ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

FORTIO---A FORTRAN I/O INTERFACE

by

L. Shalla

Applied Mathematics Division

April 1976
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT.</td>
<td>1</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>II. Subroutine Usage</td>
<td>2</td>
</tr>
<tr>
<td>A. OPEN.</td>
<td>2</td>
</tr>
<tr>
<td>B. BSAM/BPAM</td>
<td>4</td>
</tr>
<tr>
<td>C. QSAM.</td>
<td>6</td>
</tr>
<tr>
<td>D. BPAM.</td>
<td>8</td>
</tr>
<tr>
<td>E. CLOSE</td>
<td>9</td>
</tr>
<tr>
<td>F. POSITIONING</td>
<td>10</td>
</tr>
<tr>
<td>G. DCB</td>
<td>11</td>
</tr>
<tr>
<td>H. FILE CONTROL</td>
<td>15</td>
</tr>
<tr>
<td>I. LOAD.</td>
<td>16</td>
</tr>
<tr>
<td>J. STORAGE MANAGEMENT</td>
<td>17</td>
</tr>
<tr>
<td>III. Macro Usage</td>
<td>21</td>
</tr>
<tr>
<td>IV. Examples</td>
<td>22</td>
</tr>
<tr>
<td>A. BSAM.</td>
<td>22</td>
</tr>
<tr>
<td>B. QSAM.</td>
<td>25</td>
</tr>
<tr>
<td>C. BPAM.</td>
<td>26</td>
</tr>
<tr>
<td>V. Outline of Program Modules</td>
<td>34</td>
</tr>
<tr>
<td>VI. Error Processing</td>
<td>39</td>
</tr>
<tr>
<td>VII. Program Materials Available</td>
<td>39</td>
</tr>
</tbody>
</table>
A set of OS/370 Basic Assembly Language programs is described which provides a FORTRAN IV interface with OS/370 Macros.

I. Introduction

While the IBM System/370 FORTRAN IV language provides many convenient facilities for data transfer, no really practical technique is included for processing magnetic tapes with an arbitrary number of files. The fact that a separate ID statement must be provided for each distinct file written or read precludes the FORTRAN program from handling tapes with a large number of files. Thus the development of a general purpose I/O package was suggested and undertaken at AMD.

The I/O package was implemented as a set of assembly language coded FORTRAN compatible subroutines which correspond to the OS/370 system data management macros ordinarily available only to the basic assembly language programmer. Indeed, the subroutines' names and their parameterizations are almost identical to their OS counterparts.

Although the package was intended primarily for handling BSAM magnetic tape files, it was evident that other OS access methods could very easily be included. Hence subroutines are provided for the commonly used OS access methods: BSAM, QSAM, and BPAM. It should be noted that all data transfer using these subroutines is unformatted. Thus to convert data to internal form requires the use of temporary files or in-core conversion routines.

I have attempted to write the subroutines so that they are convenient and easy to use. Nevertheless, it should be noted that they are intended primarily for the rather sophisticated FORTRAN user. To effectively use them, a knowledge of ^S/JCL and the Data Management Services Macros is necessary.

Sections II and III, which follow, describe, in detail, the calling sequences and usage of the various subroutines and macros provided. Section IV gives some examples of their usage and may perhaps be an easier starting point for the reader.
II. Subroutine Usage

A. OPEN subroutines

1. Purpose

These routines obtain and initialize storage necessary for a Data Control Block (DCB) if required, and then OPEN (connect) the data set to the processing program.

2. Calling sequences

CALL OPENB(ddname, dcb[, opt1, opt2, filesq])
CALL OPENQ(ddname, dcb[, opt1, opt2, filesq])
CALL OPENP(ddname, dcb[, opt1, opt2, filesq])

where

OPENB--opens a data set for BSAM processing (cf. READ-WRITE below).

OPENQ--opens a data set for QSAM processing (cf. GET-PUT below).

OPENP--opens a data set for BPAM processing (cf. FIND-STOW below).

ddname—A literal constant or variable containing any valid 8 character ddname (trailing blanks must be used).

dcb—A 4-byte integer variable used to save the address of the DCB. If initialized to zero, a DCB area is obtained via GETMAIN and the address is planted in dcb. If the variable is non-zero, it contains an address pointing to one of the following:

a. A user assembled DCB area (cf. SECTION III below) which may be LOADED with the LOAD subroutine (cf. SECTION II, I below).

b. An INTEGER*4 array at least 100 bytes long (i.e., 25 elements) which has been previously initialized to zero. An initialized DCB will be placed in the array provided. The IADDR subroutine may be used to plant the address of this array in the dcb variable (cf. SECTION II, G below).

c. A previously initialized DCB.

(1) In the following descriptions, all parameters within the square brackets are optional and, unless otherwise noted, may be specified in any order. Any or all such parameters may be omitted. The default value for an omitted operand is underlined in the text describing that operand.
opt1--specifies processing method as a literal constant or variable containing one of the following character strings:

- **INPUT**--input data set (default if not specified)
- **OUTPUT**--output data set
- **RDBACK**--input data set, positioned to read backwards (QSAM, BSAM magnetic tape only)
- **INOUT**--input data set first, then output data set (BSAM only)
- **OUTIN**--output data set first, then input data set (BSAM only)
- **UPDAT**--data set or blocks to be added in place (direct access only)

opt2--specifies the final data set disposition as a literal constant or variable containing one of the following strings:

- **LEAVE**--no repositioning performed
- **REREAD**--position at beginning of data set
- **DISP**--perform volume positioning based on DISP parameter of the DD statement (default if not specified):
  - **PASS**--at end of data set
  - **DELETE**--rewind volume
  - **KEEP,CATLG,UNCATLG**--rewind and unload volume

fileseq--4-byte integer variable containing the file sequence number where the tape is to be positioned after OPEN. Note that this is valid for magnetic tape only and overrides the file sequence number of the LABEL parameter on the DD card for the file.

3. Notes

Only the minimum information required (DSORG,MACRF) is specified by the OPEN subroutine prior to OPENing the DCB. Therefore, all other required parameters (e.g., record length or blocksize) must be specified either through JCL or data set labels. Subroutines are provided for setting some DCB parameters prior to OPEN (cf. SECTION II, G). Errors detected during OPEN (e.g., missing DD statement) will always cause the program to abnormally terminate with an indicative message in the users SYMSG data set.
B. BSAM/BPAM subroutines

1. Purpose

These routines transfer blocks of data between storage and the data set (READ, WRITE), and WAIT for completion of I/O (CHECK). If not provided, a BSAM DCB and data event control block (DECB) are built. Any DCB provided will be assumed valid and opened for BSAM (cf. OPENB).

2. Calling sequences

CALL READ(decb, type, dcb, area, length[, filesq, ddname])
CALL WRITE(decb, type, dcb, area, length[, filesq, ddname])
CALL CHECK(decb, abc, err)

where

READ--transfers a block of unformatted data from the data set into area. For sequential tape files, a READ subsequent to one raising an end-of-file condition results in the transfer of the first (last if type=SB) block of data from the succeeding (preceding if type=SB) file if no intervening CLOSE was executed.

WRITE--transfers a block of unformatted data from area to the data set.

CHECK--WAIT's on the DECB for completion of the READ or WRITE operation.

decb--a 4-byte integer variable used to save the address of the DECB. This variable should be initialized to zero prior to the first call of the subroutine. If non-zero, it points to a previously initialized DECB.

type--a literal constant or variable containing the characters "SF" (sequential forward) or "SB" (sequential backward-READ only).

dcb--a 4-byte integer variable whose value is the address of a previously opened DCB (cf. OPENB). If the value is zero, a DCB area is obtained and opened for INPUT, DISP(READ-SF), OUTPUT, DISP(WRITE), or RDBACK, DISP(READ-SB).

area--an array large enough for the smaller of one block of data or the number of bytes specified in the length parameter.
**length**—is used for format U records only and is a 4-byte integer variable specifying the number of bytes to be read or written. This is taken from the DCB (block length) if the literal constant 'S' is used.

**filseq**—a 4-byte integer variable specifying the file sequence for tape positioning. Note, this is ignored if the DCB has already been opened.

**ddname**—a literal constant or variable containing any valid 8 character ddname (trailing blanks must be used).

**rbc**—a 4-byte integer variable in which the residual byte count is planted. The difference between the original count (specified by the length parameter) and the rbc is the number of bytes transferred.

**err**—a 4-byte integer variable where the result of the last I/O operation is stored:

1. 0 if normal completion
2. 1 if end-of-file (EOD)
3. 2 if automatic end-of-volume processing has occurred for concatenation of unlike data sets(1)
4. the following word for erroneous or exceptional conditions--

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **DS**—is the device status(2)
- **CS**—is the channel status(2)
- **S0**—is sense byte zero(3)
- **S1**—is sense byte one(3)

---

**Notes:**

(1) The DCBOFLGS must be set to X'08' (cf. SOFLGS, CONCAT below—SECTION II, G) to allow concatenation of unlike data sets.

(2) Refer to IBM System/370 Principles of Operation for a definition of status bytes (Form GA22-7000).

(3) Refer to the manual for device type in question for a definition of sense bytes.
3. Notes

The user is responsible for buffering and de-blocking the data. In addition, the READ and WRITE subroutines only cause initiation of the I/O operations returning control to the calling program immediately thereafter. Therefore, control may be returned prior to the completion of data transfer. The CHECK subroutine must be executed for the given DECB to ensure that the I/O transfer has been completed. Thus, the program can be written so that I/O and computation are overlapped:

```fortran
CALL READ(IDECB,'SF',IDCB,AREA,ILNGTH)
  Perform calculations not dependent on block just read.
CALL CHECK(IDECB,IREC,IERR)
```

Once the CHECK subroutine returns control, the I/O operation has completed. I/O errors are detected when the CHECK subroutine is executed. When an error is encountered (i.e., err=0 or 1), the operating system assumes that the bad record has been accepted. Finally, an end-of-file condition (err=1) causes the file to be positioned after the last record (READ, type=SB). In both cases, the file sequence number is still that of the current file (i.e., remains unchanged).

C. QSAM subroutines

1. Purpose

These routines transfer a record of data to the user provided area and WAIT for completion of the transfer before returning control to the MAIN program.

2. Calling sequences

```fortran
CALL GET(dcb,area,err[,filesq,ddname])
CALL PUT(dcb,area,err[,filesq,ddname])
CALL RELSE(DCB)
CALL TRUNC(DCB)
```

where

GET--transfers a record of unformatted data from the data set to area. For sequential tape files, a GET subsequent to one raising an end-of-file condition results in the transfer of the first record of data from the succeeding file if no intervening CLOSE was executed.

PUT--transfers a record of unformatted data from area to the data set.
RELSE--causes the remaining contents of the input block to be ignored. A subsequent GET obtains the first record from the next block.

TRUNC--causes the current output block to be written.

dcb--a 4-byte integer variable whose value is the address of a previously opened DCB (cf. OPENQ). If the value is zero, a DCB area is obtained and opened for INPUT,DISP(GET), or OUTPUT,DISP(PUT).

area--an array large enough to hold the largest record of the data sets being processed. For variable and undefined records, the LRECL field in the DCB is used to determine the actual record size written.

err--a 4-byte integer variable where the result of the last I/O operation is stored:

(i) 0 if normal completion
(ii) 1 if end-of-file (EOD)
(iii) 2 if automatic end-of-volume processing has occurred for concatenation of unlike data sets
(iv) the following word for erroneous or exceptional conditions--

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td>CS</td>
<td>S0</td>
<td>S1</td>
<td></td>
</tr>
</tbody>
</table>

DS--is the device status
CS--is the channel status
S0--is sense byte zero
S1--is sense byte one

filesq--a 4-byte integer variable specifying the file sequence for tape positioning. Note, this is ignored if the DCB has already been opened.

ddbname--a literal constant or variable containing any valid 8 character ddname (trailing blanks may be used).

---

(1) The DCBOFLGS must be set to Z08 (cf. SOFLGS, CONCAT below--SECTION II, G) to allow concatenation of unlike data sets. Also, if this is the case, the GET is automatically reissued--the first record of the next data set is returned to area.

(2) Refer to IBM System/370 Principles of Operation for a definition of status bytes (Form #GA22-7000).

(3) Refer to the manual for device type in question for a definition of sense bytes.
3. Notes

Blocking or deblocking is automatically provided with the QSAM subroutines. For unblocked files (i.e., RECFM=U), the LRECL field of the DCB is used to determine the length of the record transferred. (This can be retrieved or set using the GLRECL and SLRECL subroutines—cf. SECTION II, G below). For variable-length blocked files (i.e., RECFM=V, VB, VBS) the 4-byte record descriptor word (RDW) is included with the record (GET), or must be provided by the programmer (PUT). Buffering of blocks of data is automatic and anticipatory (i.e., look ahead). Since the transfer of a record of data is synchronized (i.e., program control is returned after data transfer), the CHECK subroutine must not be called to test for I/O completion. Data errors will cause abnormal termination unless the EROPT parameter of the DCB is specified. After an end-of-file (err=1), the file is positioned after the last record of the current file. (The file sequence number remains unchanged.)

D. BPAM subroutines

1. Purpose

These routines position partitioned data sets for subsequent READ’s on a member (FIND, BLDL) and changes, deletes, replaces or adds a member name to a directory (STOW).

2. Calling sequences

CALL BLDL(dcb, lst, err)
CALL FIND(dcb, nam, type, err)
CALL STOW(dcb, lst, direct, err)

where

BLDL—causes information from a PDS directory to be placed in lst in main storage.

FIND—positions a PDS so that the next READ is the first block of the member in question.

STOW—updates the PDS directory by adding, changing, replacing or deleting an entry.

dcb—is a 4-byte integer variable which points to a DCB previously opened for BPAM (cf. OPENP).

lst—is an array containing a BLDL list which must be initialized by the programmer.\(^1\)

\(^1\) Cf. IBM System/360 Operating System Supervisor and Data Management Macros Instructions (C28-6647) for the precise format of the BLDL list.
nam—if type = "D", a double word variable (READ*8) containing the left justified name of the member to be accessed. This must be padded with blanks. If type = "C", then nam is an array containing a BLDL list which must be initialized by the programmer. (1)

type—a literal constant or variable preset to the character "P" or "C".

direct—a literal constant or variable containing one of the following characters which describes the type of directory action required:
   A - add an entry to the directory
   C - change the name of a member in the directory
   D - delete an entry in the directory
   R - replace an entry in the directory

erg—a 4-byte integer variable containing the error code:
  a) BLDL or FIND
     0 - successful completion
     4 - list could not be filled (member name not found)
     8 - permanent I/O error in directory
  b) STOW
     0 - member name STOWed
     4 - directory already contains member name
     8 - specified name could not be found
     12 - no space left in directory
     16 - permanent I/O error in directory

E. CLOSE subroutines

1. Purpose

These subroutines CLOSE (disconnect) a data set from the processing program, return the storage occupied by the DCB area, and position the volume.

(1) Cf. IBM System/360 Operating System Supervisor and Data Management Macros Instructions (C28-6647) for the precise format of the BLDL list.
2. Calling sequences

CALL CLOSE(dcb[,opt,type])
CALL CLOSEK(dcb[,opt,type])

where

CLOSE--disconnects the data set and frees its storage area if the
    type option is not coded.

CLOSEK--disconnects the data set but does not free the storage.
    (The same DCB may be reopened.)

dcb--is a 4-byte integer variable which points to a DCB.

opt--a literal constant or variable containing one of the
    following character strings:

RERead--positions the volume to reprocess the data set.
LEAVE--positions the volume at the end of the data set.
REWIND(1)--positions the volume at load point.
DISP(1)--positions the volume according to the DISP
    parameter of the dd statement:
    (i) PASS--at end of current data set
    (ii) DELETE--rewind current volume
    (iii) KEEP,CATLG,UNCATLG--rewind and unload volume

T--a literal constant specified as:

T--creates labels and positions volumes according to the
    LEAVE or RERead option. The data set can be processed
    without issuing another OPEN call. Note that CLOSE
    TYPE T does not free the DCB storage area.

F. POSITIONING subroutines

1. Purpose

These subroutines position files before OPEN(SETSEQ), after
OPEN(SKIP), and write end-of-file (WEOF).

2. Calling sequences

CALL SETSEQ(dcb,filesq)
CALL GETSEQ(dcb,filesq)
CALL SKIP(dcb,filesq)
CALL WEOF(dcb)

(1) If the type option is specified, this parameter is ignored and CLOSE
defaults to LEAVE.
where

SETSEQ--causes a subsequent OPEN to position the volume to the file specified by \( \text{filesq} \) parameter.

GETSEQ--retrieves the current number and places it in \( \text{filesq} \).

SKIP--will position the volume to the current file plus that specified by the \( \text{filesq} \) parameter.

WBOF--writes an end-of-file and positions the volume at the beginning of the next file to be written.

dcb--is a 4-byte integer variable which must contain a dcb address. For SKIP and WBOF, the DCB must be opened.

\( \text{filesq} \)--is a 4-byte integer variable (\(+\)) containing the file sequence number.

3. Notes

The file skipping subroutines accomplish their task via the operating system—the DCB is closed, the file sequence number is incremented, and the DCB is then reopened. Consequently, shipping beyond the last file of a tape may cause unpredictable results but should result in an abnormal termination of the program. It is recommended that these subroutines be used only when the files are known to exist.

G. DCB subroutines

1. Purpose

These routines get and initialize storage for DCB's, retrieve and set certain fields in the DCB but do not OPEN the DCB.

2. Calling sequences

\[
\begin{align*}
\text{CALL GDCBB}(\text{ddname}, & \text{dcbj}, \text{opt1}, \text{opt2}) \\
\text{CALL GDCBQ}(\text{ddname}, & \text{dcbj}, \text{opt1}, \text{opt2}) \\
\text{CALL GDCBP}(\text{ddname}, & \text{dcbj}, \text{opt1}, \text{opt2}) \\
\text{CALL GRECRM}(\text{dcb}, & \text{recjm}) \text{ or } \text{recjm} = \text{GRECRM}(\text{dcb}) \\
\text{CALL GLRECL}(\text{dcb}, & \text{rec} \text{ecl}) \text{ or } \text{rec} \text{ecl} = \text{GLRECL}(\text{dcb}) \\
\text{CALL GBLKSZ}(\text{dcb}, & \text{blksz}) \text{ or } \text{blksz} = \text{GBLKSZ}(\text{dcb}) \\
\text{CALL GDEN}(\text{dcb}, & \text{den}) \text{ or } \text{den} = \text{GDEN}(\text{dcb}) \\
\text{CALL GDSORG}(\text{dcb}, & \text{dsorg}) \text{ or } \text{dsorg} = \text{GDSORG}(\text{dcb}) \\
\text{CALL GTRTOI}(\text{dcb}, & \text{trtch}) \text{ or } \text{trtch} = \text{GTRTOI}(\text{dcb}) \\
\text{CALL GDEN}(\text{dcb}, & \text{dsn}) \\
\text{CALL GVOL}(\text{dcb}, & \text{vol})
\end{align*}
\]

(1) When programs are referred to as FUNCTION subprograms, their names must appear in a type statement declaring them as INTBGER*4.
CALL GLAB(dcb,lab) or lab = GLAB(dcb)
CALL GOFLG(dcb,o6l) or o6l = GOFLG(dcb)
CALL SRECFM(dcb,recfm)
CALL SLRECL(dcb,reccl)
CALL SBLKSZ(dcb,blk5z)
CALL SDEN(dcb,den)
CALL SDSORG(dcb,dsorg)
CALL STRTCH(dcb,xtch)
CALL SDSN(dcb,dsn)
CALL SOSFLG(dcb,o6l)
CALL SBURCB(dcb,area)
CALL SBUFNO(dcb,bufno)
CALL CONCAT(dcb)

where

GDCBB--
GDCBO--
GDCBP--
GRECFM--
GLRECL--
GBLKSZ--
GDEN--
GDSORG--
STRTCH--
SDSN--
GVO--
GLAB--
GOFLG--
SRECFM--
SLRECL--
SBLKSZ--
SDEN--
SDDSORG--
STRTCH--
SDSN--
SOFWE--
SBUCB--
SSWO--

GDCBB--
GDCBO--
GDCBP--

GRECFM--
GLRECL--
GBLKSZ--
GDEN--
GDSORG--
STRTCH--
SDSN--
GVO--
GLAB--
GOFLG--

COMCAT- - sets the DCBOFLGS field to allow concatenation of unlike data sets.

ddname--a literal constant or variable containing any valid 8 character ddname (trailing blanks must be used).

dcb--a 4-byte integer variable where the address of the initialized DCB is planted (or located).

dsn--a 44-byte area containing the data set name (left justified, blank fill).

vol--a 30-byte area containing the first five volume serial numbers for the data set.
lab--a 4-byte integer variable containing the following masks:

<table>
<thead>
<tr>
<th>byte 0</th>
<th>byte 1</th>
<th>byte 2</th>
<th>byte 3</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>BLP Bypass label processing</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>SUL User label</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>NSL Nonstandard label</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>SL Standard label</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>NL No label</td>
</tr>
</tbody>
</table>

optl--specifies the processing method as a literal constant or variable containing one of the following character strings:

- INPUT--input data set (default if not specified)
- OUTPUT--output data set
- RDBACK--input data set, positioned to read backwards (QSAM, BSAM magnetic tape only)
- INOUT--input data set first, then output data set (BSAM only)
- OUTIN--output data set first, then input data set (BSAM only)
- UPDAT--data set or blocks to be added in place (direct access only)

opt2--specifies the final data set disposition as a literal constant or variable containing one of the following strings:

- LEAVE--no repositioning performed.
- REREAD--position at beginning of data set.
- DISP--perform volume positioning based on DISP parameter of the DD statement:
  - PASS--at end of data set
  - DELETE--rewind volume
  - KEEP,CATLG,UNCATLG--rewind and unload volume

optl--a 4-byte integer variable containing the value of DCBOFLGS:

<table>
<thead>
<tr>
<th>byte 0</th>
<th>byte 1</th>
<th>byte 2</th>
<th>byte 3</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>last operation was write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>last operation was read</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.1</td>
<td>last operation was read backwards</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..1</td>
<td>close routine for concatenation</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>....1</td>
<td>open</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>....1</td>
<td>problem program concatenation</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>....1</td>
<td>tape mark read</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>....0</td>
<td>user exit taken</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>....1</td>
<td>return from user exit</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>....1</td>
<td>dcb to be processed</td>
</tr>
</tbody>
</table>
recfm—a 4-byte integer variable containing a combination of the following binary masks:

<table>
<thead>
<tr>
<th>byte 0</th>
<th>byte 1</th>
<th>byte 2</th>
<th>byte 3</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F Fixed length</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>V Variable length</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>U Undefined</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.1</td>
<td>T Track overflow</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.1</td>
<td>B Blocked</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.10</td>
<td>S Spanned</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.01</td>
<td>M Machine control</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.00</td>
<td>no control character</td>
</tr>
</tbody>
</table>

reccl—a 4-byte integer variable containing the record length.

blksz—a 4-byte integer variable containing the block size.

den—a 4-byte integer variable containing one of the following density codes:

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
<th>7-tracks</th>
<th>9-tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000003</td>
<td>3</td>
<td>200 bpi</td>
<td>----</td>
</tr>
<tr>
<td>00000043</td>
<td>67</td>
<td>550 bpi</td>
<td>----</td>
</tr>
<tr>
<td>00000083</td>
<td>131</td>
<td>800 bpi</td>
<td>800 bpi</td>
</tr>
<tr>
<td>000000C3</td>
<td>195</td>
<td>----</td>
<td>1600 bpi</td>
</tr>
<tr>
<td>000000D3</td>
<td>211</td>
<td>----</td>
<td>6250 bpi</td>
</tr>
</tbody>
</table>

dorg—a 4-byte integer variable containing one of the following data set organization codes:

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
<th>Code</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000080</td>
<td>128</td>
<td>IS</td>
<td>Indexed sequential</td>
</tr>
<tr>
<td>00000040</td>
<td>64</td>
<td>PS</td>
<td>Physical sequential</td>
</tr>
<tr>
<td>00000020</td>
<td>32</td>
<td>DA</td>
<td>Direct</td>
</tr>
<tr>
<td>00000002</td>
<td>2</td>
<td>K</td>
<td>Partitioned</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
<td>U</td>
<td>Unmovable</td>
</tr>
</tbody>
</table>

area—an area in which a buffer pool is to be built.

(Cf. BUILD subroutine—SECTION II, I.)
The area to be structured must have a length 8+(number of buffers)*(length of each buffer).

ttech—a 4-byte integer variable containing one of the following tape-recording-technique codes:

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
<th>Code</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000023</td>
<td>35</td>
<td>E</td>
<td>Even parity</td>
</tr>
<tr>
<td>0000003B</td>
<td>59</td>
<td>T</td>
<td>BCD/EBCDIC translation</td>
</tr>
<tr>
<td>00000013</td>
<td>19</td>
<td>C</td>
<td>Data conversion</td>
</tr>
<tr>
<td>0000002B</td>
<td>43</td>
<td>ET</td>
<td>Even parity translation</td>
</tr>
</tbody>
</table>

buysno—a 4-byte integer variable whose value is the number of buffers in the buffer pool (§255).
H. FILE CONTROL subroutines

1. Purpose

These routines can be used to position a tape within a file (CNTRL), position a tape or disk at a particular physical block (POINT), or record the position of a physical block (NOTE).

2. Calling sequences

CALL CNTRL(dcb, opt1, err, opt2)
CALL NOTE(dcb, blkadd)
CALL POINT(dcb, blkadd, err)

where

CNTRL--positions a magnetic tape according to the opt parameters.

NOTE--returns the relative position of the last block read to blkadd. The dcb must be OPENed for NOTE.\(^1\)

POINT--positions the file to the block specified by blkadd. The dcb must be opened for POINT.\(^1\)

dcb--is a 4-byte integer variable whose value is the address of the dcb.

opt1--is a literal constant or variable containing one of the following character strings:

BSM--backspace to beginning of file
FSM--forward space to end of file
BSR--backspace one block (or number of blocks specified by opt2)
FSR--forward space one block (or the number of blocks specified by opt2)

opt2--is a 4-byte positive integer specifying the number of blocks to be spaced (forward or backward). This parameter should only be specified when opt1=FSR/BSR.

err--a 4-byte integer variable containing the error code:

(i) 0 operation completed successfully
(ii) 1 for CNTRL with OPT1 = FSR or BSR, a tape mark was encountered. OPT2 contains the number of uncompleted forward or backspaces in this case.

\(^1\)Unless the data set is partitioned (i.e., OPENed for BPAM), a special DCB with the P(point) option for the MACRF parameter must be preassembled using the DCBAREA macro (cf. Section III) and LOADED prior to execution of the OPEN module. The DCBPOINT module is such a DCB (cf. Section V).
(iii) the following word for erroneous conditions--

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td>CS</td>
<td>S0</td>
<td>S1</td>
<td></td>
</tr>
</tbody>
</table>

DS--is the device status\(^{(1)}\)
CS--is the channel status\(^{(1)}\)
S0--is sense byte zero\(^{(2)}\)
S1--is sense byte one\(^{(2)}\)

blkadd—a 4-byte integer of the following form (binary):

(i) for tape

```
Z Z C C
```

(ii) for disk

```
T T R Z
```

CC is a two byte block number.
TT is a 2-byte track number.
R is a one-byte block number.
Z is zero.

I. LOAD subroutines

1. Purpose

These subroutines can be used to dynamically load other modules and transfer control to them. They are provided primarily so that the programmer can load and delete a preassembled DCB and use it.

2. Calling sequences

```
CALL LOAD(modnam, eploc)
CALL LINK(modnam, pumlst)
CALL DELETE(eploc)
CALL CALL(eploc, pumlst)
```

\(^{(1)}\) Refer to IBM System/370 Principles of Operation for a definition of status bytes (FORM #GA22-7000).

\(^{(2)}\) Refer to the manual for the device type in question for a definition of sense bytes.
where

LOAD--causes the module named in modnam to be loaded and saves its entry point in eploc.

LINK--causes the module named in modnam to be loaded, executed and deleted, passing it the parameters in pmist.

DELETE--causes the module whose address is specified in eploc to be deleted.

CALL--causes the module whose address is specified in eploc to be executed passing to it the parameters specified by pmist.

modnam--a literal constant or variable containing any 8-character module name (trailing blanks must be used).

eptoe--a 4-byte integer variable whose value is the address of the entry point of the loaded module.

pmist--the proper calling sequence for the module in question.

J. STORAGE MANAGEMENT subroutines

1. Purpose

These subroutines can be used to dynamically obtain and free storage from the user's region (GETINN,FREBIN), structure I/O buffer pools (GETPL,FREEPL,BUILD), and manage I/O buffers from the pools (GETBF,FREEBF).

2. Calling sequences

CALL GETINN(type,length,add[,subp])
CALL FREBIN(type[,length],add[,subp]) (1)
CALL IADDR(var,add) or add = IADDR(var)
CALL GETPL(dcb,buyno[,length])
CALL FREEPL(dcb)
CALL BUILD(area,buyno,length)
CALL GETBF(dcb,add) or add = GETBF(dcb) (2)
CALL FREEBF(dcb,add)

(1) If included, the length parameter must be the second operand in the parameter list. Moreover, this parameter must only be present for FREBIN, type = E.

(2) When referred to as a FUNCTION subprogram, GETBF must be declared as INTEGER*4 in a type statement.
where

**GETMN**—allocates an area of length (in bytes) specified by the value of `Length` (rounded up to a multiple of 8). The area is allocated from the main storage assigned to the job step and begins at a doubleword boundary. Depending on the `type` parameter, the request may be variable (`Length` specified as upper and lower bounds) or fixed; also, the request may be conditional (job step not terminated if the request is unsatisfied) or unconditional—cf. the `type` parameter. The address of the area obtained is returned to `add`. If the request is conditional and is not satisfied, `add` is set to zero.

**FREEMN**—returns an area previously allocated via **GETMN** to the pool of storage available to the job step. If the `type` parameter is variable ("V"), the `Length` parameter must be omitted and the address and length of the area to be released are both specified by the `add` parameter (see below). If the type is specified as "E", then the `Length` parameter must be present and specifies the length (in bytes) of the area to be released (rounded up to a multiple of 8).

**IADDR**—returns the address of `vaA` to `add`. This subroutine provides addressability to storage areas dynamically allocated (cf. Addressing below).

**GETPL**—allocates a main storage area and builds a buffer pool in it. The area is allocated from main storage assigned to the job step. The number of buffers is specified by `buna` and their length by `Length`. If the optional length parameter is not specified, the blocksize from the DCB is used. This subroutine must be called before the file is opened. If insufficient storage exists to build the pool, the job step is abnormally terminated.

**BUILD**—structures a previously allocated area for use as a buffer pool. `Area` is an integer*4 array in which the buffer pool is built; the `buna` and `Length` parameters specify the number and length of the buffers. The BUFNO and BUFCSB are required in the DCB. BUFNO may be specified via JCL or the 5BUFNO subroutine; BUFCSB must be set by the 5BUFCSB subroutine and is the address of the buffer pool (value of `add`). The format of the buffer pool is:

```
+8 bytes+  buffer          buffer          ............          buffer
```

```
buffer pool control block
```

Address

```
area address
```

`buna = number of buffers`
FREEPL--releases the main storage allocated to a buffer pool. The main storage area to which the pool was assigned must have been obtained via GETMN or GETPL.

GETBF--obtains an I/O buffer from the buffer pool associated with the dcb and places its address in add. If no buffer is available, add is set to zero.

FREEBF--returns an I/O buffer to the buffer pool associated with the DCB.

type--a literal constant or variable containing one of the following character strings:

(i) for GETMN

EC--a fixed length area with length specified by LENGTH. The request is conditional--add is set to zero if unsatisfied.

BU--a fixed length area with length specified by LENGTH. The request is unconditional--the job step is terminated if unsatisfied.

VC--the largest area whose size is \\
LENGTH(1) and \\
LENGTH(2). If the request cannot be satisfied, add(1) is set to zero; otherwise add(1) and add(2) respectively contain the address and length of the area obtained.

VU--same as VC except the job step is terminated if the request is unsatisfied.

(ii) for PREMN

I--releases a single area of storage whose length is specified by the LENGTH parameter.

V--releases a single area of storage whose length is specified by add(2).

LENGTH--an array of type INTEGER*4 dimensioned as follows:

(i) For GETPL, BUILD or GETMN

   type E, EC, or BU (see type above),
   one element whose value is the length
   (in bytes) of the buffer or main storage
   area requested.

(ii) For GETMN type VC or VU (see type above),
    a two element array specified as:

```
<table>
<thead>
<tr>
<th>element 1</th>
<th>element 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>length</td>
</tr>
<tr>
<td>required</td>
<td>required</td>
</tr>
</tbody>
</table>
```
add--an array of type INTEGER*4 dimensioned as follows:

(i) GETMN or FREEMN
   (a) type E, EC, or EJ (see type above),
       one element whose value is the
       address of the main storage area
       obtained or released.
   (b) type V, VC or VU (see type above),
       a two element array whose values are

       | element 1 | element 2 |
       |-----------|-----------|
       | address of |
       | storage    |
       | area       |
       | length of area |
       | obtained or |
       | released    |

(ii) GETBFT or FREBFT, one element containing the
    address of the buffer.

(iii) IADDR, one element containing the address of
     the variable.

area--an INTEGER*4 array used to structure a buffer pool.
The size of this array (in bytes) must be greater
than or equal to (buffer length)*(number of buffers)+8.

var--any variable (REAL, INTEGER, LOGICAL, etc.) whose
     address is to be obtained.

dcb--a 4-byte integer variable containing the address of
     the DCB with which the subpool is associated.

bufno--a 4-byte integer variable whose value is the number
     of buffers to be built ($255$).

subp--a 4-byte integer variable whose value is the subpool
     number from which the main storage is to be allocated
     (between 0 and 127).

3. Addressing

   The IADDR subroutine may be used to address dynamically obtained
   storage areas (e.g., areas obtained via GETBFT, GETMN). For the
   sake of illustration, suppose the INTEGER*4 array IAREA(1)
   contains the address of such an area, then the term:

   IAREA((IAREA(1)-IADDR(IAREA))/4+1)

   will provide addressability to that area. It is the programmer's
   responsibility to ensure that the proper boundary alignment
   requirements are met.

4. Buffering

   In most applications two buffers are sufficient to obtain close
   to maximum I/O speed provided that an input or output request
   to fill (or to empty, for an output file) a buffer is issued as
   soon as one becomes available. In certain cases, additional
   buffering depth may increase I/O overlap. One such example
   would be a program which sporadically outputs 3 blocks of data
   at once. In this case, an additional buffer would increase
   the amount of I/O overlap.
For QSAM, I/O overlap is automatically provided by the system, and the buffer pool is also automatically built by the operating system (2 buffers are assigned unless BUFNO is specified for the DCB). Thus there is no advantage to issuing the GETPL subroutine unless the programmer wishes for some reason to determine the level of buffering at program execution time.

Buffering for the BSAM access method, on the other hand, is not automatic. Thus the GETPL subroutine must be called to obtain a buffer pool, and the programmer is required to overlap his I/O. The following DO loop illustrates one possible technique for overlapping I/O:

```
DO 200 I=1,K
200 CALL READ(I,DECBR(I), 'SF', IDCBR, IAREA(IAREA(I)-IADDR(IAREA))/4+1), 'S')
```

This loop issues K READs without executing a CHECK for the DECB. The array IAREA must contain the address of K buffers gotten via the GETPL subroutine. Before the data can be processed, CHECK must be issued for each DECB:

```
DO 300 I=1,K
   CALL CHECK(I,DECBR(I),IRBC(I),IB=RF)
   * PROCESS INPUT DATA
300 CONTINUE
```

A complete example illustrating I/O overlap with BSAM is given below (cf. SECTION IV, A, Example 3).

III. Macro Usage

If the Fortran programmer finds that the foregoing DCB setting subroutines (cf. II, Sec. G) are insufficient, a macro has been provided which formats the DCB area properly for the various subroutines and which can be parameterized as the user wishes. The macro must be assembled and the resulting CSECT must be loaded prior to OPEN. Its usage is as follows:

```
[name] DCBAREA TYPE=CSECT,
   DSORG=dsorg,MACRP=macr
   [,RECFM=recfm] [,LRECL=lrecl]
   [,BSIZE=bsize] [,DEVDEV=devd]
   [,DEN=dien] [,DNSNAME=dname]
```

The values of the various parameters are identical to those for the DCB macro described in the OS/360 System Supervisor and Data Management Macros Instructions (C28-6647).

Following is an example of the assembly language statements necessary to assemble a DCB allowing the BSAM POINT (NOTE) option:

```
DCB   TITLE 'BSAM DCB WITH POINT/NOTE OPTION'
DCBPOINT START
   REGISTER
   DCBAREA TYPE=CSECT,DSORG=PS,MACRP=(KP,MP)
END
```
IV. Examples

A. BSAM

1. Write ten files of data on an output tape.

a. JCL

//FT08F001 DD UNIT=TAPE,DISP=(NEW,PASS),LABEL=(1,NL),
// DCB=(RECFM=U,BLKSIZE=80)

b. PROGRAM

The ddname linking the file to the program is FT08F001. The blocks of data to be written are in the array ODAT. (Note: BSAM performs no blocking.) ODCB, OLN, ODECB, IRBC are save areas for the DCB address, record length, DECB address, and residual byte count respectively. On the first WRITE, the DCB and DECB are initialized, and the former is OPENed since that was not done with a previous call to OPENB. After the I/O request is issued, CHECK waits for completion of I/O and sets error codes. When 1000 records are written, WEOF writes an end-of-file and positions the tape at the next file. If an I/O error occurs, the status word is written (ISN 0018).

2. Read files 3 and 8 from input tape.

a. JCL

//FT08F001 DD UNIT=TAPE,DISP=(OLD,PASS),LABEL=(3,NL),
// DCB=(RECFM=U,BLKSIZE=80),VOL=SER-MYTAPE
This program is almost identical to the preceding example except only files 3 and 8 are actually read into the data area IDAT. The PASS subparameter for the DISP parameter ensures that the tape is not rewound between files.

3. Copy a file buffering I/O.

a. JCL

```
//FT08F001 DD DSN=INPUT,DISP=SHR,
  // DCB=NCP=99

//FT08F002 DD DSN=COPY,DISP=(NEW,KEEP),
  // DCB=NCP=99

//FT05F001 DD *
```

b. PROGRAM

```
ISN 0002 10 FORMAT (1H,I15)
ISN 0003 20 FORMAT(10(1H ),80(A1))
ISN 0004 REAL*3 DDNAME
ISN 0005 LOGICAL*1 IDAT(80)
ISN 0006 DATA DDNAME/'FT08F001'/
ISN 0007 LOGICAL*1 FLAG
ISN 0008 DATA FLAG/.FALSE./
ISN 0009 DATA IDCB,ILN,IDECB,IRBC /0,80,0,0/
ISN 0010 1000 CALL READ (IDECB,'SF',IDCB,IDAT,ILN,DDNAME)
ISN 0011 CALL CHECK(IDECB,IRBC,IERR)
ISN 0012 IF(IERR.EQ.1) GO TO 2000
ISN 0014 IF (IERR.GT.1)GO TO 3000
ISN 0016 WRITE(6,20) IDAT
ISN 0017 1000 IF(FLAG) GO TO 4000
ISN 0020 FLAG=.TRUE.
ISN 0021 CALL SKIP(IDCB,5)
ISN 0022 GO TO 1000
ISN 0023 3000 WRITE (6,10) IERR
ISN 0024 4000 STOP
ISN 0025 END
```
The ddname of the file to be copied is FT08F001, its copy is FT08F002. The level of buffering is K (read from FT05F001). The buffer pool consisting of K buffers is obtained and built in ISN 0012 and ISN 0015. The input dataset's attributes are copied in ISN 0016 through ISN 0021. The buffer pool addresses are obtained in ISN 0023 and data is read into these buffers at ISN 0032. Notice that K reads are issued. After checking the reads ISN 0036, processing of the data may commence (in this case a WRITE--ISN 0041). The WRITEs are then checked in ISN 0028.
B. QSAM

1. Write ten files of data on an output tape.
   
   a. JCL

   //FT08F001 DD UNIT=TAPE,DISP=(NEW,PASS),LABEL=(1,NL),
   // DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)

   b. PROGRAM

   ISN 0002 10 FORMAT (IH ,I15)
   ISN 0003 REAL*8 DDNAME,IDAT(10,10)
   ISN 0004 DATA DDNAME /'FT08F001'/
   ISN 0005 DATA IDCB /0/
   ISN 0006 DO 2000 I=1,10
   ISN 0007 DO 1000 J=1,100
   ISN 0008 DO 500 K=1,10
   ISN 0009 CALL PUT(IDCB,IDAT,IERR,DDNAME)
   ISN 0010 IF (IERR.GT.1) GO TO 3000
   ISN 0012 500 CONTINUE
   ISN 0013 1000 CONTINUE
   ISN 0014 CALL WEOF(IDCB)
   ISN 0015 2000 CONTINUE
   ISN 0016 GO TO 4000
   ISN 0017 3000 WRITE (6,10) IERR
   ISN 0018 4000 CONTINUE
   ISN 0019 STOP
   ISN 0020 END

   The ddname connecting the file to the program is FT08F001. The blocks of data to be written are in the array IDAT. (Note: QSAM does the blocking of records automatically.) IDCB is the save area for the DCB address; and, since OPENQ is not directly called, a DCB area is obtained, initialized and OPENed on the first call to PUT. Notice also that check is not used since QSAM automatically WAITs for completion of the I/O. After 1000 records are written, WEOF writes an end-of-file and positions the tape at the next file.

2. Read all files of an input tape.

   a. JCL

   //FT08F001 DD UNIT=TAPE,DISP=(OLD,PASS),LABEL=(1,NL),
   // DCB=(RECFM=FB,BLKSIZE=800,LRECL=80),VOL=SER=MIXTAPE
b. **PROGRAM**

```
FT08F001 is the ddname for the input file. This program counts the
number of records in each file on the tape. Note that the end-of-
tape is denoted by a double end-of-file (tape mark). It is assumed
here, of course, that all files are identically blocked.

C. BPAM

1. List selected members of a partitioned data set.

   a. **JCL**

   //FT08F001 DD DSN=MYPDS,DISP=SHR
   //FT08F002 DD DSN=SASCR,DSN=TEMP,SPACE=(CYL,(1,1))
```
b. PROGRAM

ISN 0002  INTEGER*4 BLK
ISN 0003  DATA IRCFB /Z000000C0/
ISN 0004  LOGICAL*1 AREA(4000)
ISN 0005  REAL*8 MEMBER
ISN 0006  5  FORMAT(1H ,Z8)
ISN 0007  10  FORMAT(10(1H ),80(A1))
ISN 0008  15  FORMAT(1H ,**************END-OF-MEMBER***************)
ISN 0009  20  FORMAT(1H1,30(1H ),'MEMBER=',A8)
ISN 0010  25  FORMAT (A8)
ISN 0011  30  FORMAT(IH1,30(1H ),'MEMBER=',A8,' NOT FOUND')
ISN 0012  DATA DCBP,DECBP,DCBB,DECBB,DCBQ /0,0,0,0,0/
ISN 0013  CALL OPENP('FT08F001',DCBP,'INPUT')
ISN 0014  CALL GRECFM(DCBP,IRCFM)
ISN 0015  CALL GLRECL(DCBP,ILRCL)
ISN 0016  CALL GBLKSZ(DCBP,IBLKSZ)
ISN 0017  CALL GDCBB('FT08F002',DCBB,'0UTPUT')
ISN 0018  CALL GDCBQ('FT08F002',DCBQ,'INPUT')
ISN 0019  CALL SRECFM(DCBB,IRCFB)
ISN 0020  CALL SBLKSZ(DCBB,IBLKSZ)
ISN 0021  CALL SRECFM(DCBQ,IRCFB)
ISN 0022  CALL SLRECL(DCBQ,ILRCL)
ISN 0023  CALL SBLKSZ(DCBQ,IBLKSZ)
ISN 0024  500  READ(5,25,END=6000) MEMBER
ISN 0025  600  CALL FIND(DCBP,MEMBER,'D',IERR)
ISN 0026  IF (IERR.EQ.0) GO TO 750
ISN 0027  WRITE(6,30) MEMBER
ISN 0028  READ (5,25,END=6000)
ISN 0029  GO TO 600
ISN 0030  750  CONTINUE
ISN 0031  GO TO 600
ISN 0032  1000  CALL READ(DECBP,'SF',DCBP,AREA,'S')
ISN 0033  CALL CHECK(DECBP,BLK,IERR)
ISN 0034  IF (IERR.GT.1) GO TO 5000
ISN 0035  ILNGTH=IBLKSZ-BLK
ISN 0036  CALL WRITE(DECBP,'SF',DCBP,AREA,ILNGTH)
ISN 0037  CALL CHECK(DECBP,BLK,IERR)
ISN 0038  IF (IERR.GT.1) GO TO 5000
ISN 0039  GO TO 1000
ISN 0040  2000  CALL CLOSEK(DCBB)
ISN 0041  CALL OPENQ('FT08F002',DCBQ,'INPUT')
ISN 0042  WRITE(6,20) MEMBER
ISN 0043  GO TO 4000
ISN 0044  GO TO 4000
ISN 0045  3000  CALL GET(DCBQ,AREA,IERR)
ISN 0046  IF (IERR.EQ.1) GO TO 4000
ISN 0047  WRITE(6,10) (AREA(J),J-1,80)
ISN 0048  GO TO 3000
ISN 0049  4000  WRITE(6,15)
ISN 0050  5000  WRITE(6,5) IERR
ISN 0051  6000  CALL CLOSE(DCBP)
ISN 0052  STOP
ISN 0053  END
This program reads a particular member from the partitioned data set (FT08F001) and copies it to an intermediate file (FT08F002). The intermediate copy is then reread and deblocked using the QSAM access method. The PDS is OPENed for BPAM processing in ISN 0013, and its DCB parameters are retrieved (record format, record length, and blocksize). A BSAM DCB is obtained in ISN 0017 (for the intermediate copy), and the record format and blocksize are set to U(1RCFB=Z000000C0) and the blocksize of the PDS. It is necessary to use RECFM=U so that the length parameter (ILNGTH) in ISN 0040 will be permitted. It is necessary to use the length parameter so that the last block of a member can be truncated if necessary. After closing the intermediate file (ISN 0045), it is reOPENed (ISN 0046) for QSAM input. Note that the DCB parameters for DCBQ are identical to the original PDS. The members are deblocked and written to the standard output class in ISN 0048 and ISN 0051 respectively.

2. List all members of a partitioned data set.

   a. JCL

   //SYSLIB DD DSN=MYPDS,DISP=SHR
   //SYSUT1 DD UNIT=SASCR,DSN=\$TEMP,SPACE=(CYL,(1,1))

   b. PROGRAM

   ISN 0002 INTEGER*4 BLK
   ISN 0003 INTEGER*2 UCNT
   ISN 0004 INTEGER*2 ACTIVE
   ISN 0005 LOGICAL*1 DIR(256), MASK
   ISN 0006 LOGICAL*1 MEMBER(8)
   ISN 0007 REAL*8 MBR
   ISN 0008 EQUIVALENCE(MEMBER(1),MBR)
   ISN 0009 EQUIVALENCE(DIR(1),ACTIVE)
   ISN 0010 DATA MASK /Z1F/
   ISN 0011 LOGICAL*1 XFF
   ISN 0012 DATA XFF /ZFF/
   ISN 0013 LOGICAL*1 AREA(4000)
   ISN 0014 LOGICAL*1 DSN(44)
   ISN 0015 DATA IPAGE /0/
   ISN 0016 DATA IUSTRFB /192/
   ISN 0017 5 FORMAT(1H,28)
   ISN 0018 10 FORMAT(10(1H ),80(A1))
   ISN 0019 15 FORMAT(1H,'***************END-OF-MEMBER**************')
   ISN 0020 20 FORMAT(1H,30(1H ),'MEMBER=',A8)
   ISN 0021 25 FORMAT(A8)
   ISN 0022 30 FORMAT(1H,10(1H ),6(A1),1H ,3(Z2)1H ,I4)
   ISN 0023 34 FORMAT(1H,30(1H ),'DSNAME=',44(A1))
   ISN 0024 35 FORMAT(1H,30(1H ),'LIST OF PDS MEMBERS')
   ISN 0025 36 FORMAT(1H )
   ISN 0026 37 FORMAT(1H,10(1H ),' NAME ',2H ,TTR',3(1H ),'PAGE')
   ISN 0027 40 FORMAT(1H1,100(1H ),'PAGE ',I4)
ISN 0028  DCBD=0
ISN 0029  IBLS=256
ISN 0030  IDSORG=64
ISN 0031  IRECFM=128
ISN 0032  ILRCL=256
ISN 0033  CALL GDCBB('SYSLIB ',DCBD,'INPUT')
ISN 0034  CALL SRECFM(DCBD,IRECFM)
ISN 0035  CALL SDSORG(DCBD,IDSORG)
ISN 0036  CALL SBLKKSZ(DCBD,IBLS)
ISN 0037  CALL SLRECL(DCBD,ILRCL)
ISN 0038  DCBO=0
ISN 0039  DCBB=0
ISN 0040  DECBO=0
ISN 0041  DCBD=0
ISN 0042  DCBP=0
ISN 0043  DECBP=0
ISN 0044  CALL OPENP('SYSLIB ',DCBP,'INPUT')
ISN 0045  CALL GBLKSZ(DCBP,IUTBLK)
ISN 0046  CALL GRECFM(DCBP,IUTRFM)
ISN 0047  CALL GLRECL(DCBP,IUTCL)
ISN 0048  CALL GDSN(DCBP,DSN)
ISN 0049  WRITE(9,35)
ISN 0050  WRITE(9,34) DSN
ISN 0051  WRITE(9,36)
ISN 0052  WRITE(9,37)
ISN 0053  WRITE(9,36)
ISN 0054  CALL GDCBB('SYSUT1 ',DCBB,'OUTPUT')
ISN 0055  CALL GDCBQ('SYSUT1 ',DCBP,'INPUT')
ISN 0056  CALL SRECFM(DCBB,IUTRFB)
ISN 0057  CALL SRECFM(DCBQ,IUTRFM)
ISN 0058  CALL SBLKKSZ(DCBB,IUTBLK)
ISN 0059  CALL SBLKKSZ(DCBQ,IUTBLK)
ISN 0060  CALL SLRECL(DCBB,IUTCL)
ISN 0061  CALL OPENB('SYSLIB ',DCBD,'INPUT')
ISN 0062  500 CALL READ(DCBE,'SF',DCBP,AREA,'S')
ISN 0063  CALL CHECK(DCBP,GBK,IERR)
ISN 0064  IF(IERR.GT.1) GO TO 500
ISN 0065  IF( IERR.EQ.1) GO TO 6000
ISN 0066  ACTIVE=ACTIVE-2
ISN 0067  J=3
ISN 0070  600 CONTINUE
ISN 0071  IF(DIR(J).EQ.XFF) GO TO 6000
ISN 0073  IPC=IPAGE+1
ISN 0074  WRITE(9,30) DIR(J),DIR(J+1),DIR(J+2),DIR(J+3),DIR(J+4),
           DIR(J+5)*DIR(J+6),DIR(J+7),DIR(J+8),DIR(J+9),DIR(J+10),IPC.
ISN 0075  DO 800 I=1,8
ISN 0076  800 MEMBER(I)=DIR(J-I-1)
ISN 0077  CALL FIND(DCBB,MEMBER,'D',IERR)
ISN 0078  IF( IERR.GT.0) GO TO 5000
ISN 0080  CALL OPENB('SYSUT1 ',DCBB,'OUTPUT')
ISN 0081  1000 CALL READ(DECBP,'SF',DCBP,AREA,'S')
ISN 0082  CALL CHECK(DECBP,GBK, IERR)
This program functions much like the preceding example except that the directory of the partitioned data set (SYSLIB) is read in order to determine the member names. The DCB for the directory is obtained and initialized in ISN 0033 through ISN 0037. Note that the directory's record format (specified 128 DEC or 80 HEX) is fixed blocked records 256 bytes long, and the data set organization (64 DEC or 40 HEX) is physical sequential. These are standard OS/370 MVT conventions for partitioned data set directories. The PDS and intermediate files are OPENed in ISN 0044 and ISN 0061 respectively. A block of the directory is read, and the member names within that block are "deblocked" in ISN 0062, ISN 0076 and ISN 0112 through ISN 0115. As each member is found ISN 0077, the member is copied to the intermediate file (SYSUT1) in ISN 0081 through ISN 0089. After copying, the intermediate file is then deblocked (using QSAM) in ISN 0104.

(1) Cf. IBM SYSTEM/370 Operating System Supervisor and Data Management Services for the format of the directory block.
It is interesting to note the same file (SYSLIB) is OPENed for two different methods (BPAM and BSAM) simultaneously. The LAND and SHFTL are additional bit setting functions invoked by the "XL" option to the FORTRAN H compiler. These functions (or their counterparts) are necessary to manipulate the directory blocks.

3. Copy all members of a data set and resequence each member.

a. JCL

```plaintext
//INPUT DD DISP=SHR,DSN=INPDS,DCB=NCP=2
//OUTPUT DD DISP=(NEW,KEEP),DSN=COPYPDS,
// DCB=NCP=2,SPACE=(CYL,(1,1,5)),UNIT=3330
//SYSIN DD *
```

b. PROGRAM

```plaintext
ISN 0002  INTEGER*4 BLK
ISN 0003  INTEGER*2 UCNT
ISN 0004  INTEGER*2 ACTIVE
ISN 0005  LOGICAL*1 DIR(256),MASK
ISN 0006  LOGICAL*1 MEMBER(8)
ISN 0007  REAL*4 MBR(2)
ISN 0008  DATA MBR /0.0/
ISN 0009  EQUIVALENCE(MEMBER(1),MBR(1))
ISN 0010  EQUIVALENCE(DIR(1),ACTIVE)
ISN 0011  DATA MASK /ZIP/
ISN 0012  LOGICAL*1 XFF
ISN 0013  DATA XFF /ZFF/
ISN 0014  LOGICAL*1 BLANK,ZERO
ISN 0015  DATA BLANK,ZERO /'0','0' /
ISN 0016  DIMENSION IDECBR(120),IBRC(120),IER(120),IDECBN(120),
           ILENGTH(120)
ISN 0017  DATA IDECBR,IDECBN /120*0,120*0/
ISN 0018  LOGICAL*1 SWITCH,AREA(1)
ISN 0019  DATA IDECBR,IDECBN,SWITCH /0,0,.TRUE./
ISN 0020  INTEGER BUFF(120),GETBF
ISN 0021  INTEGER GRCFM,GRECBL,GBLKSZ
ISN 0022  DATA DCDIR,DECID /0,0/
ISN 0023  READ(5,50) ISTRT,INCR,K
ISN 0024   50 FORMAT(18,18,12)
ISN 0025   WRITE(6,55) ISTRT,INCR,K
ISN 0026   55 FORMAT(1I8,'START SEQUENCE=',18,'INCR=',18,9,'BUFFERS=',12)
ISN 0027  CALL GCDBP('INPUT ','IDCBR,'INPUT')
ISN 0028  CALL GCDBP('OUTPUT ','IDCBN,'OUTPUT')
ISN 0029  CALL GETPL(IDCBR,K)
ISN 0030  CALL OPENP( 'INPUT ','IDCBR,'INPUT')
ISN 0031  INCFM=GRCFM(IDCBR)
ISN 0032  ILACL=GRECBL(IDCBR)
ISN 0033  IBLKSZ=GBLKSZ(IDCBR)
ISN 0034  CALL SRBCFM(IDCBN,INCFM)
ISN 0035  CALL SBLKSZ(IDCBN,IBLKSZ)
```
CALL SLRECL(IDCBW, ILRCL)

IBLFCr = IBLKSZ/ILRCL

DO 120 i = 1, K

MFF(I) = GETBF(I, M)

CALL GDCBB('INPUT ', DCBDIR, 'INP1r')

CALL SRECI4(DCBDIR, 128)

CALL SDSORG(DCBDIR, 64)

CALL SBLKSZ(DCBDIR, 256)

CALL SLRECL(DCBDIR, 256)

CALL OPENB('INPUT ', DCBDIR, 'INPUT')

CALL READ(DCBR, 'SF', DCBDIR, 'S')

IF(KERR.EQ.1) GO TO 6000

IF(KERR.EQ.0) GO TO 520

WRITE(6,510) KERR

WRITE(6,610) KERR

IF(DIR(N).HQ.XFF) GO TO 6000

DO 630 J = 1, IBLCr

IF(AREA(IWFF(1)-IADIN(AREA)+LL+ILICL*(J-1)) .NE.BLANK) GO TO 626

AREA(IWFF(1)-IADIN(AREA)+LL+ILICL*(J-1)) = ZEEM

WRITE(6,620) KERR

CALL CHECK(IDCBW(1), IRBC(1), IERR(1))

IF(IERR(1).GE.1) D TO 3000

DO 625 LL = 73, 80

IF(AREA(BUFF(1)-IADDR(AREA)+LL+ILRCL*(J-1)).NE.BLANK) GO TO 626

IF(AREA(BUFF(1)-IADDR(AREA)+LL+ILRCL*(J-1)).NE.BLANK) = ZERO

CONTINUE

IBLFT = IBLKSZ - IRBC(1)

CALL SBLKSZ(IDCBW, IBLFT)

IGN = 1

WRITE(6, 700) KERR

CALL CHECK(IDCBW(1), IRBC(1), IERR(1))
ISN 0090 IF(IERR(1).GE.1) GO TO 1000
ISN 0092 CALL NOTE(IDCBEW,MLKADD)
ISN 0093 640 DO 700 I=1,K
ISN 0094 ICNT=I
ISN 0095 IF(SWITCH) GO TO 650
ISN 0097 CALL CHECK(IDCBEW(I),IRBC(I),IERR(I))
ISN 0098 IF(IERR(I).GE.1) GO TO 1000
ISN 0100 650 CONTINUE
ISN 0101 CALL READ(IDCBEW(I),'SF',IDCBR,AREA(IJFF(I)-IADR(AREA)+1),'S')
ISN 0102 700 CONTINUE
ISN 0103 SWITCH=.FALSE.
ISN 0104 DO 800 I=1,K
ISN 0105 ICNT=I+1
ISN 0106 CALL CHECK(IDCBEW(I),IRBC(I),IERR(I))
ISN 0107 IF(IERR(I).EQ.1) GO TO 900
ISN 0109 IF(IERR(I).GT.1) GO TO 2000
ISN 0111 DO 750 J=1,IBLRT
ISN 0112 CALL CONVO(('(18)',AREA(BUFF(I)-IADDR(AREA)+73+ILRCL*(J-1)),0,8,*ISEQ)
ISN 0113 ISEQ=ISEQ+INCR
ISN 0114 DO 740 LL=3,80
ISN 0115 IF(AREA(IJFF(I)-IADR(AREA)+LL+ILRCL*(J-1)).EQ.BLANK) GO TO 745
ISN 0117 740 AREA(BUFF(I)-IADDR(AREA)+LL+ILRCL*(J-1)).NE.BLANK) GO TO 745
ISN 0118 745 CONTINUE
ISN 0119 750 CONTINUE
ISN 0120 750 ILENGTH(I)=IBLKSZ-IRBC(I)
ISN 0121 CALL SBLKSZ(IDCBEW,ILENGTH(I))
ISN 0122 CALL WRITE(IDCBEW(I),'SF',IDCBR,AREA(BUFF(I)-IADDR(AREA)+1),'S')
ISN 0123 800 CONTINUE
ISN 0124 GO TO 640
ISN 0125 900 IF(ICNT.GT.K) GO TO 915
ISN 0127 DO 910 I=ICNT,K
ISN 0128 CALL CHECK(IDCBEW(I),IRBC(I),IERR(I))
ISN 0129 915 CONTINUE
ISN 0130 CALL STON(IDCBEW,MEMBER,'A',IERR(I))
ISN 0131 IF(IERR(I).EQ.0) GO TO 917
ISN 0133 WRITE(6,916) MEMBER,IERR(I)
ISN 0134 916 FORMAT(1H 'MEMBER NOT STOWED-' ,8(A1),',IERR=','Z8)
ISN 0135 CALL ABEND
ISN 0136 917 CONTINUE
ISN 0137 WRITE(6,920) MEMBER
ISN 0138 920 FORMAT(1H 'COPIED MEMBER=','8(A1))
ISN 0139 950 SWITCH=.TRUE.
ISN 0140 UCNT=SHIFTL(LAND(DIR(N+11),MASK),1)
ISN 0141 N=N+UCNT+12
ISN 0142 ACTIVE=ACTIVE-UCNT-12
ISN 0143 IF(ACTIVE.GT.0) GO TO 590
ISN 0145 GO TO 500
ISN 0146 1000 WRITE(6,1010) MEMBER,IERR(ICNT)
ISN 0147 1010 FORMAT(1H 'WRITE ERROR ON MEMBER=','8(A1),',IERR=','Z8)
This program assumes that each member of the PDS is to be resequenced
the same way (i.e., the starting sequence and increment are the same).
It also assumes that the data set consists of fixed block 80 character
card images. The single input card read (ISN 023) has the format:

SSSSSSSS IIIIIII BB

where SSSSSSSS is the starting sequence number for each member,
IIIIIII is the increment to be added to SSSSSSSS and BB is the level
of buffering. The buffer pool is built in ISN 0029, and the directory
is read in ISN 0047. The FIND for each member is type "C" since the
second parameter points to the in core directory just read in
(ISN 0064). This avoids much overhead reading the directory for each
FIND. The buffering technique is identical to the BSAM example
(ISN 0101 through ISN 0122) except that it was necessary to NOTE the
TTR of the first record for each member with no other outstanding I/O
(ISN 0092). The CONVO statements cause incore conversion of the
sequence numbers to external (output) formats (ISN 0077 and ISN 0112).

The CONVO subroutine is a locally available subroutine at ANL. (1)

After copying each member of the data set, a STOW executed for each
member updating the directory of the copy (ISN 0130).

V. Outline of Program Modules

A. OPEN

1. Functional Description

   This module initializes the DCB area and OPENs the data set.

2. Entry Points

   OPENB  GRECFM  SRECFM  GLAB
   OPENQ  GLRECL  SLRECL  GOFLG
   OPENP  GLKLCZ  SBLKSS  SOFLG
   GDORB  GDEN   SDEN   SBUFCC
   GDORQ  GTKCH  STKCH  SBUFCO
   GDORP  GDSORG  SDSORG  RDJFCB
   GVOL   GDEN   SDEN   CONCAT

(1) ANL K250S-1, Core to Core Conversion, by Daniel Carson (Oct. 1967)--ANL
3. Core Storage Requirements

2970 (B9A HEX) Bytes

4. Internal Logic

The OPEN subroutines (BSAM, QSAM and BPAM) check to see if a DCB has been provided (value of dcb parameter ≠ 0) and OPENs (TYPE=J) the user provided DCB is yes. Otherwise, a DCB area is obtained from subpool 0 via GETMAIN, the option list, and DCB are initialized, and the file is OPENed (TYPE=J). The GDCB subroutines get a DCB area (via GETMAIN-subpool 0), initialize the gotten area, but do not OPEN the data set. The GET/SET routines simply copy/plant the correspond parameter in the user provided DCB.

B. RDWRRT

1. Functional Description

This module contains the various BSAM processing subroutines.

2. Entry Points

READ   CNTRL
WRITE   POINT
CHECK   NOTE

3. Core Storage Requirements

1844 (734 HEX) Bytes

4. Internal Logic

The READ/WRITE subroutines obtain an area via GETMAIN for the DCB if none exists. If no DCB is provided, they first call the OPENB routine to initialize and OPEN a DCB. Finally, the appropriate operation (READ/WRITE) is executed. Note that if a previous READ resulted in an EOF condition, the subsequent READ is issued for the first block of the next file. The CHECK routine sets up SYNAD and BODAD exits in the DCB, executes the OS CHECK macro, and returns the resulting status information to the user program. CNTRL and POINT simply execute the appropriate OS macro.

C. GETPUT

1. Functional Description

This module contains the various QSAM processing routines.

2. Entry Points

GET   RELSE
PUT   TRUNC
3. Core Storage Requirements
822 (336 HEX) Bytes

4. Internal Logic

The GET/PUT subroutines first call the OPENQ routine if no DCB is provided. Then the appropriate operation is executed. Note that if a previous GET resulted in an EOF condition, the subsequent GET is issued for the first record of the next file.

D. POSTN

1. Functional Description

This module provides certain file positioning functions.

2. Entry Points

WEOF
SKIP
SETSEQ
GETSEQ

3. Core Storage Requirements

664 (298 HEX) Bytes

4. Internal Logic

The WEOF subroutine CLOSEs the file (via the CLOSEK subroutine), bumps the file sequence number in the JFCB control block, and reOPENS the file (via OPENB). The SKIP subroutine is identical except that the file sequence number is bumped by the user provided skip count. The SETSEQ subroutine simply bumps the current file sequence number in the JFCB by the user provided count. GETSEQ retrieves the file sequence number from the JFCB.

E. PAR

1. Functional Description

This module provides BPAM processing subroutines.

2. Entry Points

FIND
STCW
BLDL

3. Core Storage Requirements

620 (26CHEK) Bytes
4. Internal Logic

Each of these subroutines simply executes the appropriate OS macro.

F. LDDEL

1. Functional Description

This module provides the dynamic loading functions.

2. Entry Points

LOAD
LINK
DELETE
CALL

3. Core Storage Requirements

332 (14CHEX) Bytes

4. Internal Logic

These subroutines simply execute the appropriate OS macros.

G. CLSE

1. Functional Description

This module disconnects the data set from processing.

2. Entry Points

CLOSE
CLOSEK

3. Core Storage Requirements

641 (281 HEX) Bytes

4. Internal Logic

The appropriate CLOSE macro is executed and then--for CLOSE only--the DCB area is freed via FREEMAIN or DELETED depending upon whether the DCB was gotten via GETMAIN or a pre-assembled DCB was LOADED (cf. DCBAREA macro, SECTION III). For CLOSEK and CLOSE (TYPE=T), the DCB is retained for future use.

H. STORAG

1. Functional Description

This module performs dynamic storage allocation, formats buffer pools and provides addressability to these areas.
2. Entry Points

BUILD  FREEPL
GETPL  FREEBF
GETBF  FREEMN
GETMN  IADDR

3. Core Storage Requirements

1172 (494 HEX) Bytes

4. Internal Logic

The appropriate macro is executed and the result is returned to the calling program.

I. DCBCNTRL

1. Functional Description

This module provides a BSAM DCB allowing use of the CNTRL subroutines.

2. Entry Points

DCBCNTRL
GETMAIN

3. Core Storage Requirements

421 (1AS HEX) bytes

J. DCBPOINT

1. Functional Description

This module provides a BSAM DCB allowing use of the POINT(NOTE) subroutines.

2. Entry Points

DCBPOINT
GETMAIN

3. Core Storage Requirements

421 (1AS HEX) bytes
VI. Error Processing

Attempts are made throughout the modules to check the validity of the arguments as far as that is possible. In case an error is detected, an appropriate message is written into the SYSMSG data set (via WIO, ROUTCDE=11); and an ABEND DUMP is produced. Register 10 points to the module base which is active; register 9 points to the parameter list passed to the program.

The following data sets are available:

A. C245.FORTIO.SYMBL--a partitioned data set whose members are the FORTIO module source decks suitable for assembly with the F-level assembler.

B. C245.FORTIO.MACRO--a partitioned data set whose members are the FORTIO macros which are necessary for assembly of the modules.

C. C245.FORTIO.LOAD--load modules (partitioned data set)

D. C245.FORTIO.OBJ--object decks (sequential data set)

Finally, all pertinent materials will be available through the Argonne Code Center.

VII. Program Materials Available

<table>
<thead>
<tr>
<th>Module</th>
<th>Symbolic Deck</th>
<th>Object Deck</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLSE</td>
<td>175</td>
<td>16</td>
</tr>
<tr>
<td>DCBCNTRL</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>DCBPOINT</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>GETPUT</td>
<td>203</td>
<td>21</td>
</tr>
<tr>
<td>LDDEL</td>
<td>103</td>
<td>11</td>
</tr>
<tr>
<td>OPEN</td>
<td>511</td>
<td>68</td>
</tr>
<tr>
<td>PART</td>
<td>154</td>
<td>17</td>
</tr>
<tr>
<td>POSTN</td>
<td>178</td>
<td>18</td>
</tr>
<tr>
<td>READWRT</td>
<td>393</td>
<td>40</td>
</tr>
<tr>
<td>STORAG</td>
<td>247</td>
<td>28</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>1974</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macro</th>
<th>Deck</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>29</td>
</tr>
<tr>
<td>DCBAREA</td>
<td>83</td>
</tr>
<tr>
<td>IEFJFCB</td>
<td>116</td>
</tr>
<tr>
<td>LYNK</td>
<td>5</td>
</tr>
<tr>
<td>REGISTER</td>
<td>23</td>
</tr>
</tbody>
</table>