SHARE PROGRAM LIBRARY AGENCY

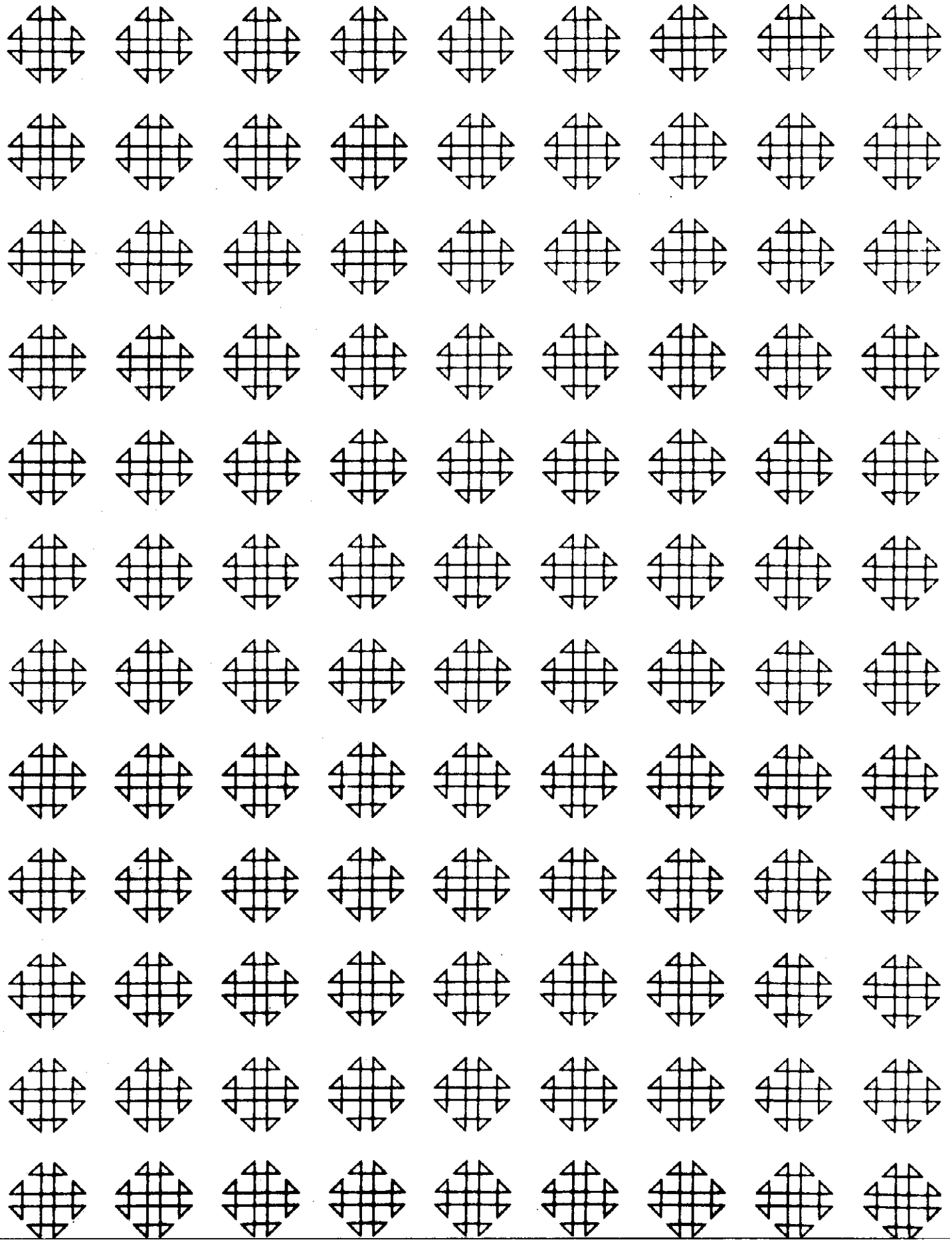
PROGRAM NUMBER

063012

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA

(305) - 284-6257



DISCLAIMER

This program and its documentation have been contributed to the Program Information Department by an IBM customer and are provided by the IBM Corporation as part of its service to customers. The program and its documentation are essentially in the author's original form and have not been subjected to any formal testing. IBM makes no warranty expressed or implied as to the documentation, function, or performance of this program and the user of the program is expected to make the final evaluation as to the usefulness of the program in his own environment. There is no committed maintenance for the program.

Questions concerning the use of the program should be directed to the author or other designated party. Any changes to the program will be announced in the appropriate Catalog of Programs; however, the changes will not be distributed automatically to users. When such an announcement occurs, users should order only the material (documentation, machine readable or both) as indicated in the appropriate Catalog of Programs.

DISCLAIMER

Triangle Universities Computation Center (TUCC) serves solely as the distribution agent for contributed programs and does not test or maintain them. They are distributed essentially in the original form submitted by the author. Neither TUCC nor SHARE, INC., makes any warranty expressed or implied, as to the documentation, function, or performance of the contributed programs.

CONTRIBUTED PROGRAM LIBRARY SUBMITTAL FORM
(for IBM S/360, 1130 and 1800)

IBM Corporation
Program Information Department (PID)
40 Saw Mill River Road
Hawthorne, New York 10532, U.S.A.
Attention: Program Control Desk

This form should be completed and submitted with the program package to PID at the address shown above. Standards and instructions for submitting programs are in your *User Group Reference Manual* or the *Contributed Program Submittal Standards Manual* available from PID.

(1) **Program Order Number** *(to be filled in by PID)* 360D-06.3.012

(2) **System Type** *(machine)* S/360

(3) **Search Key** . . . . . 360 PACKAGE FOR HI-SPEED BISYNCHRONOUS COMMUNICATIONS TO AN 1130 OR 360

(4) **Programming Language** . . . . . 360 OS MVT

(5) **Author's Name and Address** . . . . George M. Stabler

Mr. G. M. Stabler
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, MA 01701

(6) **Direct Inquiries to Name and Address** *(if different than Author)*

(7) **Title of Program** . . . . A HIGH SPEED BISYNCHRONOUS COMMUNICATIONS ACCESS METHOD

(8) **Submitter's User Group Affiliation Code and Installation Code** . . . . . S B.U.C.

(9) **Submitter's Own Program Identification and Suffix** *(optional)* . . . . .

(10) **Primary Subject Code** 06.3

(11) **Secondary Subject Codes** . . . . . 03.4 . . .

(12) **Operating or Monitor System Required** . . . . . OS/360 MVT

(13) **New or Revision Code** *(if revision, show prior Program Order Number in item 1)* . . . . . N

(14) **Year Completed** 69

(15) **Date of Submittal** 09 22 69

(16) **Documentation** *(number of original pages submitted)* 21

(17) **Abstract** *(should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.*

CONTRIBUTED PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide

a. Purpose
b. Programming Language used
c. Version and modification level or release number of IBM Programming System used, or program order number for non-IBM authored program used
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements
g. Engineering Changes (EC) level of equipment (if pertinent)

ABSTRACT

The Bisynchronous Communications Access Method is a set of 360 programs which support queued telecommunications over a high speed (40.8K BAUD) point-to-point half duplex line connecting a 360 and an 1130 or another 360.

The system, which employs EXCP for all I/O and handles all message blocking, line protocol, and error checking internally, is called at the GET/PUT level from assembly language programs. The system will support any number of logical users (message destinations) in either machine, and can support any number of remote terminals. Since all user messages are transmitted in "transparent text" mode, there are no restrictions on the type of data a user may send. Line protocol conforms to conversational Bisynchronous Communications standards.

Support analogous to this package is provided for an 1130 system by the Type 4 program entitled "An 1130 High Speed Bisynchronous Communications System".

(Please attach additional pages if necessary) . . . . . **Total pages attached** _____

Permission to Publish

"I hereby give anyone permission to reprint, reproduce, and distribute this program to anyone else."

(18) **Signature of Submitter and Date** George M Stabler 9/22/69

(19) **Signature of Installation Addressee** Mary S. Kline 9/22/69

T4SF

TABLE OF CONTENTS

TAPE KEY

This volume contains a volume label, two files, and seven tape marks. All files are unloaded partitioned data sets with LRECL =80, BLKSIZE =800. The files are EBCDIC card image and were created with IEHMOVE.

VOLUME LABEL: VOL = SER = BSCAM

T/M

Header: 3 80 byte records.

T/M

FILE 1: The PDS named BSCAM. 192 records; last record length 80.

T/M

Trailer: 2 80 byte records.

T/M

Header: 2 80 byte records.

T/M

FILE 2: The PDS named MACROS. 47 records; last record length 480.

T/M

Trailer: 2 80 byte records.

T/M

SUMMARY

The High Speed Bisynchronous Communications Access Method (BSCAM) consists of a set of 360 programs which were originally written for a low speed (2400 BAUD) line by Messrs Hendricks and Tuttle at the IBM Cambridge Scientific Center in Cambridge, Mass. These programs have been extensively rewritten and adapted at Brown University for use on a high speed (40.8K BAUD) link between a S/360 Mod 50 and 1130 Mod 2B or higher or another S/360.

The system provides queued telecommunications support for an arbitrary number of users in either machine. A user is no more than a four character identifier to which messages can be directed from the other machine.) Messages can be sent from any user in one machine to any user in the other, the routing being controlled by four character 'to' and 'from' identifiers preceeding each message. Since all user messages are sent in transparent text mode in which any 8 bit character can be sent over the line, there are no restrictions on the type of data a user may transmit. The communications package was written in assembly language and will support communications to an arbitrary number of 1130 (or 360) terminals.

An 1130 program supporting BSCAM is also in the Type IV library. It is entitled "An 1130 High Speed Bisynchronous Communications System" (Program Number 1130-06.3.005 [illegible].

ADVANTAGES-LIMITATIONS

The chief advantages of the BSCAM package are its small size and simplicity of use. Since the package was written specifically for high-speed bisynchronous communications, the user is not saddled with the overhead in time and space needed to support a more general range of communications needs. Since there are only four calls to the GET/PUT module (SIGN-ON, GET, PUT, SIGN-OFF), use of BSCAM is easily and quickly learn.

At the present time there are certain desirable facilities which have not been included in the system. Two of these

extensions are the ability to request a message from a particular user (rather than the first message to be received), and the option of providing an ECB to test for successful transmission of messages. In addition, the current system assumes that the user's program will be of the 'non-ending monitor' variety (e.g., IBM's Satellite Graphic Job Processor), thus no provisions have been made to close communications normally.

MACHINE CONFIGURATION

The configuration under which the BSCAM can operate is as follows:

1. A 360 Mod 40 or higher with 256K bytes of core storage running under OS Release 15-16 or higher (MVT).
2. A 2701 Data Adapter Unit equipped with a Synchronous Data Adapter Type II and the Transparency feature.
3. Two 301-B or equivalent data sets operating at 40.8K BAUD over a half or full duplex line (the communications package only makes use of the half duplex capabilities). The data sets currently in use were manufactured by Tuck Electronics (New Cumberland, Pa.).
4. An 1133 Multiplex Control Enclosure comprising 1865 Channel Multiplexer, 7492 Storage Access Channel II, 831551 Expansion 1133, and #21059 Communications Adapter.
5. An 1131 CPU Model 2B or higher.

Since the BSCAM logic is essentially machine independent, the same package can also be used for communications between two 360's. In this case, numbers 4 and 5 above can be replaced by 1 and 2.

SOURCE LANGUAGE AND PROGRAMMING SYSTEM

The BSCAM package was written in assembly language and uses EXCP for all I/O. All error checking is internal; no IBM supplied error routines are used. Data-link control procedures follow BSC conversational conventions.

The package runs under OS/360 Release 15-16 or higher and requires the facilities of MVT. Core requirements are as follows:

Line Monitor - 2100 bytes
GET/PUT Routine - 1100 bytes
Control Blocks - for each line: 700 bytes plus the maximum message length to be allowed (currently 842 bytes) plus sufficient space for input/output queues. (All control blocks and queue elements are acquired dynamically.)
Communications Trace Routine (optional) - 1600 bytes.

The following paragraphs provide the information needed by a user in order to use the BSCAM system. The concepts of the user, the logical message, and the functions of the various calls to the GET/PUT module are explained. The manner in which the user's program interfaces with the communications task is described. A description of the physical messages, line protocol, and generation of the system is left to the section on System's Information.

THE_USER

BSCAM considers all communications as being between users which are distinguished by four character user identifiers. These identifiers are made known to the communications routines by the SIGN-ON function of the GET/PUT routine. A user in this sense has no relation to any particular module, OS task, or machine user; one module can sign on as several different users, and a series of modules can communicate under one user name.

Once an identifier has been established by the sign-on function, it is referred to by that user in all subsequent calls to the GET/PUT module. In calls for GET, it is used to control the routing of input messages; in PUTs, it serves as a check on the validity of the user.

THE_USER_MESSAGE

A message to be sent between users consists of two parts: a message header and text or data. The message header consists of the following items:

1. The identifier of the sender. This is the four character identifier of the user who is sending the message.
2. The identifier of the addressee.
3. A halfword containing the length of the following message text.

The message text or data immediately follows the message header and can consist of from 0 to 830 bytes (the maximum length can be changed as noted under 'Generation of BSCAM Modules'). Since all user data is transmitted in transparent mode, the message text can contain any bit configuration.

When a user wishes to send a message, he passes the GET/PUT routine the address of the first byte of the header which is immediately followed by the text. A sample message could be assembled as follows:

```
MESSAGE  DC    CL4'SNDR'          IDENTIFIER OF SENDER
         DC    CL4'RCVR'          IDENTIFIER OF RECEIVER
         DC    AL2(TEXTEND-TEXT)  LENGTH OF MESSAGE
TEXT     DC    C'THIS IS THE MESSAGE TEXT'
TEXTEND  EQU   *                  END OF MESSAGE TEXT
```

USE OF THE SYSTEM

The GET/PUT module currently supports four user calls: SIGN-ON, GET, PUT, and SIGN-OFF. All calling sequences are of the form:

```
         LA    1,PARMLIST  (PARMLIST MUST BE FULL-WORD ALIGNED)
         L     15,=V(getput-routine)
         BALR  14,15
*        CHECK RETURN CODES IN REGISTER 15
```

The descriptions of the different PARMLISTs are described under the corresponding call. (Obtaining the entry point for the GET/PUT routine is described in the following section.)

SIGN-ON

Purpose: To initiate communications under a particular identification.

Parameter list:
```
DC XL1'PP',AL3(USERID)  USERID IS THE ADDRESS OF A
                        FOUR CHARACTER IDENTIFIER
```

Return Codes:
- 0 - Normal return
- 4 - Insufficient space for Sign-On
- 8 - Identifier is already in use (if PP = 04)

- 12 - Line is not initialized
- 16 - Invalid call

Notes: A user must sign on before issuing any GETs or PUTs under the particular identifier.

There are two versions of the SIGN-ON function. If the first byte of the parameter list (PP) is set to '20', any previous user (and his associated input or GET queue) using the same identifier is removed from the communications queues before the new sign-on. If PP is set to '04', the existance of a user already using the identifier will result in a return code of 8 and no other action. The first version of SIGN-ON (PP = 20) should normally be used in a environment in which it is possible for a user to ABEND without removing out of date GET requests from the system. This protects the I/O module against trying to pass a message to a non-existant user.

GET

Purpose: To request a message from a user in the other machine.

Parameter list:
```
DC  XL1'80',AL3(USERID)  THE ADDR OF THE USER
                         ISSUING THE GET
DC  H'0' RESERVED
DC  H'BUFLEN' THE LENGTH OF THE USER'S BUFFER
DC A(BUFFER) ADDRESS OF BUFFER IN WHICH MESSAGE
                         IS TO BE PLACED
DC  A(ECB) ADDRESS OF ECB TO BE POSTED ON RECEIPT
                         OF THE MESSAGE
```

Return Codes:
- 0 - Normal return
- 4 - BUFLEN too small (i.e., less than 10)
- 8 - User has not signed on
- 12 - Line is not initialized
- 16 - Invalid call

Notes: A GET call should be used in a manner similar to a BSAM READ, that is, when control returns to the user, the I/O event (in this case the receipt of a message) may or may not have occurred. The user must therefore provide an event control

block (ECB) which is posted on completion of the event. There are two possible completion codes: X'40000000', meaning normal completion, and X'4800xxxx', meaning the message was too large for the user's buffer. In this case, 'xxxx' is the length of the message text as received by the I/O monitor, and the length in the message header is the amount of data passed to the user.

Any messages directed to the user before he signs on or after he signs off are lost, however messages received after he signs on but before he issues a GET request are queued in order of reception.

PUT

Purpose: To send a message to a user in the other machine.

Parameter list:
```
DC   XL1'40',AL3(USERID)    ADDRESS OF SENDER'S IDENTIFIER
DC   AL1(LINEID),AL3(MSGHEADER)  LAST DIGIT OF DDNAME FOR THE LINE, AND ADDRESS OF THE MESSAGE HEADER
```

Return Codes:
- 0 - Normal return
- 4 - Message Length is greater than maximum allowed (see section on BSCAM generation)
- 8 - Sender has not signed on
- 12 - Line is not initialized
- 16 - Invalid call
- 20 - Line does not exist

Notes: A request to PUT a message results in the message being queued for output on the requested line. When control returns to the user, there is currently no way of knowing whether the message has actually been sent or not.

If the user to whom the message is directed has not signed on, the message is moved to the end of the output queue to await retransmission. Ten retrials without success result in the message being purged.

When control returns to the user, the first digit of the LINEID will be set to F.

SIGN OFF

Purpose: To terminate communications under an identifier.

Parameter list: DC XL1'10',AL3(USERID) ADDRESS OF USER IDENTIFIER

Return Codes:
- 0 - Normal return
- 4 - (not used)
- 8 - Identifier not in use
- 12 - Line not initialized
- 16 - Invalid call

Notes: The user's identifier is removed from the list of current users, and all GET requests or input messages waiting for the user are purged. Messages waiting for transmission will not be purged.

RUNNING A USER PROGRAM UNDER BSCAM

LINKAGE TO THE GET/PUT ROUTINE

The task structure of a program using BSCAM is shown in Figure 1. The user executes the program named ICP, which constructs the communications control blocks, ATTACHes the Line Monitor to handle I/O activity, and XCTLs to the user's program.

(Note: If the user's program in turn ATTACHes a subtask which will use communications, the ATTACH macro should include the operand 'SHSPV=123' to give the task access to the subpool used by the BSCAM routines.)

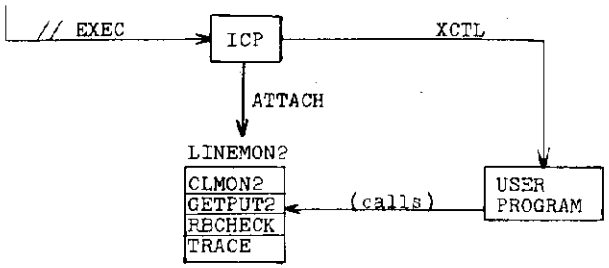


Figure 1. Task Structure Under BSCAM

Since the GET/PUT routine is part of the I/O module, it is not link-edited with the user's program, and he cannot refer to that routine by means of a VCON. Two methods are provided for obtaining this entry point:

1. When the user's program receives control, register 1 contains the address of a parameter list as described in IBM's 'Supervisor and Data Management Services' manual (Form C28-6646). Following the address of the parameters from the EXEC card is the address of SUPERBLOK which is the start of the I/O queues. Included in this block is the entry point of the GET/PUT routine. The user can obtain addressability for this by (a) including the instruction 'SUPER TYPE=DSECT' in his assembly; (b) loading a base register with the address of SUPERBLOK; and (c) referring to the symbol GETPUTAD. The following instructions demonstrate this method:

```
// EXEC ASMF
//ASM.SYSLIB DD DSN=SYS1.YOUR.MACLIB,DISP=SHR
// DD DSN=SYS1.MACLIB,DISP=SHR
//ASM.SYSIN DD *
         SUPER TYPE=DSECT
PROGRAM  CSECT
* PERFORM ENTRY FUNCTIONS
         L     10,4(1)        GET ADDRESS OF SUPERBLOK
         USING SUPER,10       ADDRESSABILITY
* OTHER USER STUFF BEFORE NEEDING COMMUNICATIONS
         LA    1,SNONPRMS     PARMS FOR SIGN-ON
```



```
         L     15,GETPUTAD    GET ENTRY POINT OF GETPUT
         BALR  14,15          CALL IT FOR SIGN ON
```

2. The other method of obtaining the entry point for GETPUT is for the user to link-edit the module IGETPUT into his program. This module scans the I/O control blocks, obtains the entry point, and passes control to the GETPUT routine. A calling sequence for GETPUT would appear as:

```
         LA    1,PARMLIST
         L     15,=V(IGETPUT)
         BALR  14,15
```

The IGETPUT module is approximately 80 bytes and is re-entrant.

JCL REQUIREMENTS

The user specifies the name of the program which he wishes to execute by use of the PARM field of the EXEC card as shown in the sample job below. The name of the user's program is optionally followed by a comma and parameters which are to be passed to the program. Communications lines which are to be used are specified by including DD cards of the form

```
//COMLINOx DD UNIT=comm-unit
```

where 'x' is the zoned number which the user refers to in PUT requests. In addition to DD cards for communications lines, the only other DD cards needed by BSCAM are

1. A JOBLIB or STEPLIB from which LINEMON2 and the user's program are to be fetched;
2. A '//TRACE DD SYSOUT=A' if a trace of line activity is desired (this assumes that the CLTRACE module has been included in LINKMON2).

A sample communications job might thus appear as follows:

```
//LINK     JOB  ACCT.INFO.,TJ
//JOBLIB   DD   DSN=SYS1.BSCAM.LOADMODS,DISP=SHR
//         DD   DSN=SYS1.USER.LINKLIB,DISP=SHR
//         EXEC PGM=ICP,PARM='USERPROG,PARMS FOR USER PROGRAM'
//COMLIN01 DD   UNIT=050     A 2701 AT 050
//TRACE    DD   SYSOUT=A     USER WANTS TO TRACE LINE ACTIVITY
//SYSUDUMP DD   SYSOUT=A     ALWAYS A GOOD IDEA
/*
```

PROGRAM LOGIC

The following paragraphs describe the functions performed by the various BSCAM modules. There are three main modules: the ICP module which initializes communications; the LINEMON2 module, which consists of the I/O monitor, the GETPUT module, an RB check routine, and one or two trace routines; and the IGETPUT module which is link-edited with the user's program.

ICP -- LINE INITIALIZATION

The ICP module is executed as the job step task. It performs the following functions:

1. The Task I/O Table (TIOT) for the job is searched for all DD cards which refer to communications lines to be used by the link. DD cards for communications lines are recognized by being of the form '//COMLINOx DD UNIT=yyyy', where 'x' can be any digit from 0 to F (hex). (This digit is also the line identifier which is used in PUTs. However the user need not include the zone in the line ID, that is, 'AL1(2)' will work as well as 'CL1'2' for referring to DDNAME 'COMLINO2'.)
2. For each line, a Line Control Block is constructed. This block contains a DCB for the line, an IOB, an I/O buffer, and workareas used by the I/O monitor.
3. For each line, the I/O monitor, which is reentrant, is attached as a subtask to initialize and maintain activity on the line.
4. Once all lines have been initialized, the initialization routine lowers its dispatching priority (thus giving the I/O monitor top priority for the job) and transfers control to the user's program (via XCTL). The name of the user's program is obtained from the PARM field of the EXEC card for the job step, and any other parameters on the EXEC card are set up as if the user's program was being executed as the job step task.

LINEMON2--THE I/O MONITOR AND GETPUT MODULE

LINEMON2 is composed of three modules--the line I/O monitor (CLMON2), the GET/PUT module (GETPUT2), and an RB check routine (RBCHECK). The I/O monitor and the GET/PUT routines operate asynchronously to each other, the I/O monitor getting control from ICP via the ATTACH function, and the GETPUT module getting control by means of user calls.

The line I/O monitor is in charge of initializing and maintaining all line activity. The monitor operates at the EXCP level and provides all its own error checking and recovery procedures. It performs the following functions:

1. The address of the line control block (passed from the initialization routine) is obtained, the 2701 is enabled, and an initial READ (actually a PREPARE followed by a READ) is issued.
2. On being notified of message(s) waiting for transmission (via a POST from the GET/PUT module) an IOHALT SVC is issued to halt the read-initial channel program. An ENQ is sent, and normal line activity commences (see Line Protocol).
3. Line activity may also be initiated by receiving an ENQ from the other processor. In this case, the I/O monitor is posted by IOS on completion of the read-initial channel program. A DLE-ACK0 control sequence is then transmitted, and line activity proceeds.
4. During line activity output messages are blocked into system messages for transmission, input messages are deblocked into user messages and are either queued or given directly to the user (the user is posted), and reply headers giving the status of messages received and sent are processed.

The GET/PUT module is called by the communications user who has obtained the entry point from the SUPRBLOK or from the IGETPUT routine. The main functions performed by this module are the following:

1. Calls for SIGN-ON and SIGN-OFF, by which the user initiates and terminates communications under a particular identifier, result in the creation and deletion of User Control Blocks (USCBs). A USCB is used as the start of the user's queues of input elements or GET requests.
2. A GET call builds an GET Request Queue Element (GRQE) which is queued on the user's USCB to await an input message directed to the user. Alternatively, if a message satisfying the GET call is already on the user's input

queue (an IPQE), it is dequeued and passed immediately to the user, and the user is posted.

3. A PUT call results in the creation of an output queue element (OPQE) which is placed on the queue of output messages for the requested line. If the line is currently inactive, the line monitor is posted to tell it that output is available for transmission.

The RB check routine is called by the I/O monitor and checks the validity of RB addresses in ECB's which are to be posted. Due to the asychronous nature of the I/O monitor, it is possible for a communications user to terminate or ABEND without the knowledge of I/O monitor. The RB check routine is passed the address of an ECB and searches the currently active TCB and RB chains to determine if the RB is still active. The condition code is set to zero or non-zero depending on the success of this search.

In addition to the three modules described above, systems using only one communications line can optionally include the two trace routines CLTRACE and TRACE (these routines are non-reentrant). The first of these prints out a complete trace of line activity including IOB information, the CCWs used, the contents of the line buffer after each received message, and the times spent in execution and waiting between I/O events. The trace can be turned on and off in two ways: by omitting the module from the LINEMON2 module and by removing the '//TRACE DD SYSOUT=A' card from the job. CLTRACE requires the small TIME module to provide elapsed times.

The TRACE routine provides a small wrap-around trace of the CCWs, status indicators, and the first byte of data received in each message (i.e., the first byte in the line buffer). The trace is kept in a 400 byte area enclosed in the identifiers 'TRACE' and 'TRACEND'. For a complete description of the format of the trace entries, see the listing of TRACE. This routine must always exist as an entry point in the LINEMON2 module. If the CCW trace is not desired, it should be replaced with a 'BR 14'.

IGETPUT -- USER INTERFACE ROUTINE

In some user applications, it may prove to be difficult to pass the address of SUPERBLOK to all user programs wishing to use communications. For this reason, the IGETPUT routine is provided for the purpose of picking up the entry point of the GET/PUT module. The entry point is found by finding the job step Task

Control Block (TCB) via the CVT and IEAFTCBPT. A search is then made of the DEB chain for a DCB which is open to a 2701 with the proper characteristics. On finding a suitable DEB, the address of the DCB is obtained and the entry point for GETPUT2, which is stored just before the DCB in the Line Control Block is loaded into register 15. The user's registers are restored, and GETPUT2 is entered by a 'BR 15'.

LINE PROTOCOL

User messages are sent across the line as part of 'system messages', which are made up of three different types of information: a block header (also termed a reply header), user message headers, and user messages. The reply header consists of a four-byte block ID, which is presently unused, and a variable number of acknowledgement bytes, one corresponding to each user message in the last system message received. If the number of acknowledgement bytes is odd, an extra dummy byte is added to preserve alignment for the 1130 program. (This greatly simplifies the code in the 1130--a non-byte oriented machine.)

Following the reply header on the system message are a variable number of user messages, each preceded by a ten byte header consisting of the sender's ID (four bytes); the ID of the user to which the message is addressed (four bytes); and the length of the following message data (two bytes). (The message as sent is also padded to an even number of bytes.) The only additional things contained in the message are the appropriate BSC control characters; an SOH as the first character; DLE STX after the reply header to initiate transparent text for the user messages; and DLE ETX as the last characters to terminate transparent text and the message itself. The number of messages to be sent in any one system message is determined by the line buffer size.

It can easily be seen that except for system initiation and termination, this structure allows all action on the line to consist of constant interchange of these 'system messages'. However, there are several special cases which must be handled in order to ensure constant maintainence of the action on the line.

With respect to line activity, each processor can be classed into one of three general states: inactive, active with no messages to send, and active with messages to send. The ideal situation described above covers only the case where both processors are in the third state described. In order to cover the other cases, it is necessary to define some other forms of

the general 'system message', and to introduce a very small subset of BSC protocol.

The system message itself takes three forms, depending on whether or not there are messages to send on either or both sides: if both are active with messages to send, it is of the 'full' form described above; if one side has no messages to send but is receiving messages from the other side, a system message consisting of a reply header only (terminated by an EFB) is utilized to acknowledge the messages being received; and if one side is receiving reply headers only but still has messages to transmit, a system message with a 'dummy' reply header (with no acknowledgement bytes) is used, since there are no messages to be acknowledged.

The first case to be considered is the case where both processors are 'inactive' (the initial state of the programs). In this state, both are monitoring the line in a READ state, and before exchange of system messages can begin, program synchronization must be established through 'handshaking' in the standard manner: the processor first desiring to send a message transmits an ENQ control character, which is responded to by a DLE ACK0 by the other processor. At this point, exchange of system messages begins with transmission of a system message consisting of a dummy reply header followed by the message(s) to be sent.

Return to inactive state for both sides is accomplished by transmission of an EOT by the program first discovering that neither side has any messages to send. This occurs when a program receives a reply header only (indicating that the other side has nothing to send) while it is in active state with no messages to send. The normal state of system message exchange may then be re-entered at a later time by again following the initialization procedures described above.

The final special case that must be taken into consideration is the problem of contention: when both programs attempt to exit from active state and transmit an ENQ at the same instant. The standard procedure in this case is to designate one processor the 'master' and the other the 'slave' (in current implementation, the 360 is the 'master' and the 1130 the 'slave'). The 'master' retransmits the ENQ and is allowed to begin the system message exchange.

If, at any point in any of the above proceedings a line error of any kind occurs, the receiving station transmits a NAK control character and expects retransmission of the message in error.

GENERATION OF THE BSCAM MODULES

The BSCAM system is distributed as a pair of unloaded partitioned data sets: a source library and a macro library. The job shown below will unload these data sets and assemble and link-edit the member CLMON2, the line I/O monitor. All parameters which are underlined should be replaced as appropriate for your installation. For example, if the data sets are to reside on 2311's, 'BLKSIZE=7280' should be replaced with 'BLKSIZE=3200'.

```
//BSCAM  JOB   ACCT,INFO,,YOURNAME
//       EXEC  PGM=IEBMOVE
//CREATE DD    UNIT=SYSDA,VOLUME=SER=SCOPER,DISP=(NEW,KEEP),
//            SPACE=(TRK,(50,10,5)),DCB=(RECFM=FB,LRECL=80,
//            BLKSIZE=7280,DSORG=PO),DSNAME=SYS1.YOUR.SOURCLIB
//CREAT2 DD    UNIT=SYSDA,VOLUME=SER=SCOPER2,DISP=(NEW,KEEP),
//            SPACE=(TRK,(20,10,5)),DCB=(RECFM=FB,LRECL=80,
//            BLKSIZE=7280,DSORG=PO),DSNAME=SYS1.YOUR.MACLIB
//SYSPRINT DD  SYSOUT=A
//SYSUT1 DD    UNIT=SYSDA,VOL=SER=SCOPER,DISP=SHR
//DD1    DD    UNIT=SYSDA,VOL=SER=SCOPER,DISP=SHR
//TAPE   DD    UNIT=2400,VOLUME=SER=BSCAM,DISP=(OLD,PASS),
//            LABEL=(1,SL),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN  DD    *,DCB=(BLKSIZE=80)
       COPY  PDS=BSCAM,TO=2314=SCOPER,FROM=2400=BSCAM,
             RENAME=SYS1.YOUR.SOURCLIB,FROMDD=TAPE
       COPY  PDS=MACROS,TO=2314=SCOPER,FROM=2400=(BSCAM,2),
             RENAME=SYS1.YOUR.MACLIB,FROMDD=TAPE
/*
//ASM    EXEC  PGM=IEUASM,PARM='LOAD'
//SYSLIB DD    UNIT=SYSDA,VOLUME=SER=SCOPER,DISP=SHR,
//            DSNAME=SYS1.YOUR.MACLIB
//       DD    DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1 DD    UNIT=SYSDA,SPACE=(TRK,(30,10))
//SYSUT2 DD    UNIT=SYSDA,SPACE=(TRK,(30,10))
//SYSUT3 DD    UNIT=SYSDA,SPACE=(TRK,(30,10))
//SYSPRINT DD  SYSOUT=A
//SYSGO  DD    UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,(10,5)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSIN  DD    UNIT=SYSDA,VOLUME=SER=SCOPER,DISP=OLD,
//            DSNAME=SYS1.YOUR.SOURCLIB(CLMON2)
/*
//LKED   EXEC  PGM=IEWL,PARM='XREF,LET,LIST,NCAL'
//SYSPRINT DD  SYSOUT=A
//SYSLIN DD    DSNAME=*.ASM.SYSGO,DISP=(OLD,DELETE)
```

```
//        DD    DDNAME=SYSIN
//SYSLMOD DD    UNIT=SYSDA,VOLUME=SER=SCOPER,DISP=(NEW,KEEP),
//              SPACE=(TRK,(50,10,5)),DCB=(RECFM=U,BLKSIZE=7294),
//              DSNAME=SYS1.YOUR.LOADLIB
//SYSIN   DD    *
        NAME  CLMON2(R)
/*
```

Following the above job, the source modules GETPUT2, RBCHECK, CLTRACE, TIME, TRACE, IGETPUT, and ICP should be assembled and link-edited. A link-edit can then be run which produces the module LINEMON2 by combining the modules CLMON2, GETPUT2, RBCHECK, CLTRACE (optional), TIME (if CLTRACE is included), and TRACE (either the routine or a dummy 'BR 14' CSECT). The entry point for LINEMON2 is CLMON2.

If it is desired to change the length of the line I/O buffer (and thus the maximum message length), statement LNC00370 in the LNCB macro (an EQU for BUFLEN) should be reset to the new buffer size, and all modules should be re-assembled. (The LNCB macro will be found in SYS1.YOUR.MACLIB.)