NATIONAL RADIO ASTRONOMY OBSERVATORY
CHARLOTTESVILLE, VIRGINIA

ELECTRONICS DIVISION INTERNAL REPORT NO. 225

A LIBRARY OF BINARY SUBROUTINES FOR APPLE II PLUS

L. R. D'ADDARIO

JANUARY 1982

NUMBER OF COPIES: 150

A LIBRARY OF BINARY SUBROUTINES FOR APPLE II PLUS

Table of Contents

I. Introduction 3
II. Memory Organization 3
III. The Pointer Reset Program 6
IV. The LINK Subroutine 7
V. Calling Library Subroutine by Name 9
VI. Library Versions 2.0 and 2.1 9

Figures

Figure 1 Memory Configuration of Apple II Plus with Binary Library 4
Figure 2 Configuration of Lower Memory (Library Larger than 3k Bytes) 4

Tables

Table I Contents of Versions 2.0 and 2.1 11

Appendices

Appendix A Internal Organization of the Binary Library 13
Appendix B Listing of "LINK" Subroutine 14
Appendix C Listing of "LIBENT" Program 15
Appendix D Listing of Logical Operation Subroutines 17

References 12

A LIBRARY OF BINARY SUBROUTINES FOR APPLE II PLUS

L. R. D'Addario

I. Introduction

In programming the Apple II Plus computer in Applesoft Basic, it often happens that some special operation is required which cannot be implemented within that language, or which is awkward or inefficient to implement. An expedient solution is to implement the operation as a subroutine written in another, more appropriate language - usually an assembly language - and to access it from Basic via the CALL statement or the ampersand (&) command.

In our setup at NRAO, where we have several Apple II Plus computers with similar peripherals, the same special operations can be expected to be useful in many programs written by different people. For example, many of our present programs need routines to control the ADIOS module [1,2]. To avoid duplication of programming effort, the code for these operations should be written, maintained, and distributed in a way that is independent of the programs that will use it, and yet is easily accessible to them. The present report describes a method of accomplishing this.

II. Memory Organization

Figure 1 shows the suggested organization of the Apple's memory when binary subroutines are to be called from Applesoft Basic. This plan is based on the following ideas:

1. The subroutines are placed in the lower part of memory, starting at $800 = 2048. The starting point of the Basic program, normally $800 also, must

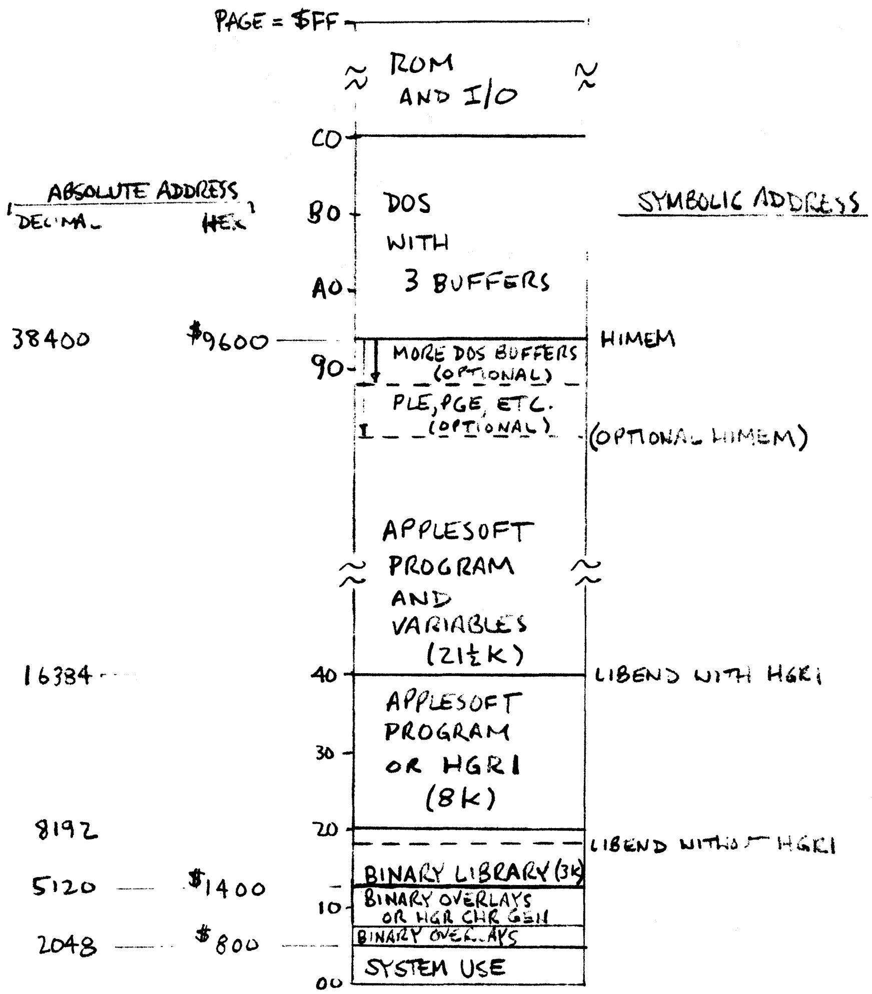

Fig. 1. Memory configuration of Apple II Plus with binary library, with and without protection of HGR1.

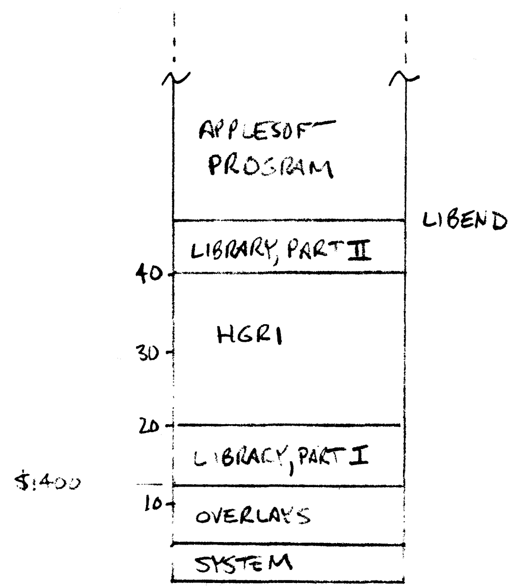

Fig. 2. Configuration of lower memory if library becomes larger than 3k bytes and HGR1 is needed.

be moved up enough to make room for the subroutines. An alternative would have been to place the subroutines in upper memory and to push HIMEM downward. But since we often want to use high resolution graphics, which requires that any long Applesoft program be placed above HGR1 ($2000-$3FFF), the space $800-$1FFF is often unused. The total usable memory is then larger with the subroutines in low memory.

2. All subroutines which are thought to be generally useful - i.e., used by at least several different main programs - are assembled together in a single block called the "binary library," starting at $1400 = 5120. Programs which use any of the library subroutines would normally load the entire library; this produces no penalty in usable memory space if HGR1 is also needed.

3. The first page of the library ($1400 thru $14FF) is reserved for JMP instructions to the entry points of the various subroutines in the library. Normally, a Basic program should CALL the appropriate JMP instruction address rather than the actual entry point address of a subroutine. This allows the library to be updated in a way that might require changing the entry point addresses of some subroutines without requiring changes in any Basic programs which use the library.

4. The space $800-$13FF is intended for loading subroutines which are not in the library but which are needed by a particular Basic program. The idea is to use this space for specialized routines which are needed by only one program (or at most a few). An exception is the high resolution graphics character generator, a purchased program [3] which occupies $C00-$FFF and requires data (a "font") occupying $1000-$13FF. If this program is required, the space for other routines is reduced to $800-$BFF, which is shared with Text Page 2.

III. The Pointer Reset Program

As mentioned earlier, the starting point of the Basic program must be moved upward to make room for the library. This can be accomplished simply by changing several pointers in the zero page, provided that the Basic program has not yet been loaded. A short program to do this is included in the initial versions of the library. The following command will load the library and reset the pointers so that subsequently-loaded Basic programs will reside above the library:

```
BRUN LIB
```

where LIB is the name of a binary file containing the library. This routine returns to the user in immediate mode, with any previously-loaded Basic program no longer accessible. An alternative is to use the utility program LOMEM [4] which actually moves a previously-loaded Basic program, and which can be executed from within the program. One can then

```
BLOAD LIB
```

so that the pointer-resetting program is not executed. However, LOMEM requires the user to specify the destination address, which might require knowing the length of the library, and the latter is variable.

The following example illustrates the procedure that I recommend. Suppose that a Basic program called WORK needs some routines from the library and also needs a special routine stored in a binary file named WORKER. The user stores the Basic program in a file named WORK.A and creates a text file named WORK containing these lines:

```
BRUN LIB
BLOAD WORKER
RUN WORK.A
```

Then, whenever he wants to run WORK, he types

EXEC WORK

and all the memory organization is automatically taken care of. Incidently, the EXEC file can also do various other useful things, such as making sure that the text window is properly set and that any resident routines required (DOS, PLE, etc.) are properly connected.

The pointer reset program knows exactly where the library ends and can place the Basic program immediately afterward, so no space is wasted. However, if Hires Graphics is to be used, then at least HGR1 ($2000-$3FFF) must also be protected from the Basic program. Therefore, another version of the library exists in which the pointer reset program causes Basic to start at $4000 = 16384 rather than at the end of the library. So far, the library does not extend up to the beginning of HGR1. If it should ever grow beyond that point, we can still have a version which protects HGR1 by skipping over it, as shown in Figure 2.

IV. The LINK Subroutine

Most useful subroutines require parameters from the calling program and produce results which must be returned to the calling program. Therefore, it is important to provide a method of passing data between a Basic program and a called subroutine. This could be done through POKE's and PEEK's, but that would be slow and cumbersome, especially for arrays.

The library includes a program which allows parameters to be passed in a simpler and more efficient manner. If a list of Basic variables is appended to the CALL statement, we can use routines within the Applesoft ROM [5] to find the addresses of these variables in memory. Our subroutine can then obtain the values of these variables or store results into them.

A subroutine called from Basic which needs the addresses of passed parameters should include the instruction

JSR LINK

where LINK is an entry point in the library. Upon return, the addresses of up to three parameters will be stored on the zero page, starting at $19, in the same order as in the CALL statement. For example, if Basic executes

CALL SUB,A,B,C

and SUB contains JSR LINK, then upon return from LINK the address of the Basic variable A will be in $19,$1A; the address of B will be in $1B,$1C; and the address of C will be in $1D,$1E. LINK also advances the text pointer to the end of the Basic statement, so that when SUB executes a RTS, the Basic program continues normally. If the CALL statement lists more than three parameters, the additional ones are ignored.

The precise syntax of the CALL statement is

CALL <adr>[{<delim><parameter>}]

where <delim> is any of the delimeter characters defined on page 33 of the Applesoft manual, except the colon and either parenthesis *; <parameter> is the name of a simple variable or an array element; and <adr> is a Basic expression which evaluates to the entry point address of the subroutine being called. Note that an array name (without a subscript, as distinguished from an array element) cannot be used as a parameter because LINK will think it is a simple variable. Another linking routine, called MLINK, has been written to pass array names without subscripts, but it will not be discussed here; contact the author if this interest you.

---

* The delimeters are thus ∿ = + - < > / * , ;

V. Calling Library Subroutine by Name

As mentioned earlier, the first page (255 bytes, actually) of the library is reserved for JMP instructions to the entry points of the various subroutines. This allows for 85 such instructions, which we'll call "indirect entry points." The addresses of these indirect entry points should remain stable through many revisions of the library. Nevertheless, the user must know at least the indirect entry address for each subroutine he calls.

To free the Basic program almost completely from the need to know absolute addresses in the library, a short program has been written which allows library subroutines to be called by name. The calling syntax is

CALL <libent>"<name>" <parameter>[{<delim><parameter>}]

where <libent> is the address of the LIBENT program, and <name> is the name of the desired subroutine. For example,

CALL LIB"CMUL"Y(0) = A(0)*B(0)

will execute the complex multiply subroutine. A slightly shorter calling sequence is obtained if the & command vector has been set to <libent>. Then

&"CMUL"Y(0) = A(0)*B(0)

will have the same effect as the previous example. Notice that, in either calling syntax, the second quote replaces the first delimeter of the parameter list.

In timing tests which compared CALL's through LIBENT to CALL's to the indirect entry points, no measurable difference was seen with a resolution of about 200 μsec.

VI. Library Versions 2.0 and 2.1

At this writing, the NRAO binary library contains the subroutines listed in Table I. The current library is called Version 2.0 or 2.1, according to whether the Basic program begins immediately after the library or after HGR1.

When the library is loaded, the version number times 10 is stored in $14FF = 5375. It is intended that this number will be odd for versions in which HGR1 is protected, and even otherwise. It is also intended that changes in the fraction part of the version number will indicate a minor revision which is downward-compatible (that is, all programs which worked with the earlier version should work with the new one), whereas changes in the integer part indicate major revisions in which some routines _may_ operate differently or be deleted. If the library is loaded through an EXEC file as suggested earlier, it might be useful to include the line

```
PRINT "LIBVERS = "PEEK(5375)/10
```

to be sure the user knows which version he got.

TABLE I: Contents of Versions 2.0 and 2.1

PROGRAM NAME	INDIRECT ENTRY Dec	Hex	NOTES
PTRSET	5120	1400	Resets pointers for Applesoft; automatically executed by BRUN LIB.
LINK	5123	1403	See text and Appendix B of this report.
ADIOS	5126	1406	ADIOS interface control; see [1,2].
ADOUT	5129	1409	
AND16	5132	140C	Logical operations on 16-bit integer variables. See Appendix D for listing.
OR16	5135	140F	
XOR16	5138	1412	
CADD	5141	1415	Complex Arithmetic Package; see [8].
CMUL	5144	1418	
CDIV	5147	141B	
POLAR	5150	141E	
RECT	5153	1421	
ADD2X2	5159	1427	
SUB2X2	5162	142A	
MUL2X2	5165	142D	
INV2X2	5168	1430	
LIBENT	5171	1433	See text and Appendix C of this report.

REFERENCES

[1] G. Weinreb and S. Weinreb, "ADIOS - Analog-Digital Input Output System for Apple Computer," NRAO Internal Report No. 212, April 1981.

[2] L. D'Addario, "Improved Software for Controlling the ADIOS Module," NRAO Internal Report No. 224, January 1982.

[3] A.P.P.L.E., "High Resolution Graphics Character Generator."

[4] N. Konzen, "The &LOMEM: Utility." The Apple Orchard, vol. 1, no. 1, p. 21 (March/April 1980). See also [3].

[5] J. Crossley, "Applesoft Internals." The Apple Orchard, vol. 1, no. 1, p. 12 (March/April 1980); also Call A.P.P.L.E. In Depth, no. 1, p. 51 (1981).

C. Bongers, "In the Heart of Applesoft." MICRO-the 6502 Journal, no. 33, p. 31 (Feb. 1981).

[6] S-C Software, P. O. Box 5537, Richardson, TX 75080; telephone (214) 324-2050.

[7] "Apple 6502 Editor-Assembler" manual.

[8] S. Keller and L. D'Addario, "Complex Math Package for Apple II Plus Computers," NRAO Internal Report No. 226, January 1982.

Appendix A

INTERNAL ORGANIZATION OF THE BINARY LIBRARY

The following information is provided for the benefit of those wanting to write programs for the library or to modify it. Familiarity with 6502 assembly language programming is assumed.

Two assemblers were evaluated for this application, the S-C Assembler II [6] and the DOS Tool Kit Editor-Assembler [7]. Several other assemblers were considered on the basis of their advertised features. The S-C Assembler II was selected, primarily because of a feature that allows the various subroutines to be maintained in separate source files but assembled together. The assembler accepts the pseudo-op

```
.INCLUDE  filename
```

which causes the specified source file to be included in the assembly in place of the pseudo-op. This assembler has the disadvantages that its editing facilities are poor and it stores the source code in a way that is not compatible with other editors.

To produce the library, a source file called LIBROOT was created containing the code for the indirect entry JMP table, the version number, the pointer reset program, and a list of .INCLUDE pseudo-ops for the subroutines to be assembled. Also in LIBROOT is a list of label definitions ("equates" or .EQ pseudo-ops) for addresses commonly needed by subroutines, including a large number of Applesoft internal subroutines and zero page addresses.

The library is maintained on a disk called the "NRAO Binary Library Master Disk," which includes a copy of the assembler; LIBROOT; the source code for each subroutine; and binary files with the current versions of the assembled library.

Appendix B

LISTING OF "LINK" SUBROUTINE

```
                     .IN LINK
               **************************************************
               * LINK PARAMETERS FROM APPLESOFT
               *
               * 810831 LRD. REV  820111.
               *-------------------------------------------------
0019-          PTABLE .EQ $19      PARAMETER ADDRESS TABLE, THRU $1E.
00FA-          PTABLE2 .EQ $FA     SECONDARY PTABLE

1525- A2 00    LINK   LDX #0
1527- 86 06    .1     STX TEMP
1529- 20 B7 00        JSR CHRGOT   NEXT CHAR OF BASIC TEXT.
152C- F0 1B           BEQ .5       END OF BASIC STATEMENT, EXIT.
152E- 20 B1 00        JSR CHRGET   SKIP DELIMETER (NORMALLY COMMA).
1531- F0 16           BEQ .5       END OF BASIC STATEMENT, EXIT.
1533- C9 2C           CMP #COMMA   DOUBLE COMMA?
1535- F0 09           BEQ .4        YES: PARAMETER HAS OMITTED.
1537- 20 E3 DF .2     JSR PTRGET   GET POINTER TO PARAMETER IN (A,Y).
153A- A6 06    .3     LDX TEMP     RESTORE X-REG (USED BY PTRGET).
153C- 94 1A           STY PTABLE+1,X
153E- 95 19           STA PTABLE,X
1540- E8       .4     INX
1541- E8              INX
1542- E0 06           CPX #06      MAXIMUM NUMBER OF PARAMETERS IS 3
1544- 30 E1           BMI .1
1546- 20 95 D9        JSR DATA     TABLE FULL, SKIP TO END OF ST.
1549- 60       .5     RTS
```

Appendix C

LISTING OF "LIBENT" PROGRAM

```
                 2410        .IN LIBENT
                 1000 ************************************************************
*
                 1010 * ROUTINE TO CALL PROPER LIBRARY PROGRAM BASED ON NAME.
                 1020 *
                 1030 * 811210 LRD
                 1040 *-----------------------------------------------------------
-
0022-            1060 QUOTE  .EQ $22
D412-            1070 ERROR  .EQ $D412
1D49- 20 B7 00   1080 LIBENT JSR CHRGOT   CHECK DELIMITER
1D4C- F0 25      1090        BEQ .4       END OF STATEMENT:SYNTAX ERR
                 1100 *
                 1110 * READ STRING FROM APPLESOFT PGM UP TO NEXT QUOTE
                 1120 * AND COMPUTE ITS HASH CODE:
1D4E- A9 00      1130        LDA #0       INITIALIZE HASH CODE
1D50- 85 06      1140        STA TEMP
1D52- 20 B1 00   1150 .1     JSR CHRGET   NEXT CHARACTER
1D55- F0 10      1160        BEQ .2       END OF STATEMENT?
1D57- C9 22      1170        CMP #QUOTE
1D59- F0 0C      1180        BEQ .2       YES, END OF NAME.
1D5B- 18         1190        CLC          HASH IT...
1D5C- E9 30      1200        SBC #48      UPPER CASE ONLY
1D5E- 06 06      1210        ASL TEMP
1D60- 65 06      1220        ADC TEMP
1D62- 85 06      1230        STA TEMP     SAVE
1D64- 4C 52 1D   1240        JMP .1       LOOP TO NEXT QUOTE
                 1250 *
                 1260 * SEARCH TABLE OF VALID HASH CODES
1D67- A2 10      1270 .2     LDX #HASHEN-HASHTB
1D69- A5 06      1280        LDA TEMP
1D6B- DD A9 1D   1290 .3     CMP HASHTB,X
1D6E- F0 20      1300        BEQ .6
1D70- CA         1310        DEX
1D71- D0 F8      1320        BNE .3
1D73- A9 7F      1330 .4     LDA #.5      CODE NOT FOUND.
1D75- A0 1D      1340        LDY /.5
1D77- 20 3A DB   1350        JSR STROUT   PRINT ERR MESSAGE
1D7A- A2 10      1360        LDX #16      CODE FOR "SYNTAX ERR"
1D7C- 4C 12 D4   1370        JMP ERROR
1D7F- 55 4E 4B
1D82- 4E 4F 57
1D85- 4E 20 4C
1D88- 49 42 20
1D8B- 53 55 42
1D8E- 52 22      1380 .5     .AS /UNKNOWN LIB SUBR"/
                 1390 **
                 1400 * CODE FOUND, SO JMP TO APPROPRIATE ROUTINE IN JUMP TABLE:
1D90- 86 06      1410 .6     STX TEMP
1D92- A9 01      1420        LDA #JTBL+1
1D94- A0 14      1430        LDY /JTBL
1D96- 18         1440        CLC
```

Appendix C (continued)

```
1D97- 65 06      1450          ADC TEMP
1D99- 65 06      1460          ADC TEMP
1D9B- 65 06      1470          ADC TEMP       3 BYTES PER ENTRY
1D9D- 90 01      1480          BCC .7
1D9F- C8         1490          INY
1DA0- 8D A7 1D   1500 .7       STA .8+1
1DA3- 8C A8 1D   1510          STY .8+2
1DA6- 6C A6 1D   1520 .8       JMP (*)
                 1530 *
                 1540 * HERE IS THE TABLE OF VALID HASH CODES:
1DA9- 00         1550 HASHTB .DA #0         *USE ZERO FOR PROTECTED ROUTINES
1DAA- 8D         1560        .DA #141
1DAB- 56         1570        .DA #86
1DAC- 7D         1580        .DA #125
1DAD- 3B         1590        .DA #59
1DAE- 7A         1600        .DA #122
1DAF- EC         1610        .DA #236
1DB0- 09         1620        .DA #9
1DB1- 64         1630        .DA #100
1DB2- 32         1640        .DA #50
1DB3- 90         1650        .DA #144
1DB4- A0         1660        .DA #160
1DB5- 4F         1670        .DA #79
1DB6- 1D         1680        .DA #29
1DB7- 62         1690        .DA #98
1DB8- F1         1700        .DA #241
1DB9- 51         1710 HASHEN .DA #81
                 1720        .EN
```

Appendix D

LISTING OF LOGICAL OPERATION SUBROUTINES

```
                          .IN LOGICAL
                 **************************************************
                 * LOGICAL OPERATIONS ON INTEGER VARIABLES.
                 *
                 *     CALL ADR,Z%=X%+Y%
                 * WILL COMBINE (X%) WITH (Y%) AND PUT RESULT IN (Z%).
                 * THE OPERATION PERFORMED WILL BE 16-BIT LOGICAL:
                 *     AND   IF (ADR)=AND16, OR
                 *     OR    IF (ADR)=OR16, OR
                 *     XOR   IF (ADR)=XOR16.
                 *-------------------------------------------------
15F7- A9 31      AND16  LDA #$31        OP CODE
15F9- 8D 1C 16          STA OPER1
15FC- 8D 23 16          STA OPER2
15FF- 4C 15 16          JMP LOGIC
1602- A9 11      OR16   LDA #$11        OP CODE FOR ORA (M),Y
1604- 8D 1C 16          STA OPER1
1607- 8D 23 16          STA OPER2
160A- 4C 15 16          JMP LOGIC
160D- A9 51      XOR16  LDA #$51        OP CODE FOR EOR (M),Y
160F- 8D 1C 16          STA OPER1
1612- 8D 23 16          STA OPER2
1615- 20 25 15   LOGIC  JSR LINK
1618- A0 00             LDY #0
161A- B1 0B             LDA (PTABLE+2),Y  GET LSB OF X.
161C- 31 0D      OPER1  AND (PTABLE+4),Y  OPERATE WITH LSB OF Y.
161E- 91 09             STA (PTABLE),Y    RESULT TO LSB OF Z.
1620- C8                INY
1621- B1 0B             LDA (PTABLE+2),Y  GET MSB OF X.
1623- 31 0D      OPER2  AND (PTABLE+4),Y  OPERATE WITH MSB OF Y.
1625- 91 09             STA (PTABLE),Y    RESULT TO MSB OF Z.
1627- 60                RTS               DONE.
```

NATIONAL RADIO ASTRONOMY OBSERVATORY

Addition to EDIR No. 225

A Library of Binary Subroutines for Apple II Plus

NRAO BINARY LIBRARY VERSIONS 3.0 and 3.1

L. R. D'Addario

May 3, 1982

I. NRAO Binary Library Versions 3.0 and 3.1

Versions 3.0 and 3.1 of the NRAO Binary Library are now current. Changes from versions 2.0 and 2.1, described in EDIR #225, are as follows:

1. LINK has been extended to allow passing up to six parameters instead of three. The addresses of the 4th, 5th, and 6th parameters are stored beginning at PTABLE2=$FA.

2. Errors in CMUL and CDIV have been corrected. The original versions did not handle properly the case where the result used the same variable as an operand.

3. POLAR has been changed to conform to the description in EDIR #225.

4. To save space, some code common to CMUL, CDIV, POLAR, and RECT has been moved to a subroutine called LINKC; the latter calculates the addresses of imaginary parts of passed parameters.

5. The new routines for handling the ADIOS module (see addition to EDIR #224, dated March 30, 1982) have been incorporated. The old routines ADIOS and ADOUT have been deleted.

6. The hash code table used by LIBENT has been moved to LIBROOT for convenience in maintaining the library.

Changes 1 through 4 have also been incorporated in versions 2.2 and 2.3, which should be compatible with 2.0 and 2.1.

Persons desiring copies of the library object code or the source code of any subroutine should contact Stowe Keller or Larry D'Addario. Notices of future revisions will be sent only to known users within NRAO and persons requesting to be put on a mailing list.

From the attached listing one can determine the entry points of each routine (direct and indirect), the name used to access each routine through LIBENT, and the memory occupied by the library.

```
:ASM

                 1000 ************************************************
                 1010 * NRAO BINARY LIBRARY
                 1020 *
                 1030 * 820325  VERSION 3.0 SK
                 1040 *-----------------------------------------------
                 1050          .OR $1400
                 1060          .TF LIB 3.0
1400-            1070 LIBRARY .EQ *
                 1080 *
                 1090 * LINKS TO SUBROUTINES:
1400- 4C 11 15   1100 JTBL     JMP LOADER
1403- 4C 36 15   1110          JMP LINK
1406- 4C 6A 16   1120          JMP AINIT
1409- 4C 84 15   1130          JMP ASERV
140C- 4C 1D 17   1140          JMP AND16
140F- 4C 28 17   1150          JMP OR16
1412- 4C 33 17   1160          JMP XOR16
1415- 4C 4E 17   1170          JMP CADD
1418- 4C 8F 17   1180          JMP CMUL
141B- 4C F6 17   1190          JMP CDIV
141E- 4C AA 18   1200          JMP POLAR
1421- 4C 45 19   1210          JMP RECT
1424- 4C 84 19   1220          JMP MLINK
1427- 4C C9 19   1230          JMP ADD2X2
142A- 4C 18 1A   1240          JMP SUB2X2
142D- 4C 67 1A   1250          JMP MUL2X2
1430- 4C 37 1C   1260          JMP INV2X2
1433- 4C F8 1D   1270          JMP LIBENT
1436-            1280          .BS LIBRARY+$FF-*
14FF- 1E         1290 VERS     .DA #30        VERSION NUMBER
                 1300 * HERE IS THE TABLE OF VALID HASH CODES FOR LIBENT:
1500- 00         1310 HASHTB .DA #0           *USE ZERO FOR PROTECTED ROUTINES
1501- 8D         1320          .DA #141       "LINK"
1502- 89         1330          .DA #137       "AINIT"
1503- C9         1340          .DA #201       "ASERV"
1504- 3B         1350          .DA #59        "AND16"
1505- 7A         1360          .DA #122       "OR16"
1506- EC         1370          .DA #236       "XOR16"
1507- 09         1380          .DA #9         "CADD"
1508- 64         1390          .DA #100       "CMUL"
1509- 32         1400          .DA #50        "CDIV"
150A- 90         1410          .DA #144       "POLAR"
150B- A0         1420          .DA #160       "RECT"
150C- 4F         1430          .DA #79        "MLINK"
150D- 1D         1440          .DA #29        "ADD2X2"
150E- 62         1450          .DA #98        "SUB2X2"
150F- F1         1460          .DA #241       "MUL2X2"
1510- 51         1470 HASHEN .DA #81          "INV2X2"

1E58-            2700 LASTLB .EQ *
1E58-            2710 LIBEND .EQ *
                 2720          .EN
```