BASEX Tape and Disk Guide
Table of Contents

0139

BASEX Tape and Disk Guide

by Paul K. Warme

Introduction

The programs described in this manual enable BASEX programs to communicate with either of two kinds of mass storage devices: North Star floppy disk drives and Meca tape drives. The basic features of these programs may also serve as an example for interfacing other types of mass storage devices with BASEX programs.

The North Star disk handler provides the following capabilities:

1. Up to four drives can be accessed and the directory of any drive can be listed.
2. Named files can be created, deleted, loaded, saved or checked for accuracy by comparison with memory.
3. As many as five files can be open simultaneously for read and/or write operations.
4. Once a file has been opened, 256 byte records may be written, read or verified in sequential or random order.
5. A BASEX program may load and execute a new program which may or may not overlay the old program and data arrays.
6. The BASEX/North Star disk handler occupies only 512 bytes of memory in addition to the normal disk operating system and the user program.
7. A Basexerciser demonstrates execution of all disk operations from the keyboard.

The Meca tape drive handler provides these features:

1. Up to four tape drives can be accessed and the directory of any drive can be listed.
2. The tape on any drive can be mounted, unloaded, rewound or initialized to zero the directory.
3. Named files can be loaded, saved with automatic verification or overlayed on the same region of the tape if space permits.
4. A BASEX program may load and execute another program which may or may not overlay the old program and data arrays.
5. The BASEX/Meca tape handler occupies only 270 bytes in addition to the normal tape operating system and the user program.
6. A Basexerciser sample program demonstrates execution of all tape operations from the keyboard.

The fast random access capability of the North Star disk drive together with the economy and large capacity of the Meca tape drive provide a mass storage capability that is hard to match at double the price. If you own both of these peripherals, you can combine disk and tape operations in your BASEX programs, as exemplified by a third Basexerciser routine which is included with this package.

I. The BASEX/North Star Disk Handler

A. Descriptions of Disk Operations

Each of the following disk operations can be invoked in a BASEX program by the DSK command, followed by the operation number (e.g., Ø=write, 7=list directory, etc.) and a variable number of arguments, as described below. The operation number may be specified either as a constant or as a regular variable, in which case the suggested mnemonic is given in parentheses below. Bear in mind that these mnemonics must be defined somewhere in your BASEX program, if they are used. In the examples, the regular variables FILE#, BUFFER, RECORD, UNIT and NSECT may optionally be replaced by a constant. NAME$ is a string variable of eight or fewer letters, and does not require the usual character range variables used with BASEX strings. Successful completion of disk operations and error conditions are indicated by the value stored at location 1Ø752 (decimal). In most cases, you should define a variable EOF=1Ø752 so that a WRD EOF command in your BASEX program can get this value in the accumulator (A). If EOF is set to -2 on return, a bad operation number was used. The meanings of other possible EOF values depend on the type of command, as discussed below.

Operation Ø(WRT): WRiTe next sector
Example: DSK WRT FILE# BUFFER

The write operation causes 256 bytes of data, beginning at the memory location specified by BUFFER, to be written into the next sequential sector of the disk file associated with FILE#. Note that before this and subsequent commands using the FILE# argument can be used, FILE# must be defined by the open operation (see operation 11 below).

At the conclusion of this operation, the sector pointer for the file is incremented to permit sequential access to the following sector during the next write operation. Normal completion is indicated by EOF=Ø, while an attempt to write past the end of file will return with EOF=1. Any other type of disk error leads to EOF=-1.

Operation 1(RED): REaD next sector
Example: DSK RED FILE# BUFFER

The read operation causes 256 bytes of data to be read from the next sequential sector of the disk file associated with FILE# and transferred into memory beginning at location BUFFER. The sector pointer for the file is then incremented to permit access to the following sector during the next read operation. The EOF indications are the same as for the write operation.

Operation 2(VFY): VeriFY next sector
Example: DSK VFY FILE# BUFFER

The verify operation will enable you to verify that the next sequential 256 byte sector associated with FILE# is identical to the data block beginning at memory location BUFFER. If a bad comparison is detected, this operation aborts by printing U XXXHD?, signalling a hard disk error on unit U in block XXX, and control is transferred to the disk operating system. In all other cases, control returns to the user program. If a correct comparison is detected, EOF is set to 0; EOF=1 means that the requested sector is beyond the end of the file, while EOF=-1 indicates that some other error condition was detected.

Operation 3(SFD): Save all remaining sectors of File on Disk
Example: DSK SFD FILE# BUFFER

The save operation is like the write operation, except that all remaining sectors in the file (from the current sector pointer onward) are filled with sequential memory data from BUFFER until the end of file condition is detected. Thus, EOF will normally be set to 1 after return to the user program. Caution: If the current sector pointer is beyond the end of the file, EOF will also be set to 1 on return, without any transfer having occurred. If EOF is not 1, it means that a disk error has occurred. Immediately after a file is opened, the sector pointer always points to the first record of the file, so a save operation at this point will save the entire file.

Operation 4(LFD): Load all remaining sectors of File on Disk
Example: DSK LFD FILE# BUFFER

The load operation parallels the read operation, except that all sectors of the file (from the current sector pointer onward) are loaded into memory, starting at BUFFER. The EOF values on return are interpreted in the same way as for the save operation. Immediately after a file is opened, a load operation will load the entire file.

Operation 5(CKD): CheckK all remaining sectors of file on Disk
Example: DSK CKD FILE# BUFFER

By use of the check operation, it is possible to compare all sectors of a disk file (beginning at the current sector pointer) with memory data stored at BUFFER. Like the verify operation, a bad comparison aborts to the disk operating system and prints U XXXHD? signalling a hard disk error on unit U in sector XXX. If control is returned to the user program, the EOF values will be like those described for the save operation.

Operation 6(SEK): SEeK a given record of disk file
Example: DSK SEK FILE# RECORD

The seek operation allows the sector pointer for a disk file to be positioned at any sector in that file. The first sector is sector 0. This provides direct (random) access to any 256 byte record in a file, without requiring that all intervening records be accessed beforehand. The seek operation itself does not access the

disk at all, but merely changes the sector pointer value to be used by a subsequent read or write operation. On return, EOF=1 signals that the requested record is beyond the end of the file, while EOF=Ø is the normal condition.

Operation 7(LST): LiST disk directory
Example: DSK LST UNIT

This operation will list all of the files available on drive UNIT, together with their length in sectors, starting disk sector and file type. The EOF value is unaffected by this operation, but any disk errors will abort to the disk operating system.

Operation 8(CHN): CHaiN (load and execute) a new program
Example:DSK CHN NAME$ UNIT

The chain operation permits a BASEX program to load and execute another program, which may or may not overlay the current BASEX program, its symbol table, data arrays or the BASEX execution subroutines. The only requirement is that the program not overlay the disk operating system (hex locations 2ØØØ-29FF). Of course, this operation does not normally return to the user program; however, if it does, EOF=-1 means that the name was of bad syntax and EOF=Ø means that the requested file is not on the disk.

Operation 9(DLD): DeLete file on Disk
Example: DSK DLD NAME$ UNIT

The delete operation will delete file NAME$ on disk drive UNIT. Successful completion is indicated by EOF=1. If the file is not found, EOF is set to Ø; EOF=-1 indicates a syntax error in NAME$.

Operation 1Ø(CRT): CReaTe file on disk
Example: DSK CRT NAME$ UNIT NSECT

This operation allocates NSECT sectors on disk drive UNIT for file NAME$. It does this by creating a new entry in the disk directory for that drive. On return, EOF=-2 means that a file already exists by that name; EOF=-1 means that there is not enough space left on the disk for that file; EOF=Ø indicates a disk error of some other type. If EOF is equal to NSECT on return, this indicates successful completion.

Operation 11(OPN): OPeN disk file for sequential or random access
Example: DSK OPN NAME$ UNIT FILE#

The open operation associates a file number (FILE#) with a particular disk file (NAME$) for use with subsequent read, write, verify, save, load, check or seek operations, all of which use the FILE# parameter. When a file is opened, the locations of the first and last sectors of the file are noted, so that other operations can determine whether disk transfers requested by the user are within the disk space allocated to that file, thus ensuring that other disk files will not be read or overwritten. Up to five files, numbered from Ø to 4, can be open at any time and the user may read and/or write to any open file. A file number can be redefined at any time by a second open operation, without any need to close the file previously defined. Caution: Other disk operations do not

check to make sure that a file has been opened before use; thus, unpredictable results will be obtained if you access an unopened file.

When a file is opened, the sector pointer for that file is set to Ø. Each subsequent read, write or verify operation advances the sector pointer by one sector, thus providing automatic sequential access. A useful technique for reading sequential sectors, modifying the data and then writing the data out to the same sector on the disk is to open the same file twice with different file numbers. In this way, you can avoid having to seek the previous record each time you want to write the data back in the same sector. The seek operation can also be used to set the pointer to any sector within the file, thus providing direct (random) access to any portion of the file.

If you attempt to open a nonexistent file, EOF will be set to -1 on return; other disk errors set EOF=Ø. If the file was opened without error, EOF will contain the number of sectors allocated for that file.

B. Assembler Source Listing for North Star Floppy Disk Handler

The disk handler normally resides at 2AØØ through 2BFF hex, above the disk operating system (DOS), which occupies hex locations 2ØØØ to 29FF. If your DOS is at a nonstandard location, you must redefine the equates at the start of the program accordingly. The addresses used in this compilation apply to DOS version 2; if you are using version 3, you must alter this program as indicated in the comments.

The following changes are required for Version 3 of DOS:

2BØ1=51, 2BØ8=31, 2B1F=ØØ, 2B2Ø=ØØ, 2B21=ØØ

2B2F=29, 2B3Ø=22, 2B32=ØC, 2B53=61.

```
ASSM 2A00

2A00                0000 *WRITE FNUM BUF
2A00                0001 *READ FNUM BUF
2A00                0002 *VERIFY FNUM BUF
2A00                0003 *SAVE FNUM BUF
2A00                0004 *LOAD FNUM BUF
2A00                0005 *CHECK FNUM BUF
2A00                0006 *SEEK FNUM REC
2A00                0007 *LIST UNIT
2A00                0008 *CHAIN NAME$ UNIT
2A00                0009 *DELETE NAME$ UNIT
2A00                0010 *CREATE NAME$ UNIT NSECT
2A00                0011 *OPEN NAME$ UNIT FNUM
2A00                0020 *BASEX HANDLER FOR NORTH STAR FLOPPY DISK
2A00                0030 *COPYRIGHT 1978 BY  PAUL K. WARME
2A00                0040 NEXT   EQU   261H     *BASEX ENTRY POINTS
2A00                0050 GETARG EQU   26FH
2A00                0060 RETRN  EQU   329H
2A00                0062 NEG    EQU   35BH
2A00                0065 X      EQU   2000H    *DOS ORIGIN OFFSET
2A00                0070 DLOOK  EQU   X+1CH    *DOS ENTRY POINTS
2A00                0080 DCOM   EQU   X+22H    * FOR DOS VERSION 2
2A00                0090 LISTER EQU   X+25H    *USE V3 FOR VERSION 3
2A00                0100 DELT   EQU   X+135H   *V3=2131H
2A00                0110 LDGO   EQU   X+162H   *V3=2150H
2A00                0120 CRENT  EQU   X+2A7H   *V3=2261H
2A00                0130 LOOKUP EQU   X+542H   *V3=250CH
2A00                0160 BLANK8 EQU   X+71BH   *V3=2229H
2A00 00 00          0165 EOF    DW    0        *ERROR INDICATOR
2A02 E1             0170 DISK   POP   H        *GET PROGRAM POINTER (PP)
2A03 CD 6F 02       0180        CALL  GETARG   *BC=NEXT ARG AT PP
2A06 79             0190        MOV   A,C
2A07 32 FC 2B       0200        STA   MODE     *OPERATION # (0-11)
2A0A FE 07          0210        CPI   7
2A0C CA 84 2A       0220        JZ    LIST
2A0F D2 B9 2A       0230        JNC   NAME
2A12 CD 6F 02       0240        CALL  GETARG   *BC=FILE#
2A15 79             0250        MOV   A,C
2A16 32 FD 2B       0260        STA   FNUM
2A19 CD 6F 02       0270        CALL  GETARG   *BC=BUFFER LOCATION
2A1C E5             0280        PUSH  H
2A1D 3A FC 2B       0290        LDA   MODE
2A20 FE 06          0300        CPI   6
2A22 CA 8F 2A       0310        JZ    SEEK
2A25 60             0320        MOV   H,B
2A26 69             0330        MOV   L,C
2A27 22 F7 2B       0340        SHLD  BUF
2A2A D6 03          0350        SUI   3        *MODE<3 MEANS ONE SECTOR
2A2C D2 35 2A       0360        JNC   MULTI    *MODE 3-5 MEANS ALL SECTORS
2A2F CD 49 2A       0370        CALL  RWV
2A32 C3 8C 2B       0380        JMP   END
2A35 32 FC 2B       0390 MULTI  STA   MODE     *MODE-3 IS TYPE
2A38 CD 49 2A       0400 NXT    CALL  RWV
2A3B B7             0410        ORA   A        *END OF FILE OR ERROR?
2A3C C2 8C 2B       0420        JNZ   END      *YES
```

```
2A3F 2A F7 2B   0430        LHLD  BUF       *BUMP BUFFER PTR+256
2A42 24         0440        INR   H
2A43 22 F7 2B   0450        SHLD  BUF
2A46 C3 38 2A   0460        JMP   NXT
2A49 06 00      0480 RWV    MVI   B,0       *READ/WRITE/VERIFY
2A4B 3A FD 2B   0490        LDA   FNUM
2A4E 4F         0500        MOV   C,A
2A4F 21 D0 2B   0510        LXI   H,NXTBLK  *NEXT SECTOR PTR
2A52 CD 95 2B   0520        CALL  BGT2ED    *DE=SECTOR #
2A55 D5         0570        PUSH  D         *CURRENT SECTOR #
2A56 13         0580        INX   D
2A57 2B         0585        DCX   H
2A58 72         0590        MOV   M,D       *SAVE NEXT SECTOR #
2A59 2B         0600        DCX   H
2A5A 73         0610        MOV   M,E
2A5B CD 92 2B   0620        CALL  LST2ED    *DE=LAST SECTOR
2A5E CD 5B 03   0630        CALL  NEG       *CHECK FOR END OF FILE
2A61 1B         0640        DCX   D         *DE=-LSTBLK(FNUM)-1
2A62 E1         0720        POP   H         *HL=NXTBLK(FNUM)
2A63 E5         0730        PUSH  H
2A64 19         0740        DAD   D
2A65 D2 6C 2A   0750        JNC   RWVOK
2A68 3E 01      0760        MVI   A,1       *PAST END OF FILE
2A6A E1         0780        POP   H
2A6B C9         0790        RET
2A6C 21 CB 2B   0800 RWVOK  LXI   H,UNIT
2A6F 09         0810        DAD   B         *FNUM OFFSET
2A70 4E         0820        MOV   C,M       *C=UNIT(FNUM)
2A71 3A FC 2B   0830        LDA   MODE
2A74 47         0840        MOV   B,A       *B=R/W/V
2A75 2A F7 2B   0850        LHLD  BUF       *RAM ADR
2A78 D1         0860        POP   D         *SECTOR #
2A79 EB         0870        XCHG
2A7A 3E 01      0880        MVI   A,1       *TRANSFER 1 SECTOR
2A7C CD 22 20   0890        CALL  DCOM      *DOS ENTRY
2A7F 3E 00      0900        MVI   A,0
2A81 D0         0910        RNC
2A82 3D         0920        DCR   A         *ERROR ON CARRY
2A83 C9         0930        RET
2A84 CD 6F 02   0950 LIST   CALL  GETARG    *BC=UNIT #
2A87 E5         0960        PUSH  H
2A88 79         0970        MOV   A,C       *A=UNIT
2A89 CD 25 20   0980        CALL  LISTER    *DOS ENTRY
2A8C C3 29 03   0990        JMP   RETRN
2A8F C5         1000 SEEK   PUSH  B         *STACK SECTOR #
2A90 3A FD 2B   1010        LDA   FNUM
2A93 4F         1020        MOV   C,A
2A94 06 00      1030        MVI   B,0
2A96 21 DA 2B   1040        LXI   H,FSTBLK
2A99 CD 95 2B   1050        CALL  BGT2ED    *DE=FSTBLK(FNUM)
2A9C E1         1100        POP   H         *HL=SECTOR #
2A9D 19         1110        DAD   D
2A9E 22 F9 2B   1120        SHLD  REC       *SECTOR ON DISK
2AA1 CD 92 2B   1130        CALL  LST2ED    *DE=LAST SECTOR
2AA4 CD 5B 03   1140        CALL  NEG       *CHECK FOR END OF FILE
2AA7 1B         1150        DCX   D         *DE=-LSTBLK(FNUM)-1
```

```
2AA8 2A F9 2B   1230         LHLD  REC
2AAB EB         1240         XCHG
2AAC 19         1250         DAD   D          *PAST END OF FILE?
2AAD 3E 00      1260         MVI   A,0
2AAF D2 B3 2A   1270         JNC   SOK        *NO
2AB2 3C         1280         INR   A          *EOF=1 FOR END OF FILE
2AB3 CD 9C 2B   1290 SOK     CALL  NXT2ED     *STORE NXTBLK(FNUM)
2AB6 C3 8C 2B   1300         JMP   END
2AB9 CD 97 2B   1370 NAME    CALL  GET2ED     *DE=(NAME$)
2ABC E5         1410         PUSH  H
2ABD 13         1420         INX   D          *DE=(FIRST CHAR)
2ABE 21 EE 2B   1430         LXI   H,NAMBUF   *MOVE NAME
2AC1 0E 09      1440         MVI   C,9
2AC3 0D         1450 LOOP    DCR   C
2AC4 CA DA 2A   1460         JZ    NAMDON
2AC7 1A         1470         LDAX  D
2AC8 FE 21      1480         CPI   33         *END OF NAME?
2ACA DA D3 2A   1490         JC    BLNK       *YES
2ACD 77         1500         MOV   M,A        *MOVE CHAR
2ACE 23         1510         INX   H
2ACF 13         1520         INX   D
2AD0 C3 C3 2A   1530         JMP   LOOP
2AD3 36 20      1540 BLNK    MVI   M,32       *FILL WITH SPACES
2AD5 23         1550         INX   H
2AD6 0D         1560         DCR   C
2AD7 C2 D3 2A   1570         JNZ   BLNK
2ADA E1         1580 NAMDON  POP   H          *HL=PP
2ADB CD 6F 02   1590         CALL  GETARG     *BC=UNIT
2ADE E5         1600         PUSH  H
2ADF 79         1610         MOV   A,C
2AE0 32 FB 2B   1620         STA   DVC
2AE3 21 EE 2B   1630         LXI   H,NAMBUF
2AE6 CD 1C 20   1640         CALL  DLOOK      *DOS DIRECTORY SEARCH
2AE9 F5         1650         PUSH  6          *SAVE STATUS
2AEA 3A FC 2B   1660         LDA   MODE
2AED FE 09      1670         CPI   9
2AEF CA 03 2B   1680         JZ    DELETE
2AF2 D2 0F 2B   1690         JNC   CROP
2AF5 F1         1700 CHAIN   POP   6          *CHECK STATUS
2AF6 DA AB 2B   1710         JC    ERR
2AF9 3A FB 2B   1720         LDA   DVC
2AFC 4F         1722         MOV   C,A
2AFD 31 5A 20   1725         LXI   6,205AH    *DOS STACK PTR
2B00 C3 62 21   1730         JMP   LDGO       *DOS ENTRY
2B03 F1         1740 DELETE  POP   6          *STATUS
2B04 DA AB 2B   1750         JC    ERR
2B07 CD 35 21   1760         CALL  DELT       *DOS ENTRY
2B0A 3E 01      1762         MVI   A,1
2B0C C3 8C 2B   1765         JMP   END
2B0F FE 0B      1770 CROP    CPI   11         *CREATE/OPEN
2B11 CA 5B 2B   1780         JZ    OPEN
2B14 D2 A6 2B   1790         JNC   ERRC       *BAD MODE
2B17 F1         1800 CREATE  POP   6          *STATUS
2B18 D2 A6 2B   1810         JNC   ERRC       *ERROR IF NAME$ FOUND
2B1B B7         1812         ORA   A
2B1C CA AF 2B   1814         JZ    ERR2       *BAD NAME$
```

```
2B1F 2A E9 28   1810        LHLD  28E9H     *!OMIT FOR DOS VERSION 3!
2B22 22 F7 2B   1820        SHLD  BUF       *HL=FIRST FREE SECTOR
2B25 E1         1830        POP   H
2B26 CD 6F 02   1840        CALL  GETARG    *BC=# SECTORS
2B29 E5         1850        PUSH  H
2B2A 79         1860        MOV   A,C
2B2B CD BD 2B   1870        CALL  EFS
2B2E 11 1B 27   1880        LXI   D,BLANK8 *8 SPACES
2B31 CD 42 25   1890        CALL  LOOKUP    *DOS ENTRY
2B34 DA AF 2B   1900        JC    ERR2      *DIRECTORY FULL
2B37 22 F9 2B   1910        SHLD  REC       *8TH BYTE OF FREE SLOT
2B3A 2A F7 2B   1920        LHLD  BUF       *FIRST FREE SECTOR
2B3D EB         1930        XCHG
2B3E 06 00      1940        MVI   B,0
2B40 3A 00 2A   1950        LDA   EOF
2B43 4F         1960        MOV   C,A       *BC=NSECT
2B44 21 A1 FE   1970        LXI   H,0FEA1H *-351
2B47 09         1980        DAD   B
2B48 DA AF 2B   1990        JC    ERR2      *FILE TOO LARGE
2B4B 19         2000        DAD   D         *FREE SECTOR+NSECT-351
2B4C DA AF 2B   2010        JC    ERR2      *TOO LONG
2B4F 2A F9 2B   2020        LHLD  REC       *DIRECTORY PTR
2B52 CD A7 22   2030        CALL  CRENT     *DOS ENTRY
2B55 DA B4 2B   2035        JC    ERR1
2B58 C3 29 03   2040        JMP   RETRN
2B5B F1         2100 OPEN   POP   6         *STATUS
2B5C DA AB 2B   2110        JC    ERR
2B5F 22 F7 2B   2120        SHLD  BUF       *DIRECTORY PTR
2B62 E1         2130        POP   H         *PP
2B63 CD 6F 02   2140        CALL  GETARG    *BC=FNUM
2B66 E5         2150        PUSH  H
2B67 21 CB 2B   2160        LXI   H,UNIT
2B6A 09         2170        DAD   B         *HL=(UNIT(FNUM))
2B6B 3A FB 2B   2180        LDA   DVC
2B6E 77         2190        MOV   M,A       *SET UNIT
2B6F 2A F7 2B   2200        LHLD  BUF
2B72 CD 97 2B   2210        CALL  GET2ED    *DE=DISK START
2B75 D5         2220        PUSH  D
2B76 CD 97 2B   2230        CALL  GET2ED    *DE=NSECT
2B79 7B         2240        MOV   A,E
2B7A E1         2260        POP   H         *HL=START
2B7B EB         2270        XCHG
2B7C 19         2280        DAD   D         *LAST SECTOR+1
2B7D 2B         2290        DCX   H
2B7E E5         2300        PUSH  H         *LAST SECTOR
2B7F CD 9C 2B   2310        CALL  NXT2ED    *STORE NXTBLK(FNUM)
2B82 01 08 00   2320        LXI   B,8       *OFFSET FOR FSTBLK
2B85 CD A0 2B   2330        CALL  PUT2ED    *STORE FSTBLK(FNUM)
2B88 D1         2340        POP   D         *DE=LAST SECTOR
2B89 CD A0 2B   2350        CALL  PUT2ED    *STORE LSTBLK(FNUM)
2B8C CD BD 2B   2360 END    CALL  EFS       *STORE EOF
2B8F C3 29 03   2370        JMP   RETRN     *BASEX RETURN
2B92 21 E4 2B   2380 LST2ED LXI   H,LSTBLK *GET LSTBLK ADR
2B95 09         2385 BGT2ED DAD   B         *ADD FNUM OFFSET
2B96 09         2390        DAD   B
2B97 5E         2395 GET2ED MOV   E,M       *LOAD DE AT HL
```

```
2B98 23          2400          INX   H
2B99 56          2405          MOV   D,M
2B9A 23          2410          INX   H
2B9B C9          2415          RET
2B9C 21 D0 2B    2420 NXT2ED   LXI   H,NXTBLK *GET NXTBLK ADR
2B9F 09          2425          DAD   B        *ADD FNUM OFFSET
2BA0 09          2430 PUT2ED   DAD   B
2BA1 73          2435          MOV   M,E      *STORE DE AT HL
2BA2 23          2440          INX   H
2BA3 72          2445          MOV   M,D
2BA4 23          2450          INX   H
2BA5 C9          2455          RET
2BA6 3E FE       2500 ERRC     MVI   A,0FEH   *BAD COMMAND OR DUPL FILE
2BA8 C3 B6 2B    2510          JMP   STORE
2BAB B7          2520 ERR      ORA   A        *BAD NAME IF A=0
2BAC C2 B4 2B    2530          JNZ   ERR1
2BAF 3E FF       2540 ERR2     MVI   A,0FFH   *BAD NAME OR DISK FULL
2BB1 C3 B6 2B    2550          JMP   STORE
2BB4 3E 00       2560 ERR1     MVI   A,0      *OTHER DISK ERRORS
2BB6 CD BD 2B    2580 STORE    CALL  EFS      *STORE EOF
2BB9 E1          2585          POP   H
2BBA C3 61 02    2590          JMP   NEXT     *NEXT BASEX COMMAND
2BBD 32 00 2A    2600 EFS      STA   EOF
2BC0 07          2610          RLC            *NEGATIVE VALUE?
2BC1 3E FF       2620          MVI   A,0FFH
2BC3 DA C7 2B    2630          JC    EFF      *YES
2BC6 3C          2640          INR   A
2BC7 32 01 2A    2650 EFF      STA   EOF+1    *STORE HIGH BYTE
2BCA C9          2660          RET
2BCB             3000 UNIT     DS    5
2BD0             3010 NXTBLK   DS    10
2BDA             3020 FSTBLK   DS    10
2BE4             3030 LSTBLK   DS    10
2BEE             3040 NAMBUF   DS    8
2BF6 20          3050 SPACE    DB    32
2BF7 00 00       3060 BUF      DW    0
2BF9 00 00       3070 REC      DW    0
2BFB 00          3080 DVC      DB    0
2BFC 00          3090 MODE     DB    0
2BFD 00          3110 FNUM     DB    0
```

C. BASEXerciser for North Star Floppy Disk Drives

This exerciser program will allow you to execute all of the disk operations by typing simple commands on the keyboard, using the suggested mnemonics given in parentheses in the preceding descriptions. For example, the command "LST 1" will list the directory on drive 1 and "CRT FILE1 1 10" will create FILE1 on drive 1 with 10 sectors. In addition, several useful utility operations are provided:

DMP I J means print the contents of memory locations I through J, starting a new line after each location that is evenly divisible by 10.

NTR I... means accept values from the keyboard and enter them into successive memory locations starting at location I. Each value is separated from the next by a space, and you can enter as many values as will fit on a single line.

BLK I J K means fill memory locations from I to J with the value K. This is useful for initializing a memory buffer prior to reading in a new block of data or creating a disk file filled with a single value.

XIT I means exit from this program and continue at location I. For example, XIT 0 will restart BASEX if it is resident, or you may wish to exit to your monitor.

HLP means print the list of options.

Before running this or any other BASEX program which uses the DSK command, it is necessary to alter several locations in the BASEX compiler and execution routine, as follows:

1. The US1 command in the BASEX compiler must be changed to DSK. To do this from BASEX, type "NTR 8279 68 83 75".
2. The jump for the US1 command in the execution routines must be changed to point to the disk handler. This may be done from BASEX by typing "NTR 2136 1 42."
3. If you wish to use the DOS I/O routines for terminal input and output, you may enter the appropriate changes as described on pages 22 and 23 of the BASEX manual. You may also change MNTR at location 6858 in the compiler symbol table to point to the DOS entry, 8232 decimal.
4. Now save this version of BASEX on your disk.

```
          SIZ
PROGRAM 12288 14126    SYMBOLS 14200 15559
14126 ? LST
12288 DIM $ 1
12296 STR $ 1 1 13
12308 PRT "BASEXERCISER FOR NORTH STAR FLOPPY" $ 1
12318 PRT "****** COPYRIGHT (C) 1978 BY PAUL K. WARME ******" $ 1
12328 *** HELP
12334 PRT "OPERATION CODES:" $ 1
12344 PRT "WRT # B    *WRITE" "ONE SECTOR," "BUFFER B," "FILE #" $ 1
12360 PRT "RED # B    *READ" "ONE SECTOR," "BUFFER B," "FILE #" $ 1
12376 PRT "VFY # B    *VERIFY" "ONE SECTOR," "BUFFER B," "FILE #" $ 1
12392 PRT "SFD # B    *SAVE" "ALL SECTORS," "BUFFER B," "FILE #" $ 1
12408 PRT "LFD # B    *LOAD" "ALL SECTORS," "BUFFER B," "FILE #" $ 1
12424 PRT "CKD # B    *CHECK" "ALL SECTORS," "BUFFER B," "FILE #" $ 1
12440 PRT "SEK # S    *SEEK SECTOR S," "FILE #" $ 1
12452 PRT "LST U      *LIST DIRECTORY," "UNIT U" $ 1
12464 PRT "CHN $ U    *CHAIN" "FILENAME $," "UNIT U" $ 1
12478 PRT "DLD $ U    *DELETE" "FILENAME $," "UNIT U" $ 1
12492 PRT "CRT $ U N *CREATE" "FILENAME $," "UNIT U" "N SECTORS" $ 1
12508 PRT "OPN $ U # *OPEN" "FILENAME $," "UNIT U" "FILE #" $ 1
12524 PRT "DMP I J    *DUMP" "MEMORY FROM I TO J" $ 1
12536 PRT "NTR I...   *ENTER DATA STARTING AT I" $ 1
12546 PRT "BLK I J K *BLOCK FILL" "MEMORY FROM I TO J" "WITH K" $ 1
12560 PRT "XIT I      *EXIT TO I" $ 1
12570 PRT "HLP        *PRINT" "OPERATION CODES:" $ 1
12582 DIM IN$ 72 OP$ 60 OPR$ 3 NAME$ 8
12602 SET EOF=10752              Error indicator address
12610 *** START                  All commands return here
12616 CHR 13                     Carriage return/linefeed
12622 PRT "OPERATION"            Input command line
12628 INP IN$ 1
12636 STR OPR$ 1 3 IN$ 1         First 3 char are operation
12650 STR IN$ 1 68 IN$ 5         Shift line buffer
12664 FOR I=1                    Search for operation name
12672 CMP OPR$ 1 3 OP$ I
12686 JMP EQ DSPCH               Found it
12694 TIL I+3 51
12706 PRT "OPERATION"            Undefined operation
12712 *** ERR
12718 PRT "ERROR"
12724 GTO START
12730 *** DSPCH                  Dispatch to disk routines
12736 DIV I/3                    OP=(position in OP$)/3
12744 SET OP=A
12752 SBT OP-6
12760 JMP LT X05                 OP=0 to 5
12768 JMP EQ SEK                 OP=6
12776 DEC A
12782 JMP EQ LST                 OP=7
12790 SBT OP-12
12798 JMP LT X811                OP=8 to 11
12806 JMP EQ DMP                 OP=12
12814 SBT OP-14
12822 JMP LT NTR                 OP=13
12830 JMP EQ BLK                 OP=14
```

```
12838 SBT OP-16
12846 JMP LT EXIT                OP=15
12854 JMP EQ HELP                OP=16
12862 *** ARGERR
12868 PRT "ARGUMENT"
12874 GTO ERR
12880 REM !GET V=NEXT VALUE IN IN$!
12886 *** ARG
12892 LEN IN$ 1                  Get command length
12900 SBT A-0                    Null?
12908 JMP LE ARGERR
12916 SET N=A                    N=length
12924 FOR J=1                    Find 1st non-numeral
12932 STR C IN$ J
12942 SBT C-47                   <0?
12950 JMP LE ARGEND
12958 SBT C-58                   >9?
12966 JMP GE ARGEND
12974 TIL J+1 N
12986 *** ARGEND                 Delimiter found
12992 SBT J-1                    Double delimiter?
13000 JMP EQ ARGERR
13008 VAL IN$ 1                  Ascii to binary
13016 SET V=A                    V=value
13024 SBT J-N                    Last argument?
13032 JMP GT JOK                 Yes
13040 INC J                      No; skip delimiter
13046 *** JOK                    Shift out argument
13052 STR IN$ 1 N IN$ J
13066 RET
13070 REM !WRT/RED/VFY/SAV/LOD/CHK!
13076 *** X05                    OP=0 to 5
13082 CAL ARG                    Get file #
13088 SET FNUM=V
13096 CAL ARG                    Get buffer location
13102 SET BUF=V
13110 DSK OP FNUM BUF            Execute OP=0 to 5
13120 SBT OP-2                   SAV or LOD?
13128 JMP GT SLC                 Yes
13136 REM !WRT/RED/VFY/SEK ERRORS!
13142 *** ERTST
13148 WRD EOF                    Get error indicator
13154 SBT A-0                    OK?
13162 JMP EQ START               Yes
13170 JMP LT DKERR               Disk error if <0
13178 PRT "END OF FILE"
13184 GTO ERR
13190 REM !SAV/LOD/CHK ERRORS!
13196 *** SLC
13202 WRD EOF                    Get error indicator
13208 SBT A-1                    Should be end of file
13216 JMP EQ START               It is
13224 *** DKERR
13230 PRT "DISK"
13236 GTO ERR
13242 *** SEK                    Seek given sector
```

```
13248 CAL ARG                          Get file #
13254 SET FNUM=V
13262 CAL ARG                          Get sector #
13268 SET REC=V
13276 DSK OP FNUM REC                  Execute OP=6
13286 GTO ERTST
13292 *** LST                          List directory
13298 CAL ARG                          Get unit #
13304 SET UNIT=V
13312 DSK OP UNIT                      Execute OP=7
13320 GTO START
13326 REM !CHN/DEL/CRT/OPN!
13332 *** X811                         Operations 8 to 11
13338 LEN IN$ 1
13346 SET N=A                          N=line length
13354 FOR J=1                          Extract file name
13362 STR C IN$ J                      C=next char
13372 REM !FIRST CHAR OF NAME CAN'T BE A NUMBER!
13378 SBT J-1
13386 JMP EQ LETR
13394 SBT C-48                         Is it a number?
13402 JMP LT NAMEND                    No
13410 SBT C-58
13418 JMP LT COK                       Yes
13426 *** LETR
13432 SBT C-65                         Is it a letter?
13440 JMP LT NAMEND                    No
13448 SBT C-91
13456 JMP GE NAMEND                    Must be a delimiter
13464 *** COK
13470 TIL J+1 N
13482 GTO ARGERR                       No delimiter
13488 *** NAMEND                       End of name
13494 DEC J
13500 JMP EQ ARGERR                    No char in name
13508 STR NAME$ 1 J IN$ 1              Transfer to NAME$
13522 ADD J+2                          Skip delimiter
13530 STR IN$ 1 N IN$ A                Shift out name
13544 CAL ARG                          Get unit#
13550 SET UNIT=V
13558 SBT OP-10
13566 JMP EQ CRT                       OP=10
13574 JMP GT OPN                       OP=11
13582 REM !CHN/DEL!
13588 DSK OP NAME$ UNIT                Execute OP=8 or 9
13598 WRD EOF                          Get error indicator
13604 SBT A-0
13612 JMP GT START                     OK
13620 *** NAMTST
13626 JMP LT NAMERR
13634 PRT "ABSENT"                     EOF=0
13640 *** NAMERR
13646 PRT "FILE NAME"                  EOF=-1 or -2
13652 GTO ERR
13658 *** CRT                          Create a new file
13664 CAL ARG                          Get # sectors
```

```
13670 SET NBLK=V
13678 DSK OP NAME$ UNIT NBLK     Execute OP=10
13690 WRD EOF                    Check errors
13696 SBT A-0
13704 JMP GT BLKSIZ              Normal return
13712 JMP EQ DKERR
13720 SBT A--1
13728 JMP LT DUP
13736 PRT "OVERFLOW"             EOF=-1
13742 GTO DKERR
13748 *** DUP
13754 PRT "DUPLICATE"            EOF=-2
13760 GTO NAMERR
13766 *** OPN                    Open an old file
13772 CAL ARG                    Get file#
13778 SET FNUM=V
13786 DSK OP NAME$ UNIT FNUM     Execute OP=11
13798 WRD EOF                    Check errors
13804 SBT A-0                    Normally, EOF=# sectors
13812 JMP LE NAMTST              Error
13820 *** BLKSIZ
13826 PRT NAME$ 1 A "SECTRS"
13838 GTO START
13844 *** DMP                    Dump memory contents
13850 CAL ARG                    Get low range
13856 SET I=V
13864 CAL ARG                    Get high range
13870 FOR I=I
13878 BRD I                      Read a byte
13884 CHR 32                     Space
13890 PRT A                      Print a byte
13896 DIV I/10                   CRLF every 10th value
13904 MLT A*10
13912 SBT I-A
13920 JMP NE DM1
13928 CHR 13
13934 *** DM1
13940 TIL I+1 V
13952 GTO START
13958 *** NTR                    Enter memory values
13964 CAL ARG                    Get starting location
13970 SET I=V
13978 *** NT1
13984 LEN IN$ 1                  Line length
13992 SBT A-0
14000 JMP EQ START               Done
14008 CAL ARG                    Get a value
14014 BRT I=V                    Store it
14022 INC I                      Bump memory ptr
14028 GTO NT1                    Loop
14034 *** BLK                    Fill memory with value
14040 CAL ARG                    Get low range
14046 SET I=V
14054 CAL ARG                    Get high range
14060 SET OP=V
14068 CAL ARG                    Get fill value
```

```
14074 FOR I=I
 14082 BRT I=V                Fill next byte
 14090 TIL I+1 OP
 14102 GTO START
 14108 *** EXIT               Exit to given location
 14114 CAL ARG                Get location
 14120 GTO V                  Go there
 14126 END 14126 ? SIZ
 PROGRAM 12288 14126    SYMBOLS 14200 15559
 14126 ? LSM
 14200 HLP       *PRINT" 14200
 14220 XIT I     *EXIT TO I" 14220
 14244 HELP 12334
 14251 EXIT 14114
 14258 OPERATION CODES:" 14258
 14278 ****** COPYRIGHT (C) 1978 BY PAUL K. WARME ******" 14278
 14331 BASEXERCISER FOR NORTH STAR FLOPPY" 14331
 14369 WITH K" 14369
 14379 BLK I J K *BLOCK FILL" 14379
 14404 NTR I...  *ENTER DATA STARTING AT I" 14404
 14443 MEMORY FROM I TO J" 14443
 14465 DMP I J   *DUMP" 14465
 14484 OPN $ U # *OPEN" 14484
 14503 N SECTORS" 14503
 14516 CRT $ U N *CREATE" 14516
 14537 DLD $ U   *DELETE" 14537
 14558 FILENAME $," 14558
 14573 CHN $ U   *CHAIN" 14573
 14593 UNIT U" 14593
 14603 LST U     *LIST DIRECTORY," 14603
 14633 SEK # S   *SEEK SECTOR S," 14633
 14662 CKD # B   *CHECK" 14662
 14682 LFD # B   *LOAD" 14682
 14701 ALL SECTORS," 14701
 14717 SFD # B   *SAVE" 14717
 14736 VFY # B   *VERIFY" 14736
 14757 RED # B   *READ" 14757
 14776 FILE #" 14776
 14786 BUFFER B," 14786
 14799 ONE SECTOR," 14799
 14814 WRT # B   *WRITE" 14814
 14834 $ 1

 14838 NT1 13984
 14844 DM1 13940
 14850 SECTRS" 14850
 14860 DUPLICATE" 14860
 14873 OVERFLOW" 14873
 14885 DUP 13754
 14891 BLKSIZ 13826
 14900 NBLK 0
 14907 FILE NAME" 14907
 14920 ABSENT" 14920
 14930 NAMERR 13646
 14939 NAMTST 13626
 14948 CHN/DEL! 14948
```

```
14959 OPN 13772
14965 CRT 13664
14971 COK 13470
14977 NAMEND 13494
14986 LETR 13432
14993 FIRST CHAR OF NAME CAN'T BE A NUMBER! 14993
15033 CHN/DEL/CRT/OPN! 15033
15052 UNIT 1
15059 REC 0
15065 SAV/LOD/CHK ERRORS! 15065
15087 DISK" 15087
15095 END OF FILE" 15095
15110 DKERR 13230
15118 ERTST 13148
15126 WRT/RED/VFY/SEK ERRORS! 15126
15152 SLC 13202
15158 BUF 0
15164 FNUM 0
15171 WRT/RED/VFY/SAV/LOD/CHK! 15171
15198 JOK 13052
15204 V 1
15208 ARGEND 12992
15217 C 49
15221 J 2
15225 N 1
15229 ARG 12892
15235 GET V=NEXT VALUE IN IN$! 15235
15262 ARGUMENT" 15262
15274 ARGERR 12868
15283 BLK 14040
15289 NTR 13964
15295 DMP 13850
15301 X811 13338
15308 LST 13298
15314 SEK 13248
15320 X05 13082
15326 OP 7
15331 ERROR" 15331
15340 ERR 12718
15346 DSPCH 12736
15354 I 22
15358 OPERATION" 15358
15371 START 12616
15379 10752 10752
15387 EOF 10752
15393 NAME$ 8
15408 OPR$ 3 LST
15417 OP$ 60 WRTREDVFYSFDLFDCKDSEKLSTCHNDLDCRTOPNDMPNTRBLKXITHLP
15482 IN$ 72
```

II. THE BASEX/MECA TAPE HANDLER

A. Descriptions of Tape Operations

Each of the following tape operations can be invoked in a BASEX program by the TAP command, followed by the operation (e.g., Ø=list directory, 3=save file) and a variable number of arguments, as described below. The operation number may be either a constant or a regular variable. Suggested mnemonics for each operation are given in parentheses following the operation number below; however, these names must be defined somewhere in the BASEX program. The regular variables UNIT, START and END used in the examples may be replaced by constants, as well. The string variable NAME$, used to denote the file name, does not require the character range specification normally used with string variables in BASEX programs. Note that the tape operating system recognizes only the first five letters of a file name. If an error is detected by the tape operating system, it types out the usual error messages (see Meca manual), and waits for you to type a carriage return before the tape system takes control and types "EH?". However, this should not happen if you use the tape operations correctly. If this does occur, you should type "EXEC Ø" to restart BASEX.

Operation Ø(DIR): List tape DIRectory
Example: TAP DIR UNIT

The directory operation will list the tape directory on the specified tape unit. The drive number and the end of tape location are typed on the first line, followed by the name, length in bytes and tape location of each file on the tape. If an escape (^Z) is typed during the listing, the listing is aborted and the tape operating system takes over.

Operation 1(REW): REWind tape
Example: TAP REW UNIT

This operation rewinds the tape on the requested UNIT.

Operation 2(LOD): LOaD tape file
Example: TAP LOD NAME$ UNIT START

The load operation loads file NAME$ on UNIT at the location specified by START. You will notice that tape operations do not permit loading segments of the file, but only complete files. However, you can simulate random access to records within a file by creating a set of files which together constitute the larger file you wish to access randomly. For example, you could create files named DATA1, DATA2,...., DATA9, each of length 256 bytes, and then define NAME$ 1 5="DATA1". Now, in your BASEX program, you can call in any "record" of file DATA by altering the last character of NAME$ (e.g., STR NAME$ 5 5="3" will generate DATA3 in NAME$). In this way, you can reduce the buffer space required for very large random access files to a manageable size. If you require more than 5Ø files on a tape, the tape operating system must be altered accordingly.

Operation 3(SAV): SAVe tape file
Example: TAP SAV NAME$ UNIT START END

The save operation saves the data or program residing between locations START and END as file NAME$ on tape drive UNIT. If a file by that name is already on the tape, an error message will be typed and no action will be taken. Otherwise, the new file will be written at the end of the tape and its accuracy will automatically be verified. In addition, an updated tape directory will be written on the tape. The remarks about "random access" files described under the load operation apply here, as well. See also operation 14 for overlaying an old file.

Operation 4(MNT): MouNT tape
Example: TAP MNT UNIT

This operation causes the tape on drive UNIT to be rewound and its directory to be read into memory, overlaying the directory of any tape previously mounted on that drive.

Operation 5(ULD): UnLoaD tape
Example: TAP ULD UNIT

The unload operation writes out an updated directory on the tape mounted on drive UNIT, if necessary, and rewinds the tape.

Operation 6(NEW): NEW tape directory
Example: TAP NEW UNIT

The new operation will zero the directory of the tape on drive UNIT and initialize the tape for subsequent read or write operations.

Operation 7(DLT): DeLeTe Tape file
Example: TAP DLT NAME$ UNIT

This operation will delete file NAME$ on drive UNIT. The tape region previously occupied by that file remains unaltered, but cannot be accessed again until the tape is compressed or the Meca directory reconstruction program is run.

Operation 8(RUN): RUN an executable program
Example: TAP RUN NAME$ UNIT

The run operation loads file NAME$ on drive UNIT at the location from which it was originally saved and transfers control to the first byte of that program. You will note that this program does not necessarily overlay the current BASEX program, symbol table or data arrays, thus permitting shared symbol tables or shared data among multiple BASEX programs.

Operations 9-13,15(ERR): Undefined

Any of these operation numbers, as well as any numbers greater than 16, will be recognized as errors, a BASEX error dump will be printed and BASEX will be restarted.

Operation 14(OVR): OVeRlay tape file
Example: TAP OVR NAME$ UNIT START END

The overlay operation will overlay a previously created file

NAME$ on drive UNIT with the program or data stored between memory locations START and END. If the file will not fit within the tape domain allocated for that file, it will be written at the end of the tape. The overlay operation updates the memory copy of the tape directory, but does not write a new directory on the tape. If you attempt to overlay a file which does not exist on the tape, the routine exits to the tape operating system and types "EH?". The remarks about "random access" tape files given under the load operation above apply here, as well.

Operation 16(SIZ): Determine SIZe of tape file
Example: TAP SIZ NAME$ UNIT BYTES

If you wish to determine the size of a tape file before loading it, you should use the size operation. This may be desireable, for instance, if you have a limited buffer area and want to certify that a file will not exceed the buffer size. The length of the file, in bytes, is returned in variable BYTES. If a length of 0 is returned, the file does not exist. Thus, the size command is useful for determining whether a file is already on a tape, so that you can overlay instead of saving it.

B. Assembler Source Listing of Meca Tape Handler

This version of the tape handler is compiled at location 2C00 so that it immediately follows the BASEX/North Star disk handler. However, if you do not use this disk system and your tape operating system is located in high memory, you may wish to relocate the tape handler. In order to do this, the value of X (now 7000) which defines the origin of the tape operating system should be adjusted, and this will redefine all entries to the Meca tape operating system, version 3. Then, you may recompile the tape handler anywhere you wish.

The following locations must be changed if your tape operating system is not located at 7000H. Change the X to the first digit of its origin.

2C19=X4, 2C13=XF, 2C80=XF, 2C83=XF,

2C86=X3, 2CDA=XF, 2CDD=X6, 2CE0=X9.

```
ASSM 2C00

2C00                0100 *** BASEX HANDLER FOR MECA TAPE DRIVES ***
2C00                0110 *** COPYRIGHT 1978 BY PAUL K. WARME ***
2C00                0120 GETARG EQU   26FH     *BASEX ENTRY POINTS
2C00                0130 PUTARG EQU   2DCH
2C00                0140 RETRN  EQU   329H
2C00                0150 ERROR  EQU   332H
2C00                0160 X      EQU   7000H    *MECA OS ORIGIN
2C00                0180 FNAME  EQU   X+3A8H   *OS ENTRY POINTS
2C00                0200 DRID   EQU   X+3BEH
2C00                0210 RTRY   EQU   X+3EAH
2C00                0220 CTAB   EQU   X+4B0H
2C00                0230 RD0    EQU   X+62DH
2C00                0240 FSDIR  EQU   X+96AH
2C00                0250 BUFR   EQU   X+0F9BH
2C00 E1             0260 TAPE   POP   H          *HL=PROGRAM PTR (PP)
2C01 CD 6F 02       0270        CALL  GETARG     *BC=OPERATION # (TOP)
2C04 E5             0280        PUSH  H          *SAVE PP
2C05 79             0290        MOV   A,C
2C06 32 0A 2D       0300        STA   MODE
2C09 FE 10          0310        CPI   16
2C0B DA 17 2C       0320        JC    CMND       *TOP=0 TO 15
2C0E C2 32 03       0330        JNZ   ERROR      *BAD OPERATION
2C11 11 9B 7F       0340        LXI   D,BUFR     *TOP=16
2C14 C3 28 2C       0350        JMP   SIZ1
2C17 21 B3 74       0360 CMND   LXI   H,CTAB+3 *OS MNEMONIC TABLE
2C1A 09             0370        DAD   B          *OFFSET=TOP*4
2C1B 09             0380        DAD   B
2C1C 09             0390        DAD   B
2C1D 09             0400        DAD   B          *HL=(OPERATION MNEMONIC)
2C1E 11 9B 7F       0410        LXI   D,BUFR
2C21 7E             0420        MOV   A,M        *MOVE MNEMONIC TO BUFR
2C22 23             0430        INX   H
2C23 12             0440        STAX  D
2C24 13             0450        INX   D
2C25 7E             0460        MOV   A,M
2C26 12             0470        STAX  D
2C27 13             0480        INX   D
2C28 3E 20          0490 SIZ1   MVI   A,32       *INSERT 3 SPACES
2C2A 12             0492        STAX  D
2C2B 13             0494        INX   D
2C2C 12             0496        STAX  D
2C2D 13             0498        INX   D
2C2E 12             0500        STAX  D
2C2F 13             0510        INX   D
2C30 D5             0520        PUSH  D          *STACK BUFPTR
2C31 21 3D 2C       0530        LXI   H,DSPCH    *GET (EXECUTION ROUTINE)
2C34 09             0540        DAD   B          *OFFSET=TOP*2
2C35 09             0550        DAD   B
2C36 5E             0560        MOV   E,M        *DE=(EXECUTION ROUTINE)
2C37 23             0570        INX   H
2C38 56             0580        MOV   D,M
2C39 EB             0590        XCHG
2C3A 3E 00          0600        MVI   A,0        *DEFAULT ZERO ARGS
2C3C E9             0610        PCHL             *GO DO IT
```

```
2C3D 5F 2C      0620 DSPCH  DW    UNIT1     *0=DIR UNIT
2C3F 5F 2C      0630        DW    UNIT1     *1=REW UNIT
2C41 8F 2C      0640        DW    ARG2      *2=LOD NAME$ UNIT BUFLO
2C43 8E 2C      0650        DW    ARG3      *3=SAV NAME$ UNIT BUFLO BUFHI
2C45 5F 2C      0660        DW    UNIT1     *4=MNT UNIT
2C47 5F 2C      0670        DW    UNIT1     *5=ULD UNIT
2C49 5F 2C      0680        DW    UNIT1     *6=NEW UNIT
2C4B 90 2C      0690        DW    ARG1      *7=DLT NAME$ UNIT
2C4D 90 2C      0700        DW    ARG1      *8=RUN NAME$ UNIT
2C4F 32 03      0710        DW    ERROR     *UNDEFINED OPERATIONS
2C51 32 03      0720        DW    ERROR
2C53 32 03      0730        DW    ERROR
2C55 32 03      0740        DW    ERROR
2C57 32 03      0750        DW    ERROR
2C59 8E 2C      0760        DW    ARG3      *14=OVR NAME$ UNIT BUFLO BUFHI
2C5B 32 03      0770        DW    ERROR
2C5D 90 2C      0780        DW    ARG1      *16=SIZ NAME$ UNIT
2C5F E1         0790 UNIT1  POP   H         *ONLY 1 ARG
2C60 E3         0800        XTHL            *STACK BUFPTR,HL=PP
2C61 CD 6F 02   0810        CALL  GETARG    *BC=UNIT#
2C64 E3         0820        XTHL            *STACK PP,HL=BUFPTR
2C65 36 3A      0830        MVI   M,':'     *UNIT DELIMITER
2C67 23         0840        INX   H
2C68 3E 30      0850        MVI   A,'0'     *CONVERT UNIT# TO ASCII
2C6A 81         0860        ADD   C
2C6B 77         0870        MOV   M,A
2C6C 23         0880        INX   H
2C6D 36 0D      0890 EXEC   MVI   M,13      *PUT CR AFTER COMMAND
2C6F 3A 0A 2D   0900        LDA   MODE      *TOP
2C72 FE 10      0910        CPI   16
2C74 CA D8 2C   0920        JZ    SIZ2      *SPECIAL CASE FOR SIZ
2C77 21 00 00   0930        LXI   H,0       *SAVE PROGRAM STACK
2C7A 39         0940        DAD   6
2C7B 22 09 2D   0950        SHLD  NARG
2C7E 31 EB 7F   0960        LXI   6,BUFR+80 *USE MECA STACK
2C81 21 9B 7F   0970        LXI   H,BUFR    *COMMAND BUFFER
2C84 CD DB 73   0980        CALL  RTRY-15   *OS ENTRY
2C87 2A 09 2D   0990        LHLD  NARG      *RESTORE PROGRAM STACK
2C8A F9         1000        SPHL
2C8B C3 29 03   1010        JMP   RETRN     *RETURN TO BASEX PROGRAM
2CE 3C          1030 ARG3   INR   A         *3 ARGS
2C8F 3C         1040 ARG2   INR   A         *2 ARGS
2C90 32 09 2D   1050 ARG1   STA   NARG      *# ARGS - 1
2C93 D1         1060        POP   D         *MOVE NAME$ TO BUFR
2C94 E1         1070        POP   H         *DE=BUFPTR,HL=PP
2C95 4E         1080        MOV   C,M       *GET (NAME$)
2C96 23         1090        INX   H
2C97 46         1100        MOV   B,M
2C98 23         1110        INX   H
2C99 03         1120        INX   B         *SKIP DIMENSION BYTE
2C9A 0A         1130 TRANS  LDAX  B         *GET SOURCE CHAR
2C9B B7         1140        ORA   A         *0 MEANS END OF STRING
2C9C CA A5 2C   1150        JZ    DONE
2C9F 03         1160        INX   B         *BUMP SOURCE PTR
2CA0 12         1170        STAX  D         *STORE CHAR
2CA1 13         1180        INX   D         *BUMP DESTINATION PTR
```

```
2CA2 C3 9A 2C   1190         JMP   TRANS
2CA5 3E 20      1200 DONE    MVI   A,32      *SPACE
2CA7 12         1210         STAX  D
2CA8 13         1220         INX   D
2CA9 D5         1230         PUSH  D         *STACK BUFPTR
2CAA CD 6F 02   1240         CALL  GETARG    *BC=UNIT#
2CAD E3         1250         XTHL            *STACK PP,HL=BUFPTR
2CAE 36 3A      1260         MVI   M,':'     *UNIT DELIMITER
2CB0 23         1270         INX   H
2CB1 3E 30      1280         MVI   A,'0'     *CONVERT UNIT# TO ASCII
2CB3 81         1290         ADD   C
2CB4 77         1300         MOV   M,A
2CB5 23         1310         INX   H
2CB6 3A 09 2D   1320         LDA   NARG      *MORE ARGS?
2CB9 3D         1330         DCR   A
2CBA FA 6D 2C   1340         JM    EXEC      *NO
2CBD 36 20      1350         MVI   M,32      *SPACE
2CBF 23         1360         INX   H
2CC0 E3         1370 NXTARG  XTHL            *STACK BUFPTR,HL=PP
2CC1 CD 6F 02   1380         CALL  GETARG    *BC=NEXT ARG
2CC4 E3         1390         XTHL
2CC5 CD EE 2C   1400         CALL  HEX       *CONVERT TO HEX
2CC8 3A 09 2D   1410         LDA   NARG      *MORE ARGS?
2CCB 3D         1420         DCR   A
2CCC CA 6D 2C   1430         JZ    EXEC      *NO
2CCF 32 09 2D   1440         STA   NARG
2CD2 36 20      1450         MVI   M,32      *SPACE
2CD4 23         1460         INX   H
2CD5 C3 C0 2C   1470         JMP   NXTARG
2CD8 11 9B 7F   1480 SIZ2    LXI   D,BUFR    *CHECK FILE LENGTH
2CDB CD 2D 76   1490         CALL  RD0       *OS ENTRY
2CDE CD 6A 79   1500         CALL  FSDIR     *OS ENTRY
2CE1 CAE7 2C    1510         JZ    LENOK
2CE4 01 00 00   1520         LXI   B,0       *LENGTH=0 IF BAD NAME$
2CE7 E1         1530 LENOK   POP   H         *HL=PP
2CE8 CD DC 02   1540         CALL  PUTARG    *STORE LENGTH
2CEB C3 2A 03   1550         JMP   RETRN+1   *RETURN TO BASEX PROGRAM
2CEE CD F2 2C   1560 HEX     CALL  HEX2      *CONVERT BINARY TO HEX
2CF1 41         1570         MOV   B,C       *LOW ORDER BYTE
2CF2 78         1580 HEX2    MOV   A,B       *CONVERT B TO HEX
2CF3 0F         1590         RRC
2CF4 0F         1600         RRC
2CF5 0F         1610         RRC
2CF6 0F         1620         RRC
2CF7 CD FB 2C   1630         CALL  HEX1
2CFA 78         1640         MOV   A,B       *SECOND DIGIT
2CFB E6 0F      1650 HEX1    ANI   15
2CFD FE 0A      1660         CPI   10
2CFF FA 04 2D   1670         JM    ANUM      *<10
2D02 C6 07      1680         ADI   7         *>10
2D04 C6 30      1690 ANUM    ADI   '0'       *ASCII OFFSET
2D06 77         1700         MOV   M,A       *STORE DIGIT IN BUFR
2D07 23         1710         INX   H
2D08 C9         1720         RET
2D09 00         1730 NARG    DB    0
2D0A 00         1740 MODE    DB    0
```

C. BASEXerciser for Meca Tape Drives

This exerciser program permits you to execute any of the tape operations described above by typing commands on your keyboard. The mnemonics for tape operations suggested in parentheses under each description are used. For example, the command "DIR Ø" will list the directory of the tape on drive Ø and the command "SAV FILE1 Ø 10000 12000" will save memory from 10000 to 12000 (decimal) as FILE1 on drive Ø. In addition, the utility routines (DMP, NTR, BLK, XIT and HLP) described under section I.C are also available here.

Before running this or any other BASEX program which uses the TAP operation, you must alter several locations in the BASEX compiler and execution routines, as follows:

1. The US2 command in the BASEX compiler must be changed to TAP. To do this from BASEX, type "NTR 8282 84 65 80".
2. The jump for the US2 command in the execution routines must be changed to point to the tape handler. This may be done (from BASEX) by typing "NTR 2139 ØØ 44". If you have relocated the tape handler, of course, this address must be adjusted to point to your relocated handler.
3. You may wish to change the definition of MNTR (location 6858 in the compiler symbol table) to point to the origin of your tape operating system+390 hex, so that typing "MON" in BASEX will exit to there.
4. Now save this version of BASEX on your tape.

```
        SIZ
PROGRAM 12288 13832    SYMBOLS 14000 15090
13832 ? LST
12288 DIM $ 1
12296 STR $ 1 1 13
12308 PRT "BASEXERCISER FOR MECA TAPE" $ 1
12318 PRT "**** COPYRIGHT (C) 1978 BY PAUL K. WARME ****" $ 1
12328 *** HELP
12334 PRT "OPERATION CODES:" $ 1
12344 PRT "DIR U       *LIST DIRECTORY," "UNIT U" $ 1
12356 PRT "REW U       *REWIND" "UNIT U" $ 1
12368 PRT "LOD $ U I   *TAPE LOAD" "FILENAME $," "UNIT U" "AT I" $ 1
12384 PRT "SAV $ U I J *TAPE SAVE" "FILENAME $," "UNIT U"
12394 PRT "FROM I TO J" $ 1
12404 PRT "MNT U       *MOUNT TAPE," "UNIT U" $ 1
12416 PRT "ULD U       *UNLOAD TAPE" "UNIT U" $ 1
12428 PRT "NEW U       *NEWTAPE ON" "UNIT U" $ 1
12440 PRT "DLT $ U     *DELETE" "FILENAME $," "UNIT U" $ 1
12454 PRT "RUN $ U     *EXECUTE" "FILENAME $," "UNIT U" $ 1
12468 PRT "OVR $ U I J *TAPE OVERLAY" "FILENAME $," "UNIT U"
12478 PRT "FROM I TO J" $ 1
12488 PRT "SIZ $ U     *PRINT SIZE OF" "FILENAME $," "UNIT U" $ 1
12502 PRT "DMP I J     *DUMP" "MEMORY" "FROM I TO J" $ 1
12516 PRT "NTR I...    *ENTER DATA..." "AT I" $ 1
12528 PRT "BLK I J K   *BLOCK FILL" "FROM I TO J" "WITH K" $ 1
12542 PRT "XIT I       *EXIT TO I" $ 1
12552 PRT "HLP         *PRINT" "OPERATION CODES:" $ 1
12564 DIM IN$ 72 TOP$ 60 OPR$ 3 NAME$ 8
12584 *** START                    All commands return here
12590 CHR 13                       Carriage return/linefeed
12596 PRT "OPERATION"              Input command line
12602 INP IN$ 1
12610 STR OPR$ 1 3 IN$ 1           First 3 char are operation
12624 STR IN$ 1 68 IN$ 5           Shift line buffer
12638 FOR I=1                      Search for operation name
12646 CMP OPR$ 1 3 TOP$ I
12660 JMP EQ TSPCH                 Found it
12668 TIL I+3 57
12680 PRT "OPERATION"              Undefined operation
12686 *** ERR
12692 PRT "ERROR"
12698 GTO START
12704 *** TSPCH                    Dispatch to tape routines
12710 DIV I/3                      TOP=(position in TOP$)/3
12718 SET TOP=A
12726 SET NARG=1                   Initialize to 1 arg
12734 SBT TOP-2
12742 JMP EQ ARG2                  TOP=2
12750 SBT TOP-3
12758 JMP EQ ARG3                  TOP=3
12766 SBT TOP-7
12774 JMP LT ARG0                  TOP=0,1,4-6
12782 SBT TOP-9
12790 JMP LT ARG1                  TOP=7,8
12798 JMP EQ DMP                   TOP=9
12806 SBT TOP-11
```

```
12814 JMP LT NTR                TOP=10
12822 JMP EQ BLK                TOP=11
12830 SBT TOP-16
12838 JMP LT ARG3               TOP=14
12846 JMP EQ ARG1               TOP=16
12854 DEC A
12860 JMP EQ EXIT               TOP=17
12868 GTO HELP                  TOP=18
12874 *** ARG3                  4 args
12880 INC NARG
12886 *** ARG2                  3 args
12892 INC NARG
12898 *** ARG1                  2 args
12904 INC NARG
12910 LEN IN$ 1                 Get command length
12918 SET N=A
12926 FOR J=1                   Extract file name
12934 STR C IN$ J               C=next char
12944 REM !FIRST CHAR OF NAME CAN'T BE A NUMBER!
12950 SBT J-1
12958 JMP EQ LETR
12966 SBT C-48                  Is it a number?
12974 JMP LT NAMEND             No
12982 SBT C-58
12990 JMP LT COK                Yes
12998 *** LETR
13004 SBT C-65                  Is it a letter?
13012 JMP LT NAMEND             No
13020 SBT C-91
13028 JMP GE NAMEND             Must be a delimiter
13036 *** COK
13042 TIL J+1 N
13054 GTO ARGERR                No delimiter
13060 *** NAMEND                End of name
13066 DEC J
13072 JMP EQ ARGERR             No char in name
13080 STR NAME$ 1 J IN$ 1       Transfer to NAME$
13094 ADD J+2                   Skip delimiter
13102 STR IN$ 1 N IN$ A         Shift out name
13116 *** ARG0                  Only 1 arg
13122 CAL ARG                   Get unit #
13128 SET UNIT=V
13136 SBT NARG-1                More args?
13144 JMP GT AR1                Yes
13152 TAP TOP UNIT              Execute command
13160 GTO START
13166 *** AR1
13172 SBT NARG-2                More args?
13180 JMP GT AR2                Yes
13188 SBT TOP-16                SIZ is special case
13196 JMP EQ SIZ
13204 TAP TOP NAME$ UNIT        Execute command
13214 GTO START
13220 *** SIZ                   Get file length
13226 TAP TOP NAME$ UNIT R2
13238 PRT NAME$ 1 R2 "BYTES"
```

```
13250 GTO START
13256 *** AR2                    3 or more args
13262 CAL ARG                    Get next arg
13268 SET R2=V
13276 SBT NARG-3                 More args?
13284 JMP GT AR3                 Yes
13292 TAP TOP NAME$ UNIT R2      Execute command
13304 GTO START
13310 *** AR3                    4 args
13316 CAL ARG                    Get last arg
13322 TAP TOP NAME$ UNIT R2 V    Execute command
13336 GTO START
13342 REM !GET V=NEXT VALUE IN IN$!
13348 *** ARG
13354 LEN IN$ 1                  Get command length
13362 SBT A-0                    Null?
13370 JMP LE ARGERR
13378 SET N=A                    N=length
13386 FOR J=1                    Find 1st non-numeral
13394 STR C IN$ J
13404 SBT C-47                   <0?
13412 JMP LE ARGEND
13420 SBT C-58                   >9?
13428 JMP GE ARGEND
13436 TIL J+1 N
13448 *** ARGEND                 Delimiter found
13454 SBT J-1                    Double delimiter?
13462 JMP EQ ARGERR
13470 VAL IN$ 1                  Ascii to binary
13478 SET V=A                    V=value
13486 SBT J-N                    Last arg?
13494 JMP GT JOK                 Yes
13502 INC J                      No; skip delimiter
13508 *** JOK                    Shift out arg
13514 STR IN$ 1 N IN$ J
13528 RET
13532 *** ARGERR
13538 PRT "ARGUMENT"
13544 GTO ERR
13550 *** DMP                    Dump memory contents
13556 CAL ARG                    Get low range
13562 SET I=V
13570 CAL ARG                    Get high range
13576 FOR I=I
13584 BRD I                      Read a byte
13590 CHR 32                     Space
13596 PRT A                      Print a byte
13602 DIV I/10                   CRLF every 10th value
13610 MLT A*10
13618 SBT I-A
13626 JMP NE DM1
13634 CHR 13
13640 *** DM1
13646 TIL I+1 V
13658 GTO START
13664 *** NTR                    Enter memory values
```

```
13 670 CAL ARG              Get starting location
 13676 SET I=V
 13684 *** NT1
 13690 LEN IN$ 1            Line length
 13698 SBT A-0
 13706 JMP EQ START         Done
 13714 CAL ARG              Get a value
 13720 BRT I=V              Store it
 13728 INC I                Bump memory ptr
 13734 GTO NT1              Loop
 13740 *** BLK              Fill memory with value
 13746 CAL ARG              Get low range
 13752 SET I=V
 13760 CAL ARG              Get high range
 13766 SET OP=V
 13774 CAL ARG              Get fill value
 13780 FOR I=I
 13788 BRT I=V              Fill next byte
 13796 TIL I+1 OP
 13808 GTO START
 13814 *** EXIT             Exit to given location
 13820 CAL ARG              Get location
 13826 GTO V                Go there
 13832 END 13832 ? SIZ
PROGRAM 12288 13832    SYMBOLS 14000 15090
 13832 ? LSM
 14000 HLP          *PRINT" 14000
 14022 XIT I        *EXIT TO I" 14022
 14048 HELP 12334
 14055 EXIT 13820
 14062 SIZ 13226
 14068 WITH K" 14068
 14078 BLK I J K    *BLOCK FILL" 14078
 14105 NTR I...     *ENTER DATA..." 14105
 14135 MEMORY" 14135
 14145 DMP I J      *DUMP" 14145
 14166 SIZ $ U      *PRINT SIZE OF" 14166
 14196 OVR $ U I J *TAPE OVERLAY" 14196
 14225 RUN $ U      *EXECUTE" 14225
 14249 DLT $ U      *DELETE" 14249
 14272 NEW U        *NEWTAPE ON" 14272
 14299 ULD U        *UNLOAD TAPE" 14299
 14327 MNT U        *MOUNT TAPE," 14327
 14355 FROM I TO J" 14355
 14370 SAV $ U I J *TAPE SAVE" 14370
 14396 AT I" 14396
 14404 FILENAME $," 14404
 14419 LOD $ U I    *TAPE LOAD" 14419
 14445 REW U        *REWIND" 14445
 14468 UNIT U" 14468
 14478 DIR U        *LIST DIRECTORY," 14478
 14510 OPERATION CODES:" 14510
 14530 **** COPYRIGHT (C) 1978 BY PAUL K. WARME ****" 14530
 14579 $ 1

 14583 ARG3 12880
```

```
14590 OP 17010
14595 NT1 13690
14601 DM1 13646
14607 ARGUMENT" 14607
14619 JOK 13514
14625 ARGEND 13454
14634 GET V=NEXT VALUE IN IN$! 14634
14661 BYTES" 14661
14670 AR3 13316
14676 R2 16384
14681 AR2 13262
14687 AR1 13172
14693 V 0
14697 UNIT 0
14704 ARG 13354
14710 ARGERR 13538
14719 COK 13042
14725 NAMEND 13066
14734 LETR 13004
14741 FIRST CHAR OF NAME CAN'T BE A NUMBER! 14741
14781 C 48
14785 J 2
14789 N 1
14793 BLK 13746
14799 NTR 13670
14805 DMP 13556
14811 ARG1 12904
14818 ARG0 13122
14825 ARG2 12892
14832 NARG 2
14839 TOP 16
14845 ERROR" 14845
14854 ERR 12692
14860 TSPCH 12710
14868 I 49
14872 OPERATION" 14872
14885 START 12590
14893 NAME$ 8 TPBXC
14908 OPR$ 3 SIZ
14917 TOP$ 60 DIRREWLODSAVMNTULDNEWDLTRUNDMPNTRBLK      OVR   SIZXITHLP
14983 IN$ 72
15060 BASEXERCISER FOR MECA TAPE" 15060
```

III. Simultaneous Use of North Star Disk and Meca Tape

A. Description of Disk/Tape Operations

The operations described in Section I for disks and the tape operations described in Section II can be used together in BASEX programs, without any modifications in the disk or tape handlers and using the same command mnemonics suggested above. You will note that both tape and disk operations use a UNIT argument to specify which drive is to be used. The disk operating system uses drive numbers 1 to 4, whereas the tape system uses drive numbers ranging from 0 to 3. Even though these ranges overlap, there is no conflict, since the DSK and TAP commands communicate with entirely distinct handler routines.

By combining the rapid random access of the North Star floppy disk with the economy and large capacity of the Meca tape, BASEX programs can handle mass storage problems traditionally relegated to much larger and more expensive computer systems.

B. BASEXerciser for Floppy Disk and Tape

This program combines the features of the Basexercisers for disk and tape described in Sections I.B and II.B, respectively. No additional patches to the BASEX compiler or execution routines are required. All capabilities and restrictions cited in those sections apply equally to this program.

```
          SIZ
PROGRAM 12288 14480     SYMBOLS 14570 15862
14480 ? LST
12288 PRT "BASEXERCISER FOR NORTH STAR FLOPPY AND MECA TAPE" $ 1
12298 PRT "**** COPYRIGHT (C) 1978 BY PAUL K. WARME ****" $ 1
12308 PRT "$=FILENAME, U=UNIT, #=FILE, R=RECORD, I/J=MEMORY" $ 1
12318 *** HELP
12324 PRT "DISK" "OPERATION CODES:" $ 1
12336 PRT "LST U" $ 1
12346 PRT "WRT/RED/VFY/SFD/LFD/CKD # I" $ 1
12356 PRT "SEK # R" $ 1
12366 PRT "CHN/DLD $ U" $ 1
12376 PRT "CRT/OPN $ U #" $ 1
12386 PRT "TAPE" "OPERATION CODES:" $ 1
12398 PRT "DIR/REW/MNT/ULD/NEW U" $ 1
12408 PRT "DLT/RUN/SIZ $ U" $ 1
12418 PRT "LOD $ U I" $ 1
12428 PRT "SAV/OVR $ U I J" $ 1
12438 PRT "DMP I J" $ 1
12448 PRT "NTR I..." $ 1
12458 PRT "BLK I J K" $ 1
12468 PRT "XIT I" $ 1
12478 PRT "HLP" $ 1
12488 DIM IN$ 72 OP$ 60 OPR$ 3 NAME$ 8
12508 SET EOF=10752              Error indicator address
12516 *** START                  All commands return here
12522 CHR 13                     Carriage return/linefeed
12528 PRT "OPERATION"            Input command line
12534 INP IN$ 1
12542 STR OPR$ 1 3 IN$ 1         First 3 char are operation
12556 STR IN$ 1 68 IN$ 5         Shift line buffer
12570 FOR I=1                    Search for operation name
12578 CMP OPR$ 1 3 OP$ I
12592 JMP EQ DSPCH               Found it
12600 TIL I+3 36
12612 DIM TOP$ 60                Tape operation?
12620 FOR I=1
12628 CMP OPR$ 1 3 TOP$ I
12642 JMP EQ TSPCH               Yes
12650 TIL I+3 57
12662 PRT "OPERATION"            Undefined operation
12668 *** ERR
12674 PRT "ERROR"
12680 GTO START
12686 *** DSPCH                  Dispatch to disk routines
12692 DIV I/3                    OP=(position in OP$)/3
12700 SET OP=A
12708 SBT OP-6
12716 JMP LT X05                 OP=0 to 5
12724 JMP EQ SEK                 OP=6
12732 DEC A
12738 JMP EQ LST                 OP=7
12746 SBT OP-12
12754 JMP LT X811                OP=8 to 11
12762 *** ARGERR
12768 PRT "ARGUMENT"
```

```
12774 GTO ERR
12780 REM !GET V=NEXT VALUE IN IN$!
12786 *** ARG
12792 LEN IN$ 1                 Get command length
12800 SBT A-0                   Null?
12808 JMP LE ARGERR
12816 SET N=A                   N=length
12824 FOR J=1                   Find 1st non-numeral
12832 STR C IN$ J
12842 SBT C-47                  <0?
12850 JMP LE ARGEND
12858 SBT C-58                  >9?
12866 JMP GE ARGEND
12874 TIL J+1 N
12886 *** ARGEND                Delimiter found
12892 SBT J-1                   Double delimiter?
12900 JMP EQ ARGERR
12908 VAL IN$ 1                 Ascii to binary
12916 SET V=A                   V=value
12924 SBT J-N                   Last argument?
12932 JMP GT JOK                Yes
12940 INC J                     No; skip delimiter
12946 *** JOK                   Shift out argument
12952 STR IN$ 1 N IN$ J
12966 RET
12970 REM !WRT/RED/VFY/SAV/LOD/CHK!
12976 *** X05                   OP=0 to 5
12982 CAL ARG                   Get file #
12988 SET FNUM=V
12996 CAL ARG                   Get buffer location
13002 SET BUF=V
13010 DSK OP FNUM BUF           Execute OP=0 to 5
13020 SBT OP-2                  SAV or LOD?
13028 JMP GT SLC                Yes
13036 REM !WRT/RED/VFY/SEK ERRORS!
13042 *** ERTST
13048 WRD EOF                   Get error indicator
13054 SBT A-0                   OK?
13062 JMP EQ START              Yes
13070 JMP LT DKERR              Disk error if <0
13078 PRT "END OF FILE"
13084 GTO ERR
13090 REM !SAV/LOD/CHK ERRORS!
13096 *** SLC
13102 WRD EOF                   Get error indicator
13108 SBT A-1                   Should be end of file
13116 JMP EQ START              It is
13124 *** DKERR
13130 PRT "DISK"
13136 GTO ERR
13142 *** SEK                   Seek given sector
13148 CAL ARG                   Get file #
13154 SET FNUM=V
13162 CAL ARG                   Get sector #
13168 SET REC=V
13176 DSK OP FNUM REC           Execute OP=6
```

```
13 186 GTO ERTST
 13192 *** LST                    List directory
 13198 CAL ARG                    Get unit #
 13204 SET UNIT=V
 13212 DSK OP UNIT                Execute OP=7
 13220 GTO START
 13226 REM !CHN/DEL/CRT/OPN!
 13232 *** NAMIN                  Get filename
 13238 LEN IN$ 1
 13246 SET N=A                    N=line length
 13254 FOR J=1                    Extract file name
 13262 STR C IN$ J                C=next char
 13272 REM !FIRST CHAR OF NAME CAN'T BE A NUMBER!
 13278 SBT J-1
 13286 JMP EQ LETR
 13294 SBT C-48                   Is it a number?
 13302 JMP LT NAMEND              No
 13310 SBT C-58
 13318 JMP LT COK                 Yes
 13326 *** LETR
 13332 SBT C-65                   Is it a letter?
 13340 JMP LT NAMEND              No
 13348 SBT C-91
 13356 JMP GE NAMEND              Must be a delimiter
 13364 *** COK
 13370 TIL J+1 N
 13382 GTO ARGERR                 No delimiter
 13388 *** NAMEND                 End of name
 13394 DEC J
 13400 JMP EQ ARGERR              No char in name
 13408 STR NAME$ 1 J IN$ 1        Transfer to NAME$
 13422 ADD J+2                    Skip delimiter
 13430 STR IN$ 1 N IN$ A          Shift out name
 13444 RET
 13448 *** X811                   Operations 8 to 11
 13454 CAL NAMIN                  Get filename
 13460 CAL ARG                    Get unit
 13466 SET UNIT=V
 13474 SBT OP-10
 13482 JMP EQ CRT                 OP=10
 13490 JMP GT OPN                 OP=11
 13498 REM !CHN/DEL!
 13504 DSK OP NAME$ UNIT          Execute OP=8 or 9
 13514 WRD EOF                    Get error indicator
 13520 SBT A-0
 13528 JMP GT START               OK
 13536 *** NAMTST
 13542 JMP LT NAMERR
 13550 PRT "ABSENT"               EOF=0
 13556 *** NAMERR
 13562 PRT "FILE NAME"            EOF=-1 or -2
 13568 GTO ERR
 13574 *** CRT                    Create a new file
 13580 CAL ARG                    Get # sectors
 13586 SET NBLK=V
 13594 DSK OP NAME$ UNIT NBLK     Execute OP=10
```

```
13606 WRD EOF                        Check errors
13612 SBT A-0
13620 JMP GT BLKSIZ                  Normal return
13628 JMP EQ DKERR
13636 SBT A--1
13644 JMP LT DUP
13652 PRT "OVERFLOW"                 EOF=-1
13658 GTO DKERR
13664 *** DUP
13670 PRT "DUPLICATE"                EOF=-2
13676 GTO NAMERR
13682 *** OPN                        Open an old file
13688 CAL ARG                        Get file#
13694 SET FNUM=V
13702 DSK OP NAME$ UNIT FNUM         Execute OP=11
13714 WRD EOF                        Check errors
13720 SBT A-0                        Normally, EOF=# sectors
13728 JMP LE NAMTST                  Error
13736 *** BLKSIZ
13742 PRT NAME$ 1 A "BLOCKS"
13754 GTO START
13760 *** DMP                        Dump memory contents
13766 CAL ARG                        Get low range
13772 SET I=V
13780 CAL ARG                        Get high range
13786 FOR I=I
13794 BRD I                          Read a byte
13800 CHR 32                         Space
13806 PRT A                          Print a byte
13812 DIV I/10                       CRLF every 10th value
13820 MLT A*10
13828 SBT I-A
13836 JMP NE DM1
13844 CHR 13
13850 *** DM1
13856 TIL I+1 V
13868 GTO START
13874 *** NTR                        Enter memory values
13880 CAL ARG                        Get starting location
13886 SET I=V
13894 *** NT1
13900 LEN IN$ 1                      Line length
13908 SBT A-0
13916 JMP EQ START                   Done
13924 CAL ARG                        Get a value
13930 BRT I=V                        Store it
13938 INC I                          Bump memory ptr
13944 GTO NT1                        Loop
13950 *** TSPCH                      Dispatch to tape routines
13956 DIV I/3                        TOP=(position in TOP$)/3
13964 SET TOP=A
13972 SET NARG=1                     Initialize to 1 arg
13980 SBT TOP-2
13988 JMP EQ ARG2                    TOP=2
13996 SBT TOP-3
14004 JMP EQ ARG3                    TOP=3
```

```
14012 SBT TOP-7
14020 JMP LT ARG0              TOP=0,1,4-6
14028 SBT TOP-9
14036 JMP LT ARG1              TOP=7,8
14044 JMP EQ DMP               TOP=9
14052 SBT TOP-11
14060 JMP LT NTR               TOP=10
14068 JMP EQ BLK               TOP=11
14076 SBT TOP-16
14084 JMP LT ARG3              TOP=14
14092 JMP EQ ARG1              TOP=16
14100 DEC A
14106 JMP EQ EXIT              TOP=17
14114 GTO HELP                 TOP=18
14120 *** ARG3                 4 args
14126 INC NARG
14132 *** ARG2                 3 args
14138 INC NARG
14144 *** ARG1                 2 args
14150 INC NARG
14156 CAL NAMIN                Get filename
14162 *** ARG0                 Only 1 arg
14168 CAL ARG                  Get unit #
14174 SET UNIT=V
14182 SBT NARG-1               More args?
14190 JMP GT AR1               Yes
14198 TAP TOP UNIT             Execute command
14206 GTO START
14212 *** AR1
14218 SBT NARG-2               More args?
14226 JMP GT AR2               Yes
14234 SBT TOP-16               SIZ is special case
14242 JMP EQ SIZ
14250 TAP TOP NAME$ UNIT       Execute command
14260 GTO START
14266 *** SIZ                  Get file length
14272 TAP TOP NAME$ UNIT R2
14284 PRT NAME$ 1 R2 "BYTES"
14296 GTO START
14302 *** AR2                  3 or more args
14308 CAL ARG                  Get next arg
14314 SET R2=V
14322 SBT NARG-3               More args?
14330 JMP GT AR3               Yes
14338 TAP TOP NAME$ UNIT R2    Execute command
14350 GTO START
14356 *** AR3                  4 args
14362 CAL ARG                  Get last arg
14368 TAP TOP NAME$ UNIT R2 V  Execute command
14382 GTO START
14388 *** BLK                  Fill memory with value
14394 CAL ARG                  Get low range
14400 SET I=V
14408 CAL ARG                  Get high range
14414 SET OP=V
14422 CAL ARG                  Get fill value
```

```
14428 FOR I=I
14436 BRT I=V              Fill next byte
14444 TIL I+1 OP
14456 GTO START
14462 *** EXIT             Exit to given location
14468 CAL ARG              Get location
14474 GTO V                Go there
14480 END 14480 ?         SIZ
PROGRAM 12288 14480     SYMBOLS 14570 15862
14480 ? LSM
14570 HLP" 14570
14577 HELP 12324
14584 **** COPYRIGHT (C) 1978 BY PAUL K. WARME ****" 14584
14633 BASEXERCISER FOR NORTH STAR FLOPPY AND MECA TAPE" 14633
14685 XIT I" 14685
14694 BLK I J K" 14694
14707 NTR I..." 14707
14719 DMP I J" 14719
14730 SAV/OVR $ U I J" 14730
14749 LOD $ U I" 14749
14762 DLT/RUN/SIZ $ U" 14762
14781 DIR/REW/MNT/ULD/NEW U" 14781
14806 TAPE" 14806
14814 CRT/OPN $ U #" 14814
14831 CHN/DLD $ U" 14831
14846 SEK # R" 14846
14857 WRT/RED/VFY/SFD/LFD/CKD # I" 14857
14888 LST U" 14888
14897 OPERATION CODES:" 14897
14917 $=FILENAME, U=UNIT, #=FILE, R=RECORD, I/J=MEMORY" 14917
14969 $ 1

14973 TOP$ 60 DIRREWLODSAVMNTULDNEWDLTRUNDMPNTRBLK      OVR   SIZXITHLP
15039 AR3 14362
15045 BYTES" 15045
15054 R2 0
15059 SIZ 14272
15065 AR2 14308
15071 AR1 14218
15077 NAMIN 13238
15085 ARG1 14150
15092 ARG0 14168
15099 ARG3 14126
15106 ARG2 14138
15113 NARG 0
15120 TOP 0
15126 TSPCH 13956
15134 EXIT 14468
15141 NT1 13900
15147 DM1 13856
15153 BLOCKS" 15153
15163 DUPLICATE" 15163
15176 OVERFLOW" 15176
15188 DUP 13670
15194 BLKSIZ 13742
15203 NBLK 0
```

```
15210 FILE NAME" 15210
15223 ABSENT" 15223
15233 NAMERR 13562
15242 NAMTST 13542
15251 CHN/DEL! 15251
15262 OPN 13688
15268 CRT 13580
15274 COK 13370
15280 NAMEND 13394
15289 LETR 13332
15296 FIRST CHAR OF NAME CAN'T BE A NUMBER! 15296
15336 CHN/DEL/CRT/OPN! 15336
15355 UNIT 1
15362 REC 0
15368 SAV/LOD/CHK ERRORS! 15368
15390 DISK" 15390
15398 END OF FILE" 15398
15413 DKERR 13130
15421 ERTST 13048
15429 WRT/RED/VFY/SEK ERRORS! 15429
15455 SLC 13102
15461 BUF 0
15467 FNUM 0
15474 WRT/RED/VFY/SAV/LOD/CHK! 15474
15501 JOK 12952
15507 V 1
15511 ARGEND 12892
15520 C 49
15524 J 2
15528 N 1
15532 ARG 12792
15538 GET V=NEXT VALUE IN IN$! 15538
15565 ARGUMENT" 15565
15577 ARGERR 12768
15586 BLK 14394
15592 NTR 13880
15598 DMP 13766
15604 X811 13454
15611 LST 13198
15617 SEK 13148
15623 X05 12982
15629 OP 7
15634 ERROR" 15634
15643 ERR 12674
15649 DSPCH 12692
15657 I 22
15661 OPERATION" 15661
15674 START 12522
15682 10752 10752
15690 EOF 10752
15696 NAME$ 8
15711 OPR$ 3 LST
15720 OP$ 60 WRTREDVFYSFDLFDCKDSEKLSTCHNDLDCRTOPN
15785 IN$ 72
```

IV. Modifying the BASEX Compiler to Allow Programs to be Saved and Loaded from Disk or Tape

The standard BASEX compiler has no built-in facilities for saving and loading programs from mass storage; thus, it is necessary to exit to your monitor via the MON command in order to save or load programs. However, once you have made the changes described below, as well as those described in Section I.C or II.C, you will be able to list directories and read or write program files using the DKL, DKR or DKW commands (for disk) or the TPL, TPR or TPW commands (for tape).

DKL # or TPL # will List the directory on drive "#".

DKR NAME or TPR NAME will Read file "NAME" into memory, starting at the beginning of the current program area (i.e., the first location typed in response to the "RANGE?" prompt). The drive number is assumed to be the same as that given in the most recent DKL or TPL command. The length of the program read into memory is determined by the actual length of the disk or tape file. After a successful read operation, BASEX is restarted and prints the "RANGE?" prompt. If the requested file is not found or some other error occurs, the disk operating system will print "ERROR" and wait for another BASEX command after the "?" prompt. The tape operating system responds to an error in a different way: it prints "EH?" and wait for a command in the tape operating system after the "OK--" prompt. You will then have to type "EX Ø" to restart BASEX.

DKW NAME or TPW NAME will Write the current program into file "NAME" on the drive specified by the most recent DKL or TPL command. There must be a file called "NAME" already on the disk or else an error will occur. The error messages are the same as those described above for the DKR and TPR commands. In the case of the DKW command, the length of the saved program is determined strictly by the number of sectors previously allocated for that file on the disk. The TPW command uses the actual program limits last given in response to the "RANGE?" query and attempts to overlay file "NAME" on the tape. If the space already allocated to that file is too small, the current program will be written at the end of the tape. If you wish, you can change the overlay command (operation 14) in the TPW routine at line 6776 to a save command (operation 3); in this case, you must save the program under a new name or else an error will occur.

Before using the tape or disk routines below, you must make the following changes to the BASEX compiler:

1. Restart BASEX at location Ø and after the "RANGE" prompt, type 2148 6724 6846 8191. This sets up the BASEX compiler to make changes to itself.

2. If you are using the disk, type the following:

```
NTR 7525 68 75 204
NTR 7454 68 75 210    NTR 7110 0 42
NTR 7370 68 75 215
NTR 8023 68 75 76 68 75 82 68 75 87
```

If you want to use the tape commands, substitute 84 for 68 and 80 for 75 in these entries. These changes convert all references to GO1, GO2 and GO3 in the BASEX compiler to read DKL, DKR, DKW or TPL, TPR, TPW.

3. Type "LOC 6688" and then enter the disk or tape routines listed below.

4. After making sure that the assembler language disk or tape handler is loaded (at 2A00 or 2C00), type DKL 1 or TPL 1 to list the directory.

5. Now, type DKW BASEX or TPW BASEX to overlay the old version of BASEX with the new one.

*** BASEX CHANGES FOR DISK ***

```
PROGRAM 2148 6838    SYMBOLS 6846 8191
6838 ? LST 6688 6838
6688 *** DKL
6694 CAL ARGET
6700 SET U=V
6708 DSK 7 U
6716 GTO CMND
6722 *** DKR
6728 SET U1=4
6736 GTO USR1
6742 *** DKW
6748 SET U1=3
6756 *** USR1
6762 CAL ARGET
6768 DSK 11 V$ U 1
6780 WRD U2
6786 SBT A-0
6794 JMP LE ERROR
6802 DSK U1 1 L0
6812 WRD U2
6818 DEC A
6824 JMP NE ERROR
6832 GTO 0
6838 END 6838 ?
```

*** BASEX CHANGES FOR TAPE ***

```
PROGRAM 2148 6802    SYMBOLS 6846 8191
6802 ? LST 6688 6802
6688 *** TPL
6694 CAL ARGET
6700 SET U=V
6708 TAP 0 U
6716 CHR 13
6722 GTO CMND
6728 *** TPR
6734 CAL ARGET
6740 TAP 2 V$ U L0
6752 CHR 13
6758 GTO 0
6764 *** TPW
6770 CAL ARGET
6776 TAP 14 V$ U L0 ST
6790 CHR 13
6796 GTO 0
6802 END 6802 ?
```

A GUIDE TO BASEX*

TABLE OF CONTENTS

A GUIDE TO BASEX*

I. General Concepts of BASEX

BASEX is a new, easy-to-learn language for microcomputers that shares some of the best features of both BASIC and assembler language. Like assembler language programs, BASEX programs execute very rapidly (typically at least five times faster than equivalent programs written in BASIC). This speed of execution is possible because the BASEX compiler produces code which makes use of the fast and powerful subroutines residing in the BASEX operating system. Since the BASEX operating system is only about 2K bytes long, BASEX programs require about 6K less memory than comparable BASIC programs,which are executed by an 8K interpreter.

In spite of the speed and compactness of BASEX, most of the powerful features of BASIC are available in BASEX. In fact, these languages are so similar that most programs can be translated directly from BASIC into BASEX. Array variables, text strings, arithmetic operations on signed 16 bit integers and versatile I/O communication functions are provided by BASEX. One indication of the completeness, speed and power of the BASEX instruction set is the fact that the BASEX compiler itself is written in BASEX.

Unlike FORTRAN compilers, the BASEX compiler combines the functions of editting, compiling and initiating execution of programs. This saves time during program development. If the program is too large to reside in memory at the same time as the BASEX compiler and operating system (about 8K), it can be compiled in several segments and then the segments can be linked by a small (2K) LOADER program. The LOADER is capable of relocating programs anywhere in memory and also provides several utility functions, such as listing and changing the contents of specified memory

locations. The LOADER also compresses the program into the smallest amount of memory, thus making it possible to run large programs in a system with modest memory capacity.

In its present form, BASEX does not allow for calculations involving floating point numbers or trigonometric functions. These operations may be available in the future as add-on modules. The BASEX compiler is designed for expansion to include functions which are customized to suit the applications of each individual user.

The set of BASEX commands includes both program commands and compiler commands. Program commands (Section III) are entered into the next available location in the program area, while compiler commands (Section IV) are executed immediately by the compiler. Each command line entered on the user terminal consists of a three-letter command mnemonic, and this may be followed by one or more arguments separated by any single character other than a letter (A-Z) or a number (0-9) or a comma. BASEX inserts standard argument delimiters (e.g., +, *, =) in certain commands when they are listed, but it is not necessary to type these standard delimiters when the command is entered; a space between arguments is the recommended form for entry. If a back arrow "←" is typed, the previous character is ignored. If a control X is typed, the entire line is ignored and may be retyped following the prompting "?".

Upon initialization, BASEX types the "RANGE ?" message. Four values should now be entered to specify the first and last memory bytes of the current program and the first and last bytes of the symbol table. If you wish to enter a new program, the first and last byte values should be equal. If you type just 0, the most recent ranges of the program and symbol table are assumed. All memory locations between the last byte of the program and the first byte of the symbol table are cleared to zero

and then BASEX types the address of the first free location, followed by a "?". At this point, the user types in a command line; if it is a compiler command, it is executed immediately. If it is not a recognized command, the "CM ERROR" message is printed. If it is a program command, the command is deposited at the location typed (by BASEX) at the beginning of the line and the remainder of the command line is examined to determine whether any additional arguments were typed. If so, the argument names are looked up in the "symbol" table and their locations in the symbol table are entered at the next available location in the program space.

The symbol table consists of a list of names of variables followed by the values associated with those variables. If one or more of the variable names encountered in the new command line is not found in the symbol table, its name is entered in the symbol table and space is allocated to store the values subsequently to be assigned to that variable. The symbol table builds downward from the end of the memory region allocated for the symbols, whereas the program builds upward from the start of the program region. If, at any time, the program overlaps the symbol table, an error message, "OM ERROR" or "OS ERROR", is typed by BASEX. After the entire command line has been processed and the address of each argument has been entered in the program space, BASEX types the address of the next available location followed by a "?", and then awaits the next command. Other compiler commands for listing, editting and running the program are described in Section IV.

A COMPARISON OF BASIC, BASEX* AND ASSEMBLER LANGUAGE

		BASIC	BASEX	ASSEMBLER
A.	General Features:			
1.	Mode of operation	Interpreter	Compiler	Compiler
2.	Program execution speed	Slow	Medium	Fast
3.	Typical memory size for compilation	12K	12K	12K
4.	Typical memory size for program execution	12K	6K	6K
5.	Maximum characters in variable name	2	10	10
6.	Storage allocation for variables	Implicit	Implicit	Explicit
7.	Dimensionality of arrays	Multiple	Single	None
8.	Powerful text string operations	Yes	Yes	No
9.	Provision for program loops	FOR---NEXT	FOR---TIL	NONE
10.	Subroutine calls	Yes	Yes	Yes
11.	Arguments passed to and from subroutines	Single	Multiple	Registers
12.	Terminal Service for input and output	Yes	Yes	No
13.	Advanced functions	SQR,EXP,LOG SIN,COS,TAN	User Definable	None
14.	Memory search command	No	Yes	No
15.	Automatic statement numbering	No	Yes	No
16.	Block data move command	No	Yes	No
B.	Arithmetic Operations (number of bits)			
1.	Addition	32/16	16	16/8
2.	Subtraction	32/16	16	8
3.	Multiplication	32/16	16	0
4.	Division	32/16	16	0
5.	Exponentiation	32/16	0	0
6.	Absolute value	32/16	16	0
7.	Increment	0	16	16/8
8.	Decrement	0	16	16/8
C.	Logical Operations (number of bits)			
1.	Logical AND	16	16	8
2.	Inclusive OR	16	16	8
3.	Exclusive OR	0	16	8
4.	One's complement NOT	16	16	Accumulator

II. Definition of Argument Types

All arguments following BASEX commands fall into one of the following classes:

Type 1. Constants

A number argument, such as 12 or -19056, is a constant. Since BASEX works with signed 16-bit integers, the available number range is from -32767 to 32767. Constants within the range of -255 to 255 are stored directly within the program, whereas the alphanumeric names of all other constants are entered in the symbol table, together with their associated values; BASEX treats large constants like regular variables.

Type 2. Regular variables

A regular variable is an argument whose name begins with a letter (A-Z) and contains only letters (A-Z) and numbers (0-9). Each regular variable is associated with a 16-bit value (initially zero); this value is referenced in a command line by typing its name. Although the names of regular variables (as well as the names of all other types of arguments) can be of any length, long names take up more space in the symbol table. However, each argument name appears only once in the symbol table, regardless of the number of times it is referenced within the program. Thus, if a longer name will improve the readability of the program, it is desireable to use it. In fact, if the symbol table is subsequently compressed by the LOADER (see Section VIII), long names consume no more space in the symbol table than short ones.

The regular variable with the name A is reserved for referencing the accumulator, which stores the results of arithmetic and logical calculations (see Section III B).

Type 3. Array variables

An argument name that begins with a letter (A-Z) and ends with "(" is called an array variable. The number of 16-bit values assigned to an array is specified by the dimension (DIM) command (Section III A). Any one of the values in the array can be referenced by typing the name of the array, followed by the number of the desired element. For example, A(1 means the first value in the array called A(, while BOX(10 means the tenth value in the array BOX(. A regular variable can be used instead of a constant to specify the desired element of an array. For example, if N has previously been assigned the value ten, BOX(N means the tenth value in the array BOX(. Notice that the closing parenthesis should not be typed after the element number, as it is not used by the compiler and will be treated as an argument delimiter.

Type 4. String variables

The name of a BASEX string variable begins with a letter and ends with "$". The number of 8 bit characters assigned to a string is specified by the DIM command (Section III A). Any Ascii character other than a comma or a carriage return can be stored within a string variable. Any particular character in the string may be referenced by typing the name of the string, followed by the number (either a constant or a regular variable) of the character. For example, S$ 1 means the first character of string S$ and TEXT$ N means the Nth character of string TEXT$. In some cases, depending on the program context (see Section III E), S$ 1 would reference an indefinite number of characters in S$, beginning with the first character. In certain program commands (STR and CMP) it is necessary to specify a range of characters; in such cases, the number of the last character to be used is also typed. For example, S$ 3 5 would specify the third through the fifth characters of S$, while TEXT$ I J would

specify the Ith through Jth characters of TEXT$.

Type 5. String constants

A BASEX argument which begins and ends with quotes (") is a string constant. Any Ascii character except a " or ! can be stored within a string constant. In the symbol table, the initial quote is discarded (to save space), and when a string constant is typed out, the quotes are omitted. A special kind of string constant which begins and ends with "!" is used in REM statements (see Section III A).

III. Description of Program Commands

In the following descriptions, the permissible argument types (1 to 5 in Section II) are enclosed in brackets [ ], while the permissible argument types for subscripts are enclosed in parentheses ().

A. Non-executable Commands (REM, DIM, ***, END)

Non-executable commands are those which are entered in the program (so that they can be listed), but have no effect during execution of the program.

1. REM [5] (e.g., REM !MESSAGE!)

The REM command serves as a REMinder of important details about the program. The message argument must be enclosed by exclamation marks. Although REM messages use up space in the symbol table, they are very beneficial as an aid to understanding and REMembering. When a program is compressed by the LOADER (see Section VIII), all REM commands (as well as all other non-executable commands) are omitted, so that more memory is made available for executable program commands.

2. DIM [3,4](1)[3,4](1)... (e.g., DIM A(5 S$ 10)

The DIM command is used to specify the DIMensions of array variables and string variables. Arrays and strings must be dimensioned before they are used in a program. Thus, it is recommended that all DIM commands be placed at the beginning of the program. When the compiler recognizes a DIM command, it allocates the required space in the symbol table. If a variable has previously been dimensioned, a double dimension error (DD ERROR) is announced. Although the DIM statement is non-executable, its presence in the program will serve as a reminder of the dimensions allocated to arrays.

3. *** [2] (e.g., *** LABEL)

The *** command is used to label a particular location in a program so that it can be referenced by other program commands, such as GTO, JMP and CAL (Section III F).

4. END

The END command appears at the end of each program module. When it is encountered during program execution, the program ceases and control is returned to the BASEX compiler. The END command is automatically appended after each command line is entered, but the END command can also be inserted at any point within a program in order to terminate execution prematurely. This will permit you to examine the values associated with particular variables (see Section IV B) and verify that the program is performing as planned. In general, it is possible to restore the original command (which was replaced by the END) and then resume execution at that point. However, if there are any pending returns (RET) from subroutines (i.e., if the END command is inserted within a subroutine), the return address is unrecoverable, so you should not attempt to resume execution from that point. Another exception is that the value in the accumulator (A) is destroyed after returning to BASEX, so you should not attempt to resume execution at that point if the value of A is critical.

B. Arithmetic Commands (ADD, SBT, MLT, DIV, ABS, INC, DEC)

1. ADD [1,2]+[1,2] (e.g., ADD BOX+10)
2. SBT [1,2]-[1,2] (e.g., SBT COLOR-BLUE)
3. MLT [1,2]*[1,2] (e.g., MLT BACTERIA*2)
4. DIV [1,2]/[1,2] (e.g., DIV CAKE/EATERS)

All of the above commands operate on signed, 16-bit integers. Notice that the arguments of these and all other arithmetic and logical commands must be either constants or regular variables. If operations are required on array variables or string variables, the values must be transferred first into a regular variable or into the accumulator (A). For example,

```
SET A=BOX( 10
MLT A*5
```

will produce a value equivalent to BOX(10*5. The result of any arithmetic operation is stored in the accumulator, A. An internal register (the condition code register) also remembers whether the most recent arithmetic operation produced a zero result (EQ or NE condition) or a result less than zero (LT or LE condition) or a result greater than zero (GT or GE condition). The condition code register is tested by the JMP command (Section III F) and is used to control the flow of the program depending on the result of an arithmetic calculation.

5. ABS [2] (e.g., ABS VALUE)

The ABS command returns the ABSolute value of its argument in the accumulator, A. In other words, if the value is negative, the positive number of equal magnitude is returned. The condition code register is unaffected.

6. INC [2] (e.g., INC COUNT)

The INC command INCrements (adds one to) its argument and places the result in both the accumulator, A, and in the memory space assigned to that argument. The condition code register is affected.

7. DEC [2] (e.g., DEC MONEY)

The DEC command DECrements (subtracts one from) its argument and places the result in both the accumulator, A, and in its assigned memory space. The condition code register is also affected.

C. Logical Commands (AND, IOR, XOR, NOT)

1. AND [1,2]&[1,2] (e.g., AND INPUT&LOHALF)

The AND command performs a logical AND between its first argument and its second and places the result in the accumulator, A. In other words, only those bits that are 1's in both arguments are 1's in the result. If the result is all 0's, this is reflected by the condition code register.

2. IOR [1,2]#[1,2] (e.g., IOR BINARY#ASCII)

The IOR command performs a logical Inclusive OR between its first argument and its second and places the result in the accumulator, A. Those bits that are 1's in either argument are 1's in the result. The zero condition is reflected by the condition code register.

3. XOR [1,2]%[1,2] (e.g., XOR RESULTS%ELSE)

The XOR command performs a logical eXclusive OR between its first argument and its second and places the result in the accumulator, A. Those bits that are 1's in either argument but not both are 1's in the result. The condition code register detects the zero condition.

4. NOT [1,2] (e.g., NOT PRESENT)

The NOT command returns the one's complement of its argument in the accumulator, A. That is, all bits that are 0's are changed to 1's

and vice versa. If the result is incremented by one, this is equivalent to negation of the argument. The zero condition is detected by the condition code register.

D. Load and Store Commands (SET, DEF, DAT, WRD, BRD, WRT, BRT)

1. SET [2]=[1,2,3,4,5] (e.g., SET SUM=A(5)

SET [3] (1,2)=[1,2,3,4,5] (e.g., SET A(5=SUM)

The SET command SETs the left hand argument equal to the right hand argument. In the case of array or string variables (types 3 and 4), an additional argument must follow the name of the array or string in order to indicate the particular element desired. When the right hand argument is a string variable, the first character specified is placed in the low order byte of the left hand variable and the next character is placed in the high order byte. If the right hand argument is a string constant, its first two characters are similarly placed in the left variable.

2. DEF [2]=[1,2] [2]=[1,2]... (e.g., DEF L=1 M=L)

DEF [3] (1,2)=[1,2]... (e.g., DEF B(3=C)

The DEF command DEFines any number of regular variables in a single command line. In contrast to the SET command, however, only regular variables and array variables can be defined by the DEF command.

3. DAT [3] (1,2)=[1,2] [1,2]... (e.g., DAT NAME(2=B 10 D)

The DAT command assigns values to an array variable, beginning with the element specified by the second argument, until the right hand arguments are exhausted. Note that only constants and regular variables can appear as right hand arguments.

4. WRD [1,2] (e.g., WRD ADDRESS)

5. BRD [1,2] (e.g., BRD 1010)

The WRD command ReaDs a WoRD (16 bits) at the address specified by its argument and places the value in the accumulator, A. The BRD command is the same, except that it ReaDs a Byte (8 bits) at the address specified and places it in the low order half of the accumulator, A, while the high order half is set to zero. These two commands provide a convenient way to access data tables created by your program.

6. WRT [1,2]=[1,2] [1,2]... (e.g., WRT WORDS=W1 W2)

7. BRT [1,2]=[1,2] [1,2]... (e.g., BRT BYTES=B1 B2)

The WRT command WRiTes one or more 16-bit Words, corresponding to the right hand arguments, into sequential memory locations beginning at the address specified by the left hand argument. Similarly, the BRT command wRiTes one or more Bytes into sequential memory locations. These commands provide a convenient means for creating data tables anywhere in memory in either word or byte format.

E. String Commands (STR, LEN, CMP, VAL)

1. STR [4] (1,2)(1,2)=[1,2,3,4,5] (e.g., STR S$ 1 4="TEXT")

STR [2]=[4] (1,2) (e.g., STR CHAR=S$ 8)

STR [3](1,2)=[4] (1,2) (e.g., STR A(1=S$ 2)

The STR command is a very versatile command for manipulating STRing variables. Since strings are made up of 8-bit characters, they cannot be handled by most of the other commands, which deal only with 16-bit words. However, the STR command provides a means for transferring 8-bit string characters into 16-bit variables and vice versa. For example, STR S$ 3 4=B will transfer the low order byte of the regular variable B into the third character position of S$ and the high order byte of B will be transferred into the fourth position. The command STR S$ 1 1=B will transfer just the low order byte of B into the first

character of S$. Whenever a string variable appears on the left side of an STR command (i.e., characters are being transferred into a string variable), a 0 byte is placed in the character position immediately following the last byte transferred. This 0 byte signals the end of the current string. In BASIC, the CHR$ function would be used to perform this function (LET S$=CHR$(B)).

In order to transfer a single character from a string variable into a regular variable or into an array variable, the destination variable is the left hand argument and the string variable is on the right. For example, STR CHAR=S$ I will transfer the Ith character of S$ into the low order byte of CHAR and the high order byte of CHAR will be set to 0. Notice that only one character is transferred into CHAR in this case, whereas the command SET CHAR=S$ I would transfer two characters from S$ into CHAR (see Section III D). In BASIC, the same result would be accomplished by the statement LET CHAR=ASC(MID$(S$,I)).

The STR command is frequently used to transfer an entire group of characters from one string variable into another. For example, STR S$ 1 5= TEXT$ 3 will transfer characters starting from the third one in TEXT$ into the first through the fifth positions in S$ and then will insert a terminating 0 into the sixth position of S$. It is not proper to specify a range of arguments for the right hand argument in an STR command (i.e., one should not specify TEXT$ 3 7 in this example) because the number of characters to be transferred is always fully specified by the left hand argument. If there are insufficient characters in the right hand string to match the range of the left hand string, a 0 is placed in the left hand string following the last character available to be transferred.

In BASIC, several commands are necessary to accomplish the functions equivalent to the STR command. The BASEX equivalents of the CHR$ and

ASC functions have already been mentioned above. The BASIC functions called LEFT$ and MID$ can be duplicated in BASEX by appropriately setting the range of the left hand (destination) string. For example, STR S$ 1 I=TEXT$ 1 is equivalent to LET S$=LEFT$(TEXT$,I) in BASIC. The command STR S$ 1 10=TEXT$ I is equivalent to the BASIC statement LET S$= MID$(TEXT$,I,I+9). The BASIC function called RIGHT$ can also be performed by BASEX. For example, the BASIC statement LET S$=RIGHT$(TEXT$,I) can be accomplished by the BASEX command STR S$ 1 20=TEXT$ I. Provided that the range of characters specified in S$ equals or exceeds the number of characters in TEXT$ from the Ith onward, the desired result will be obtained. As pointed out earlier, it doesn't matter if the left hand argument asks for more characters than are available in the right hand argument, since the left hand string will automatically be terminated with a 0 (end of string) byte after the last available character.

2. LEN [4](1,2) (e.g., LEN S$ I)

The LEN command counts the number of characters preceding the first 0 (end of string) byte in a string variable. The first character counted is specified by the subscript, which may be either a constant or a regular variable.

3. CMP [4] (1,2) (1,2) [4] (1,2) (e.g., CMP S$ 1 I TEXT$ J)

CMP [4] (1,2) (1,2) [5] (e.g., CMP S$ 1 I "TEXT")

The CMP command CoMPares string variables with other string variables or string constants. As a result of the comparison, the condition code register is set to indicate whether the strings are the same. If they are not the same, the first differing character in the right hand string is subtracted from the corresponding character in the left hand string and the condition code register is set accordingly. As described

in Section III F, the JMP command makes use of the condition code register to direct the course of the program. If the two strings are equal, the JMP EQ THERE command will jump to the location called THERE; if the strings are not equal, the program will continue on to the command following the JMP command. Alphabetic order is the criterion for deciding which string is less than or greater than the other. For example, if the following program is executed:

```
STR S$ 1 4 "ABCD"
CMP S$ 1 4 "BCDE"
JMP LT THERE
```

the program will jump to THERE because ABCD would precede BCDE in the dictionary.

4. VAL [4] (1,2) (e.g., VAL S$ 1)

The VAL command returns the numeric VALue of a string in the accumulator, A. Beginning with the character specified by the subscript, the string variable is scanned until the first non-numeric character occurs and then the preceding numeric characters are interpreted as a decimal integer. If the first specified character is not a number, the value zero is returned in A.

F. Commands to Control Program Flow (GTO, JMP, FOR---TIL, CAL, GET, PUT, RET)

1. GTO [1,2] (e.g., GTO THERE)

The GTO command causes the program to branch (GoTO) to the location specified by its argument. The location may be specified by a constant or by a regular variable. Generally, the location name is defined by a *** command (Section III A) elsewhere in the program.

2. JMP CC [1,2] (e.g., JMP NE THERE)

The JMP command is used to direct program flow in response to previous calculations by the program. As mentioned in Sections III B and III E, the condition code register (CC) stores the sign (less than, equal or greater than zero) flags, which are affected by all arithmetic and logical commands and are also affected by the string comparison (CMP) command. There are six conditions which can be tested by the JMP command: equal (EQ), not equal (NE), greater than (GT), greater than or equal (GE), less than (LT) and less than or equal (LE) to zero. The "equality" flag is set if the result of a calculation is zero. The "less than" and "greater than" flags are set according to the sign of the result. For example, SBT 10-10 would leave zero in the accumulator and would set the EQuality flag, so that a subsequent command JMP EQ THERE would jump to THERE. On the other hand, SBT 10-12 would give a result Less Than zero, so that JMP LT THERE would jump to THERE, whereas a JMP GE THERE would not jump. However, the command SBT 12-10, followed by JMP GE THERE would jump to THERE. In other words, the SBT command together with the JMP command provides a means for testing the relative magnitudes of two variables and altering the course of the program in response to the result. Reading programs, such as the one included in Section VI, is a good way to become familiar with many additional ways to use the JMP command.

3. FOR [2]=[1,2] (e.g., FOR I=1)

4. TIL [2]+[1,2] [1,2] (e.g., TIL I+J K)

The FOR command is used in conjunction with the TIL command to produce program loops, much as the FOR---NEXT construction is used

in BASIC. The FOR command sets the initial value of the loop variable. The program commands following the FOR command are executed in the normal fashion unTIL a TIL command is encountered. The TIL command adds to its first argument (the loop variable) the value of its second argument (the step size) and checks whether the result has exceeded its third argument (the termination value). The step size may be either positive or negative. If the result has not exceeded the termination value, the program loops back to the command following the FOR command having the same loop variable. If the result has exceeded the termination value, the program continues on to the command following the TIL command. One precaution should be noted: a loop always terminates when the value of the loop variable passes through zero. There is no limit to the number of FOR---TIL loops which can be contained in the program and there are no special conditions pertaining to "nesting" of loops in BASEX. The current value of the loop variable can be used normally by commands within the loop and its value can even be changed by program commands within the loop. In contrast to most other languages, it is possible to jump into the middle of a FOR---TIL loop without first executing the FOR command. In this event, the current value of the loop variable will be incremented, as usual, by the TIL command until the termination value is reached. It is also possible to jump out of a FOR---TIL loop before the termination value is reached. Consider the program

```
FOR I=1
SBT I-X
JMP EQ SMALLER
TIL I+1 10
GTO GREATER
*** SMALLER
```

If the value of X is less than or equal to 10, the loop will be prematurely terminated and will jump to location SMALLER. If the value of X is greater than 10, the loop will terminate normally (with the value of I=11) and the program will go to location GREATER.

5. CAL [1,2] [1,2] [1,2]... (e.g., CAL FUNCTION 30 VALUE)

The CAL command CALls the subroutine whose address is specified by its first argument. Usually, the name of the subroutine is defined elsewhere in the program by a *** command. However, it is possible to call a machine language subroutine by giving its address in terms of a constant (or by specifying a variable that contains its address) as the first argument. In contrast to BASIC, BASEX provides for convenient exchange of multiple variables between the calling program and the subroutine. This is done by including additional constants or regular variables in the CAL statement, following the subroutine name or address. For example, the command CAL FUNCTION 30 B will call subroutine FUNCTION and the arguments 30 and B will be used by the subroutine as outlined below. Each extra argument in the CAL command must have a corresponding GET or PUT (see below) in the subroutine or an error will result.

6. GET [1,2] (e.g., GET ARGUMENT)

The GET command GETs a value from the argument list of the CAL command and places the value in the variable specified by its own argument. The name of the variable in the GET command is not necessarily the same as the name of the variable in the CAL command. If the names are different, the passed value is a "local" variable; i.e., any change in its value during execution of the subroutine will not affect the value of the variable named in the CAL command. However, if the name of the variable is the same in both the CAL command and the GET command, the

passed variable is a "global" variable and any change in its value within the subroutine will also be recognized by the calling program.

It is possible to GET as many arguments as there are in the CAL command, but the subroutine is responsible for either GETting or PUTting (see below) each argument specified in the CAL command. If this condition is not met, an error will result.

7. PUT [1,2] (e.g., PUT RESULT)

The PUT command PUTs a value calculated within a subroutine into the next variable listed in the CAL command. PUT commands may be interspersed with GET commands in any order. Remember that there is a one-to-one correspondence between the arguments listed in the CAL command and the GET or PUT commands in the subroutine. If all arguments listed in the CAL are not used by the subroutine, an error will result.

The remarks about "local" and "global" variables given in the description of the GET command (above) pertain here, as well. In other words, if the same name is used for an argument in both the calling program and in the subroutine, it is not necessary to specifically return the value via the PUT command. However, if different names are used, the PUT command may be used to place the value calculated in the subroutine into the corresponding variable listed in the CAL statement.

8. RET

The RET command RETurns from a subroutine to the calling program, where execution will continue at the command following the CAL command. It is important to realize that each CAL command places a return address on the "stack" (an internal list of arguments which is not directly accessible to BASEX programs). Thus, if a RET command is not executed

within the subroutine, the results are unpredictable. The subroutine may contain more than one RET command, permitting optional returns from several different points within the subroutine.

9. Machine Language Subroutine Conventions

If the subroutine is in machine language and not in BASEX, there are some special considerations which must be taken into account. If the passed argument is a constant in the range -255 to 255, the following code will GET it in the BC register pair:

```
POP H     *HL now contains the address of the value
MOV C,M
INX H
MOV B,M
INX H
PUSH H    *Save the pointer.
```

If the passed argument is a regular variable or a constant outside the range above, the following code will GET it in the BC register pair:

```
POP H
MOV E,M
INX H
MOV D,M   *DE now contains the address of the value
INX H
PUSH H    *Save the pointer
XCHG
MOV C,M
INX H
MOV B,M   *BC contains the passed value
```

To return (PUT) a regular variable from the BC register pair to the calling program, include this code:

```
POP H
MOV E,M
INX H
MOV D,M   * DE now contains the address for storing the value
INX H
PUSH H    *Save the pointer
XCHG
MOV M,C
INX H
MOV M,B   *BC value has now been transmitted
```

You should not attempt to PUT a value into a constant defined in the calling program, since this will redefine the constant and permanently alter the program.

The machine language equivalent of the RET command can be implemented by the following code after all arguments in the CAL command have been disposed of:

```
POP H
INX H
PCHL
```

There is no restriction on use of the registers in machine language subroutines, but any data placed on the stack must be removed from it before executing the RET sequence above.

G. Input and Output Commands (INP, PRT, PRN, CHR, TAB, XIN, XOT, WAT, DSI, ENI)

1. INP [2,3,4] [2,3,4] ... (e.g., INP B C(2 S$ 5)

The INP command INPuts data from the user terminal into the specified list of variables. When an INP command is encountered in a

BASEX program, a "?" is typed, followed by a space. The arguments may then be entered. Any character other than a number serves as a delimiter for numeric arguments, but string arguments must be separated by commas. Note that array variable and string variable names in the input list must be followed by a single subscript which indicates the first element to be entered. In the case of string variables, characters are entered into successive elements of the string until a comma is encountered in the input line. If the line entered does not contain enough arguments, an error will occur; any extra arguments entered will be ignored.

If a back-arrow (←) is typed, the previous character is ignored, while a control X will delete the entire line, type a carriage return and line feed, and then will accept a new input line. Up to 80 characters (not counting deleted ones) can be entered on a line, but an error will result if more than this are entered.

The INP command makes use of one simple subroutine which must be supplied by the user. This subroutine should read a single character from your terminal and return it in the A register (e.g., LOOP: IN STATUS; ANI READY; JZ LOOP; IN TERMINAL; RET). The address of this subroutine must be entered at location 44B (hex) before the BASEX compiler is ready to operate.

2. PRT [1,2,3,4,5] [1,2,3,4,5]... (e.g., PRT 10 B C(2 S$ 5 "TEXT")

3. PRN [1,2,3,4,5] [1,2,3,4,5]... (e.g., PRN S$ 1 T$ 1)

The PRT command PRinTs its arguments on the user terminal. One space is typed after each argument. The PRN command is the same, except that the space after each argument is omitted. Note that array variables and string variables must be followed by a single subscript, which may be either a constant or a regular variable. In the case of a string variable, characters will be typed from the first one indicated by the subscript until a 0 (end of string) byte is encountered or until the last character of the string has been typed.

In BASEX, a carriage return/line feed (CR/LF) is not automatically typed after each PRT command, as it is in BASIC. In order to type a CR/LF, you may define a string variable containing the Ascii character for a carriage return and then append this string to each PRT command where you want a CR/LF to be typed. A line feed is automatically typed after a carriage return. For example, the following commands will create such a string variable:

```
DIM CR$ 1
STR CR$ 1 1=13
```

Now, you can insert a CR/LF after any PRT command, as illustrated by this command:

```
PRT 10 B CR$ 1
```

An alternative way to type a CR/LF is to insert the CHR 13 command (see below). If a line is printed that exceeds 80 characters in length, a CR/LF is automatically performed after the 80th character.

The PRT command makes use of one simple subroutine that must be supplied by the user. This subroutine should type a character, supplied in the B register (e.g., LOOP: IN STATUS; ANI READY; JZ LOOP; MOV A,B; OUT TERMINAL; RET). The address of this subroutine must be entered at locations 40E, 41F and 430 (hex) before the BASEX compiler is ready to operate. If you are using a video terminal for output, it would be advisable to insert a delay loop into your subroutine, since BASEX is capable of very high output rates.

4. CHR [1,2] (e.g., CHR LETTER)

The CHR command types one CHaRacter specified by its argument, which should be a valid Ascii character. As noted above, a carriage return/line feed will be generated by the CHR 13 command. The following program will type all printing characters on your terminal, followed

by CR/LF.

```
FOR I=20
CHR I
TIL I+1 126
CHR 13
```

5. TAB [1,2] (e. g., TAB 10)

The TAB command will print spaces on your terminal until the column specified by its argument is reached. If the current carriage position is beyond the specified column, no spaces are typed.

6. XIN [1,2] (e.g., XIN TAPE)

The XIN command INputs a byte from the input port specified by its argument, places it in the low order half of the accumulator, A, and sets the high order half to zero.

7. XOT [1,2] (e.g., XOT PUNCH)

The XOT command OuT puts the byte contained in the low order half of the accumulator, A, to the output port specified by its argument.

8. WAT [1,2] [1,2] (e.g., WAT PUNCH READY)

The WAT command WAiTs until the input port specified by its first argument gives a nonzero result when it is ANDed with the second argument. The accumulator (A) is not altered.

9. DSI

10. ENI

The DSI command DiSables Interrupts, whereas the ENI command ENables Interrupts.

It is possible to write many versatile input and output routines in BASEX using the preceding commands XIN, XOT, WAT, DSI and ENI. The high execution speed of BASEX should enable you to interface moderately fast devices, such as paper tape readers, punches, and digital cassette recorders.

H. Memory Search and Move Commands (WSR, BSR, MCV)

1. WSR 1,2 1,2 1,2 (e.g., WSR LOW HIGH WORD)

2. BSR 1,2 1,2 1,2 (e.g., BSR BEGIN END BYTE)

The WSR command SeaRches for a Word specified by its third argument throughout the memory region specified by its first and second arguments. Similarly, the BSR command SeaRches for a Byte. When the specified word or byte is found, its memory address is stored in the accumulator (A) and the LT condition code flag is set. Thus, a successful search can be detected by a JMF LT FOUND command. Although the search function could be performed by a simple BASEX program, the WSR and BSR commands operate faster on large data tables because they are coded in assembler language.

3. MCV 1,2 1,2 1,2 (e.g., MCV BEGIN END NEWLOC)

The MCV command rapidly MOVes large blocks of data from one place in memory to another. The first and second arguments specify the limits of the block to be moved and the third argument specifies the first location of the new position for the block. The MCV command tests whether the block is being moved upward or downward in memory; if it is being moved upward, the last byte of the block is moved first and if it is being moved downward, the first byte is moved first. Thus, there is no danger that data will be destroyed during the MOV operation.

IV. Compiler Commands

A. Listing the Program (LST) (e.g., LST 9000 10000)

The LST command enables you to LiST one or more program statements. If no arguments are given, the entire program will be listed. If just one argument is given, only the command line at that location is listed. The presence of a second argument requests a listing of all command lines within the range of the first and second arguments. If you specify a location which is not the beginning of a command line, the "LS ERROR" message will be printed. If an END command is encountered before the line specified by the second argument is reached, the listing will terminate.

B. Listing the Symbol Table (LSM) (e.g., LSM 10000 11000)

The LSM command LiSts the SyMbol table between the limits specified by its arguments. The symbol table contains the names and associated values of all variables used by the program. If no arguments are given, the entire symbol table is listed; if one argument is given, only the variable name at that location is listed; if two arguments are given, the variables within that range will be listed. After each variable, the value currently associated with that variable is listed. For array variables and string variables, the dimension of the variable is printed, followed by the current contents of each element (of an array) or all characters preceding the end-of-string byte (of a string). The LSM command can often be used to debug a program by examining the values of variables before and after execution.

C. Determining the Size of the Program (SIZ)

The SIZ command requires no arguments, and prints out the current range of the program and symbol table in the following format:

PROGRAM 9000 10000 SYMBOLS 10000 11000

When BASEX is initialized or reinitialized after an execution error, these four values should be typed in this same order, separated by spaces, in response to the "RANGE ?" message.

D. Dumping the Contents of Memory (DMP) (e.g., DMP 100 200)

The DMP command DuMPs the contents of all memory locations within the range of its arguments out on the user terminal. If no second argument is given, only the value at the location specified by the first argument is typed. A carriage return is automatically typed after each location that is evenly divisible by ten.

E. Entering Values in Memory (NTR) (e.g., NTR 100 1 2...)

The NTR command is used to eNTeR values into sequential memory locations, beginning at the location specified by the first argument. The second and subsequent arguments are interpreted as (8-bit) decimal integers.

F. Changing the Location Pointer (LOC) (e.g., LOC 8500)

The position where the next program command will be entered is called the LOCation pointer. BASEX types out the value of the location pointer at the beginning of each command line. However, its value may be changed by the LOC command. For example, if the previous command line contained an error, the location pointer may be reset to the value at the beginning of that line and the correct command can be written over the incorrect one. Considerable care must be exercised if it is necessary to change a command that is followed by other command lines which you want to keep. This is not permitted unless the number of arguments (including subscripts) in the new command is exactly the same as the number of arguments in the old command. In all other cases, the INS and DLT commands must be used for modifying the program.

G. Inserting Program Commands (INS) (e.g., INS 8000 8100)

The INS command allows you to INSert one or more command lines in a previously existing program. The first and second arguments specify a region of memory which is to be vacated by pushing the program upward in memory. Before invoking this command, be sure that the location pointer is positioned at the end of the program and be careful not to insert so much space that the end of the program overlaps the symbol table. After the INS command is completed, BASEX will type out the new position of the end of the program and will set the location pointer to

the value of the first argument. You may now enter commands in the vacated region, remembering that you must not go beyond the location specified as the second argument of the INS command. In general, you should insert more than the number of locations required to contain your corrections, because the delete (DLT) command (see below) must follow an INS command, anyway. After typing in your program corrections, set the location pointer to the end of the program (using the LOC command) and then execute a DLT command to recover all unused program locations.

H. Deleting Parts of the Program (DLT) (e.g., DLT 8050 8100)

The DLT command will DeLeTe program locations from its first to its second argument, inclusive. As the DLT command is executed, the locations of *** commands are updated in the symbol table and the ranges of FOR---TIL loops are corrected, if necessary. This is why the DLT command must follow an INS command.

After an INS command has been executed and corrections have been inserted, the location pointer should be set to the end of the program (as printed out immediately after the INS command is executed), using the LOC command. Then, the unused space within the vacated region can be recovered by executing the DLT command. Generally, the first argument of the DLT command will be the same as the location pointer value printed after the last inserted program line; the second argument of the DLT command will be the same as the second argument of the previous INS command. After the DLT command is executed, the location pointer is automatically set to the end of the program and its value is printed out, as usual, at the beginning of the next command line. The vacated region beyond the end of the program is filled with zeroes.

down the range of both the program and the symbol table (as printed by the SIZ command) so that after the program is reloaded and BASEX is reinitialized, you can enter these ranges in response to the "Range ?" query.

V. Error Messages

A. Compiler Error Messages

1. CM ERROR means that the CoMmand given as the first three characters of the current line was not recognized.

2. DD ERROR means that the current DIM command has attempted to redimension (Double Define) a string or array variable that already is present in the symbol table.

3. DM ERROR means that an error has been detected in a DIM command. Either a subscript is invalid (outside the range of 1 to 255) or else the last character of a variable name was neither "$" nor "(".

4. LS ERROR means that an error has occurred while LiSting a command line due to failure to recognize the command type. This message generally means that the first argument of the LST command does not coincide with the beginning of a command line or that a gap exists in the program at the point where the LS ERROR occurred.

5. NF ERROR means that a TIL command has used a loop variable which has No corresponding For command (using the same loop variable).

6. OM ERROR means that the current command line has exceeded the available memory (Out of Memory). In other words, the program has overlapped the symbol table.

7. OS ERROR means that the current command line has attempted to expand the symbol table to the point where the symbol table would overlap the program (Out of Symbol space).

ST1, ST2, ST3 = the first, second and third values on the stack at the time of the error

The ERR value (the 6th value printed) is the most important one for the BASEX user because it will tell you the type of error, as listed below. The location in the BASEX program where the error occurred will be given by either HL or ST1, depending on the type of error (see below). This location will not exactly correspond to the beginning of a command line, but will point to an argument within the command which was being processed at the time of the error. The other values printed out in the error code may be useful to the experienced assembly language programmer who has a source listing of the BASEX execution routines.

The ERR values may be interpreted as follows (ERR values preceded with * are cases where the program location is given by HL; all others place the program location in ST1):

ERR = 323: Attempt to divide by zero.

*606: Incorrect back pointer in TIL command. The back pointer addresses the command following the corresponding FOR command.

*702: Subscript error in array or string variable.

*754: Improper second subscript for a string variable.

*817: Phase error; the next instruction in the program is not a CALL instruction, as required.

1144: Line exceeds 80 characters.

1162: Attempt to redefine a constant in INP command.

1170: Attempt to redefine a string constant in INP command.

*1226: During conversion of an input string or string variable (VAL) into a number, the number exceeded the range -32767 to 32767.

1262: In an INP command, the size of a string variable in the input line has exceeded the allocated size of the string.

1287: In an INP command, no value has been entered to satisfy one or more of the arguments in the INP command.

*1330: The number of arguments in a DAT command exceeds the allocated dimension of the array variable.

*1339: A constant appears on the left side of an STR command.

*1347: A string constant appears on the left side of an STR command.

*1399: A variable that is not a string variable is present on left side of a CMP command.

*1510: In a DEF command, an attempt has bee.. made to redefine a constant.

*1515: A string variable or string constant is present in a DEF command.

*1540: The argument of a LEN command is not a string variable.

*1564: The argument of a VAL command is not a string variable.

VI. Sample Programs

A. Number Base Conversion and Variable Base Arithmetic

You have noticed that BASEX consistently works with decimal (base 10) numbers, as most humans do. However, assembler languages and many "monitor" programs operate in octal (base 2) or hexadecimal (base 16) notation and frequently, there is a need to convert numbers from one base to another. Why should we humans have to go through the drudgery of converting numbers from one number base to another? Texas Instruments has recently come out with a calculator that works in hexadecimal notation, so obviously, there is a need for some help with this task.

The accompanying BASEX program does it all and then some. You merely specify the number base (from 2 to 36) which you will use for entering numbers (they must be positive integers less than 32768, base 10) and the number base (from 2 to 36) that BASEX should use for printing out answers. For digits greater than "9", the letters A through Z are used; A=10, B=11, etc. As illustrated by the sample run, you can also do arithmetic and logical calculations in any number base, using the standard BASEX operators for addition (+), subtraction (-), multiplication (*), division (/), logical AND (&), inclusive OR (#) and exclusive OR (%). In compressed form (see Section VIII), this program occupies less than 1K, so you may wish to keep it in an empty corner of memory for frequent use.

```
         SIZ
PROGRAM 18944 19820    SYMBOLS 20120 20479
19820 ? LST
18944 REM !BASE CONVERSION AND VARIABLE BASE MATH!
18950 DIM V$ 10 X$ 20 C$ 1 CR$ 1
18970 STR CR$ 1 1 13
18982 *** INIT
18988 PRT "INPUT AND OUTPUT BASES"
18994 INP B1 B2
19002 *** NEXT
19008 PRT "NUMBER OR EXPRESSION"
19014 INP X$ 1
19022 LEN X$ 1
19030 SET L=A
19038 FOR J=1
19046 STR C$ 1 1 X$ J
19060 CMP C$ 1 1 "/"
19072 JMP LE OPR
19080 TIL J+1 L
19092 *** OPR
19098 DEC J
19104 STR V$ 1 J X$ 1
19118 CAL NIN
19124 SBT V-0
19132 JMP EQ INIT
19140 SBT L-J
19148 JMP EQ CNVRT
19156 SET V1=V
19164 ADD J+2
19172 STR V$ 1 10 X$ A
19186 CAL NIN
19192 ADD V1+V
19200 CMP C$ 1 1 "+"
19212 JMP EQ PRT
19220 SBT V1-V
19228 CMP C$ 1 1 "-"
19240 JMP EQ PRT
19248 MLT V1*V
19256 CMP C$ 1 1 "*"
19268 JMP EQ PRT
19276 DIV V1/V
19284 CMP C$ 1 1 "/"
19296 JMP EQ PRT
19304 IOR V1#V
19312 CMP C$ 1 1 "#"
19324 JMP EQ PRT
19332 AND V1&V
19340 CMP C$ 1 1 "&"
19352 JMP EQ PRT
19360 XOR V1%V
19368 CMP C$ 1 1 "%"
19380 JMP NE ERROR
19388 *** PRT
19394 SET V=A
```

```
19402 *** CNVRT
19408 CAL NOUT
19414 GTO NEXT
19420 *** ERROR
19426 PRT "ERROR" CR$ 1
19436 GTO NEXT
19442 REM !CONVERT V$ TO DECIMAL V!
19448 *** NI.
19454 LEN V$ 1
19462 SET N=A
19470 SET V=0
19478 FOR I=1
19486 MLT V*B1
19494 SET V=A
19502 STR C V$ I
19512 SBT C-57
19520 JMP LE NUM
19528 SBT A-7
19536 *** NUM
19542 ADD A+9
19550 ADD A+V
19558 SET V=A
19566 TIL I+1 N
19578 RET
19582 REM !PRINT V IN BASE B2!
19588 *** NOUT
19594 SET B3=1
19602 SET N=1
19610 SET C=32767
19618 *** LOOP
19624 INC N
19630 MLT B2*B3
19638 SET B3=A
19646 DIV C/B2
19654 SET C=A
19662 SBT C-B2
19670 JMP GT LOOP
19678 FOR I=1
19686 DIV V/B3
19694 SET C=A
19702 MLT A*B3
19710 SBT V-A
19718 SET V=A
19726 DIV B3/B2
19734 SET B3=A
19742 SBT C-9
19750 JMP LE OK
19758 ADD A+7
19766 *** OK
19772 ADD A+57
19780 STR X$ I I A
19792 TIL I+1 N
19804 PRT X$ 1 CR$ 1
19816 RET
19820 END
```

```
          19820 ? LSM
20120 OK 19772
20125 LOOP 19624
20132 32767 32767
20140 B3 0
20145 PRINT V IN BASE B2! 20145
20167 NUM 19542
20173 C 48
20177 I 2
20181 N 1
20185 CONVERT V$ TO DECIMAL V! 20185
20212 ERROR" 20212
20221 NOUT 19594
20228 ERROR 19426
20236 %" 20236
20241 &" 20241
20246 #" 20246
20251 *" 20251
20256 -" 20256
20261 PRT 19394
20267 +" 20267
20272 V1 0
20277 CNVRT 19408
20285 V 0
20289 NIN 19454
20295 OPR 19098
20301 /" 20301
20306 J 1
20310 L 1
20314 NUMBER OR EXPRESSION" 20314
20338 NEXT 19008
20345 B2 10
20350 B1 16
20355 INPUT AND OUTPUT BASES" 20355
20381 INIT 18988
20388 CR$ 1

20394 C$ 1 0
20399 X$ 20 0
20423 V$ 10 0
20437 BASE CONVERSION AND VARIABLE BASE MATH! 20437
```

```
19820 ? SIZ
PROGRAM 18944 19820    SYMBOLS 20120 20479
19820 ? RUN 18944
INPUT AND OUTPUT BASES ? 10 16
NUMBER OR EXPRESSION ? 678
02A6
 NUMBER OR EXPRESSION ? 256
0100
 NUMBER OR EXPRESSION ? 50+35
0055
 NUMBER OR EXPRESSION ? 879-45
0342
 NUMBER OR EXPRESSION ? 666*5
0D02
 NUMBER OR EXPRESSION ? 1998/5
018F
 NUMBER OR EXPRESSION ? 240#15
00FF
 NUMBER OR EXPRESSION ? 255&15
000F
 NUMBER OR EXPRESSION ? 255%15
00F0
 NUMBER OR EXPRESSION ? 0
INPUT AND OUTPUT BASES ? 10 2
NUMBER OR EXPRESSION ? 57
000000000111001
 NUMBER OR EXPRESSION ? 64
0000000001000000
 NUMBER OR EXPRESSION ? 255
0000000011111111
 NUMBER OR EXPRESSION ? 240
0000000011110000
 NUMBER OR EXPRESSION ? 15
0000000000001111
 NUMBER OR EXPRESSION ? 240
0000000011110000
 NUMBER OR EXPRESSION ? 240#15
0000000011111111
 NUMBER OR EXPRESSION ? 255&15
0000000000001111
 NUMBER OR EXPRESSION ? 255%15
0000000011110000
 NUMBER OR EXPRESSION ? 0
INPUT AND OUTPUT BASES ? 0
151 111 256 20347 19056 1287 19001 766 855
RANGE ? 0
PROGRAM 18944 19820    SYMBOLS 20120 20479
19820 ?
```

B. Execution Speed Benchmark Program

An article appeared in Kilobaud magazine (June, 1977, p. 66) entitled "Basic Timing Comparisons . . . Information for Speed Freaks" by Tom Rugg and Phil Feldman. A series of seven benchmark programs were run using 21 different versions of BASIC in order to assess their speed of execution. The BASEX version of their Benchmark Program 7 (the most advanced one) is given below. The execution time for BASEX was about 7.1 seconds, which compares very favorably with the times for execution in floating point BASIC, which ranged from 51.8 to 235.6 seconds. The Apple 6K integer BASIC took 28.0 seconds which is probably a better comparison, since floating point math routines take longer than integer routines. Many computer programs can be written so that floating point math is avoided and in such cases, BASEX can be used to great advantage. Of course, the relative speed advantage of BASEX depends to some extent on the program. Benchmark Program 7 was designed to exercise most of the commonly used control structures of BASIC and it is quite thorough in spite of its small size. The original BASEX compiler was written in BASIC (which partly explains the relatively simple structure of the current BASEX compiler) and its execution time was slower by a factor of about 20. It appears that BASEX excels at table searches, string matching and other tasks which are extensively used in compilers and text editors.

```
        SIZ       *** BENCHMARK PROGRAM 7
PROGRAM 13000 13152    SYMBOLS 13438 13500
13152 ? LST
13000 PRT "START"
13006 SET K=0
13014 DIM M( 5
13022 *** LOOP
13028 INC K
13034 DIV K/2
13042 MLT A*3
13050 ADD A+4
13058 SBT A-5
13066 SET B=A
13074 CAL SUB
13080 FOR L=1
13088 SET M(=L B
13098 TIL L+1 5
13110 SBT 1000-K
13118 JMP NE LOOP
13126 PRT "END"
13132 CHR 13
13138 END 13152 ? LST 13142 13148
13142 *** SUB
13148 RET
13152 ? LSM
13438 END" 13438
13445 1000 1000
13452 L 0
13456 SUB 13148
13462 B 0
13466 LOOP 13028
13473 M( 5 0 0 0 0 0
13487 K 0
13491 START" 13491
13152 ? RUN 13000
START END
RANGE ? 0
PROGRAM 13000 13152    SYMBOLS 13438 13500
13152 ? RUN 13000
START END
RANGE ? 0
PROGRAM 13000 13152    SYMBOLS 13438 13500
13152 ? LSM
13438 END" 13438
13445 1000 1000
13452 L 6
13456 SUB 13148
13462 B 1499
13466 LOOP 13028
13473 M( 5 1499 1499 1499 1499 1499
13487 K 1000
13491 START" 13491
13152 ?
```

```
*Same program in BASIC
300 PRINT "START"
400 K=0
430 DIM M(5)
500 K=K+1
510 A=K/2*3+4-5
520 GOSUB 820
530 FOR L=1 TO 5
535 M(L)=A
540 NEXT L
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
820 RETURN
```

VII. Suggestions for User Modifications

A. Details of program organization

It is relatively easy to add customized commands to BASEX, once the format of the compiler program and symbol table are understood. As an example, the compiled code for the instruction SET B(1=C would look like this:

First byte of command	205	CALL instruction
		Low byte of exeuction routine address
		High byte
		Low byte of address of array B(in symbol table
		High byte
	01	Small constant 1 is stored inline
	00	
		Low byte of address of C in symbol table
		High byte
Last byte of command	13	Special end of command token

The execution routine address is a pointer to a jump table residing at the end of the BASEX execution routines (location 1994). Each argument of the command has a corresponding 2-byte entry in the program that either points to a value in the symbol table or else it contains the value itself, in the case of small constants. Let us suppose that the array B(has

previously been dimensioned by a DIM B(2 command. The symbol table would then look like this:

First byte of symbol table	195	'C'+128
	Low byte of value	
	High byte	
	0	End of symbol token
	66	'B'
	168	'('+128
	2	Maximum subscript
	Low byte of first element	
	High byte	
	Low byte of second element	
	High byte	
Last byte of symbol table	0	End of symbol token

Notice that the program code builds upward in memory (i.e., increasing memory address), whereas the symbol table builds downward. Each argument in a new command line is evaluated first to determine whether it is a small constant. If so, its value is entered directly in the program code. If not, the symbol table is searched to determine whether that symbol name has been used before. If the name is found, the address of the byte following the last character of the symbol name is entered in the program. The last character of the symbol name is indicated by adding 128 to its Ascii value. In the case of an array or string, the address entered in the program is the address of the maximum subscript value.

B. Memory allocation for user modifications

Since the BASEX compiler itself is written in BASEX, the remarks above apply to it, as well as to other BASEX programs. The string variable VRB$ at location 8033 in the symbol table of the BASEX compiler stores the set of three-letter names of program commands. At the end of this list are four command names (US1, US2, US3, US4) which have been reserved for custom user commands. After these commands have been initialized as described here, they may be used in your programs, just like any other command. As an example, suppose you want to add a new command to negate a variable. The first step is to change the name in VRB$ to an appropriate command mnemonic like NEG. Your first custom command will replace US1 in VRB$, so you should change the characters of US1 (location 8179) to NEG. Next, you should change the address in the jump table (location 858 hex) to point to the address of your execution routine for the NEG command(prior to initialization, US1 will merely jump to location 0 and restart BASEX). About 84 bytes are available between locations 71A and 76D (hex) for user commands. If more space is needed, the line buffer (BUF) can be relocated anywhere else in memory and this will leave a total of 164 bytes free. The code for the NEG command would look like this, if we assume that NEG has a single argument of type 1 or 2 and returns a result in the accumulator A:

```
NEG POP H        Get program pointer
    CALL GETARG  Get value of argument in BC
    PUSH H       Stack program pointer
    MOV A,B      Low byte of argument
    CMA          Complement
    MOV L,A      Place result in HL
    MOV A,B      High byte
```

```
CMA          Complement
MOV H,A
INX H        Two's complement
MOV A,H
RAL          Set carry if result is negative
JMP FLAG     Store HL in A, set flags and return
```

Three compiler commands (GO1, GO2, GO3) have been reserved for user modifications and 122 bytes at the end of the BASEX compiler (locations 6724 to 6845) are available for user commands. The names of these commands can be changed by altering the VIM$ array at location 7988. Before initialization, all three commands cause a jump to location 0, which merely restarts BASEX. However, the GTO addresses at locations 6694, 6706 and 6718 may be changed to point to your machine language subroutine which should return to BASEX by jumping to location 0 (RST 0). Arguments may be passed to the user subroutine by changing the program as follows:

```
*** GO1
CAL ARGET          Get first argument in V
SET V1=V           Transfer to V1
CAL ARGET          Get second argument in V
CAL USERSUB V1 V
GTC CMND           Return to location 0 will also work
```

The protocols for passing arguments to machine language subroutines (Section III F 9) should be followed in this case.

It is also possible to implement short compiler commands entirely in BASEX, as illustrated by the following simple routines to punch and read binary programs on paper tape:

```
*** SAV              Change GO1 to SAV
FOR I=LO             LO is first byte of program
```

```
WAT STATUS READY      Wait for ready bit in status port
BRD I                 A=next byte of program
XCT PUNCH             Send A to output port
TIL I+1 ST            Loop to end of symbol table
GTC CMND
*** LOD               Change GO2 to LOD
FOR I=LO              LO is first byte
WAT STATUS READY      Wait for ready bit in status port
XIN READER            Read byte from input port
BRT I=A               Enter in program
TIL I+1 ST            Loop to end of symbol table
GTC CMND
```

All symbols beginning with the letter 'U' in the BASEX symbol table are available for the user; their names may be changed by altering the characters of the symbol name.

In general, it isn't possible to modify the BASEX compiler without creating a second version of BASEX to do the modification (see Section VIII). However, in the cases discussed above, the changes are at the end of BASEX and aren't executed until the modifications are completed. Thus, these changes can be performed by entering the appropriate range values for BASEX itself in response to the "RANGE?" query and then editting in the normal fashion.

VIII. Using the LOADER to Relocate and Compress Programs

When a BASEX program is compiled, the variable addresses in the program are tied to a particular location in the symbol table and therefore, the program and symbol table cannot be relocated by the compiler. However, the LOADER program is capable of relocating a program and/or its symbol table. The LOADER can also be used to move the BASEX compiler or the

LOADER itself to a more desireable location. In addition, the LOADER is capable of compressing debugged programs to a form which saves 20 to 25% of the original memory requirement. In the compressed form, the program cannot be changed by the BASEX compiler and it cannot be relocated again by the LOADER.

When the LOADER is started, it types a "?" and then one of the following commands can be typed (those that operate exactly like BASEX compiler commands are preceded by *):

*MON - exit to the system monitor.

*DMP - dump bytes of memory between the limits specified by the first and second arguments.

*NTR - starting at the location specified by the first argument, enter the bytes specified by the second and subsequent arguments into successive memory locations. This command echoes an extra "?" after completion.

*RUN - run the program at the location specified by the first argument.

MCV - move the block of memory between the limits specified by the first and second arguments to the memory region beginning at the location given by the third argument. Blocks can be moved upward or downward in memory without overwriting themselves.

PRG - adjust the *** command addresses in the program segment which has been MCVed to the region specified by the first and second arguments. After this command has been executed, the program can be run at its new address. The PRG command assumes that the symbol table has either remained at its original location or has previously been updated by the SYM command.

SYM - adjust all symbol table addresses in the program located between the limits given by the first two arguments to reference the relocated symbol table starting at the location specified by the fifth argument. The third and fourth arguments indicate the range of the symbol table at the time the program was compiled (or where it was relocated by a previous SYM command).

FIX - compress the program located between the first and second arguments by removing all non-executable commands (REM, DIM, ***). Also, compress the symbol table located between the third and fourth arguments to remove all comments, abbreviate symbol names to a single letter and remove the end of symbol tokens. As the symbol table is compressed the new location of the first letter of each symbol is printed and then the original symbol name and the current value of the symbol are printed. The position of a particular command in the compressed program can be determined by checking the symbol value for the nearest *** command in the original program and counting from there. The compressed program and symbol table are moved to the location specified by the fifth argument. The fifth argument should be less than or equal to the first argument or else should be greater than the fourth argument in order to avoid overwriting the old program prematurely during the compression operation. The old program is destroyed during the compression, regardless of whether it is overwritten by the new program or not.

An example will help to clarify the method for using the LOADER. Suppose that you have compiled a program between locations 10000 and 11000 with a symbol table between 11100 and 11500. You want to move the program to location 8000, with the symbol table immediately following the program. The following command sequence should be used:

```
? MOV 10000 11000 8000          Move program
MOW 8000 9000                   LOADER prints new range
? MOV 11100 11500 9001          Move symbols
MOW 9001 9401                   New range of symbol table
? SYM 8000 9000 11100 11500 9001  Update symbol table references
? PRG 8000 9000                 Update *** command
? RUN 8000                      Run the relocated program
```

If you want to move the program and symbol table and compress at the same time, a single command will do the job:

```
? FIX 10000 11000 11100 11500 8000
10800 SYMBOL 0                  List the new symbol table
10803 . . .                     More symbols . . .
? RUN 8000                      Execute the program
```

The best way of handling very large programs that will not fit in memory while the BASEX compiler is resident is to compile the program in several segments, all of which use the same symbol table. To do this, type in the first program segment until the available memory is used up and then save only the program region (not the symbols) on cassette or disk. Now, type in the next program segment, starting at the same location where you typed in the first segment, but using the same symbol table left by the previous program segment. After the second segment has been

compiled, you can read in the first program segment at location 2148 (the starting point of the BASEX compiler) and use the LOADER to MOV the second program segment down to the location just after the first segment. Of course, more than two segments can be handled in similar fashion. If desired, the symbol table can be moved down to follow the last program segment, but then the SYM command must be invoked to adjust the addresses in the entire program. The final step is to apply the PRG command to update all of the *** commands. It is also possible to compress all of the concatenated program segments and the symbol table in one operation by use of the FIX command.

If memory is tight, you may also wish to FIX the BASEX compiler and this will decrease its size by almost 1000 bytes. If you don't plan to use the FIX command, you may create a shorter version of the LOADER by inserting the command GTO ERR after the *** FIX command in the LOADER and eliminating all of the symbols above ?" in the symbol table. This shorter version occupies less than 2K bytes of memory and can be FIXed to shorten it to less than 1.5K bytes. The compressed form of the complete loader requires about 2.5K bytes of memory. The listing of the LOADER shows it positioned at location 2148, immediately following the BASEX execution routines. However, you can relocate it to any convenient position by applying the MOV, SYM, and PRG operations to the LOADER itself. If you wish to FIX the LOADER, an extra step is required due to the fact that the original program (in this case, the LOADER itself) is destroyed by the FIX operation. Thus, you should first relocate the LOADER with the MOV, SYM and PRG sequence and then FIX the resulting relocated version of the LOADER.

```
                *** BASEX Execution Routines--Version 1.0 ***
                Copyright June,1977 by Paul K. Warme
                                       423 Kemmerer Rd.
                                       State College,PA 16801
                                       Abbreviations used in this program:
                                        arg = argument
                                        char = character
                                        adr = address
                                        ptr = pointer
                                        ctr = counter
                                        ( X ) = address of X
                                        BC = register pair BC
                                        DE = register pair DE
                                        HL = register pair HL
                                        PP = program pointer
0064            0100 FL EQU 1989        FL = flag register
0064            0110 AC EQU 1992        AC = accumulator
0064            0120 PRVAL EQU 3A8HThis program is compiled in 3 segme
0064            0130 TYPE EQU 40BH  to facilitate changes on a 16K byt
0064            0140 INDX EQU 1987  computer. The definitions at the s
0064            0150 MAX EQU 1988   of each segment link them together
0064 E1         0300 SET POP H      *** SEGMENT 1
0065 5E         0305  MOV E,M       SET arg1=arg2
0066 23         0310  INX H
0067 56         0315  MOV D,M       DE=(arg1)
0068 23         0320  INX H
0069 1B         0325  DCX D         Check for array
006A 1A         0330  LDAX D        Last char of name
006B FE A8      0335  CPI 168       '('+128
006D C2 74 00   0340  JNZ NORAY
0070 CD AE 02   0345  CALL ARY      DE=(array element)
0073 1B         0350  DCX D
0074 13         0355 NORAY INX D
0075 D5         0360  PUSH D        Stack (arg1)
0076 CD 86 02   0365  CALL GETADR   DE=(arg2)
0079 E3         0370  XTHL          Stack PP, HL=(arg1)
007A 1A         0375  LDAX D        Move arg1 to arg2
007B 13         0380  INX D
007C 77         0385  MOV M,A
007D 23         0390  INX H
007E 1A         0392  LDAX D
007F 77         0394  MOV M,A
0080 C3 29 03   0396  JMP RETRN     Standard return
0083 E1         0400 FOR POP H      *Initialize loop ctr
0084 5E         0410  MOV E,M       DE=(arg1)
0085 23         0420  INX H
0086 56         0430  MOV D,M
0087 23         0440  INX H
0088 D5         0450  PUSH D        Stack (arg1)
0089 CD 6F 02   0460  CALL GETARG   BC=arg2
008C E3         0470  XTHL          Stack PP, HL=(arg1)
008D 71         0480  MOV M,C       Move arg2 to arg1
008E 23         0490  INX H
008F 70         0500  MOV M,B
0090 C3 29 03   0510  JMP RETRN
```

```
0093 E1          0520 LBL POP H      *Skip over *** or REM command
0094 23          0530  INX H        Skip 3 bytes
0095 23          0540  INX H
0096 23          0550  INX H
0097 E9          0560  PCHL         Go to start of next command
0098 E1          0570 INC POP H     *Increment arg1
0099 CD 6F 02    0575  CALL GETARG  BC=arg1, DE=(arg1)+1
009C E5          0580  PUSH H       Stack PP
009D 21 01 00    0590  LXI H,1      Add 1
00A0 09          0600  DAD B
00A1 C3 AE 00    0610  JMP RESTOR   This code same as DEC
00A4 E1          0620 DEC POP H     *Decrement arg1
00A5 CD 6F 02    0625  CALL GETARG  BC=arg1, DE=(arg1)+1
00A8 E5          0630  PUSH H       Stack PP
00A9 21 FF FF    0640  LXI H,-1     Subtract 1
00AC 09          0650  DAD B
00AD 3F          0660  CMC
00AE EB          0670 RESTOR XCHG   Put new value back
00AF 72          0680  MOV M,D
00B0 2B          0690  DCX H
00B1 73          0700  MOV M,E
00B2 EB          0705  XCHG
00B3 C3 12 03    0710  JMP FLAG     Store AC & FL
00B6 E1          0720 ADD POP H     *Add arg1+arg2
00B7 CD 6F 02    0722  CALL GETARG  BC=arg1
00BA C5          0724  PUSH B       Stack arg1
00BB CD 6F 02    0726  CALL GETARG  BC=arg2
00BE E3          0728  XTHL         Stack PP,HL=arg1
00BF 09          0730  DAD B        Add
00C0 C3 12 03    0740  JMP FLAG     Store AC & FL
00C3 E1          0750 SBT POP H     *Subtract arg1-arg2
00C4 CD 6F 02    0752  CALL GETARG  BC=arg1
00C7 C5          0754  PUSH B       Stack arg1
00C8 CD 6F 02    0756  CALL GETARG  BC=arg2
00CB E3          0758  XTHL         Stack PP, HL=arg1
00CC 7D          0760  MOV A,L      Subtract
00CD 91          0770  SUB C        *** The following multiply and
00CE 6F          0780  MOV L,A      divide routines are based on
00CF 7C          0790  MOV A,H      "Tiny Basic, Extended Version"
00D0 98          0800  SBB B        by Dick Whipple and John Arnold
00D1 67          0810  MOV H,A      Dr. Dobb's Journal of Computer
00D2 C3 12 03    0820  JMP FLAG     Calisthenics and Orthodontia,
00D5 E1          0830 MLT POP H     Volume I, No. 1, P. 19.
00D6 CD 6F 02    0832  CALL GETARG  Copyright (c) 1976 by People's
00D9 C5          0834  PUSH B       Computer Company, 1263 El Camino
00DA CD 6F 02    0836  CALL GETARG  Real, Menlo Park, CA 94025.
00DD E3          0838  XTHL         Reprinted by permission of
00DE EB          0840  XCHG         People's Computer Company
00DF 21 00 00    0850  LXI H,0      and the authors.
00E2 09          0860  DAD B        HL=arg2
00E3 06 00       0870  MVI B,0      B=sign indicator
00E5 7A          0880  MOV A,D      Check sign of arg2
00E6 B7          0890  ORA A
00E7 FC 5A 03    0900  CM INEG      If -, negate arg2, B=B+1
00EA EB          0910  XCHG         DE=arg1, HL=arg2
```

```
00EB 7A          0920  MOV A,D       Check sign of arg1
00EC B7           0930  ORA A
00ED FC 5A 03     0940  CM INEG       If -, negate arg1, B=B+1
00F0 C5           0950  PUSH B       Save sign indicator
00F1 44           0960  MOV B,H      BC=arg1
00F2 4D           0970  MOV C,L      Product ends up in BC
00F3 21 00 00     0980  LXI H,0      HL=accumulator
00F6 3E 11        0990  MVI A,17     Set up for 17 cycles
00F8 32 C3 07     1000 ML1 STA INDX  INDX=cycle counter
00FB 78           1010  MOV A,B      Shift arg1 1 bit right
00FC 1F           1020  RAR
00FD 47           1030  MOV B,A
00FE 79           1040  MOV A,C
00FF 1F           1050  RAR
0100 4F           1060  MOV C,A      Carry=rightmost bit of arg1
0101 D2 05 01     1070  JNC ML2      If 0, don't add arg2
0104 19           1080  DAD D        If 1, add arg2
0105 7C           1090 ML2 MOV A,H   Shift accumulator 1 bit right
0106 1F           1100  RAR
0107 67           1110  MOV H,A
0108 7D           1120  MOV A,L
0109 1F           1130  RAR          Bit shifted into carry here
010A 6F           1140  MOV L,A       is shifted into BC in next cycle
010B 3A C3 07     1150  LDA INDX     Decrement cycle ctr
010E 3D           1160  DCR A
010F C2 F8 00     1170  JNZ ML1      If not 0, continue
0112 50           1180  MOV D,B      Done; move product to DE
0113 59           1190  MOV E,C
0114 C1           1200 ML3 POP B     Get sign indicator
0115 05           1210  DCR B
0116 C2 1C 01     1220  JNZ ML4      Product is positive if B=0 or 2
0119 CD 5B 03     1230  CALL NEG     If only 1 arg is negative, so is
011C EB           1240 ML4 XCHG      Product in HL
011D 7C           1245  MOV A,H      A=high byte
011E 84           1250  ADD H        Set carry if negative
011F C3 12 03     1260  JMP FLAG     Store AC & FL
0122 E1           1280 DIV POP H     *Divide arg1/arg2
0123 CD 6F 02     1282  CALL GETARG  BC=arg1
0126 C5           1284  PUSH B       Stack arg1
0127 CD 6F 02     1286  CALL GETARG  BC=arg2
012A E3           1288  XTHL         Stack PP, HL=arg1
012B 50           1290  MOV D,B      DE=arg2
012C 59           1300  MOV E,C
012D 06 00        1310  MVI B,0      B=sign indicator
012F 7A           1320  MOV A,D      Check sign of arg2
0130 B7           1330  ORA A
0131 FC 5A 03     1340  CM INEG      If -, negate arg2, B=B+1
0134 EB           1350  XCHG         DE=arg1, HL=arg2
0135 7A           1360  MOV A,D      Check sign of arg1
0136 B7           1370  ORA A
0137 FC 5A 03     1380  CM INEG      If -, negate arg1, B=B+1
013A AF           1390  XRA A        Check for divide by 0
013B BC           1400  CMP H
013C C2 43 01     1410  JNZ DI1       H not 0
013F BD           1420  CMP L
```

```
0140 CC 32 03   1430  CZ ERROR      ERROR (323) divide by 0
0143 C5         1440 DI1 PUSH B     Stack sign indicator
0144 06 01      1450  MVI B,1       B counts # of subtractions requi
0146 7C         1460 DI2 MOV A,H    Check for a 1
0147 E6 40      1470  ANI 40H        in left bit of arg2
0149 C2 51 01   1480  JNZ DI3       If so, arg2 is left justified
014C 29         1490  DAD H         Shift HL left
014D 04         1500  INR B         Bump subtraction ctr
014E C3 46 01   1510  JMP DI2        and continue
0151 78         1520 DI3 MOV A,B    Save subtraction ctr
0152 32 C3 07   1530  STA INDX
0155 44         1540  MOV B,H       BC=justified divisor
0156 4D         1550  MOV C,L
0157 21 00 00   1560  LXI H,0       HL=accumulator for quotient
015A 7B         1570 DI4 MOV A,E    Subtract BC from DE
015B 91         1580  SUB C
015C 5F         1590  MOV E,A
015D 7A         1600  MOV A,D
015E 98         1610  SBB B
015F 57         1620  MOV D,A
0160 D2 7A 01   1630  JNC DI0       If BC<DE, put a 1 in quotient
0163 7B         1640  MOV A,E       Add BC to DE if BC>DE
0164 81         1650  ADD C
0165 5F         1660  MOV E,A
0166 7A         1670  MOV A,D
0167 88         1680  ADC B
0168 57         1690  MOV D,A
0169 29         1700  DAD H         Shift HL left, put 0 in quotient
016A 3A C3 07   1710 DI5 LDA INDX   Decrement subtraction ctr
016D 3D         1720  DCR A
016E CA 7F 01   1730  JZ DI7       If done, exit loop
0171 32 C3 07   1740 DI6 STA INDX   If not, store ctr
0174 EB         1750  XCHG          Shift arg1 left
0175 29         1760  DAD H
0176 EB         1770  XCHG
0177 C3 5A 01   1780  JMP DI4       Subtract again
017A 29         1790 DI0 DAD H      Shift quotient left
017B 23         1800  INX H          and insert a 1
017C C3 6A 01   1810  JMP DI5       Continue subtracting
017F EB         1820 DI7 XCHG       Done; put quotient in DE
0180 C3 14 01   1830  JMP ML3        and check sign
0183 E1         1840 AND POP H      *AND arg1&arg2
0184 CD 6F 02   1842  CALL GETARG   BC=arg1
0187 C5         1844  PUSH B        Stack arg1
0188 CD 6F 02   1846  CALL GETARG   BC=arg2
018B E3         1848  XTHL          Stack PP, HL=arg1
018C 79         1850  MOV A,C       AND low bytes
018D A5         1860  ANA L
018E 6F         1870  MOV L,A
018F 78         1880  MOV A,B       AND high bytes
0190 A4         1890  ANA H
0191 67         1900  MOV H,A
0192 C3 12 03   1910  JMP FLAG      Store AC & FL
```

```
0195 E1          1920 NOT POP H        *One's complement arg1
0196 CD 6F 02    1925  CALL GETARG     BC=arg1
0199 E5          1930  PUSH H         Stack PP
019A 79          1940  MOV A,C        Complement low byte
019B 2F          1950  CMA
019C 6F          1960  MOV L,A
019D 78          1970  MOV A,B        Complement high byte
019E 2F          1980  CMA
019F 67          1990  MOV H,A
01A0 C3 12 03    2000  JMP FLAG       Store AC & FL
01A3 E1          2010 IOR POP H       *Inclusive OR arg1+arg2
01A4 CD 6F 02    2012  CALL GETARG     BC=arg1
01A7 C5          2014  PUSH B         Stack arg1
01A8 CD 6F 02    2016  CALL GETARG     BC=arg2
01AB E3          2018  XTHL           Stack PP, HL=arg1
01AC 79          2020  MOV A,C        OR low bytes
01AD B5          2030  ORA L
01AE 6F          2040  MOV L,A
01AF 78          2050  MOV A,B        OR high bytes
01B0 B4          2060  ORA H
01B1 67          2070  MOV H,A
01B2 C3 12 03    2080  JMP FLAG       Store AC & FL
01B5 E1          2090 XOR POP H       *Exclusive OR arg1+arg2
01B6 CD 6F 02    2092  CALL GETARG     BC=arg1
01B9 C5          2094  PUSH B         Stack arg1
01BA CD 6F 02    2096  CALL GETARG     BC=arg2
01BD E3          2098  XTHL           Stack PP,HL=arg1
01BE 79          2100  MOV A,C        XOR low bytes
01BF AD          2110  XRA L
01C0 6F          2120  MOV L,A
01C1 78          2130  MOV A,B        XOR high bytes
01C2 AC          2140  XRA H
01C3 67          2150  MOV H,A
01C4 C3 12 03    2160  JMP FLAG       Store AC & FL
01C7 E1          2170 GTO POP H       *Go to arg1
01C8 CD 6F 02    2180 GT1 CALL GETARG BC=arg1
01CB C5          2190 GT2 PUSH B      Stack arg1
01CC C9          2200  RET            Use it as destination
01CD E1          2240 JMP POP H       *Jump on condition code
01CE 3A C5 07    2250  LDA FL         Get condition code flags
01D1 A6          2260  ANA M          AND with next byte in program
01D2 23          2265  INX H          Bump PP
01D3 CA DC 01    2270  JZ SKP1        If 0, condition not satisfied
01D6 A6          2280  ANA M          AND with next byte in program
01D7 23          2285  INX H          Bump PP
01D8 CA C8 01    2290  JZ GT1         If 0, condition satisfied
01DB 2B          2295  DCX H           Else back up PP
01DC 23          2300 SKP1 INX H      Condition not satisfied
01DD 23          2305  INX H          Skip jump adr
01DE 23          2310  INX H
01DF C3 2A 03    2320  JMP RET1       Return to adr in HL
01E2 E1          2330 CAL POP H       *Call subroutine arg1
01E3 CD 6F 02    2335  CALL GETARG     BC=arg1
01E6 E5          2340  PUSH H         Stack PP
01E7 C5          2345  PUSH B         Stack arg1
01E8 C9          2350  RET             and use it as destination
```

```
01E9 3E 01      2360 WRD MVI A,1    *Read a 16 bit word at arg1
01EB C3 EF 01   2362  JMP BR1       Set type to 1
01EE AF         2364 BRD XRA A      *Read a byte at arg1; type 0
01EF 32 C3 07   2366 BR1 STA INDX   INDX=word/byte indicator
01F2 E1         2368  POP H         HL=PP
01F3 CD 6F 02   2370  CALL GETARG   BC=arg1
01F6 E5         2372  PUSH H        Stack PP
01F7 0A         2374  LDAX B        A=low byte
01F8 6F         2376  MOV L,A       Save in L
01F9 3A C3 07   2378  LDA INDX      Check word/byte
01FC 67         2380  MOV H,A       H=0 if byte
01FD 3D         2382  DCR A
01FE C2 0C 03   2384  JNZ NOFLG     If byte, store HL in AC
0201 03         2386  INX B
0202 0A         2388  LDAX B        Read high byte
0203 67         2389  MOV H,A       Store in H
0204 C3 0C 03   2390  JMP NOFLG     Store HL in AC
0207 E1         2400 GET POP H      *Get arg from CALler
0208 E3         2410  XTHL          Stack PP, HL=PP of CAL
0209 CD 6F 02   2420  CALL GETARG   BC=next word after CAL
020C E3         2430  XTHL          Stack PP of CAL, HL=PP
020D CD DC 02   2440  CALL PUTARG   Transfer BC to arg1
0210 C3 2A 03   2450  JMP RET1      Continue at HL
0213 E1         2460 PUT POP H      *Return arg to CALler
0214 CD 6F 02   2465  CALL GETARG   BC=arg1
0217 E3         2470  XTHL          Stack PP, HL=next word after CAL
0218 CD DC 02   2480  CALL PUTARG   Move BC to CAL variable
021B E3         2490  XTHL          Stack PP of CAL, HL=PP
021C C3 2A 03   2500  JMP RET1      Continue at HL
021F E1         2510 RET POP H      *Return to CALling program
0220 C3 29 03   2520  JMP RETRN     Discard current PP, return to CAL
0223 E1         2522 TIL POP H      *Test for end of loop
0224 CD 6F 02   2524  CALL GETARG   BC=arg1
0227 D5         2526  PUSH D        Stack (arg1)+1
0228 C5         2528  PUSH B        Stack arg1
0229 CD 6F 02   2530  CALL GETARG   BC=arg2
022C D1         2532  POP D         DE=arg1
022D E3         2534  XTHL          Stack PP, HL=(arg1)+1
022E EB         2536  XCHG          HL=arg1, DE=(arg1)+1
022F 09         2538  DAD B         Add step size
0230 EB         2540  XCHG          HL=(arg1)+1, DE=updated loop variab
0231 72         2542  MOV M,D       Update loop variable
0232 2B         2544  DCX H
0233 73         2546  MOV M,E
0234 EB         2548  XCHG          HL=updated loop ctr
0235 E3         2550  XTHL          Stack loop ctr, HL=PP
0236 C5         2555  PUSH B        Stack step size
0237 CD 6F 02   2560  CALL GETARG   BC=arg3=end of loop
023A 50         2565  MOV D,B       Move it to DE
023B 59         2570  MOV E,C
023C C1         2575  POP B         BC=step size
023D E3         2580  XTHL          Stack PP, HL=loop ctr
023E 3E 7F      2585  MVI A,127     Check sign of step
0240 90         2590  SUB B
0241 DA 45 02   2595  JC  MNS       If -, reverse signs
```

```
0244 EB          2600  XCHG              HL=end of loop, DE=loop ctr
0245 CD 5B 03    2605 MNS CALL NEG      Negate DE
0248 19          2610  DAD D             Subtract
0249 DA 52 02    2615  JC LOOP           Not done yet
024C E1          2640 DONE POP H        HL=PP
024D 23          2650  INX H             Skip back ptr
024E 23          2660  INX H
024F C3 2A 03    2670  JMP RET1          Continue at PP
0252 E1          2680 LOOP POP H        HL=PP
0253 4E          2685  MOV C,M           BC=back ptr
0254 23          2690  INX H
0255 46          2700  MOV B,M
0256 23          2710  INX H
0257 09          2720  DAD B             Add back ptr to PP
0258 7E          2730  MOV A,M           Check valid adr
0259 FE CD       2732  CPI 205           CALL code?
025B C4 32 03    2734  CNZ ERROR         ERROR (606)-phase error in TIL
025E E9          2736  PCHL              Go to HL
025F E1          2740 DIM POP H         *DIM command is non-executable
0260 23          2750 DM1 INX H         Search for end of command
0261 7E          2760  MOV A,M
0262 FE 0D       2770  CPI 13             13 is end token
0264 C2 60 02    2780  JNZ DM1           Look again
0267 23          2790  INX H             CALL code should follow
0268 7E          2800  MOV A,M
0269 FE CD       2810  CPI 0CDH
026B C2 60 02    2820  JNZ DM1           If not, keep looking
026E E9          2830  PCHL              Go to next command
026F 5E          3060 GETARG MOV E,M    *Get next arg from HL=PP
0270 23          3070  INX H             Arg must be type 0 or 1
0271 56          3080  MOV D,M           DE=arg or (arg)
0272 23          3090  INX H
0273 AF          3100  XRA A             Is high byte 0?
0274 BA          3110  CMP D
0275 CA 83 02    3120  JZ INT            Yes=small integer
0278 3D          3130  DCR A             Is high byte -1?
0279 BA          3140  CMP D
027A CA 83 02    3150  JZ INT            Yes=small integer
027D EB          3160  XCHG              HL=(arg), DE=PP
027E 4E          3170  MOV C,M           Move arg into BC
027F 23          3180  INX H
0280 46          3190  MOV B,M
0281 EB          3200  XCHG              DE=(arg)+1, HL=PP
0282 C9          3210  RET
0283 42          3220 INT MOV B,D       Move arg into BC
0284 4B          3230  MOV C,E
0285 C9          3240  RET
0286 5E          3260 GETADR MOV E,M    *Get (arg) from HL=PP
0287 23          3270  INX H             Any type arg is OK
0288 56          3280  MOV D,M           D=high byte of arg or (arg)
0289 23          3290  INX H
028A AF          3300  XRA A             Is it 0?
028B BA          3310  CMP D
028C CA 94 02    3320  JZ CON            Yes=small integer
```

```
028F 3D          3330  DCR A          Is it -1?
0290 BA          3340  CMP D
0291 C2 9B 02    3350  JNZ NOC        Not a small integer
0294 54          3360 CON MOV D,H     DE=(arg)+2
0295 5D          3370  MOV E,L
0296 1B          3380  DCX D          DE=(arg)
0297 1B          3390  DCX D
0298 06 00       3400  MVI B,0        Arg type 0
029A C9          3410  RET
029B 1B          3420 NOC DCX D       Back up (arg)-1
029C 1A          3430  LDAX D         A=last char of name
029D FE A4       3440  CPI 164        Is it $?
029F CA AE 02    3450  JZ ARY         Yes=string
02A2 FE A8       3460  CPI 168        Is it (?
02A4 CA AE 02    3470  JZ ARY         Yes=array
02A7 DA D2 02    3475  JC TEX         ! or "=string constant
02AA 06 01       3480  MVI B,1        Others=regular variable
02AC 13          3490  INX D          Set DE=(arg)
02AD C9          3500  RET
02AE D5          3510 ARY PUSH D      Save (last char)
02AF 13          3520  INX D
02B0 1A          3530  LDAX D         A=max subscript
02B1 32 C4 07    3540  STA MAX        Save it
02B4 CD 6F 02    3550  CALL GETARG    BC=arg2=subscript
02B7 3A C4 07    3560  LDA MAX        Check illegal subscript
02BA B9          3570  CMP C
02BB DC 32 03    3580  CC ERROR       ERROR (702)-illegal subscript
02BE E3          3590  XTHL           Stack PP, HL=(last char)
02BF 7E          3595  MOV A,M        A=last char
02C0 09          3600  DAD B          Add subscript
02C1 FE A4       3602  CPI 164        Test for string
02C3 CA CC 02    3604  JZ STR         String
02C6 09          3610  DAD B          Add again for array
02C7 D1          3620  POP D          DE=PP
02C8 EB          3630  XCHG           HL=PP, DE=(arg)
02C9 06 02       3640  MVI B,2        Arg type 2
02CB C9          3650  RET
02CC 23          3750 STR INX H       HL=(specified char)
02CD D1          3760  POP D          DE=PP
02CE EB          3770  XCHG           HL=PP, DE=(arg)
02CF 06 03       3780  MVI B,3        Arg type 3
02D1 C9          3790  RET
02D2 13          3800 TEX INX D       DE=(ptr to start of string)
02D3 1A          3810  LDAX D         Get ptr in DE
02D4 13          3820  INX D
02D5 4F          3830  MOV C,A
02D6 1A          3840  LDAX D
02D7 57          3850  MOV D,A
02D8 59          3860  MOV E,C
02D9 06 04       3870  MVI B,4        Arg type 4
02DB C9          3880  RET
02DC 5E          3900 PUTARG MOV E,M  *Get (arg) from CALler
02DD 23          3910  INX H
02DE 56          3920  MOV D,M        DE=(arg)
02DF 23          3930  INX H
```

```
02E0 EB          3940  XCHG              HL=(arg), DE=PP
02E1 71          3950  MOV M,C           Move BC to (arg)
02E2 23          3960  INX H
02E3 70          3970  MOV M,B
02E4 EB          3980  XCHG              HL=PP, DE=(arg)
02E5 C9          3990  RET               *Get second subscript of string
02E6 D5          4000 GETLEN PUSH D      Stack (first char of string)
02E7 C5          4010  PUSH B            Stack arg type and first subscr
02E8 CD 86 02    4020  CALL GETADR       DE=(second subscript)
02EB 78          4030  MOV A,B           A=arg type of second subscript
02EC FE 02       4040  CPI 2             Should be 0 or 1
02EE C1          4050  POP B             BC=arg type and first subscript
02EF D4 32 03    4060  CNC ERROR         ERROR (754)-invalid subscript
02F2 1A          4070  LDAX D            A=second subscript value
02F3 47          4080  MOV B,A           Move to B
02F4 3A C4 07    4090  LDA MAX           Valid second subscript?
02F7 B8          4100  CMP B
02F8 F2 FC 02    4110  JP GE1            Yes
02FB 47          4120  MOV B,A           No-set to MAX
02FC D1          4130 GE1 POP D          Restore DE=(first char of strin
02FD C9          4140  RET               *Check for last arg
02FE CD 86 02    4150 NDCHK CALL GETADR  DE=(next arg in program)
0301 3E 0D       4160  MVI A,13          End of command token?
0303 BB          4170  CMP E
0304 C0          4180  RNZ               No-continue
0305 3E CD       4190  MVI A,205         CALL code next?
0307 BA          4200  CMP D
0308 C0          4210  RNZ               No-continue
0309 D1          4220  POP D             Dump return adr
030A 2B          4230  DCX H             Back up PP to CALL
030B E9          4240  PCHL               and go to HL=PP
030C 22 C8 07    4250 NOFLG SHLD AC      Store HL=result in AC
030F C3 29 03    4260  JMP RETRN          but don't set flags
0312 7C          4270 FLAG MOV A,H      Check for negative result
0313 17          4275  RAL
0314 3E 03       4280 FLLT MVI A,3      FL=3 means LT
0316 DA 23 03    4285  JC SHAC
0319 7C          4290  MOV A,H          Check for zero result
031A B5          4300  ORA L
031B 3E 42       4310 FLEQ MVI A,42H    FL=42 means EQ
031D CA 23 03    4320  JZ SHAC
0320 3E 02       4330 FLGT MVI A,2      FL=2 means GT
0322 00          4340  NOP
0323 22 C8 07    4350 SHAC SHLD AC      Store result in AC
0326 32 C5 07    4360 SFLG STA FL       Store flags in FL
0329 E1          4370 RETRN POP H       HL=PP
032A 23          4380 RET1 INX H        Skip end token
032B 7E          4390 RET2 MOV A,M      Is next byte CALL code?
032C FE CD       4400  CPI 205
032E C4 32 03    4410  CNZ ERROR        ERROR (817)-phase error
0331 E9          4420  PCHL              OK, go to PP
0332 E5          4430 ERROR PUSH H      *ERROR dump; save registers
0333 D5          4440  PUSH D
0334 C5          4450  PUSH B
0335 F5          4460  PUSH 6           PSW=6
0336 D1
```

```
0336 D1          4470  POP D          D=A,E=flags
0337 4A          4480  MOV C,D        Put A value in BC
0338 AF          4490  XRA A
0339 47          4500  MOV B,A
033A 57          4510  MOV D,A        Put flag value in DE
033B C5          4520  PUSH B         Stack A
033C D5          4530  PUSH D         Stack flags
033D 3E 09       4540  MVI A,9        Print ctr=9
033F 32 C3 07    4545 RGPRT STA INDX  INDX=print ctr
0342 D1          4550  POP D          DE=next register value on stack
0343 CD A8 03    4555  CALL PRVAL     Print it
0346 3E 20       4560  MVI A,' '      Print space
0348 CD 0B 04    4565  CALL TYPE
034B 3A C3 07    4570  LDA INDX       Decrease print ctr
034E 3D          4575  DCR A
034F C2 3F 03    4580  JNZ RGPRT      If more, continue
0352 3E 0D       4582  MVI A,13       Print CRLF
0354 CD 0B 04    4584  CALL TYPE
0357 C3 00 00    4590  JMP 0          Restart after error
035A 04          4610 INEG INR B     Increment sign indicator
035B 7A          4620 NEG MOV A,D    *Negate DE
035C 2F          4630  CMA
035D 57          4640  MOV D,A
035E 7B          4650  MOV A,E
035F 2F          4660  CMA
0360 5F          4670  MOV E,A
0361 13          4680  INX D          2's complement
0362 C9          4690  RET

                                      *** SEGMENT 2
                                      Statements 3060 to 4780 link thi
0064             3000 BSXG DS 20BH     to segments 1 and 3
026F             3060 GETARG DS 17H
0286             3260 GETADR DS 28H
02AE             3510 ARAY DS 2EH
02DC             3900 PUTARG DS 0AH
02E6             4000 GETLEN DS 18H
02FE             4150 NDCHK DS 0EH
030C             4250 NOFLG DS 6
0312             4270 FLAG DS 14H
0326             4360 SFLG DS 3
0329             4370 RETRN DS 1
032A             4380 RET1 DS 8
0332             4430 ERROR DS 41
035B             4620 NEG DS 8
0363             4700 BUF EQU 1902
0363             4710 B1 EQU 1982
0363             4720 B2 EQU 1984
0363             4730 PPOS EQU 1986
0363             4740 INDX EQU 1987
0363             4750 MAX EQU 1988
0363             4760 AC EQU 1992
0363             4770 COUT EQU 2900H
0363             4780 CIN EQU 2929H
```

```
0363 E1          5000 TAB POP H         *Horizontal tab to arg1
0364 CD 6F 02    5005  CALL GETARG      BC=arg1
0367 E5          5010  PUSH H          Stack PP
0368 3A C2 07    5015 TB1 LDA PPOS      PPOS=carriage position
036B 91          5020  SUB C           Compare arg1
036C D2 29 03    5025  JNC RETRN       Done
036F 3E 20       5030  MVI A,' '       Type a space
0371 CD 0B 04    5035  CALL TYPE
0374 C3 68 03    5040  JMP TB1         Loop
0377 3E 01       5050 PRT MVI A,1       *Print with space between args
0379 C3 7D 03    5060  JMP PR0
037C AF          5070 PRN XRA A         *Print with no space after arg
037D 32 C3 07    5080 PR0 STA INDX      INDX=space indicator
0380 E1          5110  POP H           HL=PP
0381 CD 86 02    5115  CALL GETADR     DE=(arg1), B=arg type
0384 E5          5120 PR1 PUSH H        Stack PP
0385 78          5130  MOV A,B         Check arg type
0386 FE 03       5140  CPI 3
0388 CA F2 03    5150  JZ PRSTR        String=3
038B D2 FE 03    5160  JNC PRTEX       String constant=4
038E EB          5170  XCHG            HL=(arg)
038F 5E          5180  MOV E,M
0390 23          5190  INX H
0391 56          5192  MOV D,M         DE=arg
0392 CD A8 03    5194  CALL PRVAL      Print value for arg types 0,1,
0395 3A C3 07    5195 DONE LDA INDX     Check space indicator
0398 B7          5196  ORA A
0399 CA A1 03    5197  JZ NO           If 0, don't space
039C 3E 20       5198  MVI A,' '
039E CD 0B 04    5199  CALL TYPE       Space
03A1 E1          5200 NO POP H          HL=PP
03A2 CD FE 02    5202  CALL NDCHK      Check for last arg
03A5 C3 84 03    5204  JMP PR1         No-continue printing
03A8 7A          5210 PRVAL MOV A,D     Print value in DE
03A9 B7          5220  ORA A           Negative value?
03AA F2 B5 03    5230  JP POS          No
03AD CD 5B 03    5240  CALL NEG        Negate DE
03B0 3E 2D       5250  MVI A,'-'       Print minus sign
03B2 CD 0B 04    5260  CALL TYPE
03B5 0E 00       5270 POS MVI C,0       C=leading zero suppressor
03B7 EB          5280  XCHG            HL=value
03B8 11 F0 D8    5290  LXI D,-10000    Break value down by decades
03BB CD D7 03    5300  CALL CNVRT      Type 10000's digit
03BE 11 18 FC    5310  LXI D, -1000
03C1 CD D7 03    5320  CALL CNVRT      1000's digit
03C4 11 9C FF    5330  LXI D,-100
03C7 CD D7 03    5340  CALL CNVRT      100's digit
03CA 11 F6 FF    5350  LXI D,-10
03CD CD D7 03    5360  CALL CNVRT      10's digit
03D0 7D          5370  MOV A,L
03D1 C6 30       5380  ADI '0'
03D3 CD 0B 04    5390  CALL TYPE       1's digit
03D6 C9          5400  RET
03D7 06 FF       5410 CNVRT MVI B,255   B=digit value accumulator
```

```
03D9 04          5420 CNV1 INR B     Bump digit value
03DA 19          5430  DAD D         Subtract DE
03DB DA D9 03    5440  JC CNV1        until negative
03DE CD 5B 03    5450  CALL NEG      Negate DE
03E1 19          5460  DAD D         Add DE
03E2 78          5470  MOV A,B       Suppress leading 0's
03E3 B9          5480  CMP C         Is digit 0?
03E4 C8          5490  RZ            Yes
03E5 0D          5500  DCR C         Reset leading 0 suppressor
03E6 C6 30       5510  ADI '0'       Convert to Ascii
03E8 CD 0B 04    5520  CALL TYPE     Type digit
03EB C9          5530  RET
03EC             5540  DS 6          Unused space
03F2 1A          5640 PRSTR LDAX D   *Print a string
03F3 B7          5650  ORA A         Check for end of string
03F4 CA 95 03    5660  JZ DONE
03F7 CD 0B 04    5670  CALL TYPE     Type character
03FA 13          5680  INX D         Bump pointer
03FB C3 F2 03    5690  JMP PRSTR      and loop
03FE 1A          5700 PRTEX LDAX D   *Print a string constant
03FF FE 80       5710  CPI 128       Last char >128?
0401 D2 95 03    5720  JNC DONE
0404 CD 0B 04    5730  CALL TYPE     Type char
0407 13          5740  INX D         Bump ptr
0408 C3 FE 03    5750  JMP PRTEX      and loop
040B C5          5760 TYPE PUSH B    Type char in A
040C 47          5770  MOV B,A       Char in b
040D CD 00 29    5780  CALL COUT     *User defined subroutine
0410 FE 0D       5790  CPI 13        Carriage return?
0412 CA 1C 04    5800  JZ TY1        Yes
0415 3A C2 07    5810  LDA PPOS      Bump carriage position
0418 3C          5820  INR A
0419 C3 25 04    5830  JMP TY2
041C 06 0A       5840 TY1 MVI B,10   Type line feed after CR
041E CD 00 29    5850  CALL COUT     *User defined subroutine
0421 06 0D       5860  MVI B,13      Restore CR in B
0423 3E 01       5870  MVI A,1       Reset carriage position
0425 32 C2 07    5880 TY2 STA PPOS   Store it
0428 DE 51       5882  SBI 81        Beyond 80 char?
042A FA 35 04    5884  JM TY3        No
042D 06 0D       5886  MVI B,13      Yes; type CR
042F CD 00 29    5888  CALL COUT     *User defined subroutine
0432 C3 1C 04    5889  JMP TY1        and LF
0435 78          5890 TY3 MOV A,B    Put char in B
0436 C1          5900  POP B         Restore original BC
0437 C9          5910  RET
0438 3E 3F       5940 INP MVI A,'?'  *Input values from terminal
043A CD 0B 04    5950  CALL TYPE     Type '?'
043D 3E 20       5960  MVI A,' '      and a space
043F CD 0B 04    5970  CALL TYPE
0442 21 6E 07    5980 IN0 LXI H,BUF  *BUF location may be altered
0445 22 BE 07    5985  SHLD B1       B1=start of buffer ptr
0448 06 00       5990  MVI B,0       B=char ctr
044A CD 29 29    6000 IN1 CALL CIN   *User defined subroutine
```

```
044D FE 18      6010  CPI 24        Control X?
044F C2 5A 04   6020  JNZ IN2       No
0452 3E 0D      6030  MVI A,13      Yes; type CRLF
0454 CD 0B 04   6040  CALL TYPE
0457 C3 38 04   6050  JMP INP       Accept new line from start
045A CD 0B 04   6060 IN2 CALL TYPE  Echo char
045D FE 0D      6070  CPI 13        CRLF?
045F CA 78 04   6080  JZ EOL       Yes; end of line
0462 FE 5F      6090  CPI 95        Backspace?
0464 C2 6C 04   6100  JNZ IN3       No
0467 05         6110  DCR B         Back up char ctr
0468 2B         6120  DCX H          and BUF ptr
0469 C3 4A 04   6130  JMP IN1       Next char
046C 04         6140 IN3 INR B      Bump char ctr
046D 77         6150  MOV M,A       Move char to BUF
046E 23         6160  INX H         Bump BUF ptr
046F 3E 50      6170  MVI A,80      Line >80 char?
0471 B8         6180  CMP B
0472 D2 4A 04   6190  JNC IN1       No; accept more
0475 CD 32 03   6200  CALL ERROR    ERROR (1144) line >80 char
0478 36 2C      6210 EOL MVI M,',' Insert comma at end of line
047A 22 C0 07   6220  SHLD B2      B2=end of buffer ptr
047D E1         6225  POP H         HL=PP
047E CD 86 02   6230  CALL GETADR   DE=(arg), B=arg type
0481 E5         6240  PUSH H        Stack PP
0482 2A BE 07   6250 IN4 LHLD B1    HL=start of BUF ptr
0485 78         6260  MOV A,B       B=arg type
0486 B7         6270  ORA A         Small constant?
0487 CC 32 03   6280  CZ ERROR      ERROR (1162) tried to alter constar
048A DE 03      6290  SBI 3
048C CA D7 04   6300  JZ INSTR      Type 3=string
048F F4 32 03   6310  CP ERROR      ERROR (1170) tried to alter str cor
0492 7E         6320  MOV A,M       A=first char
0493 FE 2D      6330  CPI '-'       Minus sign?
0495 CA A1 04   6340  JZ MINUS      Yes
0498 CD AE 04   6350  CALL INTIN    Translate input to binary
049B 73         6360  MOV M,E       Store value at (arg)
049C 23         6362  INX H
049D 72         6364  MOV M,D
049E C3 F4 04   6370  JMP INX       Next arg
04A1 23         6380 MINUS INX H    Bump BUF ptr
04A2 CD AE 04   6390  CALL INTIN    Convert input to binary
04A5 CD 5B 03   6400  CALL NEG      Negate value
04A8 73         6410  MOV M,E       Store value at (arg)
04A9 23         6412  INX H
04AA 72         6414  MOV M,D
04AB C3 F4 04   6420  JMP INX       Next arg
04AE D5         6430 INTIN PUSH D   Stack (arg)
04AF EB         6440  XCHG          DE=BUF adr
04B0 21 00 00   6450  LXI H,0       HL=value accumulator
04B3 1A         6460 INXT LDAX D    A=next char in BUF
04B4 13         6465  INX D
04B5 FE 30      6470  CPI '0'       <0?
04B7 DA D1 04   6480  JC IDON       Yes; done
04BA FE 3A      6490  CPI '9'+1     >9?
04BC D2 D1 04   6500  JNC IDON      Yes; done
```

```
04BF E6 0F      6510  ANI 15        Strip Ascii bits
04C1 44         6520  MOV B,H       BC=HL
04C2 4D         6530  MOV C,L       Multiply HL*10
04C3 29         6540  DAD H         *2
04C4 29         6550  DAD H         *4
04C5 09         6560  DAD B         *5
04C6 29         6570  DAD H         *10
04C7 DC 32 03   6580  CC ERROR      ERROR (1226) input value too large
04CA 4F         6581  MOV C,A       Add new digit
04CB 06 00      6582  MVI B,0
04CD 09         6583  DAD B
04CE C3 B3 04   6584  JMP INXT      Next digit
04D1 EB         6586 IDON XCHG      HL=BUF ptr, DE=value
04D2 22 BE 07   6587  SHLD B1       Update start of BUF
04D5 E1         6588  POP H         HL=PP
04D6 C9         6589  RET
04D7 7E         6590 INSTR MOV A,M  A=next char in BUF
04D8 23         6600  INX H         Bump BUF ptr
04D9 FE 2C      6610  CPI ','       End of string?
04DB CA EF 04   6620  JZ STDON     Yes
04DE 12         6630  STAX D        Move char into string
04DF 13         6640  INX D         Bump string ptr
04E0 0C         6650  INR C         Bump char ctr
04E1 3A C4 07   6660  LDA MAX      Check for string overflow
04E4 B9         6670  CMP C
04E5 D2 D7 04   6680  JNC INSTR     Not yet
04E8 7E         6690  MOV A,M       Next char must be ','
04E9 FE 2C      6700  CPI ','        to avoid overflow
04EB C4 32 03   6710  CNZ ERROR     ERROR (1262) input string overflow
04EE 23         6720  INX H         Bump BUF ptr
04EF 22 BE 07   6730 STDON SHLD B1  Update start of BUF ptr
04F2 AF         6732  XRA A         Put 0 in next char
04F3 12         6734  STAX D        to signal end of string
04F4 E1         6740 INX POP H      Next arg; HL=PP
04F5 CD FE 02   6745  CALL NDCHK    Check for last arg
04F8 E5         6750  PUSH H       Not yet; stack PP
04F9 3A BE 07   6760  LDA B1
04FC 6F         6770  MOV L,A       L=start of BUF ptr
04FD 3A C0 07   6780  LDA B2        A=end of BUF ptr
0500 BD         6790  CMP L         Is start>end?
0501 D2 82 04   6800  JNC IN4       No
0504 CD 32 03   6810  CALL ERROR    ERROR (1287) missing argument
0507 E1         6820 DAT POP H      *Initialize array values
0508 CD 86 02   6825  CALL GETADR   DE=(arg) BC=arg type & subscript
050B C5         6830 DA1 PUSH B     Stack arg type & subscript
050C D5         6840  PUSH D        Stack (array element)
050D CD 6F 02   6850  CALL GETARG   BC=next value
0510 D1         6860  POP D         DE=(array element)
0511 79         6870  MOV A,C       Move value into array
0512 12         6880  STAX D
0513 13         6890  INX D
0514 78         6900  MOV A,B
0515 12         6910  STAX D
0516 13         6920  INX D
```

```
0517 C1          6930  POP B          BC=arg type & subscript
0518 7E          6940  MOV A,M        A=next byte in prog
0519 FE 0D       6950  CPI 13         End of command token?
051B C2 27 05    6960  JNZ MORE       No; get another value
051E 23          6970  INX H          Look at next byte
051F 7E          6980  MOV A,M
0520 FE CD       6990  CPI 205        CALL code?
0522 2B          7000  DCX H          HL points to CALL
0523 C2 27 05    7010  JNZ MORE       No; continue
0526 E9          7020  PCHL           Next command
0527 0C          7021 MORE INR C      Bump subscript
0528 3A C4 07    7023  LDA MAX        Check for array overflow
052B B9          7025  CMP C
052C D2 0B 05    7027  JNC DA1        Not yet; continue
052F CD 32 03    7029  CALL ERROR     ERROR (1330) array overflow
0532 E1          7030 STR POP H       *String equate;HL=PP
0533 CD 86 02    7035  CALL GETADR    DE=(arg1), BC=arg type
0536 78          7040  MOV A,B        Check valid arg type
0537 B7          7050  ORA A
0538 CC 32 03    7060  CZ ERROR       ERROR (1339) constant left of STR
053B DE 03       7070  SBI 3          Type 3=string?
053D CA 50 05    7080  JZ ST1         Yes
0540 F4 32 03    7090  CP ERROR       ERROR (1347) str const left of STR
0543 D5          7100  PUSH D         Must be type 1 or 2; stack(destinati
0544 CD 86 02    7110  CALL GETADR    DE=(source)
0547 E3          7120  XTHL           Stack PP, HL=(destination)
0548 1A          7130  LDAX D         A=low byte of source
0549 77          7140  MOV M,A        Move to destination
054A 23          7150  INX H          Bump destination adr
054B 36 00       7160  MVI M,0        Put 0 in high byte of destination
054D C3 29 03    7170  JMP RETRN      Exit
0550 CD E6 02    7180 ST1 CALL GETLEN BC=last & first char to change
0553 D5          7190  PUSH D         Stack (first char to enter)
0554 C5          7200  PUSH B         Stack subscript range
0555 CD 86 02    7210  CALL GETADR    DE=(source)
0558 C1          7220  POP B          BC=string subscript range
0559 E3          7230  XTHL           Stack PP, HL=(first char to enter)
055A 1A          7240 ST2 LDAX D      A=next char from source
055B 13          7250  INX D          Bump source ptr
055C 77          7260  MOV M,A        Transfer to string
055D B7          7262  ORA A          Check end of string
055E CA 29 03    7264  JZ RETRN       If so, quit
0561 23          7270  INX H          Bump string ptr
0562 0C          7280  INR C          Bump string subscript
0563 78          7290  MOV A,B        A=last char to transfer
0564 B9          7300  CMP C          There yet?
0565 D2 5A 05    7310  JNC ST2        No; continue
0568 36 00       7315  MVI M,0        Insert end of string token
056A C3 29 03    7320  JMP RETRN
056D E1          7330 CMP POP H       *Compare strings
056E CD 86 02    7335  CALL GETADR    DE=(left arg) BC=arg type,subscrip
0571 78          7340  MOV A,B        Check valid arg type
0572 FE 03       7350  CPI 3          Type 3=string?
0574 C4 32 03    7360  CNZ ERROR      ERROR (1399) left side of CMP not s
```

```
0577 CD E6 02   7370  CALL GETLEN   BC=last & first subscripts
057A D5         7380  PUSH D       Stack (left arg)
057B C5         7390  PUSH B       Stack subscripts
057C CD 86 02   7400  CALL GETADR  DE=(right arg)
057F C1         7410  POP B        BC=subscripts for string
0580 E3         7420  XTHL         Stack PP, HL=(left arg)
0581 EB         7425  XCHG         HL=(right arg), DE=(left arg)
0582 1A         7430 CM1 LDAX D    A=next char of left arg
0583 13         7440  INX D        Bump left ptr
0584 BE         7450  CMP M        Same as right arg?
0585 C2 94 05   7460  JNZ NEQ      No; exit
0588 23         7470  INX H        Yes-bump right ptr
0589 0C         7480  INR C        Bump char ctr
058A 78         7490  MOV A,B      Test for last char
058B B9         7500  CMP C         to be compared
058C D2 82 05   7510  JNC CM1      No-go on
058F 3E 42      7520  MVI A,42H    Yes-strings are equal
0591 C3 26 03   7530  JMP SFLG     Store FL
0594 3E 02      7540 NEQ MVI A,2   Reset LT flag
0596 D2 26 03   7550  JNC SFLG     Store in FL if left>right
0599 3C         7560  INR A        Otherwise set LT flag
059A C3 26 03   7570  JMP SFLG      and store in FL
059D 3E 01      7572 WRT MVI A,1   *Write a word
059F C3 A3 05   7574  JMP BW1
05A2 AF         7576 BRT XRA A     *Write a byte
05A3 32 C3 07   7578 BW1 STA INDX  INDX=word/byte indicator
05A6 E1         7580  POP H       HL=PP
05A7 CD 6F 02   7585  CALL GETARG  BC=(first byte location)
05AA C5         7590  PUSH B      Stack (destination)
05AB CD 6F 02   7600 PO1 CALL GETARG BC=next value to be written
05AE 7B         7610  MOV A,E     End of command token?
05AF FE 0E      7620  CPI 14
05B1 C2 BA 05   7630  JNZ PO2      No-transfer
05B4 7A         7640  MOV A,D      CALL code?
05B5 FE CD      7650  CPI 205
05B7 CA CA 05   7660  JZ PO3      Yes-done
05BA E3         7670 PO2 XTHL      Stack PP HL=(destination)
05BB 71         7680  MOV M,C      Transfer low byte
05BC 23         7690  INX H        Bump (destination)
05BD 3A C3 07   7692  LDA INDX     Check word/byte
05C0 3D         7694  DCR A        1?
05C1 C2 C6 05   7696  JNZ PO4      No-byte transfer
05C4 70         7698  MOV M,B      Yes-move high byte of word
05C5 23         7700  INX H        Bump (destination)
05C6 E3         7702 PO4 XTHL      Stack (destination) HL=PP
05C7 C3 AB 05   7710  JMP PO1      Next arg
05CA D1         7720 PO3 POP D     Clear stack
05CB 2B         7730  DCX H        Set PP to CALL
05CC E9         7740  PCHL         Next command
05CD E1         7810 ABS POP H     *Return absolute value of arg
05CE CD 6F 02   7815  CALL GETARG  BC=arg
05D1 E5         7820  PUSH H       Stack PP
05D2 50         7830  MOV D,B      Move arg to DE
05D3 59         7835  MOV E,C
05D4 7A         7840  MOV A,D      Check sign of arg
```

```
05D5 17          7845  RAL          Carry=sign bit
05D6 DC 5B 03    7850  CC NEG       If -, negate arg
05D9 EB          7860  XCHG         HL=abs value
05DA C3 0C 03    7870  JMP NOFLG    Store HL in AC
05DD E1          7880 DEF POP H     *Define multiple variables
05DE CD 86 02    7885  CALL GETADR  DE=(arg1)
05E1 78          7890 DE1 MOV A,B   A=arg type
05E2 B7          7900  ORA A        Type 0?
05E3 CC 32 03    7910  CZ ERROR     ERROR (1510)-attempt to redef cons
05E6 DE 03       7920  SBI 3        Arg type 3 or 4?
05E8 D4 32 03    7930  CNC ERROR    ERROR (1515)-string or str const
05EB D5          7940  PUSH D       Stack (arg1)
05EC CD 6F 02    7950  CALL GETARG  BC=arg2
05EF E3          7960  XTHL         Stack PP HL=(arg1)
05F0 71          7970  MOV M,C      Move arg2 to arg1
05F1 23          7980  INX H
05F2 70          7990  MOV M,B
05F3 E1          7995  POP H        HL=PP
05F4 CD FE 02    8000  CALL NDCHK   Check for last arg
05F7 C3 E1 05    8010  JMP DE1      NO-continue
05FA E1          8020 LEN POP H     *Find length of string variable
05FB CD 86 02    8025  CALL GETADR  DE=(arg1) BC=arg type & subscript
05FE 78          8030  MOV A,B      Check arg type
05FF DE 03       8040  SBI 3        Type 3=string?
0601 C4 32 03    8050  CNZ ERROR    ERROR (1540)-nonstring in LEN
0604 E5          8060  PUSH H       Stack PP
0605 67          8070  MOV H,A      H=0=end of string token
0606 6F          8080  MOV L,A      L=0=char ctr
0607 1A          8090 LE1 LDAX D    A=next char of string
0608 BC          8100  CMP H        0?
0609 CA 0C 03    8110  JZ NOFLG     Yes-store HL in AC
060C 2C          8120  INR L        Bump char ctr
060D 13          8130  INX D        Bump string ptr
060E C3 07 06    8140  JMP LE1      Loop
0611 E1          8150 VAL POP H     *Convert string to number
0612 CD 86 02    8160  CALL GETADR  DE=(arg1) BC=arg type & subscript
0615 E5          8170  PUSH H       Stack PP
0616 3E 03       8180  MVI A,3      Check for string type 3
0618 B8          8190  CMP B
0619 C4 32 03    8200  CNZ ERROR    ERROR (1564)-nonstring in VAL
061C EB          8210  XCHG         HL=(arg1) DE=PP
061D 7E          8220  MOV A,M      A=first char
061E FE 2D       8230  CPI '-'      Minus sign?
0620 CA 37 06    8240  JZ NGTV      Yes
0623 CD AE 04    8250  CALL INTIN   Convert string to #
0626 EB          8260  XCHG         HL=number
0627 C3 0C 03    8270  JMP NOFLG    Store AC=HL
062A             8275  DS 13        Unused space
0637 23          8280 NGTV INX H    Bump string ptr
0638 CD AE 04    8290  CALL INTIN   Convert string to #
063B CD 5B 03    8300  CALL NEG     Negate result
063E EB          8310  XCHG         HL=number
063F C3 0C 03    8320  JMP NOFLG    Store AC=HL
```

```
0642 E1          8330 XIN POP H     *Input byte from port X to AC
0643 CD 6F 02    8340  CALL GETARG  C=port #
0646 79          8350  MOV A,C
0647 32 4B 06    8360  STA $+1      Move to IN instruction
064A DB 00       8370  IN 0         Input byte
064C 32 C8 07    8380  STA AC       Store in low byte of AC
064F AF          8390  XRA A
0650 32 C9 07    8400  STA AC+1     High byte of AC=0
0653 C3 2A 03    8410  JMP RET1
0656 E1          8420 XOT POP H     *Output AC to port X
0657 CD 6F 02    8430  CALL GETARG  C=port #
065A 79          8440  MOV A,C
065B 32 62 06    8450  STA $+4      Move to OUT instruction
065E 3A C8 07    8460  LDA AC       Get AC low byte
0661 D3 00       8470  OUT 0         Output byte
0663 C3 2A 03    8480  JMP RET1
0666 E1          8490 WAT POP H     *Wait on status of port X
0667 CD 6F 02    8500  CALL GETARG  C=port #
066A C5          8510  PUSH B       Stack port #
066B CD 6F 02    8520  CALL GETARG  BC=AND mask
066E E3          8530  XTHL         Stack PP HL=port #
066F 7D          8540  MOV A,L      Stash port #
0670 32 74 06    8550  STA $+1       in IN instruction
0673 DB 00       8560 W1 IN 0       Input status byte
0675 A1          8570  ANA C        AND with mask
0676 CA 73 06    8580  JZ W1        Loop until not 0
0679 C3 29 03    8590  JMP RETRN
067C F3          8600 DSI DI        *Disable interrupts
067D C3 29 03    8610  JMP RETRN
0680 FB          8620 ENI EI        *Enable interrupts
0681 C3 29 03    8630  JMP RETRN
0684 E1          8640 MOV POP H     *Move a block of memory
0685 CD 6F 02    8650  CALL GETARG  BC=start of block
0688 C5          8660  PUSH B       Stack start
0689 CD 6F 02    8670  CALL GETARG  BC=end of block
068C C5          8680  PUSH B       Stack end
068D CD 6F 02    8690  CALL GETARG  BC=new location
0690 D1          8700  POP D        DE=end
0691 E3          8710  XTHL         Stack PP HL=start
0692 22 BE 07    8720  SHLD B1      Save start
0695 EB          8730  XCHG         DE=start HL=end
0696 CD 5B 03    8740  CALL NEG     Negate start
0699 EB          8750  XCHG         DE=end HL=-start
069A 09          8760  DAD B        HL=new location-start
069B DA B3 06    8770  JC UP        Negative; move to higher adr
069E 2A BE 07    8780  LHLD B1      Recall start
06A1 13          8790  INX D        Bump end ptr
06A2 7E          8800 DN MOV A,M    A=next byte
06A3 23          8810  INX H        Bump old block ptr
06A4 02          8820  STAX B       Store in new
06A5 03          8830  INX B        Bump new block ptr
06A6 7D          8840  MOV A,L      Check for end
06A7 93          8850  SUB E        Low byte correct?
06A8 C2 A2 06    8860  JNZ DN       No-continue
```

```
06AB 7C          8870  MOV A,H      High byte correct?
06AC 9A          8880  SBB D
06AD C2 A2 06    8890  JNZ DN       No-continue
06B0 C3 29 03    8900  JMP RETRN    Done
06B3 19          8910 UP DAD D     HL=last byte of new block
06B4 44          8920  MOV B,H      Move to BC
06B5 4D          8930  MOV C,L
06B6 2A BE 07    8940  LHLD B1      Recall start
06B9 EB          8950  XCHG         DE=start HL=end of old block
06BA 1B          8960  DCX D        Back up start
06BB 7E          8970 UP1 MOV A,M   A=next byte
06BC 2B          8980  DCX H        Back up old ptr
06BD 02          8990  STAX B       Store byte in new block
06BE 0B          9000  DCX B        Back up new ptr
06BF 7D          9010  MOV A,L      Check for end
06C0 93          9020  SUB E        Low byte correct?
06C1 C2 BB 06    9030  JNZ UP1      No-continue
06C4 7C          9040  MOV A,H
06C5 9A          9050  SBB D        High byte correct?
06C6 C2 BB 06    9060  JNZ UP1      No-continue
06C9 C3 29 03    9070  JMP RETRN    Done
```

```
0064            0300 SET DS 1FH  *** SEGMENT 3
0083            0400 FOR DS 10H Statements 300 to 8640 link this segme
0093            0520 LBL DS 5     to segments 1 and 2
0098            0570 INC DS 12
00A4            0620 DEC DS 18
00B6            0720 ADD DS 13
00C3            0750 SBT DS 18
00D5            0830 MLT DS 4DH
0122            1280 DIV DS 61H
0183            1840 AND DS 18
0195            1920 NOT DS 14
01A3            2010 IOR DS 18
01B5            2090 XOR DS 18
01C7            2170 GTO DS 6
01CD            2240 JMP DS 21
01E2            2330 CAL DS 7
01E9            2360 WRD DS 5
01EE            2364 BRD DS 19H
0207            2400 GET DS 12
0213            2460 PUT DS 12
021F            2510 RET DS 4
0223            2522 TIL DS 3CH
025F            2740 DIM DS 16
026F            3060 GETARG DS 17H
0286            3260 GETADR DS 28H
02AE            3510 ARAY DS 2EH
02DC            3900 PUTARG DS 0AH
02E6            4000 GETLEN DS 18H
02FE            4150 NDCHK DS 0EH
030C            4250 NOFLG DS 6
0312            4270 FLAG DS 14H
0326            4360 SFLG DS 3
0329            4370 RETRN DS 1
032A            4380 RET1 DS 8
0332            4430 ERROR DS 41
035B            4620 NEG DS 8
0363            5000 TAB DS 20
0377            5050 PRT DS 5
037C            5070 PRN DS 2CH
03A8            5210 PRVAL DS 63H
040B            5760 TYPE DS 2DH
0438            5940 INP DS 0CFH
0507            6820 DAT DS 2BH
0532            7030 STR DS 3BH
056D            7330 CMP DS 30H
059D            7572 WRT DS 5
05A2            7576 BRT DS 2BH
05CD            7810 ABS DS 16
05DD            7880 DEF DS 1DH
05FA            8020 LEN DS 23
0611            8150 VAL DS 31H
0642            8330 XIN DS 20
0656            8420 XOT DS 16
0666            8490 WAT DS 22
067C            8600 DSI DS 4
0680            8620 ENI DS 4
0684            8640 MOV DS 48H
```

```
06CC 3E 01      9110 WSR MVI A,1   *Search for a word
06CE C3 D2 06   9120  JMP SR0
06D1 AF         9130 BSR XRA A     *Search for a byte
06D2 32 C3 07   9140 SR0 STA INDX  INDX=word/byte indicator
06D5 E1         9150  POP H        HL=PP
06D6 CD 6F 02   9160  CALL GETARG  BC=start of search
06D9 C5         9170  PUSH B       Stack start
06DA CD 6F 02   9180  CALL GETARG  BC=end of search
06DD C5         9190  PUSH B       Stack end
06DE CD 6F 02   9200  CALL GETARG  BC=object of search
06E1 D1         9210  POP D        DE=end
06E2 13         9220  INX D        Bump end
06E3 CD 5B 03   9225  CALL NEG     Negate end+1
06E6 E3         9230  XTHL         Stack -(end+1) HL=start
06E7 7E         9240 SR1 MOV A,M   A=next byte
06E8 B9         9250  CMP C        Same as low byte of object?
06E9 3A C3 07   9260  LDA INDX     A=word/byte indicator
06EC CA FF 06   9270  JZ SR3       Yes-check high byte
06EF B7         9280  ORA A        Byte search?
06F0 CA F4 06   9290  JZ SR2       Yes
06F3 23         9300  INX H        Skip next byte for word search
06F4 23         9310 SR2 INX H     Bump block ptr
06F5 E5         9320  PUSH H       Stack it
06F6 19         9330  DAD D        Subtract end+1
06F7 E1         9340  POP H        Recall block ptr
06F8 D2 E7 06   9350  JNC SR1      Not done yet
06FB 3F         9380  CMC          Unsuccessful search-reset LT flag
06FC C3 20 03   9390  JMP FLGT     Store block ptr in AC, set FL
06FF B7         9400 SR3 ORA A     Byte search?
0700 CA 0A 07   9410  JZ SR4       Yes-object is found
0703 23         9420  INX H        Bump block ptr
0704 7E         9430  MOV A,M      A=high byte
0705 B8         9440  CMP B        Same as object?
0706 C2 F4 06   9450  JNZ SR2      No-skip on
0709 2B         9460  DCX H        Back up ptr to low byte
070A 37         9470 SR4 STC       Successful search-set LT flag
070B C3 14 03   9490  JMP FLLT     Store AC & FL
070E E1         9500 CHR POP H     *type a character
070F CD 6F 02   9520  CALL GETARG  BC=char
0712 E5         9530  PUSH H       Stack PP
0713 79         9540  MOV A,C      Type char
0714 CD 0B 04   9550  CALL TYPE
0717 C3 29 03   9560  JMP RETRN
071A            9700 EMPTY DS 84   ***Space for user modifications
076E            9710 BUF DS 80     Line buffer
07BE            9720 B1 DS 2
07C0            9730 B2 DS 2
07C2            9740 PPOS DS 1     Carriage position
07C3            9750 INDX DS 1
07C4            9760 MAX DS 1      Maximum subscript
07C5 00 00      9770 FL DW 0       Flags
07C7 C1         9780  DB 'A'+128   This makes A printable as a symbol
07C8 00 00      9790 AC DW 0       Accumulator
```

```
07CA C3 64 00    9901  JMP SET ***Jump table for BASEX
07CD C3 93 00    9902  JMP LBL
07D0 C3 98 00    9903  JMP INC
07D3 C3 A4 00    9904  JMP DEC
07D6 C3 B6 00    9905  JMP ADD
07D9 C3 C3 00    9906  JMP SBT
07DC C3 D5 00    9907  JMP MLT
07DF C3 22 01    9908  JMP DIV
07E2 C3 83 01    9909  JMP AND
07E5 C3 95 01    9910  JMP NOT
07E8 C3 A3 01    9911  JMP IOR
07EB C3 B5 01    9912  JMP XOR
07EE C3 C7 01    9913  JMP GTO
07F1 C3 CD 01    9914  JMP JMP
07F4 C3 E2 01    9915  JMP CAL
07F7 C3 E9 01    9916  JMP WRD
07FA C3 07 02    9917  JMP GET
07FD C3 13 02    9918  JMP PUT
0800 C3 1F 02    9919  JMP RET
0803 C3 83 00    9920  JMP FOR
0806 C3 23 02    9921  JMP TIL
0809 C3 5F 02    9922  JMP DIM
080C C3 EE 01    9923  JMP BRD
080F C3 77 03    9924  JMP PRT
0812 C3 38 04    9925  JMP INP
0815 C3 93 00    9926  JMP LBL
0818 C3 00 00    9927  JMP 0
081B C3 07 05    9928  JMP DAT
081E C3 32 05    9929  JMP STR
0821 C3 6D 05    9930  JMP CMP
0824 C3 9D 05    9931  JMP WRT
0827 C3 A2 05    9932  JMP BRT
082A C3 CD 05    9933  JMP ABS
082D C3 DD 05    9934  JMP DEF
0830 C3 FA 05    9935  JMP LEN
0833 C3 11 06    9936  JMP VAL
0836 C3 7C 03    9937  JMP PRN
0839 C3 63 03    9938  JMP TAB
083C C3 42 06    9939  JMP XIN
083F C3 56 06    9940  JMP XOT
0842 C3 66 06    9941  JMP WAT
0845 C3 7C 06    9942  JMP DSI
0848 C3 80 06    9943  JMP ENI
084B C3 84 06    9944  JMP MOV
084E C3 CC 06    9945  JMP WSR
0851 C3 D1 06    9946  JMP BSR
0854 C3 0E 07    9947  JMP CHR
0857 C3 00 00 U  9948  JMP US1
085A C3 00 00 U  9949  JMP US2
085D C3 00 00 U  9950  JMP US3
0860 C3 00 00 U  9951  JMP US4
0863             9998 COUT EQU 2900H
0863             9999 CIN EQU 2929H
```

```
                    *** BASEX Compiler Version 1.0 ***
                        Copyright June, 1977 by                    72
                          Paul K. Warme
                          423 Kemmerer Rd.
                          State College, PA 16801
PROGRAM 2148 6724     SYMBOLS 6846 8191
6724 ? LST 2148 2612
2148 DAT VMA( 1 LST LSM MON LOC INS DLT SIZ RUN DMP NTR GO1 GO2 GO3
2182 BRT 0=49 94 0 195 100 8  Initialize stack and restart address
2200 *** INIT                *** Initialize program and symbol table
2206 PRT "RANGE"                 ranges
2212 STR S$ 1 1 32            Dummy space expected by ARGET
2224 INP S$ 2                 Input values
2232 CAL ARGET                Get program start in V
2238 SBT V-0                  Check for first arg=0
2246 JMP EQ SIZ               If so, use previous ranges
2254 SET L0=V                 L0=start of program
2262 CAL ARGET                Get program end from S$
2268 SET LP=V                 LP=current line pointer
2276 SBT LP-L0                If LP<L0, try again
2284 JMP LT INIT
2292 CAL ARGET                Get start of symbols from S$
2298 SET S0=V                 S0=start of symbol table
2306 SBT S0-LP                Check for overlap of program
2314 JMP LE INIT              If so, try again
2322 CAL ARGET                Get end of symbols from S$
2328 SET ST=V                 ST=end of symbol table
2336 SBT ST-S0                Check length of symbol table
2344 JMP LT INIT              If negative, try again
2352 SBT LP-1                 Put end of line token and CALL code
2360 WRT A=ENDER               in current line position
2368 INC LP                   LP normally points to second byte of line
2374 SBT S0-1
2382 FOR I=LP                 Zero out memory from
2390 BRT I=0                   end of program
2398 TIL I+1 A                 to beginning of symbols
2410 *** CMND                ***All commands return here
2416 SBT LP-1                 Print current line adr
2424 PRT A
2430 INP S$ 1                 Get command line
2438 STR CMD$ 1 3 S$ 1        Strip 3-letter command
2452 STR S$ 1 60 S$ 4         Shift line buffer past command
2466 FOR I=1                  Check for program command
2474 CMP CMD$ 1 3 VRB$ I       in verb list
2488 JMP EQ ASMBL             If so, enter it in program
2496 TIL I+3 153              Loop over 51 program commands
2508 FOR I=1                  Check for compiler command
2516 CMP CMD$ 1 3 VIM$ I       in verb list
2530 JMP EQ IMMED             If so, decipher
2538 TIL I+1 39               Loop over 13 compiler commands
2550 PRT "CM"                 If not a known verb, print CM
2556 *** ERROR               ***"ERROR" message printout
2562 PRT ERR$ 1               Print ERROR
2570 CHR 13                    and CRLF
2576 GTO CMND                 Get next command
2582 *** IMMED               ***Decipher compiler commands
2588 DIV I/3                  Loop counter/3+1 is command #
2596 INC A
2602 SET ADR=VMA( A           VMA array contains addresses
2612 GTO ADR                  Go to requested routine
```

```
        LST 2618 2952
 2618 *** LST              ***List the program
 2624 ADD L0+1              Default set first line of program as
 2632 SET L=A                first line to list=L
 2640 SET V1=LP             Default set current line as last to list
 2648 CAL ARGET             Get start of list range
 2654 SBT V-0               If no args entered,
 2662 JMP EQ LS2             use default range
 2670 INC V                 Skip CALL in first line
 2676 SET L=V               L=first line + 1
 2684 CAL ARGET             Get end of list range
 2690 SET V1=V              V1=last line; if no second arg, list 1
 2698 INC V1                Increment so last line is listed
 2704 *** LS2              ***Interpret command type
 2710 CAL VALUE             Get CALL adr from program
 2716 SBT ADR-JUMPS         Subtract beginning of jump table - 1
 2724 JMP LE LS21           Not a valid command
 2732 SET K=A               K=first char of verb in VRB$
 2740 SBT K-151             Is it valid?
 2748 JMP LE LS3            Yes
 2756 *** LS21             ***Error during list
 2762 SBT ADR-0             Exception for CALL 0 at end of program
 2770 SET K=79              Interpret CALL 0 as END command
 2778 JMP EQ LS3
 2786 PRT "LS"              Print LS ERROR
 2792 GTO ERROR
 2798 *** LS3              ***Type verb mnemonic
 2804 STR CMD$ 1 3 VRB$ K   Extract 3 letters of verb name
 2818 SBT L-3               A=command address
 2826 PRT A CMD$ 1          Print address and mnemonic
 2836 SBT K-79              K=argument counter
 2844 JMP EQ CMND           If it is an END command,
 2852 SET N=0                terminate listing
 2860 *** LS4               Print arguments
 2866 INC N                 Increment argument counter
 2872 CAL VALUE             Get next 2 bytes from program
 2878 SBT ADR-ENDER         52493 means end of this command line
 2886 JMP NE LS5            If not, continue
 2894 CHR 13                Print CRLF
 2900 SBT L-V1              Check for end of list range
 2908 JMP LE LS2            If not, continue
 2916 GTO CMND              Done listing
 2922 *** LS5               Check for  TIL command (61)
 2928 SBT K-61
 2936 JMP NE LS7
 2944 SBT N-4              4th arg of TIL is pointer to beginning of loop
 2952 JMP EQ LS4            so don't print it
```

```
        LST 2960 3338
 2960 *** LS7              Check for JMP command (40)
 2966 SBT K-40
 2974 JMP NE LS8           Continue if not
 2982 SBT N-1              First arg of JMP is special
 2990 JMP NE LS8           Not special
 2998 FOR I=1              Interpret condition code
 3006 SET A=CC( I          Compare with possible condition codes
 3016 SBT A-ADR
 3024 JMP NE LS71          Loop if no match
 3032 ADD I+I              Get position of condition mnemonic
 3040 DEC A
 3046 STR C2$ 1 2 CC$ A    Transfer 2 letters
 3060 PRT C2$ 1            Print them
 3068 GTO LS4              Next arg
 3074 *** LS71             Loop over 6 possible condition codes
 3080 TIL I+1 6
 3092 GTO LS21             Error-invalid JMP condition
 3098 *** LS8             ***Check for small constant
 3104 ABS ADR               between -255 and 255
 3110 SBT A-255
 3118 JMP GT LS9           If not, look it up in symbol table
 3126 PRN ADR              Print small constant value
 3132 GTO LS11             Print delimiter
 3138 *** LS9             ***Print argument name
 3144 SET J=0              J=character count
 3152 DEC ADR              Check last char of name
 3158 BRD ADR
 3164 SBT A-162            Is it ! or "+128?
 3172 JMP GT LS91          No
 3180 ADD A+162
 3188 CHR A                Print at beginning of name
 3194 *** LS91             Find beginning of name
 3200 DEC ADR              Back up address
 3206 BRD ADR              Read a byte
 3212 SBT A-0              0 means end of previous symbol
 3220 JMP NE LS91          If not, back up more
 3228 *** LS10            ***Print name
 3234 INC ADR              Next char address
 3240 BRD ADR              Read char
 3246 CHR A                Print char
 3252 SBT A-129            Check for last char of name
 3260 JMP LT LS10          If not; print next
 3268 *** LS11            ***Print argument delimiter
 3274 SBT N-1              Special delimiters
 3282 JMP EQ LS12           after first arg only
 3290 CHR 32               Otherwise print a space
 3296 GTO LS4              On to next
 3302 *** LS12             Print first arg delimiter
 3308 DIV K/3              Get address of delimiter in VD$
 3316 INC A
 3322 SET C=VD$ A          Get char in C
 3332 CHR C                Print it
 3338 GTO LS4              On to next arg
```

```
      LST 3344 3742
3344 *** VALUE       ***Read 2-byte value at L
3350 WRD L
3356 SET ADR=A        ADR=value
3364 INC L            Increment pointer
3370 INC L             twice
3376 RET
3380 *** ARGET       ***Extract an argument from S$
3386 STR V$ 1 1 Ø     V$=Ascii name, initially null string
3398 LEN S$ 1         Get length of S$
3406 SET N=A          N=length of argument name, initially same as S$
3414 SBT N-Ø          Check for null string
3422 JMP EQ AR9       If so, go to exit
3430 DEC N            Strip off argument delimiter following previous ar
3436 STR S$ 1 N S$ 2  Shift left
3450 FOR I=1          Search string for next argument delimiter
3458 STR C S$ I       C=Ascii value of next char
3468 SBT C-45         Check for minus sign (45)
3476 JMP NE AR1       No
3484 SBT I-1          Yes-allow if first char of arg
3492 JMP EQ AR3
3500 *** AR1         ***Check for valid char in name
3506 SBT C-47         Is it a number?
3514 JMP LE AR2       No
3522 SBT C-58         Less than 9?
3530 JMP LT AR3       If so, accept
3538 *** AR2         ***Check special delimiters
3544 SBT C-33         ! special case
3552 JMP EQ AR4
3560 SBT C-34         " treated same as !
3568 JMP EQ AR4
3576 SBT C-36         $ special case for strings
3584 JMP EQ AR3
3592 SBT C-40          and arrays(
3600 JMP EQ AR3
3608 SBT C-65         Any other char except letters
3616 JMP LT AR7        is an argument delimiter
3624 *** AR3          Valid char gets here
3630 TIL I+1 N        Loop next char
3642 GTO AR7          Entire string is valid name
3648 *** AR4         ***Special delimiters ! and "
3654 STR S$ 1 N S$ 2  Omit ! or." at start of string
3668 DEC N            Decrement string length
3674 FOR I=1          Look for another ! or "
3682 STR C S$ I       Next char in C
3692 SBT C-32         Anything except ! or " is accepted in string
3700 JMP LE AR5       Less than !
3708 SBT C-35         " is 34
3716 JMP LT AR6       Found end of string
3724 *** AR5          Not end yet
3730 TIL I+1 N        Loop next char
3742 GTO ARA          Error-no delimiter
```

```
      LST 3748 4114
3748 *** AR6
3754 INC I              Include ! or " at end of string
3760 *** AR7           ***Ith char is delimiter
3766 SBT I-1            Omit Ith char from arg name
3774 SET N=A            N=length of name
3782 STR V$ 1 A S$ 1    Transfer name to V$
3796 STR S$ 1 65 S$ I   Shift current arg out of S$
3810 SBT N-0            Check for name of zero length
3818 JMP EQ ARA         Error-2 delimiters in a row
3826 *** AR9           ***Convert arg name to number
3832 VAL V$ 1           Evaluate up to first non-numeric char
3840 SET V=A            V=value (0 if not a number)
3848 RET
3852 *** ARA           ***Syntax error
3858 PRT "SN"           Print SN ERROR
3864 GTO ERROR
3870 *** LSM           ***List the symbol table
3876 SET L=S0           Default list from beginning of symbols
3884 SET V1=ST          Default list to end of symbols
3892 DEC V1             Don't try to list last symbol +1
3898 CAL ARGET          Get start of range
3904 SBT V-0            If no args, use default range
3912 JMP EQ LM1
3920 SET L=V            L=start of listing
3928 CAL ARGET          Get end of range
3934 SET V1=V           V1=end of listing; if no second arg, list only one
3942 *** LM1           ***List next symbol
3948 PRT L              Print address of first char of symbol name
3954 *** LM2           ***Print symbol name
3960 BRD L              Read next char
3966 SET C=A            Save as C
3974 CHR C              Print char
3980 INC L              Bump pointer
3986 SBT C-128          Check for last char
3994 JMP LT LM2         If not, loop to next char
4002 CHR 32             Space after name
4008 SBT C-168          Is it an array(
4016 JMP GT LM4         No
4024 SBT C-164          Is it a string$
4032 JMP LT LM4         No
4040 BRD L              For string or array, next byte is dimension
4046 SET N=A            N=dimension
4054 PRT N              Print it
4060 SBT C-164          Is it a string?
4068 JMP EQ LM3         Yes-go print it
4076 INC L              Print values in array
4082 FOR I=1
4090 CAL VALUE          Get value at L,L=L+2
4096 PRT ADR            Print it
4102 TIL I+1 N          Loop over N values
4114 GTO LM5             and exit
```

```
      LST 4120 4468
4120 *** LM3              ***Print the string
4126 SET I=L              I=char pointer
4134 *** LM6              ***Print next char
4140 INC I                Bump string pointer
4146 BRD I                Read char
4152 CHR A                Print it
4158 SBT A-Ø              Check for Ø means end of string
4166 JMP NE LM6           If not, loop
4174 CHR 32               Print a space
4180 ADD L+N              Add string length+1
4188 INC A
4194 SET L=A              L points to end of string space
4202 GTO LM5              Go exit
4208 *** LM4              ***Gets here if not string or array
4214 CAL VALUE            Read value at L; L=L+2
4220 PRT ADR              Print it
4226 *** LM5              ***Exit
4232 CHR 13               Print CRLF
4238 INC L                Skip Ø at end of symbol
4244 SBT L-V1             Check for last line
4252 JMP LE LM1           No-continue listing
4260 GTO CMND             Get next command
4266 *** RUN              ***Execute program
4272 CAL ARGET            Get address (no argument causes restart)
4278 GTO V                 and go
4284 *** LOC              ***Set line pointer
4290 CAL ARGET            Get value
4296 INC V                Skip CALL byte
4302 SET LP=A             LP=line pointer
4310 GTO CMND             Next command
4316 *** SIZ              ***Print range of program & symbol table
4322 SBT LP-1             Address of current command line
4330 PRT "PROGRAM" LØ A   Print program range
4344 PRT "SYMBOLS" SØ ST   symbol range
4354 CHR 13                and CRLF
4360 GTO CMND             Next command
4366 *** DMP              ***Memory dump routine
4372 CAL ARGET            Get value in V
4378 SET V1=V             V1=start of dump
4386 CAL ARGET            V=end of dump, if no second arg, list one byte
4392 FOR I=V1
4400 BRD I                Read a byte
4406 PRT A                Print value
4412 DIV I/1Ø             Check whether address is evenly divisible by 1
4420 MLT A*1Ø
4428 SBT I-A              Mod 1Ø
4436 JMP NE DM1           No-continue
4444 CHR 13               Yes-print CRLF
4450 *** DM1
4456 TIL I+1 V            Loop from V1 to V
4468 GTO CMND             Next command
```

```
        LST 4474 4866
4474 *** NTR            ***Enter bytes in memory
4480 CAL ARGET           Get value in V
4486 SET V1=V            V1=adr of first value
4494 *** NX1            ***Enter next byte
4500 CAL ARGET           Get value
4506 SBT N-Ø             If arg length=Ø, done
4514 JMP EQ CMND         Yes-next command
4522 BRT V1=V            Enter value at V1
4530 INC V1              Bump pointer
4536 GTO NX1             Do it again
4542 *** ASMBL          ***Compile program commands
4548 DEF K I VØ LP       K=3*command#; VØ=start of current line
4560 ADD K+JUMPS         Add offset of jump table in execution routines
4568 SET V=A             Store routine adr in program
4576 CAL STOR1
4582 SBT K-64            Is it a DIM command?
4590 JMP NE AS4          No
4598 *** AS1            ***Compile DIM
4604 CAL ARGET           Get array or string name
4610 SBT N-Ø             Null argument means end of DIM
4618 JMP EQ AS3          If so, finish up
4626 STR D V$ N          Check last char of variable name
4636 SBT D-4Ø            Is it an array(?
4644 JMP EQ AS11         Yes
4652 SBT D-36            Is it a string$?
4660 JMP NE AS2          Error if neither
4668 *** AS11           ***Process variable name
4674 CAL SRCH            Look it up in symbol table
4680 SBT ST-1            If returned value of S exceeds
4688 SBT S-A              end of symbol table, name wasn't found
4696 JMP GE AS12         Not found
4704 PRT "DD"            Double dimension DD ERROR
4710 GTO BAK             Erase command
4716 *** AS12           ***Process dimension
4722 SBT 66-N            Save variable name at end of S$
4730 SET N1=A            N1=Position for temporary save
4738 STR S$ N1 65 V$ 1   Transfer V$ to end of S$
4752 CAL ARGET           Get dimension
4758 SBT V-Ø             If dimension=Ø,
4766 JMP EQ AS2           error
4774 SBT V-255           If dimension>255,
4782 JMP GT AS2           error
4790 LEN S$ N1           Get length of saved variable name
4798 STR V$ 1 A S$ N1    Transfer variable name back into V$
4812 SET N=A             N=length of name
4820 SBT SØ-V            Update pointer to start of symbols
4828 INC A                by subtracting dimension
4834 SET SØ=A            SØ=symbol pointer
4842 SBT D-4Ø            Is variable an array(?
4850 JMP NE AS13         No
4858 SBT SØ-V            Allocate 2 bytes for each array element
4866 SET SØ=A            Update symbol pointer
```

```
        LST 4874 5286
 4874 *** AS13          ***Enter variable in symbol table and program
 4880 CAL ENTR           Enter name and dimension in symbols
 4886 DEF V1 V V S       S=address of dimension in symbol table
 4898 CAL STOR1          Store address in program
 4904 SET V=V1           Recall dimension
 4912 CAL STOR1          Store in program
 4918 GTO AS1            Back for more dimensioned variables
 4924 *** AS2           ***Error in DIM
 4930 PRT "DM"           Print DM and ERROR after BAK routine
 4936 GTO BAK            Erase this command from program
 4942 *** AS3           ***Done, so prepare for next command
 4948 SET V=ENDER        End of line token (13) and CALL code (205)
 4956 CAL STOR1          Store in program
 4962 SBT LP-S0          Does program overlap symbols?
 4970 JMP LT CMND        No-continue
 4978 PRT "OM"           Yes-print OM ERROR
 4984 GTO ERROR           means Out of Memory
 4990 *** AS4           ***Process JMP command
 4996 SBT K-40           Is it a JMP (40)?
 5004 JMP NE AS6         No
 5012 STR V$ 1 2 S$ 2    Skip delimiter, next 2 char are condition code
 5026 STR S$ 1 50 S$ 4   Update command line
 5040 FOR I=1            Find condition code
 5048 CMP V$ 1 2 CC$ I    in CC$
 5062 JMP EQ AS5         Found
 5070 TIL I+2 11         Check 6 possible conditions
 5082 *** RGER          ***Error in aRGument
 5088 PRT "RG"           Print RG ERROR
 5094 GTO BAK            Erase command in program
 5100 *** AS5           ***Enter condition code in program
 5106 DIV I/2            Get condition code value
 5114 INC A               from CC( array
 5120 SET V=CC( A
 5130 CAL STOR1          Store in program
 5136 *** AS6           ***Enter next argument in program
 5142 CAL ARGET          Arg name in V$, value in V, name length in N
 5148 SBT N-0            Null arg means
 5156 JMP EQ AS20         end of command
 5164 SBT V-0            If value not 0,
 5172 JMP NE AS7          process as number
 5180 STR C V$ 1         Check for number 0
 5190 SBT C-48           "0"=48
 5198 JMP NE AS8         Not a number
 5206 *** AS7           ***Process numbers
 5212 ABS V              Is value between
 5218 SBT A-255           -255 and 255
 5226 JMP LE AS9         Yes-enter directly in program
 5234 *** AS8           ***Enter arg name in symbol table
 5240 CAL SRCH           Look for name
 5246 SBT ST-1           ST=end of symbols
 5254 SBT S-A            If name is found, S<ST
 5262 JMP LT AS81        Name already present
 5270 SBT S-ACCUM        Arg A returns S=1992
 5278 JMP EQ AS81        Accept A address as valid
 5286 CAL ENTR           Otherwise create new entry in symbols
```

```
          LST 5292 5634
 5292 *** AS81        ***Enter arg address in program
 5298 SET V=S          S is address
 5306 *** AS9         ***Enter value or address in program
 5312 CAL STOR1        Store V at LP
 5318 GTO AS6          Next arg
 5324 *** AS20        ***Post-processing of *** and TIL
 5330 SBT K-4          Is it a *** label?
 5338 JMP NE AS21      No
 5346 ADD LP+1         Address of next command
 5354 SET V=A
 5362 CAL STOR3        Store in symbol table
 5368 *** AS21        ***Process TIL command
 5374 SBT K-61         Is it a TIL command?
 5382 JMP NE AS3       No-prepare for next command
 5390 CAL TIL          Calculate pointer to top of loop and enter in pro
 5396 GTO AS3          Finish up
 5402 *** TIL         ***Calculate TIL loop pointer
 5408 SBT LP-6         A points to loop variable
 5416 WRD A            Get loop variable address
 5422 SET V=A           in V
 5430 SBT LP-10        Search for corresponding FOR
 5438 SET N=A           starting at N
 5446 SET K=FORCM      2051 means FOR command
 5454 *** TI1         ***Search program for FOR
 5460 DEC N            Back up
 5466 DEC N            2 bytes
 5472 SBT L0-N         L0=start of program
 5480 JMP GT TI2       Error if not found yet
 5488 WRD N            Read word at N
 5494 SBT A-K          Is it a FOR command?
 5502 JMP NE TI1       No-continue
 5510 ADD N+2          Look at next word
 5518 SET L=A
 5526 CAL VALUE        Loop variable address
 5532 SBT ADR-V        Is it same as address in TIL?
 5540 JMP NE TI1       No
 5548 SBT L-LP         Found it
 5556 INC A            Calculate back pointer
 5562 SET V=A           value
 5570 CAL STOR1        Store in program
 5576 RET
 5580 *** TI2         ***Error in TIL command
 5586 PRT "NF"         Print NF ERROR means No For
 5592 *** BAK         ***Erase command line due to error
 5598 FOR I=V0         V0=start of current command
 5606 BRT I=0          Enter 0
 5614 TIL I+1 LP        up to LP pointer
 5626 SET LP=V0        Reset LP to beginning of line
 5634 GTO ERROR        Print ERROR
```

```
        LST 5640 6032
 5640 *** STOR1        ***Store value V at LP; LP=LP+2
 5646 WRT LP=V          Write word
 5654 INC LP            Bump pointer
 5660 INC LP             twice
 5666 RET
 5670 *** STOR3        ***Store value V at S; S=S+2
 5676 WRT S=V           Write word
 5684 INC S             Bump pointer
 5690 INC S              twice
 5696 RET
 5700 *** SRCH         ***Search for variable name in symbol table
 5706 CMP V$ 1 N "A"    Is it A (accumulator)?
 5718 JMP NE SR1        No
 5726 SET S=ACCUM       Special address for A
 5734 RET
 5738 *** SR1          ***Start search at start of symbols
 5744 SBT S0-1          S0=current symbol pointer
 5752 SET S=A           S=address returned to program
 5760 *** SR6          ***Compare next symbol name
 5766 FOR I=1
 5774 STR C V$ I        Next char of name sought
 5784 SBT I-N           Last char?
 5792 JMP NE SR2        No
 5800 ADD C+128         Set marker bit on
 5808 SET C=A            last char
 5816 *** SR2          ***Compare a char
 5822 ADD S+I           Read corresponding char
 5830 BRD A              in this symbol
 5836 SBT A-C           Same?
 5844 JMP NE SR3        No
 5852 TIL I+1 N         Check all char of name sought
 5864 ADD S+N           Name found-set S to
 5872 INC A              point to associated value
 5878 SET S=A
 5886 RET
 5890 *** SR3          ***Not a match
 5896 INC S             Read next char
 5902 BRD S              in symbol table
 5908 SBT A-128         End of symbol name?
 5916 JMP LT SR3        No-keep looking
 5924 SBT A-36          Is it a string?
 5932 JMP LT SR5        No
 5940 SBT A-4           Is it an array?
 5948 JMP GT SR5        No
 5956 SET I=A           Save array/string indicator
 5964 ADD S+1           Read dimension at
 5972 BRD A              next location
 5978 SET L=A           Skip over space
 5986 ADD S+L            allocated for string
 5994 DEC A             Correct symbol pointer
 6000 SET S=A            so same as regular variable
 6008 SBT I-0           Is it an array?
 6016 JMP NE SR5        No; string
 6024 ADD S+L           Add more space
 6032 SET S=A            for array
```

```
      LST 6040 6412
6040 *** SR5          ***Search next symbol
6046 ADD S+3          Point to first char
6054 SET S=A           of next name
6062 SBT ST-1         Past end of symbols?
6070 SBT S-A
6078 JMP LT SR6       No; go on
6086 RET
6090 *** ENTR         ***Enter new symbol name and value
6096 SBT SØ-N         Subtract N spaces for name
6104 SBT A-3           and 3 more for value and end token
6112 SET SØ=A         SØ=new start of symbols
6120 SBT SØ-LP        Do symbols overlap program?
6128 JMP GT E1        No
6136 PRT "OS"         Print OS ERROR-Out of Symbol Space
6142 GTO ERROR
6148 *** E1           ***Enter symbol name
6154 SET SP=SØ        SP=symbol pointer
6162 FOR I=1
6170 STR C V$ I       Get next char of name
6180 SBT I-N          Last char?
6188 JMP NE E2        No
6196 ADD C+128        Set marker in last char
6204 SET C=A
6212 *** E2           ***Insert a char
6218 BRT SP=C
6226 INC SP           Bump pointer
6232 TIL I+1 N         until N char done
6244 SET S=SP         S=position for value
6252 SBT V-Ø          Is it Ø?
6260 JMP NE E3        No
6268 SBT C-162        Is last char " or !?
6276 JMP GT E4        No
6284 SET V=SØ         V points to start of name
6292 *** E3           ***Enter value
6298 WRT SP=V
6306 *** E4           ***Exit
6312 RET
6316 *** INS          ***Insert space in program
6322 CAL ARGET
6328 SET V1=V         V1=start of space
6336 CAL ARGET
6342 SET V2=V         V2=end of space
6350 INC V            Get length-1
6356 SBT V2-V1        Add to end of program
6364 ADD A+LP         Print location of dislocated segment
6372 PRT "RELOC" V A   and crlf
6382 CHR 13           End+1
6388 MOV V1 LP V      Move segment above inserted space
6398 INC V1           Set line pointer
6404 SET LP=V1         to start of inserted space
6412 GTO CMND         Next command
```

```
        LST 6418 6724
6418 *** DLT                    ***Delete part of program
6424 CAL ARGET
6430 SET V1=V                   V1=start of range
6438 CAL ARGET                  V=end of range
6444 INC V                      End+1 is first byte retained
6450 MOV V LP V1                Move rest of program down
6460 SBT LP-V                   Length of segment moved
6468 ADD A+V1                    + start is new end of program+1
6476 FOR K=A                    Fill extra space
6484 BRT K=Ø                     with Ø's
6492 TIL K+1 LP                  up to old line pointer
6504 SET LP=A                   Update end of program pointer
6512 INC V1                     Fix up *** and TIL commands
6518 SET L=V1                    from start of deletion
6526 *** D1                     ***Search for *** and TIL commands
6532 SBT L-LP
6540 JMP GE CMND
6548 CAL VALUE                  Get next word
6554 SBT ADR-TILCM              Is it a TIL?
6562 JMP EQ D2                  Yes
6570 SBT ADR-LBLCM              Is it a ***?
6578 JMP NE D1                  No; continue
6586 CAL VALUE                  Get address of label
6592 INC L                      L points to CALL code of next command
6598 WRT ADR=L                  Put pointer in symbol table
6606 INC L                      Point to next command address
6612 GTO D1                      and continue search
6618 *** D2                     ***Patch up TIL loop pointer
6624 ADD L+6                    L=address of loop pointer
6632 DEF S LP J L LP A VØ       Set up parameters for TIL subroutine
6652 CAL TIL                     and save local parameters
6658 DEF LP S L J               Restore local parameters
6670 GTO D1                     Back for more
6676 *** MON                    ***Call to monitor
6682 GTO MNTR                   *User definable address
6688 *** GO1                    ***USER COMPILER COMMANDS
6694 GTO Ø
6700 *** GO2
6706 GTO Ø
6712 *** GO3
6718 GTO Ø
6724 END
```

LSM 6846 7203

```
6846 INIT 2206
6853 V2 13020
6858 MNTR 8232
6865 VD$ 51 =    +-*/& #%        =+
6921 ACCUM 1992
6929 U 0
6933 U1 0
6938 E4 6312
6943 E3 6298
6948 E2 6218
6953 OS" 6953
6959 E1 6154
6964 SR5 6046
6970 SR3 5896
6976 SR2 5822
6982 SR6 5766
6988 USR1 0
6995 SR1 5744
7001 A" 7001
7006 NF" 7006
7012 TI2 5586
7018 TI1 5460
7024 TIL 5408
7030 STOR3 5676
7038 AS21 5374
7045 AS81 5298
7052 AS9 5312
7058 AS8 5240
7064 AS7 5212
7070 AS20 5330
7077 RG" 7077
7083 RGER 5088
7090 AS5 5106
7096 AS6 5142
7102 OM" 7102
7108 U2 0
7113 DM" 7113
7119 ENTR 6096
7126 AS13 4880
7133 BAK 5598
7139 N1 63
7144 DD" 7144
7150 AS12 4722
7157 S 1992
7161 SRCH 5706
7168 AS2 4930
7174 AS11 4674
7181 D 36
7185 AS3 4948
7191 AS1 4604
7197 AS4 4996
7203 STOR1 5646
```

LSM 7211 7531

```
7211 JUMPS 1993
7219 V0 13001
== 7224 NX1 4500
7230 DM1 4456
7236 SYMBOLS" 7236
7247 PROGRAM" 7247
7258 LM6 4140
7264 LM5 4232
7270 LM3 4126
7276 LM4 4214
7282 LM2 3960
7288 LM1 3948
7294 V1 7531
7299 SN" 7299
7305 ARA 3858
7311 AR7 3766
7317 A61 15682
7323 AR5 3730
7329 U3 0
7334 AR6 3754
7340 AR4 3654
7346 AR2 3544
7352 AR3 3630
7358 AR1 3506
7364 AR9 3832
7370 GO3 6718
7376 USER3 0
7384 C 195
7388 LS10 3234
7395 LS91 3200
7402 LS11 3274
7409 LS9 3144
7415 LS71 3080
7422 LS8 3104
7428 LS7 2966
7434 LS5 2928
7440 ENDER -13043
7448 LS4 2866
7454 GO2 6706
7460 LS" 7460
7466 LS3 2804
7472 K 79
7476 LS21 2762
7483 USER2 0
7491 VALUE 3350
7499 V 7531
7503 LS2 2710
7509 N 4
7513 ARGET 3386
752? L 7522
7525 GO1 6694
7531 D2 6624
```

```
          LSM 7536 8033
 7536 ADR 3876
 7542 ASMBL" 7542
 7551 RELOC" 7551
 7560 ERROR 2562
 7568 CM" 7568
 7574 IMMED 2588
 7582 ASMBL 4548
 7590 I 5
 7594 J 0
 7598 CMND 2416
 7605 SP 13993
 7610 LP 6725
 7615 RANGE" 7615
 7624 LS12 3308
 7631 ST 8191
 7636 USER1 0
 7644 D1 6532
 7649 TILCM 2054
 7657 S0 6846
 7662 LBLCM 1997
 7670 I1 18394
 7675 FORCM 2051
 7683 L0 2148
 7688 NTR 4480
 7694 DMP 4372
 7700 RUN 4272
 7706 SIZ 4322
 7712 DLT 6424
 7718 INS 6322
 7724 LOC 4290
 7730 MON 6682
 7736 LSM 3876
 7742 LST 2624
 7748 CC( 6 64 16450 1 16707 65 259
 7765 VMA( 13 2624 3876 6682 4290 6322 6424 4322 4272 4372 4480 6
694 6706 6718
 7797 ERR$ 6  ERROR
 7809 BL$ 3
 7817 C2$ 2 EQ
 7824 C$ 1
 7829 CMD$ 3 LSM
 7838 V$ 60 8033
 7902 S$ 65
 7971 CC$ 12 EQNELTGTLEGE
 7988 VIM$ 39 LSTLSMMONLOCINSDLTSIZRUNDMPNTRGO1GO2GO3
 8033 VRB$ 153 SET***INCDECADDSBTMLTDIVANDNOTIORXORGTOJMPCALWRDGE
TPUTRETFORTILDIMBRDPRTINPREMENDDATSTRCMPWRTBRTABSDEFLENVALPRNTAB
XINXOTWATDSIENIMOVWSRBSRCHRUS1US2US3US4
```

```
       SIZ          *** BASEX LOADER ***
PROGRAM 2148 4538    SYMBOLS 4541 5058
4538 ? LST 2148 2594
2148 BRT 0=49 94 0 195 100 8 Set up stack & restart
2166 DIM S$ 65 V$ 10         I/O buffer allocation
2178 *** CMD                 All commands return here
2184 INP S$ 1                Input command
2192 STR V$ 1 3 S$ 1         V$=3-letter command
2206 STR S$ 1 60 S$ 5        Omit left 4 char
2220 CMP V$ 1 3 "MON"        MON command?
2232 JMP EQ 8232             Yes-go to monitor
2240 CMP V$ 1 3 "DMP"        DMP command?
2252 JMP NE NTR              No
2260 CAL ARG                 V=first arg
2266 SET V1=V                Store in V1
2274 CAL ARG                 V=second arg
2280 FOR I=V1                Loop from V1
2288 BRD I                   Read byte
2294 PRT A                   Print it
2300 DIV I/10                Mod 10 of adr=0?
2308 MLT A*10
2316 SBT I-A
2324 JMP NE DM1              No
2332 CHR 13                  Print CRLF
2338 *** DM1
2344 TIL I+1 V               Loop til arg2
2356 GTO CMD                 Next command
2362 *** NTR
2368 CMP V$ 1 3 "NTR"        NTR command?
2380 JMP NE RUN              No
2388 CAL ARG                 V=start adr
2394 SET V1=V                Store as V1
2402 *** NT1
2408 CAL ARG                 V=next arg;if none,ERR
2414 BRT V1=V                Enter value in memory
2422 INC V1                  Bump memory ptr
2428 GTO NT1                  and loop
2434 *** RUN
2440 CMP V$ 1 3 "RUN"        RUN command?
2452 JMP NE PRG              No
2460 CAL ARG                 V=execution adr
2466 GTO V                   Go
2472 *** PRG
2478 CMP V$ 1 3 "PRG"        PRG command?
2490 JMP NE SYM              No
2498 CAL ARG                 V=start of program
2504 DEC V
2510 SET V1=V                V1=byte before start
2518 CAL ARG                 V=end of program
2524 *** PR1
2530 WSR V1 V LBLCM          Search for *** commands
2540 JMP GT CMD              Done if LT flag reset
2548 ADD A+2
2556 SET V1=A                V1=adr of symbol ptr
2564 WRD V1                  Read symbol ptr
2570 SET L=A
2578 ADD V1+3                A=start of next command
2586 WRT L=A                 Enter in symbol table
2594 GTO PR1                  and loop
```

```
2600 *** SYM
2606 CMP V$ 1 3 "SYM"  SYM command?
2618 JMP NE MOV        No
2626 CAL ARG
2632 SET V1=V          V1=start of program
2640 CAL ARG
2646 SET V2=V          V2=end of program
2654 CAL ARG
2660 SET V3=V          V3=start of symbols
2668 CAL ARG
2674 SET V4=V          V4=end of symbols
2682 CAL ARG           V=start of new symbols
2688 INC V1            V1=second byte of program
2694 FOR I=V1          Start loop at V1
2702 WRD I             Read program word
2708 SET L=A           L=next word
2716 SBT L-V4          Past end of symbols?
2724 JMP GE SM1        Yes
2732 SBT L-V3          Before start of symbols?
2740 JMP LT SM1        Yes
2748 SBT L-NE          Special NE code=16450
2756 JMP EQ JM
2764 SBT L-GT          Special GT code=16707
2772 JMP NE SM0        Not special case
2780 *** JM            Test for preceding JMP
2786 SBT I-2
2794 WRD A             Read preceding word
2800 SBT A-JMP         JMP command?
2808 JMP EQ SM1        Yes-leave it alone
2816 *** SM0           No
2822 SBT L-V3          Deduct old symbol offset
2830 ADD A+V           Add new symbol offset
2838 WRT I=A           Update program word
2846 *** SM1
2852 TIL I+2 V2        Loop to end of program
2864 SBT V4-V3         A=length of symbols
2872 ADD A+V           Add to new symbol start
2880 SET V1=A          V1=end of new symbols
2888 FOR I=V           Update string constant ptr
2896 BRD I             Read next byte of symbols
2902 SBT A-162         "+128?
2910 JMP EQ SM2        Yes
2918 INC A             !+128?.
2924 JMP NE SM3        No
2932 *** SM2
2938 ADD I+3           Check for end token
2946 BRD A              at I+3
2952 SBT A-0           End token?
2960 JMP NE SM3        No
2968 INC I
2974 WRD I             Read ptr to start
2980 SBT A-V3          Deduct old symbol offset
2988 ADD A+V           Add new symbol offset
2996 WRT I=A           Replace old ptr
3004 *** SM3
3010 TIL I+1 V1        Loop to end of symbols
3022 GTO CMD           Done
```

```
      LST 3028 3360
3028 *** MOV
3034 CMP V$ 1 3 "MOV"   MOV command?
3046 JMP NE FIX         No
3054 CAL ARG
3060 SET V1=V           V1=start of block
3068 CAL ARG
3074 SET V2=V           V2=end of block
3082 CAL ARG
3088 SET V3=V           V3=new location
3096 PRT "NOW" V3       Print new start
3104 SBT V2-V1          A=length of block
3112 ADD A+V3           Add start
3120 PRT A              Print new end of block
3126 CHR 13              & CRLF
3132 MOV V1 V2 V3       Move block
3142 GTO CMD            Next command
3148 *** ARG            Read arg from S$
3154 LEN S$ 1           Get length
3162 SBT A-0            Null string?
3170 JMP EQ ERR         Yes-print ? & restart
3178 SET L=A            L=length
3186 FOR I=1
3194 STR V S$ I         V=next char
3204 SBT V-48           <'0'?
3212 JMP LT AR1         Found delimiter
3220 SBT V-57           >'9'?
3228 JMP GT AR1         Must be delimiter
3236 TIL I+1 L          Loop til delimiter found
3248 *** AR1             or end of string
3254 DEC I              I=# of digits
3260 JMP EQ ERR         Error if 0
3268 VAL S$ 1           Evaluate string
3276 SET V=A            V=value
3284 SBT L-I            Whole string used?
3292 JMP GT MOR         No
3300 STR S$ 1 1 0       Null string
3312 RET
3316 *** MOR            Strip used part of string
3322 ADD I+2            A=delimiter+1
3330 STR S$ 1 L S$ A    Shift string
3344 RET
3348 *** ERR            Error
3354 PRT "?"            Extra'?' means error
3360 GTO 0              Restart
```

```
      LST 3366 3744
3366 *** FIX
3372 CMP V$ 1 3 "FIX"   FIX command?
3384 JMP NE ERR         Not recognized command
3392 CAL ARG
3398 SET V1=V           V1=start of program
3406 CAL ARG
3412 SET V2=V           V2=end of program
3420 CAL ARG
3426 SET V3=V           V3=start of symbols
3434 CAL ARG
3440 SET V4=V           V4=end of symbols
3448 CAL ARG
3454 SET V0=V           V=V0=new start of program
3462 DEC V              V=new start-1
3468 DEC V1             V1=old start-1
3474 *** FI1
3480 WRT V=52493        Insert end token & CALL code
3488 INC V              V=new program ptr (PP)
3494 INC V
3500 *** FI2
3506 INC V1             V1=old program ptr (PP)
3512 INC V1
3518 WRD V1             Read old command
3524 SET C=A            Save as C
3532 SBT C-REM          Skip REM
3540 JMP EQ SKIP
3548 SBT C-DIM          Skip DIM
3556 JMP EQ SKIP
3564 SBT C-LBL          Update *** and skip
3572 JMP EQ LABL
3580 SBT C-TIL          TIL needs special update
3588 JMP NE POK         Regular commands
3596 ADD V1+2           A=adr of loop ptr
3604 WRD A              Read loop variable ptr
3610 SET I=A            Save it
3618 SET J=V            Save new PP
3626 *** GTFOR          Find corresponding FOR
3632 DEC J
3638 DEC J              Back up new PP
3644 SBT J-V0           Past start of new program?
3652 JMP LE ERR         If so, error
3660 WRD J              Read word
3666 SBT A-FOR          FOR command?
3674 JMP NE GTFOR       No
3682 ADD J+2            Read loop variable ptr
3690 WRD A
3696 SBT A-I            Same as TIL variable?
3704 JMP NE GTFOR       No
3712 SBT J-V            Compute back ptr
3720 SBT A-3
3728 SET J=A
3736 ADD V1+8           Insert in old program
3744 WRT A=J            ***DESTROYS OLD PROGRAM
```

```
        LST 3752 4140
3752 *** POK          Move commands to new program
3758 WRT V=C          Move command
3766 INC V
3772 INC V            Bump new PP
3778 SBT V1-V2        Check for last
3786 JMP GT DONE
3794 INC V1
3800 INC V1           Bump old PP
3806 WRD V1           Read old word
3812 SET C=A
3820 SBT C-52493      Check for end token
3828 JMP NE POK       Move it
3836 GTO FI1          On to next command
3842 *** LABL         Update *** ptr
3848 SBT V-1          CALL code in new program
3856 SET I=A          I=value for label
3864 INC V1
3870 INC V1           V1=adr of adr in old program
3876 WRD V1
3882 WRT A=I          Insert new ptr at adr
3890 *** SKIP         Skip REM,DIM,***
3896 INC V1
3902 INC V1
3908 WRD V1           Read next word
3914 SBT A-52493      End of command?
3922 JMP NE SKIP      No-skip more
3930 GTO FI2          Yes-next command
3936 *** DONE
3942 SET V1=V0        V1=start of new prog+1
3950 INC V1
3956 SET V2=V         V2=end of new prog+1
3964 *** LOC          Print adr of symbol
3970 SET V0=V         V0=new symbol ptr (SP)
3978 SBT V4-V3        Check for end of symbols
3986 JMP LE CMD       If so, done
3994 CHR 13           Print CRLF
4000 PRT V0           Print SP
4006 *** NAM
4012 BRD V3           Read first char of name
4018 CHR A            Print it
4024 BRT V=A          Move to new SP
4032 SBT A-128        Check for last char
4040 JMP GT LAST
4048 INC V3           Bump old SP
4054 INC V             and new SP
4060 GTO NAM          Print more
4066 *** LAST         End of name-process value
4072 CHR 32           Print space
4078 ADD A+128        Get value of last char
4086 SET C=A          Store in C
4094 SBT C-161        Is it !?
4102 JMP NE C2        No
4110 PRT "OMIT"       Omit comments
4116 SET V=V0         Reset new SP to start
4124 ADD V3+4         Skip value
4132 SET V3=A         Set old SP
4140 GTO LOC          Print same location again
```

```
        LST 4146 4538
4146 *** C2             Retain string constants
4152 SBT C-162          Last char "?
4160 JMP NE C3          No
4168 INC V              Bump new SP
4174 INC V3             Bump old SP
4180 CAL CHG            Change all prog references
4186 SET A=V0           A=start of symbol
4194 GTO C6             Enter it & finish up
4200 *** C3             Abbreviate name to last char
4206 SET V=V0           Reset new SP to start
4214 BRT V=C            Insert last char
4222 INC V              Bump new SP
4228 INC V3              and old SP
4234 CAL CHG            Change prog references
4240 SBT C-168          Was last char ( or $?
4248 JMP GT C5          No-go move value
4256 BRD V3             Read max subscript
4262 SET D=A            Store it
4270 PRT A              Print it
4276 BRT V=D            Move to new SP
4284 *** ARY            Transfer array values
4290 FOR I=1
4298 INC V3             Bump old SP
4304 INC V               and new SP
4310 BRD V3             Read a byte
4316 BRT V=A            Move to new symbols
4324 TIL I+1 D          Til D values moved
4336 SBT C-164          Was last char $?
4344 JMP EQ C4          Yes-done
4352 SET C=164          Must be (
4360 GTO ARY            Move D more bytes
4366 *** C4             Finish up string or array
4372 INC V
4378 BRT V=0            Insert end token
4386 INC V              Bump new SP
4392 INC V3              and old SP
4398 INC V3              past end token
4404 GTO LOC            Next symbol
4410 *** C5             Move value of variable
4416 WRD V3             Get value at old SP
4422 *** C6             Put in new symbol table
4428 PRT A              Print value
4434 WRT V=A            Move it
4442 INC V              Bump new SP
4448 INC V
4454 ADD V3+3           Skip value & end token
4462 SET V3=A            in old symbol table
4470 GTO LOC            Next symbol
4476 *** CHG            Change V3 to V in program
4482 SET A=V1           V1=start of new program
4490 *** CH1
4496 WSR A V2 V3        Search for V3
4506 JMP GE CH2         If not LT, not found
4514 WRT A=V            Change value to V
4522 GTO CH1            Look for more
4528 *** CH2            Done
4534 RET
4538 END
```

LSM 4541 4761

```
4541 JMP 2033
4547 SM0 2822
4553 GO 0
4558 JM 2786
4563 CH1 4496
4569 CH2 4534
4575 C4 4372
4580 ARY 4290
4586 D 0
4590 C5 4416
4595 C6 4428
4600 CHG 4482
4606 C3 4206
4611 OMIT" 4611
4619 C2 4152
4624 LAST 4072
4631 NAM 4012
4637 LOC 3970
4643 DONE 3942
4650 FOR 2051
4656 GTFOR 3632
4664 J 0
4668 POK 3758
4674 TIL 2054
4680 LABL 3848
4687 LBL 1997
4693 DIM 2057
4699 SKIP 3896
4706 REM 2069
4712 FI2 3506
4718 52493 -13043
4726 FI1 3480
4732 V0 0
4737 FIX" 4737
4744 CMDS 0
4751 GT 16707
4756 NE 16450
4761 V4 16639
```

LSM 4766 4989

```
4766 ?" 4766
4771 MOR 3322
4777 AR1 3254
4783 ERR 3354
4789 NOW" 4789
4796 MOV" 4796
4803 SM3 3010
4809 SM2 2938
4815 SM1 2852
4821 C 292
4825 V3 4541
4830 V2 17151
4835 MOV 3034
4841 SYM" 4841
4848 L 4
4852 LBLCM 1997
4860 PR1 2530
4866 SYM 2606
4872 PRG" 4872
4879 PRG 2478
4885 FIX 3372
4891 RUN" 4891
4898 NT1 2408
4904 RUN 2440
4910 NTR" 4910
4917 DM1 2344
4923 I 4
4927 V 4541
4931 V1 16634
4936 ARG 3154
4942 NTR 2368
4948 DMP" 4948
4955 8232 8232
4962 MON" 4962
4969 CMD 2184
4975 V$ 10 MOV
4989 S$ 65
```

L. Getting Started with BASEX

If you have been furnished with a paper tape version of BASEX, the following information will assist you. The paper tapes follow the Intel format and consist of 16 byte data blocks of the following form:

Bytes	Information
0	Record mark (:)
1,2	Record length (10 hex, in most cases)
3-6	Load address
7,8	Record type (0)
9-40	Data bytes in Ascii hex format (0-9, A-F).
41,42	Checksum, the negative of the sum of all 8 bit bytes since the record mark (:), ignoring carries out of an 8-bit sum.
43,44	Carriage return, line feed

A listing of a bootstrap loader for Intel format tapes is provided on page 95.

The BASEX execution routines occupy locations 0-5 and 64-863 (hex), while the BASEX compiler occupies locations 864 to 1FFF (hex). In addition, the stack is placed at 5E (hex) and will not go beyond 40 (hex) unless your I/O routines use the stack excessively. If you use interrupt routines at locations 8 to 40 (hex) and find that they are overwritten by BASEX, you may shift the stack to some other location by entering the low order byte of the new stack address at locations 1 and 88E (hex) and entering the high order byte at locations 2 and 88F (hex).

The LOADER program is provided on a separate tape, since it overlays the BASEX compiler and occupies locations 864 to 13C2 (hex). Of course, the BASEX execution routines must also be present in memory before the LOADER can be run. As discussed in Section VIII, the LOADER may be used to relocate itself, the BASEX compiler or any other BASEX program to any desired location, so you may configure your own system as required.

However, the BASEX execution routines cannot be moved from their present location unless the compiler is altered extensively and all previously compiled BASEX programs are recompiled.

After loading the BASEX tapes, you must alter the address of the character input subroutine (see page 22), the character output subroutine (see page 23) and the monitor location (see page 29). After this has been done, BASEX may be started at location 0 and will respond with the "RANGE?" prompt, as described on page 2. From there on, the manual should provide sufficient information to permit you to carry on.

CAUTION: Proper performance of the NDCHA routine (location 2FE hex in the execution routines) requires that the value stored at location CD12 hex must be greater than A8. If you don't have memory in this range, the value FF will be read from CD12 hex, so no problem will be encountered.

These changes may be used to correct the problem with NDCHK mentioned on p. 94 of the BASEX manual. This is necessary only if you have memory at CD12 hex.

```
02E0 EB          3940  XCHG            HL=(arg), DE=PP
02E1 71          3950  MOV M,C         Move BC to (arg)
02E2 23          3960  INX H
02E3 70          3970  MOV M,B
02E4 EB          3980  XCHG            HL=PP, DE=(arg)
02E5 C9          3990  RET             *Get second subscript of string
02E6 D5          4000 GETLEN PUSH D    Stack (first char of string)
02E7 C5          4010  PUSH B          Stack arg type and first subscript
02E8 CD 86 02    4020  CALL GETADR     DE=(second subscript)
02EB 78          4030  MOV A,B         A=arg type of second subscript
02EC FE 02       4040  CPI 2           Should be 0 or 1
02EE C1          4050  POP B           BC=arg type and first subscript
02EF D4 32 03    4060  CNC ERROR       ERROR (754)-invalid subscript
02F2 1A          4070  LDAX D          A=second subscript value
02F3 47          4080  MOV B,A         Move to B
02F4 3A C4 07    4090  LDA MAX         Valid second subscript?
02F7 B8          4100  CMP B
02F8 F2 FC 02    4110  JP GE1          Yes
02FB 47          4120  MOV B,A         No-set to MAX
02FC D1          4130 GE1 POP D        Restore DE=(first char of string)
02FD C9          4140  RET
02FE 5E          4150 NDCHK MOV E,M    *Check for last arg
02FF 23          4160  INX H          Skip end of command token
0300 3E CD       4170  MVI A,205      CALL code next?
0302 BE          4180  CMP M
0303 C2 88 02    4190  JNZ GETADR+2    No-get (next arg)
0306 D1          4200  POP D          Yes-dump return adr
0307 E9          4210  PCHL           Execute next program command
0308             4220  DS 4           Unused space
030C 22 C8 07    4230 NOFLG SHLD AC    Store HL=result in AC
030F C3 29 03    4240  JMP RETRN       but don't set flags
0312 7C          4250 FLAG MOV A,H     Check for negative result
0313 17          4260  RAL
0314 3E 03       4270 FLLT MVI A,3     FL=3 means LT
0316 DA 23 03    4280  JC SHAC
0319 7C          4290  MOV A,H         Check for zero result
031A B5          4300  ORA L
031B 3E 42       4310 FLEQ MVI A,42H   FL=42 means EQ
031D CA 23 03    4320  JZ SHAC
0320 3E 02       4330 FLGT MVI A,2     FL=2 means GT
0322 00          4340  NOP
0323 22 C8 07    4350 SHAC SHLD AC     Store result in AC
0326 32 C5 07    4360 SFLG STA FL      Store flags in FL
0329 E1          4370 RETRN POP H      HL=PP
032A 23          4380 RET1 INX H       Skip end token
032B 7E          4390 RET2 MOV A,M     Is CALL code next?
032C FE CD       4400  CPI 205
032E C4 32 03    4410  CNZ ERROR       ERROR (817)-phase error
0331 E9          4420  PCHL            OK, go to PP
0332 E5          4430 ERROR PUSH H     *ERROR dump; save registers
0333 D5          4440  PUSH D
0334 C5          4450  PUSH B
0335 F5          4460  PUSH 6          PSW=6
```