code with two characters per computer word. The message is formed by subtracting octal 240 from any USASC-II eight-bit code in the octal range 240 to 336, inclusively. The two-digit, octal results representing a character are placed into the first half, followed by a character in the last half of each computer word and continued into successive core storage locations. The two-digit, octal code 77 is used as the message termination signal and must appear in either the first or last half of the final computer word used for the messages. Messages may be as long as desired and may overlap page boundaries. The code includes the use of the alphabet, numerals, punctuation, etc. The routine is called without regard to the c(Ac) and requires the address of the first word of the message to follow the call. This feature was provided rather than following the call with the message itself to enable portions of a message to be reusable and to permit the placement of the message into core locations which would not otherwise be used. The routine differs from previously written routines in that the link register (a one-bit, two-state register) is used to alternately point to the character to be printed. Also, the switching of the link pointer is performed without command through the arithmetic in testing for the terminating character.

Example 1 - Using Subroutine TEXT to type "NO GO"

```
      :
JMS TEXT            /JUMP TO SUBROUTINE TEXT
MSGADR
            ← PROCESSING CONTINUES HERE
      :
MSGADR, 5657        /NO
0047                /-G
5777                /0 TERMINATION
```

Subroutine TEXT always returns with the c(Ac)=0.

Subroutine ECHO

This routine performs the input and typing of a single USASC-II, eight-bit code character. When called, the routine will "wait" until a keyboard entry is made and will return to the location immediately following with c(Ac)=0 if the RUB-OUT key is entered. If any other key is entered, the routine will return to resume past the RUB-OUT return point and will contain the USASC-II character in the c(Ac). Additionally, the character will be stored at location CHAR in the listing of the subroutine ECHO used. When the Floating Point Package is to be used, the character is returned at octal location CHAR=0057. The Stand-Alone version of ECHO deposits the character at location CHAR=TEXT and it will be destroyed upon any subsequent call to either TAB, TEXT or ECHO.

Example 2 - Using subroutine ECHO to input a character.

```
      :
AGAIN, JMS ECHO
JMP AGAIN           /WHEN RUB-OUT IS ENTERED
            ← PROCESSING CONTINUES HERE
              WHEN THE CHARACTER IS NOT
              THE RUB-OUT KEY.
```

THE ARRAY ACCESSING SUBROUTINE PACKAGE (AASP)

This package provides subroutines which can be used to access single subscripted variable arrays of the form ARRAY (a*J+b). The use of the Floating Point Package is required in that several of the Floating Point Package internal routines and services are used. Also, work regions of the Floating Point Package are shared.

The Array Accessing Subroutine Package requires 110 (decimal) storage locations and is designed to reside beginning at core location octal 4400. When a Floating Point Package version is used which does not include the subroutine FIX at location 4557, the Array Accessing Subroutine Package will supply subroutine FIX. In this event, all of the locations on the page between octal 4400-4577 are required. Ten of the locations required reside at the end of two pages within the Floating Point Package (octal 6173-6177, and 6573-6577).

The routines to be used are named as follows and will perform the attendant procedure:

Name	Procedure
LOAD	Clear the floating accumulator c(FAC) and load from a floating point array.
ILOAD	Clear c(FAC) and load from an integer data array.
STORE	Store c(FAC) into a floating point data array.
ISTORE	Store c(FAC) into an integer data array.
FIND	Without disrupting c(FAC), load a temporary accumulator defined as ANS at location 52, 53 and 54 from a floating point data array.
IFIND	Without disrupting c(FAC) load c(ANS) from an integer data array.

The routines are entered while operating within the Floating Interpreter via the table provided at octal locations 6556 through 6563. The proper contents of these locations will be furnished in the Array Accessing Subroutine Package. A user must define the table entry naming convention by providing the following octal equates to the Assembler:

```
LOAD   = 0012
ILOAD  = 0013
STORE  = 0014
ISTORE = 0015
FIND   = 0016
IFIND  = 0017
```

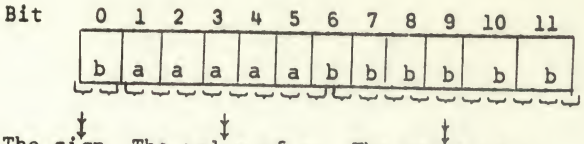
The FIXTAB pseudo-op may be used for this purpose if desired. It may be well to note that these routines may not be called while not within the floating interpreter. Notice that the use of the ILOAD, ISTORE initiates both data array accessing and c(FAC) conversion. The routines can certainly be used for the simpler usage where conversion only is desired. The resulting value will be found in the c(FAC) after either the ILOAD (load & fix/float) or the ISTORE (float/fixed & store).

Each routine requires either a two or three-word calling sequence as follows:

```
ENTRY NAME                The routine to be entered.
ARGUMENT 1                The address of the array.
ARGUMENT 2 (OPTIONAL)     The values of a and b.
```

The ENTRY NAME is coded by selecting the desired array operation, namely, LOAD, ILOAD, etc. Argument one is the array starting address which can be supplied by writing the same name used to identify the beginning of the data array in the program. If the optional argument two will follow argument one, bit zero of argument one must be turned on. This can be accomplished in a variety of ways, but perhaps the most direct way is to add octal 4000 to the array name coded as argument one.

The second argument is optional and should be coded only when the first argument has bit zero set to a one. This argument two contains the subscript values for a and b in the subscript (a*J+b). If the second argument is not supplied, these values are taken as a=1 and b=0. Otherwise both of these values are supplied in argument two as shown in the following 12-bit computer word:



The sign of b. The value of a. The value of b.

If b is negative, b must be supplied as a twos complement number.

Lastly, the variable parameter, J, of the array subscript is provided as an integer value by the user on page zero at core location 0050. This location is internal to the Floating Point Package requirements and, hence, does not constitute an additional core storage requirement. The value of J should be set to 1, 2, 3, ... etc. to access the first, second, third, ... etc. data array number regardless of whether the data is stored as floating point or fixed point data. Additionally, J may be given the value of zero as a special case in which the first number of either a floating or fixed data array will be accessed.

Arrays may consist of either integer values stored successively in core storage or as three-word groupings of floating point numbers. In either case, the first computer word of an array should be given a name in a PAL-III program to be used as the array starting address. Arrays may be of any desired length and may overlap page boundaries in any way desired. Also, arrays may consist of only one value which is not really an array; however, the ability to access any symbolic named integer in a program is useful in that the value can be loaded with conversion to floating point for computational use. The load with conversion characteristic is also useful in that arrays which otherwise might have been maintained as floating point arrays can, in some cases, be stored as integer arrays with a significant savings in core space.

The following examples will illustrate the usage of the Array Accessing Subroutine Package:

Example 3 - Move array named HERE to occupy the array named THERE. Both arrays are 10 words long.

```
     :
CLA
TAD NEGTEN     /PREPARE FOR LOOP
DCA CTR
DCA J          /ZERO J
LOOP, ISZ J    /INCREMENT J
FINT           /ENTER INTERPRETER
LOAD           /CLEAR AND LOAD C(FAC)
HERE           /FROM HERE (J)
STORE          /STORE C(FAC)
THERE          /INTO THERE (J)
FEXT           /EXIT INTERPRETER
ISZ CTR
JMP LOOP
     :
/DATA SECTION
J=50
ANS=52
FINT=JMS I 7   /TO ENTER THE INTERPRETER
NEGTEN,-12
CTR,0
```

```
HERE=.          /DIMENSION HERE(10),THERE(10)
*.+36           /THAT'S OCTAL LENGTH OF 10 FLOATING
                /POINT NUMBERS

THERE=.
*.+36
```

In the preceding example, the coding of a loop is seen to be a direct feature of the PAL-III language Also, the equivalent of a FORTRAN DIMENSION statement is shown. The coding of the equivalents to a FORTRAN 'DO' loop and a FORTRAN 'IF' statement is shown in the following example, in addition to further use of the Array Accessing Routines, LOAD and FIND.

Example 4 - Write the following FORTRAN coding in PAL-III with use of the Array Accessing Subroutine Package (AASP). Notice that the use of mixed mode computation is indicated.

FORTRAN

```
DO 180 J=1,4
IF(DATA(2*J+3)-IVAR(J)) 155,160,160
      .
      .
155   .
160   .
180 CONTINUE
      .
      .
```

Using AASP

```
CLA
TAD NEG4          /THAT'S -4
DCA CTR
DCA J             /ZERO J
DO180,ISZ J       /BEGINS DO LOOP, DO 180
FINT              /FLOATING INTERPRETER
LOAD
DATA+4000
0203              /LOADS DATA (2*J+3)
IFIND             /NOTICE MIXED MODE IS SUPPORTED
IVAR              /C(ANS) HAS IVAR(J) NOW CONVERTED
FSUB ANS          /SUBTRACT
FEXT              /EXIT INTERPRETER
TAD 45            /GET WORD WITH SIGN FOR 'IF'
SMA CLA           /SKIP 'IF' MINUS
JMP S160
S155,   .
        .
S160,   .
        .
S180, ISZ CTR     /180 CONTINUE
JMP DO180
        .
        .
/DATA SECTION
DATA=.            /DIMENSION DATA(4),IVAR(4)
*.+14             /FOUR FLOATING POINT NUMBERS
IVAR=.
*.+4              /FOUR INTEGER NUMBERS
```

Limitations of AASP

The Array Accessing Subroutine Package has several deliberate limitations; however, the major objective in permitting savings in core storage is the underlying justification for these limitations. The limits are as follows:

Consider the term ARRAY (a*J+b)

1. Range of a is given by $0 \leq a \leq 31$
2. Range of b is given by $-64 \leq b \leq 63$
3. Range of J is given by $-2048 \leq J \leq 2047$ (Range of 4K machine)
4. The starting address of any array to be accessed must be in the first 2K of storage (0000-3777 octal).
5. The temporary accumulator, ANS, located at c(0052, 0053, 0054) will be destroyed during the use of the square root, square, or an I/O controller INPUT or OUTPUT. Note that in the case of an OUTPUT operation, the c(FAC) is destroyed also.

Numeric Input and Output

The Array Accessing Subroutine Package provides an additional service to augment the use of the standard Floating Point Package numeric input and output routines. The subroutine entry table is supplied with input and output transfer addresses at octal locations 6554 and 6555. Hence, I/O can be performed while within the interpreter if the octal equates INPUT=0010 and OUTPUT=0011 are supplied to the Assembler. Numeric input in integer, fixed point or floating point notation may be encoded by simply writing INPUT while within the interpreter. Similarly, output of the c(FAC) may be initiated by encoding OUTPUT. Use of the output controller is expected when using this OUTPUT feature; however, if desired, the omission of the output controller must be indicated by placing octal 7200 into location 6555 by the user. When the output controller is used, the octal storage locations at c(62) and c(63) control the output format where c(62) should be given the field width, c(63) the decimal field width. These locations must be maintained by the user and are not initialized. If the output controller is not used, the locations are not required. A routine is provided to place the decimal field width into the accumulator as required when using the Floating Point Package output controller.

OTHER TECHNIQUES

Relational Operators

It might be valuable to point out that the Relational Operators usable in 'IF' statements of recent FORTRAN versions can be equivalently coded in PAL-III through the use of the following convenient mnemonics:

MNEMONIC		SKIP NEXT INSTRUCTION IF C(AC) ...
SGT=7550	/SPA SNA	SKIP IF .GT. Ø
SGE=7510	/SPA	SKIP IF .GE. Ø
SEQ=7440	/SZA	SKIP IF .EQ. Ø
SNE=7450	/SNA	SKIP IF .NE. Ø
SLE=7540	/SMA SZA	SKIP IF .LE. Ø
SLT=7500	/SMA	SKIP IF .LT. Ø

These mnemonics have been found to be extremely beneficial to the FORTRAN-oriented mind and have been made a permanent part of our PAL-III Assembler.

Coding Across Page Boundaries

Another procedural benefit in using FORTRAN or FOCAL is the absence of 'page' considerations in the PDP-8 family of computers. A technique can be employed in the PAL-III language which can greatly reduce the problems associated with pages.

Coding may begin at a convenient location (say octal 0200) and continue straight through crossing page boundaries.

Adopt the following coding conventions to obtain this end:

1. Use page zero for integer constants, a small work area (loop control, etc.) and for indirectly addressing absolute references to floating point values or subroutines.

2. Use the Array Accessing Subroutine Package for array processing.

3. Use the Teletype Text I/O package for referencing messages.

4. Code all jump (JMP) instructions without using indirect addressing.

Upon assembling a program written by these conventions, the Assembler will report which jump instructions must be changed to indirect addresses. Changing the jump instructions and adding the indirect addresses to page zero will produce a program which will execute across page boundaries.

SUMMARY

When combined with the Floating Point Package, the Teletype Text I/O Package and the Array Accessing Subroutine Package provide a PAL-III programming methodology. The discussion and several examples presented show the coding of the following techniques:

1. Multiple tabulating and line spacing.

2. Multiple-word, message output stored as two characters per word.

3. USASC-II, eight-bit code character input with RUB-OUT key recognition.

4. Numeric input and output in any of the formats: integer, fixed or floating point.

5. Accessing single variable, subscripted array terms of the general form ARRAY (a*J+b) to perform loading, storing, locating and, inherently, data conversion and in supporting mixed mode data calculation.

6. The directly available features of PAL-III in coding equivalents to FORTRAN 'DO' loops, 'IF' statements with either arithmetic or relational arguments, DIMENSION statements, and coding across page boundaries.

The emphasis in developing and presenting these techniques has been to utilize a minimum of core storage for each procedural task presented. These routines have been intensively used and appear to be completely reliable.

A brief testimonial is contained in my initial usage of these routines. A biomedical computer application,[2] which was only partly written in FORTRAN, occupied 96 per cent of available core storage. Text information was stored as two characters per word and routines not required within the FORTRAN Controller were overlaid when the program was in working order. The need for additional computational application features prompted the rewrite of this program to eliminate the use of the FORTRAN controller and the remaining FORTRAN code. The resulting program (which also worked) not only occupied less core space, now requiring only 63 per cent of core storage, but also executed much faster. The former FORTRAN program version required 12 seconds per compute cycle to initiate plotter movements while the program version employing the techniques of this paper required less than three seconds per compute cycle to initiate plotter movements.

REFERENCES

1. Floating-Point System Programming Manual, 8-5-S, Digital Equipment Corporation, Maynard, Massachusetts, 1965.

2. Savaglio, Fred J., Pacela, Allan F., "Monitoring and Display of Continuous Real-Time Blood Chemistry Data", Proceedings of the 21st Annual Conference on Engineering in Medicine and Biology, November 1968.

```
/
/  TTY TEXT I/O SUBROUTINE PACKAGE, STAND-ALONE VERSION
*5400     / PLACE WHERE WANTED
/     THIS PACKAGE PERFORMS INPUT AND OUTPUT (I/O)
/
/     PREPARED BY D G FRUTCHEY, BECKMAN INST.S, JUNE, 1968
/
/  SUBROUTINE TAB
/  THIS ROUTINE OUTPUTS BLANKS, CARRIAGE RETURNS (CR),
/  AND LINE FEEDS (LF) ACCORDING TO THE ACCUMULATOR CONTENTS.
/     IF C(AC) = POSITIVE INTEGER, N, GET CR & N+1 LF'S
/     IF C(AC) = ZERO, GET CRLF
/     IF C(AC) = -1, GET A CR
/     IF C(AC) = NEGATIVE INTEGER, N, GET ABS(N)-1 BLANKS
TAB,NOP
STL IAC
SMA
CMA CLL     / C(L) = 0 MARKS CRLFS
DCA TEMP
SNL
TAD POS3
TAD BLANK
SNL
TAD DIFF
JMS TYPE
ISZ TEMP
JMP .-5
JMP I TAB
POS3,3
DIFF,3752
BLANK,0240
TEMP=TEXT     / TEMPORARY STORAGE
/
/       SUBROUTINE TEXT
/     OUTPUT STRIPPED CODE BEGINNING AT ADDRESS PASSED.
/     EXAMPLE OF USAGE:
/     JMS TEXT          /TO CALL THIS ROUTINE
/     MSG1              /THE SYMBOL USED TO BEGIN THE MESSAGE
TEXT,NOP
CLA CLL     / INSURE THAT LINK IS CLEAR
TAD I TEXT
ISZ TEXT
DCA TEMP2
GET1,TAD I TEMP2
SZL
JMP .+5
RTR      /FIRST TIME THROUGH
RTR
RTR
CLL
AND MSK
TAD M77
SNA
JMP I TEXT     / DONE, RETURN
TAD MSK /RESTORE AC AND COMP LINK
TAD BLANK
JMS TYPE
SNL
ISZ TEMP2     / SECOND TIME THROUGH
JMP GET1
MSK,0077
M77,7701
TEMP2=TAB     / TEMPORARY STORAGE
/
/       SUBROUTINE ECHO
/       USAGE :
/       JMS ECHO          /TO GET ASCII CHAR INTO C(AC) & TYPED
/       JMP AGAIN         /IF RUBOUT WAS INPUT
ECHO,NOP
KSF      / SKIP ON INPUT
JMP .-1 / WAIT
KRB      / CLEAR C(AC), READ BUFFER, CLEAR FLAG
DCA CHAR
TAD CHAR
SNA
JMP ECHO+1    / ALSO WAIT WHEN NULL OR BLANK TAPE .
JMS TYPE     / ECHO THE CHARACTER
TAD CHAR
TAD MRBOUT    / HAVE RUB-OUT?
SNA CLA
JMP I ECHO    / RUB-OUT WAS ENTERED
ISZ ECHO      / INCREMENT TO BYPASS
TAD CHAR
JMP I ECHO
MRBOUT,-377
CHAR=TEXT
/
/  SUBROUTINE TYPE
TYPE,NOP
TSF
JMP .-1
TLS
CLA
JMP I TYPE
$
```

```
/  TTY TEXT I/O SUBROUTINE PACKAGE, FPP VERSION
*5400     / PLACE WHERE WANTED
/     THIS PACKAGE PERFORMS INPUT AND OUTPUT (I/O)
/     BY USING THE ROUTINES FURNISHED IN THE
/     FLOATING POINT PACKAGE (INPUT & OUT) TO
/     PERFORM SINGLE CHARACTER TRANSFER.
/
/     PREPARED BY D G FRUTCHEY, BECKMAN INST.S, JUNE, 1968
/
/  SUBROUTINE TAB
/  THIS ROUTINE OUTPUTS BLANKS, CARRIAGE RETURNS (CR),
/  AND LINE FEEDS (LF) ACCORDING TO THE ACCUMULATOR CONTENTS.
/     IF C(AC) = POSITIVE INTEGER, N, GET CR & N+I LF'S
/     IF C(AC) = ZERO, GET CRLF
/     IF C(AC) = -1, GET A CR
/     IF C(AC) = NEGATIVE INTEGER, N, GET ABS(N)-I BLANKS
TAB,NOP
STL IAC
SMA
CMA CLL     / C(L) = 0 MARKS CRLFS
DCA TEMP
SNL
TAD POS3
TAD BLANK
SNL
TAD DIFF
JMS I TYPE
ISZ TEMP
JMP .-5
JMP I TAB
POS3,3
DIFF,3752
BLANK,0240
TEMP=TEXT     / TEMPORARY STORAGE
/
/       SUBROUTINE TEXT
/     OUTPUT STRIPPED CODE BEGINNING AT ADDRESS PASSED.
/     EXAMPLE OF USAGE:
/     JMS TEXT          /TO CALL THIS ROUTINE
/     MSG1              /THE SYMBOL USED TO BEGIN THE MESSAGE
TEXT,NOP
CLA CLL     / INSURE THAT LINK IS CLEAR
TAD I TEXT
ISZ TEXT
DCA TEMP2
GET1,TAD I TEMP2
SZL
JMP .+5
RTR      /FIRST TIME THROUGH
RTR
RTR
CLL
AND MSK
TAD M77
SNA
JMP I TEXT     / DONE,RETURN
TAD MSK /RESTORE AC AND COMP LINK
TAD BLANK
JMS I TYPE
SNL
ISZ TEMP2     / SECOND TIME THROUGH
JMP GET1
MSK,0077
M77,7701
TEMP2=TAB     / TEMPORARY STORAGE
/
/       SUBROUTINE ECHO
/       USAGE :
/       JMS ECHO          /TO GET ASCII CHAR INTO C(AC) & TYPED
/       JMP AGAIN         /IF RUBOUT WAS INPUT
ECHO,NOP
CLA CLL
TAD ECHO
DCA I RESTRT
JMS I INPUT
CLA CLL
TAD INPKG
DCA I RESTRT
ISZ ECHO          /TO RETURN PAST THE RUBOUT JMP
TAD CHAR          /RETURNS WITH CHAR IN C(AC) & AT LOC 0057
JMP I ECHO
RESTRT,7167
INPUT,7142        /FPP ROUTINE INPUT
INPKG,7401
TYPE,7344     /FPP ROUTINE OUT ADDRESS
CHAR=0057
$
```

```
/
/
/       ARRAY ACCESSING SUBROUTINE PACKAGE
/       LOAD AFTER LOADING THE FLOATING POINT PACKAGE.
/       THESE ROUTINES ARE USED TO ACCESS ARRAYS OF THE
/       FORM, ARRAY(A*J+B) AND TO PROVIDE FIXED/FLOAT
/       AND FLOAT/FIXED DATA CONVERSION.
/   THIS PACKAGE WAS PREPARED BY D G FRUTCHEY,
/     BECKMAN INSTRUMENTS, FULLERTON, CALIF, OCTOBER, 1968
*6554
FINT=4407       /JMS I 7
7400    / INPUT = 10
DECOUT    / OUTPUT = 11
LOAD      / LOAD = 12
ILOAD     / ILOAD = 13
STORE     / STORE = 14
ISTORE    / ISTORE = 15
FIND      / FIND = 16
IFIND     / IFIND = 17
*6573
DECOUT,NOP
DEC=63
TAD DEC /LOAD NR OF DEC PLACES
JMS I FPPOUT
JMP I DECOUT
FPPOUT,7200
*6173
/
/  SUBROUTINE GET
GET,NOP    / RESTORE THE C(FAC) FROM ANS
FINT
FGET ANS
FEXT
JMP I GET
/
/  EQUATE SYMBOLS
A=40
B=41
SEE=42
J=50    /USER SETS J HERE
ADR=51
ANS=52    / TEMPORARY ACCUMULATOR, ANS
/
/  SUBROUTINE LOAD
*4400
LOAD,NOP    / LOAD C(FAC) WITH FLOATING POINT NR
JMS FIND
JMS I AGET
JMP I LOAD
/
/  SUBROUTINE ILOAD
ILOAD,NOP    / LOAD C(FAC) WITH FIXED POINT NR
LFIND=IFIND
JMS LFIND
JMS I AGET
JMP I ILOAD
/
/  SUBROUTINE STORE
STORE,NOP
JMS FORM
FINT
FPUT I ADR
FEXT
JMP I STORE
/
/  SUBROUTINE ISTORE
ISTORE,NOP    / CONVERT C(FAC) TO FIXED & STORE
JMS POINT
TAD ADR
DCA ADR    / POINTS TO PLACE TO STORE
JMS FIX
TAD 45
DCA I ADR
TAD P13
DCA 44
DCA 46
JMS I NORM
JMP I ISTORE
/
/  SUBROUTINE FIND
FIND,NOP    / LOAD ARRAY(A*J+B) INTO ANS
JMS FORM
TAD I ADR    / MOVE INTO ANS
DCA ANS
ISZ ADR
TAD I ADR
DCA ANS+1
ISZ ADR
TAD I ADR
DCA ANS+2
JMP I FIND
/
/  SUBROUTINE IFIND
IFIND,NOP    / LOAD FIXED INTO ANS & FLOAT
JMS POINT
TAD ADR
DCA ADR    / POINTS TO FIXED POINT ARRAY MEMBER
TAD I ADR
DCA P13+1    / SET UP INTEGER
FINT
FPUT BIN    / HOLD C(FAC)
FGET P13
FNOR        / NORMALIZE
FPUT ANS
FGET BIN    / RESTORE C(FAC)
FEXT
JMP I IFIND
/
/  SUBROUTINE FORM
FORM,NOP    / FORM ADDRESS FOR FLOATING POINT
JMS POINT    / RETURN WITH A*J+B-1
TAD B
TAD B     / TRIPLE
TAD ADR   / FORM POINTER
DCA ADR
JMP I FORM
/
/  SUBROUTINE POINT
/    CALCULATE A*J+B-1 AND RETURN IN C(AC)
/    ALSO, RETURN ARRAY ADDRESS IN ADR
POINT,NOP
TAD I GO2    / GET POINTER TO CALLING SEQUENCE
DCA SEE
TAD I SEE    / GET ARRAY ADDRESS
AND MSK1    / 3777  DELETE ZEROTH BIT
DCA ADR
TAD I SEE   / ARRAY ADDRESS AGAIN
ISZ I GOC    / TO BYPASS ARGUMENT 1
SPA CLA
JMP A5
TAD J    / NO 2ND ARGUMENT
SZA    / J MAY BE 0 OR 1 TO GET ARRAY(1)
TAD NEG1
JMP A10
A5,ISZ SEE
TAD I SEE    / GET   BAA AAA BBB BBB   WORD
AND MSK2    / 4077  GET SIGN AND B
SPA
TAD MSK3    / 3700  EXTEND MINUS BITS
DCA B    / -64<B<+64  DECIMAL
TAD I SEE
ISZ I GO2    / TO BYPASS ARGUMENT 2
AND MSK3    / 3700  GET A
CLL RTR    / SHIFT
RTR
RTR
DCA A     / 0<=A<+32  DECIMAL
/  EVALUATE A*J+B-1
TAD J
DCA I MP2P    / MULTIPLICAND
TAD A    / MULTIPLIER
SZA      / NO NEED TO MULT IF A=0
JMS I MP4P    / 12 BIT MULTIPLY
TAD B
TAD NEG1
A10,DCA B
TAD B
JMP I POINT    / RETURN
/  CONSTANTS
GO2,5655     / IN THE INTERPRETOR
MSK1,3777
MSK2,4077
MSK3,3700
MP2P,6471    / LOCATION OF MULTIPLICAND
MP4P,6437    / FPP ROUTINE FOR FIXED POINT MULTIPLY
AGET,GET
NEG1,-1
NORM,6600    / FLOATING POINT NORMALIZE
P13,13
0
0
BIN,0
0
0
/
/  SUBROUTINE FIX
*4557
FIX,NOP    / FIX C(FAC)
TAD 44
SMA SZA
JMP .+3
CLA
JMP FIXEND
TAD M13
DCA 44
LOOP,TAD 44
SMA CLA
JMP I FIX
JMS I .+2
JMP LOOP
6200     / DIVI IN INTERPRETER
FIXEND,DCA 45
JMP I FIX
M13,-13
$
```

Beckman FLOWCHARTING WORKSHEET

Program Name ARRAY ACCESSING SUBROUTINE PACKAGE Deck Name AASP

Programmer DAVID G. FRUTCHEY Date NOVEMBER 18, 1968 Chart Page 1 Of 2

1 2 3 4 5

A

Subroutine services performed by the Floating Point Package. See documentation in manual 8-5-S

Special Symbols Used:
c(AC) The Accumulator
c(FAC) The Floating Accumulator at c(44,45,46)

SUBROUTINE LOAD

B LOAD

C FIND MOVE ARRAY(a*J+b) INTO c(ANS)

D GET MOVE c(ANS) INTO c(FAC)

E RETURN

SUBROUTINE STORE

STORE

FORM c(ADR) = ARRAY +3(a*J+b-1)

FINT ENTER FLOATING INTERPRETER

MOVE c(FAC) INTO LOCATION ADDRESSED BY c(ADR)

FEXT EXIT FLOATING INTERPRETER

RETURN

SUBROUTINE ISTORE

ISTORE

POINT a*J+b-1 = c(AC)

ADD THE STARTING ADDRESS AND STORE

FIX CONVERT c(FAC) FLOAT TO FIXED

MOVE c(45) INTO LOCATION ADDRESSED BY ADR

MAKE EXPONENT c(44) = 0013₈

ZERO c(46)

DNORM NORMALIZE c(FAC)

RETURN

SUBROUTINE FIND

FIND

FORM c(ADR) = ARRAY +3(a*J+b-1)

MOVE WORD ADDRESSED BY c(ADR) INTO ANS

INCREMENT c(ADR)

MOVE WORD ADDRESSED BY c(ADR) INTO ANS+1

INCREMENT c(ADR)

MOVE WORD ADDRESSED BY c(ADR) INTO ANS+2

RETURN

SUBROUTINE IFIND

IFIND

POINT a*J+b-1 = c(AC)

ADD THE STARTING ADDRESS AND STORE

MOVE INTEGER INTO A THREE WORD UNNORMALIZED FLOATING POINT WORD

FINT ENTER FLOATING INTERPRETER

SAVE THE c(FAC) IN c(BIN)

GET FLOATING POINT INTEGER, NORMALIZE, AND PLACE INTO c(ANS)

RESTORE c(FAC)

FEXT EXIT FLOATING INTERPRETER

RETURN

SUBROUTINE ILOAD

G ILOAD

H IFIND MOVE IARRAY(a*J+b) INTO c(ANS)

J GET MOVE c(ANS) INTO c(FAC)

K RETURN

Beckman FLOWCHARTING WORKSHEET

Program Name ARRAY ACCESSING SUBROUTINE PACKAGE Deck Name AASP

Programmer DAVID G. FRUTCHEY Date NOVEMBER 18, 1968 Chart Page 2 Of 2