CENTUR~ON

CPU-6

APLIB APPLICATION

July 31, 1977

Includes 9/15/83 Revisions

CENTURION COMPUTER CORPORATION
1780 Jay Ell Drive
Richardson, Texas 75081

Printed in the USA

CPU-6 PROGRAMMERS MANUAL - APLIB

TABLE OF CONTENTS

APLIB REFERENCE

INTRODUCTION

The APLIB (APplication LIBrary) is a Type E file containing the executable form of subroutines that are useful in application programming. APLIB subroutines are combined with the executable form of a CPL program to produce the final program. A subroutine must be specified as an external and then called (with a CPL CALL statement) before the Linkage Editor adds it to the program at the end of compilation.

Some subroutines provide entrypoints that may also be used in a CPL program. The entrypoints are only available, however, if the main subroutine has been referenced and called somewhere in the program.

Each subroutine is described independently. These have been grouped into instructional sections by function:

File Access
: These subroutines enable the writing and reading of data to and from indexed files. They differ primarily in the size of the key or argument that can be used to access records. Also included is a subroutine that checks the success of an access. Some of these subroutines, marked by the warning at p the top of the page, are support for existing CPU 4/5 applications and should not be used for the development of new applications unless these are to be compatable with the CPU 4/5.

Data Input/Output (Console)
: These subroutines enable the writing and reading of data to and from console devices assigned to the partition in which the program is running.

Data Input/Output (File)
: These subroutines enable the writing and reading of data to and from indexed files.

Data Manipulation
: These subroutines enable the manipulation of data within the partition in which the program is running.

Interjob Communication
: These subroutines enable the writing and reading of partition parameters so that different jobstreams and programs may communicate.

In these different sections will be found a number of subroutines which are headed with the following warning:

> WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF EXISTING APPLICATIONS. NOT FOR USE IN NEW DEVELOPMENT!

There are also a few subroutines with specialized warnings about their limited use in new CPU 4/5-compatable development.

The following terms are used in this reference:

file-name	defined in a CPL FILE statement
format-name	defined in a CPL FORMAT statement
integer-name	defined in a CPL SET or INTEGER statement
number	numeric value, including an integer-name or numeric constant
record-name	defined in a CPL RECORD statement
string-name	defined in a CPL DEFINE or STRING statement
variable-name	includes string-name or integer-name

```
****************
*              *
*    DOTUSE    *
*              *
****************
```

1. General Form

Call DOTUSE (A, B, C, D)

Where

A = String containing device/file name to be assigned or literal zero if the device/file is to be released.

B = Disk number where the file is located or -1 if a device is to be assigned.

C = File name where the file is to be assigned (used for the sys number only)

D = Assignment option

- Ø = No Shar, No Pass
- 1 = No Shar, Pass
- 2 = Shar, No Pass
- 3 = Shar, Pass

2. Effect

The DOTUSE subroutine will assign a file or device to a specified program logical unit (but not a system logical unit) for use within the CPL program. This subroutine will also release a device or file if parameter "A" is a literal zero.

Upon exit, status will reflect the success of the operation.

Status of Ø indicates successful assignment/release.

Status of 1 indicates file/device assigned elsewhere.

Status of 2 indicates the file is not on the specified disk.

3. Purpose

The DOTUSE subroutine allows the CPL programmer greater flexibility by allowing a dynamic file assignment or release ("on the fly"); this assignment or release can be based upon an operator input or program decision.

4. Parameters

A. Parameter "A" is the name of the file or device to be assigned or released, the file name must be within a CPL string.

B. Parameter "B: is the disk number (-1 for a device) of the file to be assigned. This may be either a four byte integer or a literal.

C. Parameter "C" is defined as the name on a CPL file statement.

D. Parameter "D" is the assignment option and may be either a four byte integer or a literal. The possible values are:

Ø	=	No Shar, No Pass
1	=	No Shar, Pass
2	=	Shar, No Pass
3	=	Shar, Pass

5. Externals and Entrypoints

An external statement in the CPL program must contain the following:

```
EXTERNAL DOTUSE
```

The DOTUSE subroutine has no entrypoints.

6. Caution

   A. Status must be checked after each use of DOTUSE. Failure to do so may cause program aborts.

   B. System Logical Units connot be assigned or released with this subroutine.

7. Example of Use

```
.
.
external dotuse
;
;
file infile: sys0, class=2, sequential
open input file
.
.
call cget ('enter 1 for file 1, 2 for file 2', f001,option)
if (option .eq. 1) 'filename'='fileone'
else 'filename' = 'filetwo'
call dotuse (filename, dsknum, infile,0)
.
.
.
format f001: n01
set    dsknum:0, option:0
string filename (10)
```

```
****************
*              *
*     GETK     *
*              *
****************
```

1. General Form

CALL GETK (file-name, variable-name)

where

file-name	specifies the file that should be searched for a record, and
variable-name	specifies the key, or search argument.

2. Effect

The GETK subroutine attempts to find an index record with the indicated argument in the specified file.

a. STATUS reflects the success of the search:

VALUE OF STATUS	MEANING
0	Record was found
1	Record was not found
2	I/O error occurred during access, or argument was invalid

b. If the record is found (STATUS=0), the relative key for the data record is stored in the key integer for the specified file. (The key integer is associated with the file in a CPL FILE statement). Any data input/output operation using the file will then access that data record.

NOTE: The value of the key integer is only changed by another GETK, NEWK, or DELK call, or by a CPL equate statement.

3. Purpose

The GETK subroutine enables a CPL program to use a four-or six-byte integer or a 7- to 35-character string to access a data record in a Type I file.

The GETK subroutine is designed to replace the GETKEY, ?GETKEY, and AGKEY subroutines.

4. Parameters

a. File-name is defined in a CPL FILE statement as a Class 2, indexed (IND) file.

b. Variable-name specifies an argument for a record that may be in the specified file. The length of the argument depends on the key length specified for the file when it was initialized with the .NEW Job Control Statement, as follows:

KEYLEN	VARIABLE-NAME MAY BE
4	An integer-name defined as a four-byte integer in a CPL SET or INTEGER statement.
6	An integer-name defined as a six-byte integer in a CPL SET or INTEGER statement.
35	A string-name defined as a 7-35 character string in a CPL STRING statement. The string must contain only ASCII characters and spaces. While stored in the index record as specified, each string argument is left-justified and converted to uppercase for processing by the GETK subroutine. Thus, the arguments of 'ABC ' and ' abc' are equivalent, while the strings 'A BC' and 'AB C' are not.

The GETK subroutine rejects as invalid (STATUS=2) any argument that does not fit these specifications.

See the section on NEWK for an example.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

GETK

The GETK subroutine has no entrypoints.

6. Cautions

   a. Data I/O for any file which is accessed with the NEWK/GETK subroutines may be performed by the PUTR/GETR/HLDR/FRER subroutines. These subroutines also allow CPU 5 compatability.

      The CPL I/O statements READB, WRITEB, HOLD, FREE may be used with an indexed file.

   b. If the GETK subroutine does not find a record with the key specified, one may be created by the NEWK subroutine.

7. See also:

   In this manual:

NEWK	NEWKEY	GETKEY	?NKEY	?GKEY
PUTR	GETR	HLDR	FRER	

```
***************
*             *
*    NEWK     *
*             *
***************
```

1. General Form

CALL NEWK (file-name, variable-name)

where

file-name	specifies the file to which a data record will be added, and
variable-name	specifies the key, or argument

2. Effect

The NEWK subroutine writes a record in the index area and reserves a record in the data area of the specified file. The index record contains the argument and the relative key (pointer) of the associated data record.

STATUS reflects the success of the creation attempt:

VALUE OF STATUS	MEANING
0	Index record written; data record reserved
1	No room in file for either record
2	I/O error occurred during access, or argument was invalid
4	Index chain pointer is invalid or damaged

NOTE: The common routines HASH and IOC are linked only once even though more than one subroutine using them is called by the program.

3. Purpose

The NEWK subroutine enables a CPL program to use a four- or six-byte integer or a 7-35 character string as an argument and add a data record to an indexed Type I file.

The NEWK subroutine is designed to replace the NEWKEY,?NKEY, and ANKEY subroutines.

4. Parameters

a. File name is defined in CPL FILE statement as a Class 2, indexed (IND) file.

b. Variable-name specifies the argument for the record that is being added to the specified file. The length of the argument depends on the key length specified for the file when it was initialized with the .NEW Job Control Statement, as follows:

KEYLEN	VARIABLE-NAME MAY BE
4	An interger-name defined as a four-byte integer in a CPL SET or INTEGER statement.
6	An integer-name defined as a six-byte integer in a CPL SET or INTEGER statement.
7-35	A string-name defined as a 7-35 character string in a CPL STRING statement. The string must contain only ASCII characters and spaces. While each string argument is right-justified and converted to uppercase for processing by the NEWK subroutine. Thus, the arguments 'ABC ' and ' abc' are equivalent, while the strings 'A BC' and 'AB C' are not.

The NEW subroutine rejects as invalid (STATUS=2) any argument that does not fit these specifications.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

NEWK

The NEWK subroutine has no entrypoints.

6. Cautions

a. For CPU-5 compatibility any file which is accessed with the NEWK/GETK subroutines should also use PUTR/GETR/HLDR/FRER subroutines instead of READB/WRITEB/HOLD/FREE.

The CPL I/O statements READB, WRITEB, HOLD, and FREE may be used with an indexed file.

b. If STATUS=4 at the end of the NEWK subroutine, the data file is damaged. It should be rebuilt immediately.

c. The NEWK subroutine does not check for the indicated argument in the specified file before processing the key. Therefore, the GETK subroutine should be called with the same parameters before NEWK is used.

For example:

```
FILE MASTER: SYS0,CLASS=2,IND,KEY=MSTKEY
SET MSTKEY:0
;
RECORD MSTREC(395)
.
.
.
A10:
CALL CGET (M05,F01,CUST) ; ENTER CUSTOMER NUMBER
CALL GETK (MASTER,CUST) ; ON FILE?
GO TO (A20,IOERR) ON STATUS
CALL GETR (MASTER,MSTREC) ; READ RECORD
CALL STAT(1) ; CHECK ON ACCESS
GO TO A50
;
A20:
CALL NEWK (MASTER,CUST) ; CREATE NEW INDEX
GO TO (NOROOM,IOERR) ON STATUS
CALL CLREC (MSTREC) ; PREPARE FOR NEW DATA
;
A50:
.
.
.
CALL PUTR (MASTER,MSTREC) ; PUT DATA INTO FILE
CALL STAT(1) ; CHECK ON ACCESS
GO TO A10 ; GO GET ANOTHER
.
.
.
SET CUST:0
DEFINE M05:'ENTER CUSTOMER NUMBER'
FORMAT F01:N6
```

where

MASTER is the file-name,

CUST is the argument, i.e., the name of a four-byte integer and,

MSTKEY is the key integer for the file MASTER.

7. See also:

In this manual:

GETK	NEWKEY	GETKEY	?GKEY	?NKEY
PUTR	GETR	HLDR	FRER	

```
***************
*             *
*    NEXK     *
*             *
***************
```

1. General Form

   CALL NEXK (file-name, variable-name)

   where

   file-name - specifies the file that should be searched for a record;

   variable-name - specifies the argument.

2. Effect

   The NEXK subroutine attempts to retrieve the next index record.

   A. STATUS reflects the success of the search:

VALUE OF STATUS	MEANING
0	Next record was found.
1	End of index (nothing changed).
2	I/O error occurred during access.

   B. If the next record is found (STATUS=0), the relative key for the data record is stored in the key integer for the specified file (i.e. the key integer is associated with the file in a CPL FILE statement). Any data input/output operation using the file will then access that data record. Also, the variable-name will be set to the argument.

   NOTE: The value of the key integer is only changed by another GETK, NEWK or DELK call; it is also changed by a CPL "equate" statement.

3. Purpose

   The NEXK subroutine enables a CPL program to access the next key in the index and set the variable-name to the argument value.

   NOTE: The keys within te index are not in sorted sequence; neither are then in sequence of original entry.

4. Parameters

A. File-name is defined in a CPL FILE statement as a Class 2, indexed (IND) file.

B. Variable-name specifies an argument for a record that may be in the specified file. The length of the argument depends on the key length specified for the file when it was initialized with the .NEW Job Control Statement as follows:

KEYLEN	VARIABLE-NAME MAY BE
4	An integer-name defined as a four-byte integer in a CPL SET or INTEGER statement.
6	An integer-name defined as a six-byte integer in a CPL SET or INTEGER statement.
7 - 35	A string-name defined as a 7-35 character string in a CPL STRING statement. The string must contain only ASCII characters and spaces. While stored in the index record as specified, each string argument is left-justified and converted uppercase for processing by the GETK subroutine. Thus, the arguments of 'ABC' and 'abc' are equivalent, while the strings 'A BC' and 'AB C' are not.

5. Externals and Entrypoints

An external statement in CPL program must contain the following entry:

NEXK

The NEXK subroutine has no entrypoints

6. Cautions

A. Data I/O for any file which is accessed with the NEXK/NEWK/GETK subroutines may be performed by the PUTR/GETR/HLDR/FRER subroutines. These subroutines also allow CPU-5 compatability.

The CPL I/O statemets READB, WRITEB, HOLD, FREE may be used with an indexed file.

B. When the NEXK subroutine runs out of records, status will be set to value of 1; neither the key or the argument will be changed.

C. To sequentially process an entire file, pre-set the key to a value of 1. The first NEXK will return the first key and argument.

D. If the key is set to 0, the values returned by NEXK are unpredictable.

7. See also in this manual:

NEWK	NEWKEY	GETKEY	?NKEY	?GKEY
PUTR	GETR	HLDR	FRER	GETK

```
****************
*              *
*    DELK      *
*              *
****************
```

1. General Form

CALL DELK (file-name, variable-name)

where

file-name	specifies the file that should be searched for a record, and
variable-name	specifies the key, or search argument.

2. Effect

The DELK subroutine attempts to find an index record with the indicated argument in the specified file.

a. STATUS reflects the success of the search:

VALUE OF STATUS	MEANING
0	Record was found
1	Record was not found
2	I/O error occurred during access, or argument was invalid
4	Index chain pointer is invalid or damaged

b. If the index record is found (STATUS=0), the relative key (pointer) in it is set to zero, releasing the referenced data record for use with the NEWK subroutine; the index record is also removed from the index chain.

NOTE: The common routines HASH and IOC are linked only once even though more than one subroutine using them is called by the program.

3. Purpose

The DELK subroutine enables a CPL program to free a data

record for use with another argument.

4. Parameters

a. File-name is defined in a CPL FILE statement as a Class 2, indexed (IND) file.

b. Variable-name may be an integer-name defined as a four- or six-byte integer in a CPL SET or INTEGER statement, or a string-name defined as a 7-35 character string in a CPL STRING statement. See sections on NEWK and GETK for detailed requirements for this parameter.

For example:

```
FILE MASTER: SYS0,CLASS=2,IND,KEY=MSTKEY
SET MSTKEY:0
;
RECORD MSTREC(135)
.
.
.
CALL CGET (M05,F01,CUST) ; ENTER CUSTOMER NUMBER
CALL GETK (MASTER,CUST) ; ON FILE?
GO TO (A20,IOERR) ON STATUS
CALL GETR (MASTER,MSTREC) ; READ RECORD
CALL STAT(1) ; CHECK ON ACCESS
GO TO A50
;
A20:
.
.
.
A50:
;
WRITE (CRT,F02) CUSNAM
.
.
.
WRITE (CRT,F02) M12
WRITE (CRT,F03) M13
WRITEN (CRT,F03) M14
CALL CGET (NULL,F01,OPTION) ; WHAT TO DO?
GOTO (B10,A60,A10) ON OPTION
;
A60:
;
CALL DELK (MASTER,CUST) ; GET RID OF HIM AND FREE RECORD
GOTO A10
.
```

```
.
.
SET CUST:0
DEFINE M05:'ENTER CUSTOMER NUMBER '
.
.
DEFINE M12:'ENTER OPTION: 1 CHANGE FIELD ON THIS DISPLAY'
DEFINE M13:'2 DELETE THIS CUSTOMER'
DEFINE M14:'9 FINISHED WITH THIS CUSTOMER'
.
.
FORMAT F01:N6
FORMAT F02:C79
FORMAT F03:X16C50
```

where

MASTER is the file-name, and

CUST is the argument, i.e., the name of a four-byte integer.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

DELK

The DELK subroutine has no entrypoints.

6. Cautions

The DELK subroutine is designed to operate only on indexed files (NEWK and GETK are used for file access).

7. See also:

NEWK GETK PUTR GETR HLDR
FRER

```
***************
*             *
*   GETKEY    *
*             *
***************
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF EXISTING APPLICATIONS. NOT FOR USE IN NEW DEVELOPMENT!!

1. General Form

CALL GETKEY (file-name, integer-name)

where

file-name	specifies the file that should be searched for a record, and
integer-name	specifies the four-byte key, or search argument

2. Effect

The GETKEY subroutine attempts to find an index record with the indicated argument in the specified file.

a. STATUS reflects the success of the search:

VALUE OF STATUS	MEANING
0	Record was found
1	Record was not found
2	I/O error occurred during access

b. If the record is found (STATUS=0), the relative key for the data record is stored in the key integer for the specified file. (The key integer is associated with the file in a CPL FILE statement). Any read or write statement using the file will then access that data record.

If the record was not found, the key integer contains a 0 or -1.

NOTE: The value of the key integer is only changed by another GETKEY or NEWKEY call, or by a CPL equate statement.

3. Purpose

The GETKEY subroutine enables a CPL program to use a four-byte integer to access a data record in a Type C file.

4. Parameters

a. File-name is defined in a CPL FILE statement as a Class 2, indexed (IND) file.

b. Integer-name is defined as a four-byte integer in a CPL SET or INTEGER statement.

See the section on NEWKEY for an example.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

GETKEY

The GETKEY subroutine provides the following entrypoint:

NEWKEY

6. Cautions

a. The GETKEY subroutine is designed to operate only on Type C files that have been initialized with the system utility XRINT.

b. If the GETKEY subroutine does not find a record with the key specified, one may be created by the NEWKEY subroutine.

7. See also:

In this manual:

NEWK GETK ?NKEY ?GKEY NEWKEY

In the System Utilities Reference:

XRINT XWTAG

```
***************
*             *
*    NEWKEY   *
*             *
***************
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF EXISTING APPLICATIONS. NOT FOR USE IN NEW DEVELOPMENT!

1. General Form

CALL NEWKEY (file-name, integer-name)

where

file-name	specifies the file to which a data record will be added, and
integer-name	specifies the four-byte key, or argument.

2. Effect

The NEWKEY subroutine writes a record in the index area and reserves a record in the data area of the specified file. The index record contains the argument and the relative key (pointer) of the associated data record.

STATUS reflects the success of the creation attempt:

VALUE OF STATUS	MEANING
0	Index record written; data record reserved
1	No room in file for either record
2	I/O error occurred during access

3. Purpose

The NEWKEY subroutine enables a CPL program to use a four-byte integer as a key and add a data record to a Type C file.

4. Parameters

   a. File-name is defined in a CPL FILE statement as a Class 2, random (RND) file.

   b. Integer-name is defined as a four-byte integer in a CPL SET or INTEGER statement.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entries:

```
GETKEY
NEWKEY
```

   The NEWKEY subroutine has no entrypo nts.

6. Cautions

   a. The NEWKEY subroutine is designed to operate only on Type C files that have been initialized with the system utility XRINT.

   b. The NEWKEY subroutine does not check for the indicated key in the specified file before processing the key. Therefore, the GETKEY subroutine should be called with the same parameters before NEWKEY is used.

NOTE: The GETKEY subroutine must be called somewhere in the CPL program so that the Linker Utility will add it (and the entrypoint NEWKEY) to the program.

For example:

```
FILE MASTER: SYS0,CLASS=2,RND,RECSIZ=395,KEY=MSTKEY
SET MSTKEY:0
;
RECORD MSTREC(395)
.
.
.
A10:
CALL CGET (M05,F01,CUST) ; ENTER CUSTOMER NUMBER
CALL GETKEY (MASTER,CUST) ; ON FILE?
GO TO (A20,IOERR) ON STATUS
READB (MASTER,MSTREC) ; READ RECORD
CALL STAT(1) ; CHECK ON ACCESS
GO TO A50
```

```
;
A20:
CALL NEWKEY (MASTER,CUST) ; CREATE NEW INDEX
GO TO (NOROOM,IOERR) ON STATUS
CALL CLREC (MSTREC) ; PREPARE FOR NEW DATA
;
A50:
.
.
.
WRITEB (MASTER,MSTREC) ; PUT DATA INTO FILE
CALL STAT(1) ; CHECK ON ACCESS
GO TO A10 ; GO GET ANOTHER
.
.
.
SET CUST:0
DEFINE M05:'ENTER CUSTOMER NUMBER'
FORMAT F01:N6
```

where

MASTER is the file-name,

CUST is the four-byte integer-name, and

MSTKEY is the key integer for the file MASTER.

7. See also:

In this manual:

GETK NEWK GETKEY ?GKEY ?NKEY

In the Systems Utilities Reference:

XRINT XWTAG

```
****************
*              *
*    ?GKEY     *
*              *
****************
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF EXISTING APPLICATIONS. NOT FOR USE IN NEW DEVELOPMENT!

1. General Form

CALL ?GKEY (file-name, integer-name)

where file-name specifies the file that should be searched for a record, and

integer-name specifies the six-byte key, or search argument.

2. Effect

The ?GKEY subroutine attempts to find an index record with the indicated argument in the specified file.

a. STATUS reflects the success of the search:

VALUE OF STATUS	MEANING
0	Record was found
1	Record was not found
2	I/O error occurred during access

b. If the record is found (STATUS=0), the relative key (pointer) for the data record is stored in the key integer for the specified file. (The key integer is associated with the file in a CPL FILE statement.) Any read or write statement using the file will then access that data record. NOTE: The value of the key integer is only changed by another ?GKEY call or by a CPL equate statement.

If the record as not found, the key integer contains a 0 or -1.

3. Purpose

The ?GKEY subroutine enables a CPL program to use a six-byte integer to access a data record in a Type C file.

4. Parameters

a. File-name is defined in a CPL FILE statement as a Class 2, random (RND) file.

b. Integer-name is defined as a six-byte integer in a CPL SET or INTEGER statement, i.e., the name begins with a question mark (?).

See the section on ?NKEY for an example.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

?GKEY

The ?GKEY subroutine provides the following entrypoint:

?NKEY

6. Cautions

a. The ?GKEY subroutine is designed to operated only on Type C files that have been initialized with the system utility X?RINT.

b. If the ?GKEY subroutine does not find a record with the key specified, one may be created by the ?NKEY subroutine.

7. See also:

In this manual:

NEWK GETK ?NKEY GETKEY NEWKEY

In the System Utilities Reference:

X?RINT X?WTAG

```
**********
*        *
*  ?NKEY *
*        *
**********
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF EXISTING APPLICATIONS. NOT FOR USE IN NEW DEVELOPMENT!

1. General Form

```
CALL ?NKEY (file-name, integer-name)
```

where

file-name	specifies the file that should be searched for a record, and
integer-name	specifies the six-byte key, or search argument.

2. Effect

The ?NKEY subroutine writes a record in the index area and reserves a record in the data area of the specified file. The index record contains the argument and the relative key (pointer) of the associated data record.

STATUS reflects the success of the creation attempt:

VALUE OF STATUS	MEANING
0	Index record written; data record reserved
1	No room in file for the index record
2	I/O error occurred during access

3. Purpose

The ?NKEY subroutine enables a CPL program to use a six-byte integer as a key and add a data record to a Type C file.

4. Parameters

   a. File-name is defined in a CPL FILE statement as a Class 2, random (RND) file.

   b. Integer-name is defined as a six-byte integer in a CPL SET or INTEGER statement, i.e., the name begins with a question mark (?).

5. Externals and Entrypoints

   An External statement in the CPL program must contain the following entries:

   ```
   ?GKEY
   ?NKEY
   ```

   The ?NKEY subroutine has no entrypoints.

6. Cautions

   a. The ?NKEY is designed to operate only on Type C files that have been initialized with the system utility X?RINT.

   b. The ?NKEY subroutine does not check for the indicated key in the specified file before processing the key. Therefore, the ?GKEY subroutine should be called with the same parameters before ?NKEY is used.

   NOTE: The ?GKEY subroutine must be called somewhere in the CPL program so that the Linker Utility will add it to the program.

   For example:

```
FILE MASTER: SYS0,CLASS=2,RND,RECSIZ=395,KEY=MSTKEY
SET MSTKEY: 0
;
RECORD MSTREC(395)
.
.
.
A10
CALL CGET (M05,F01,?CUST) ; ENTER CUSTOMER NUMBER
CALL ?GKEY (MASTER,?CUST) ; ON FILE?
GO TO (A20,IOERR) ON STATUS
READB (MASTER,MSTREC) ; READ RECORD
CALL STAT(1) ; CHECK ON ACCESS
GO TO A50
```

```
;
A20
CALL ?NKEY (MASTER,?CUST) ; CREATE NEW INDEX
GO TO (NOROOM,IOERR) ON STATUS
CALL CLREC (MSTREC) ; PREPARE FOR NEW DATA
;
A50
.
.
.
WRITEB (MASTER,MSTREC) ; PUT DATA INTO FILE
CALL STAT(1) ; CHECK ON ACCESS
GO TO A10 ; GO GET ANOTHER
.
.
.
SET ?CUST:0
DEFINE M05:'ENTER CUSTOMER NUMBER'
FORMAT F01:D6
```

where

MASTER is the file-name,

?CUST is the six-byte integer-name, and

MSTKEY is the key integer for the file MASTER.

7. See also:

In this reference:

NEWK GETK ?GKEY GETKEY NEWKEY

In the System Utilities Reference:

X?RINT X?WTAG

```
***************
*             *
*    IOERR    *
*             *
***************
```

1. General Form

GOTO (label, IOERR) ON STATUS

IF(STATUS.EQ.2)CALL IOERR

where

label is the beginning of a routine in the CPL program that should be used when STATUS = 1.

2. Effect

The IOERR entrypoint of the STAT routine terminates a CPL program with the following message if STATUS is not equal to zero when it is referenced:

***** I/O ERROR SYS99 ADDRESS=aaaa STATUS=s *****

where aaaa is one of the following:

- an unrelated address within the DOS Supervisor if IOERR is referenced in a CPL GOTO statement
- the approximate address (within 10 bytes on an assembled listing of the program) of the CPL statement that called the IOERR entrypoint

and

s is the value of STATUS that caused the abort.

Completion Code is set to 100 before the program is stopped.

3. Purpose

The IOERR entrypoint provides a simple method of testing STATUS and terminating a program when further processing might damage data.

4. Parameters

IOERR is used as a label in a CPL CALL, IF, IFSTRING, or GOTO statement.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entries:

```
STAT
IOERR
```

The CPL program must contain the following statement:

```
ENTRYPOINT CRT
```

6. Cautions

a. The jobstream that executes the CPL program should include a check on the Completion Code (#C) immediately following the .RUN JCL statement. If #C = 100, the program was aborted by means of IOERR and the jobstream should also terminate immediately.

b. The STAT subroutine must be called at least once by the CPL program before the Linker Utility will add it and the IOERR entrypoint to the program.

7. See also:

STAT

```
***************
*             *
*    STAT     *
*             *
***************
```

1. General Form

CALL STAT(n)

where

n specifies the type of status check the subroutine should perform.

2. Effect

The STAT subroutine checks the value of STATUS. If STATUS=0, control is returned to the calling CPL program. If STATUS is not equal to zero, the STAT subroutine:

a. Terminates the CPL program

b. Sets the Completion Code as follows:

	STATUS	CC
n=1	1	100
	2	100
	3	100
n=2	1	100
	2	100
	3	PLU+1
n=file-name	1	100
	2	100
	3	PLU+1

where PLU is the number of the Programmer Logical Unit (SYS number) to which the file was assigned, and

file-name specifies a disk file.

c. Displays one of the following messages on the console device assigned to the partition in which the CPL program is running:

1. STATUS=3

***** EOM ON OUTPUT SYSnn ADDRESS=aaaa STATUS=s *****

2. STATUS=1 or 2

***** I/O ERROR SYSnn ADDRESS=aaaa STATUS=s *****

where nn is the number of the Programmer Logical Unit to which the file was assigned,

aaaa is the address (on a source listing produced by the assembler) of the statement following the CALL STAT statement that caused the abort, and

s is the value of STATUS that caused the abort.

3. Purpose

The STAT subroutine provides a simple method of analyzing STATUS after an I/O routine. If an error is discovered, the subroutine displays information about that error on the console device.

4. Parameters

a. The values of 1 and 2 should be used for n following CPL READB and WRITEB statements for Type B and C files, as follows:

1. n should equal 1 if a READB or if a WRITEB that should not encounter end-of-medium (EOM) has been executed, i.e., any non-zero STATUS should cause a complete abort of the program.

2. n should equal 2 if a WRITEB that might encounter EOM has been executed, i.e., if STATUS equals 3, the file needs to expanded, but the disk is full.

b. A file-name should be specified as the parameter (n=file) if:

1. The file is a Type A file

2. The subroutine call cannot immediately follow a CPL WRITEB statement and EOM may have been encountered.

See sections on ?NKEY, NEWK, and NEWKEY for examples.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

STAT

The CPL program must contain the following statement:

ENTRYPOINT CRT

The STAT subroutine provides the following entrypoint:

IOERR

6. Cautions

The jobstream which executes the CPL program should include a check on the Completion Code (#C) immediately following the .RUN JCL statement.

a. If #C=1-99, one of the files used by the program needs to be expanded (the value of #C indicates which file): the disk needs to be .REORG'd or the file's FSI needs to be made smaller.

b. If #C=100, an I/O or other error has occurred during an I/O operation: the jobstream should be terminated immediately.

7. See also:

IOERR

```
***************
*             *
*    CGET     *
*             *
***************
```

1. General Form

   CALL CGET (string-name, format-name, variable-name)

   where

string-name	specifies a message,
format-name	specifies the type of data to be entered, and
variable-name	specifies the string or integer that should receive the data.

2. Effect

   a. The message is displayed on the console device assigned to the partition in which the CPL program is running.

   b. The subroutine adds to the message a space, a slash (/), and:

      1. At-signs (@) if a C-type format was specified

      2. Number-signs (#) if a N- or D-type format was specified

      The number of signs displayed after the slash depends on the number of characters specified by the format, e.g., an N6 format causes this display:

      /######

   c. The subroutine accepts any data entered from the console device and sets STATUS to indicate if the data fit the format specified, as follows:

      1. STATUS=0: data conformed to format specified and was moved to specified variable

2. STATUS=2: data was either the wrong type, e.g. letters entered for numeric format, or longer than the number of characters specified; value of the variable is unchanged

3. Purpose

The CGET subroutine displays a prompting message to the operator and returns properly-formatted data to the CPL program.

4. Parameters

a. String-name may be either a string of characters defined in a CPL DEFINE statement or a null string defined in a CPL STRING statement.

b. Format-name is defined in a CPL FORMAT statement. If the format includes multiple fields, the CGET subroutine uses only the first field specification.

c. Variable-name may be the name of a string defined in a CPL STRING statement or the name of an integer defined in a CPL SET or INTEGER statement.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

```
CGET
```

The following statement must be in the CPL program:

```
ENTRYPOINT CRT
```

The CGET subroutine has no entrypoints.

6. Cautions

a. STATUS should be checked immediately after the subroutine call to insure that the proper data was entered and moved to the specified variable.

b. The format field specification must agree with the receiving variable, i.e., an integer variable requires an N- or D-type format specification.

c. Even though a numeric format and an integer variable are specified, the subroutine will not reject alphabetic data. However, since the subroutine cannot write alphabetic data into an integer, it will not attempt to do so, leaving the value of the integer unchanged, i.e., the integer contains the value it had before the subroutine call. Of course, STATUS is set to 2.

7. See also:

YNGET

```
***************
*             *
*    ?NGET    *
*             *
***************
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF EXISTING APPLICATIONS. NOT FOR USE IN NEW DEVELOPMENT!

1. General Form

CALL ?NGET (string-name, integer-name)

where

string-name	specifies a message, and
integer-name	specifies the six-byte integer that should receive the input data.

2. Effect

a. The message is displayed on the console device assigned to the partition in which the CPL program is running.

b. The subroutine accepts a string of numeric characters from the console device and places their value in the specified six-byte integer. Only the ASCII digits 0-9 are accepted; decimal points are ignored; all other characters are rejected (the message is redisplayed).

3. Purpose

The ?NGET subroutine displays a prompting message to the operator and returns properly-formatted data to the CPL program.

4. Parameters

a. String-name identifies a string of characters defined in a CPL DEFINE statement or a null string defined in a CPL STRING statement.

b. Integer-name is defined as a six-byte integer in a CPL SET or INTEGER statement, i.e., the name begins with a question mark (?).

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

?NGET

The following statement must be in the CPL program:

ENTRYPOINT CRT

The ?NGET subroutine has no entrypoints.

6. Cautions

The ?NGET subroutine is no longer considered standard. The CGET subroutine is now used to gather numeric data.

7. See also:

CGET

```
***************
*             *
*    NGET     *
*             *
***************
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF EXISTING APPLICATIONS. NOT FOR USE IN NEW DEVELOPMENT!

1. General Form

CALL NGET (string-name, integer-name)

where

string-name	specifies a message, and
integer-name	specifies the four-byte integer that should receive the input data.

2. Effect

a. The message is displayed on the console device assigned to the partition in which the CPL program is running.

b. The subroutine accepts a string of numeric characters from the console device and places their value in the specified four-byte integer. Only ASCII digits 0-9 are accepted; decimal points are ignored; all other characters are rejected (the message is redisplayed).

3. Purpose

The NGET subroutine displays a prompting message to the operator and returns properly-formatted data to the CPL program.

4. Parameters

a. String-name identifies a string of characters defined in a CPL DEFINE statement or a null string defined in a CPL STRING statement.

b. Integer-name is defined as a four-byte integer in a CPL SET or INTEGER statement.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

NGET

The following statement must be in the CPL program:

ENTRYPOINT CRT

The NGET subroutine has no entrypoints.

6. Cautions

The NGET subroutine is no longer considered standard. The CGET subroutine is now used to gather numeric data.

7. See also:

CGET

```
***************
*             *
*    YNGET    *
*             *
***************
```

1. General Form

   CALL YNGET (string-name)

   where

   string-name specifies a message.

2. Effect

   a. The message is displayed on the console device assigned to the partition in which the CPL program is running.

   b. The subroutine adds a space, a slash (/), and an asterisk (*) at the end of the message.

   c. The subroutine accepts either an affirmative or a negative answer and sets STATUS as follows:

ANSWER TYPE	CHARACTER ENTERED	STATUS
Affirmative:	character Y (followed by NEWLINE)	1
	character plus (+) (followed by NEWLINE)	1
	plus bar (+)	1
Negative:	character N (followed by NEWLINE)	0
	character minus (-) (followed by NEWLINE)	0
	minus bar (-)	0

   Characters other than those specified are rejected: the device beeps, the entered character(s) are erased, and the asterisk is redisplayed with the cursor underneath it.

3. Purpose

   The YNGET subroutine displays a prompting message to the operator and returns a yes (Y) or no (N) indicator in STATUS to the CPL program.

4. Parameters

String-name is defined in a CPL DEFINE statement.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

YNGET

The following statement must be in the CPL program:

ENTRYPOINT CRT

The YNGET subroutine has no entrypoints.

6. See also:

CGET

```
**********
*        *
*  MSG   *
*        *
**********
```

1. General Form

   CALL MSG (string-name)

   where

   string-name - specifies a message

2. Effect

   The MSG subroutine causes the specified message to be displayed on the console device assigned to the partition in which the CPL program is running. The cursor is left at the first position of the line below the message.

   If a null string is specified as a parameter, the subroutine will output a blank. The cursor is left at the first position of the line below the message.

3. Purpose

   The MSG subroutine displays messages to guide and inform operators without requiring input from them.

4. Parameters

   String-name is defined as:

   a. A string of characters (a message) in a CPL DEFINE statement

   b. A data variable in a CPL STRING statement

   c. A null string in a CPL STRING statement

5. Externals and Entry points

   An EXTERNAL statement in the CPL program must contain the following entry:

   MSG

The following statement must be in the CPL program:

ENTRYPOINT CRT

The MSG subroutine provides the following entrypoints:

F90

NOTE: F90 is defined in a FORMAT statement as C79.

6. Cautions

All messages displayed via the MSG subroutine are limited to 79 characters in length by the format F90.

7. See also:

CGET YNGET

```
****************
*              *
*     MSGN     *
*              *
****************
```

1. General Form

   CALL MSGN (string-name)

where

   string-name - specifies a message

2. Effect

   The MSGN subroutine causes the specified message tc be displayed on the console device assigned to the partition in which the CPL program is running. However, the subroutine does not issue a cursor return/line feed command sequence after the message is displayed; the cursor remains to the right of the last character displayed.

   If a null string is specified as a parameter, the subroutine will output a blank. The cursor will remain to the right of the blank that was just displayed.

3. Purpose

   The MSGN subroutine displays messages to guide and inform operators without requiring input from them.

4. Parameters

   a. A string of characters (a message) in a CPL DEFINE statement

   b. A data variable in a CPL STRING statement

   c. A null string in a CPL STRING statement

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

```
MSGN
```

The following statement must be in the CPL program:

```
ENTRYPOINT CRT
```

The MSGN subroutine has no entrypoints.

6. Cautions

All messages displayed via the MSGN subroutine are limited to 79 characters in length.

7. See also:

CGET YNGET

```
***************
*             *
*    LFEED    *
*             *
***************
```

1. General Form

   CALL LFEED (file-name, number)

   where

file-name	specifies an output device such as a printer or CRT, and
number	specifies the number of null lines to be output.

2. Effect

   The LFEED subroutine causes the specified device to output null lines, thereby advancing its output medium the number of lines indicated. For instance, a printer will advance the paper the number of lines, while a CRT will move the cursor down the screen the number of lines.

3. Purpose

   The LFEED subroutine allows a CPL program to effect spacing in its output with economical use of partition memory.

4. Parameters

   a. File-name is the name of an output device defined in a CPL FILE statement.

   b. Number may be either a numeric constant or an integer-name defined in a CPL SET or INTEGER statement.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   LFEED

The LFEED subroutine has no entrypoints.

6. Cautions

The LFEED subroutine uses a format named F90. This format may be provided:

a. By the program in a CPL DEFINE statement with a C-type field specification at least one character long.

b. As an entrypoint in the MSG subroutine. An EXTERNAL statement in the CPL program must then contain the following entries:

MSG

NOTE: The MSG subroutine must be called at least once by the CPL program before it and the F90 entrypoint are incorporated into the program by the Linker Utility.

7. See also:

MSG

****
*   *
* CLREOL *
*   *
****

1. General Form

   Call **CLREOL (H,V)**

   where

   H is a horizontal screen position
   V is a vertical screen position

2. Effect

   The **CLREOL** subroutine causes the cursor to be positioned at the specified coordinates and clear from that position to the end of the line. When **CLREOL** finishes the cursor is returned to the specified position.

3. Purpose

   To clear variable data from CRT screen without disturbing fixed data.

4. Parameters

   "H" and "V" may be either a positive literal value or the name of a four byte integer containing the value.

5. Externals and Entry Points.

   An external statement in the CPL program must contain the following entries:

   External **CLREOL**
   Entry point CRT

6. Cautions

   This routine may not function on some older model CRT's (prior to Adds Regent 40)/

7. See also:

   CLREOP

```
****************
*              *
*    CLREOP    *
*              *
****************
```

1. General Form

   Call **CLREOP (H,V)**

   Where

   H is a horizontal screen position
   V is a vertical screen position

2. Effect

   The **CLREOP** subroutine causes the cursor to be positioned at the specified coordinates and clears from that position to the end of the screen. When **CLREOP** finishes the cursor is returned to the specified position.

3. Purpose

   To clear variable data from CRT screen without disturbing fixed data, such as column headers or program name.

4. Parameters

   "H" and "V" may be either a positive literal value or the name of a four byte integer containing the value.

5. Externals and Entry Points

   An external statement in the CPL program must contain the following entries:

   External **CLREOP**
   Entry point CRT

6. Cautions

   This routine may not function on some older model CRT's (prior to Adds Regent 40)

7. See also:

   CLREOL

```
**********
*        *
*  GETR  *
*        *
**********
```

1. General Form

CALL GETR (file-name, record-name)

where

file-name	specifies the file containing the data that should be read, and
record-name	specifies the area within the partition where the data should be stored.

2. Effect

The GETR subroutine provides compatability with CPU 5 programs.

STATUS reflects the success of the operation:

VALUE OF STATUS	MEANING
0	Record read and data moved successfully
1	Record indicated by key integer was outside file boundaries. No data transfer occurred.
2	I/O error occurred during access; data in record area is unspecified.

3. Purpose

The GETR subroutine provides linkage to CPL READB statements.

4. Parameters

a. File-name is defined in a CPL FILE statement as a Class 2, indexed (IND) or spanned (SPN) file. A record size

must also be specified in the statement.

b. Record-name is defined in a CPL RECORD statement. The record may contain up to 2047 bytes.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

GETR

The GETR subroutine provides the following entrypoint:

PUTR

6. Cautions

a. The number of bytes transferred by the GETR subroutine is determined by the smaller of the record sizes defined in the:

1. The CPL RECORD statement

2. The CPL FILE statement

3. The RECSIZ specified in the .NEW JCL statement.

b. The CPL HOLD and FREE statements may be used with a file that is read by the GETR subroutine.

7. See also:

In this manual:

NEWK GETK DELK PUTR HLDR
FRER

```
**********
*        *
*  PUTR  *
*        *
**********
```

1. General Form

CALL PUTR (file-name, record-name)

where

file-name	specifies the file into which data should be written, and
record-name	specifies the area within the partition where the data is stored.

2. Effect

The PUTR subroutine provides compatability with CPU 5 CPL programs. STATUS reflects the success of the operation:

VALUE OF STATUS	MEANING
0	Record written and data moved successfully
2	I/O error occurred during access; data in file record area is unspecified.
3	Record indicated by key integer was outside file boundaries and file was unable to expand. No data transfer occurred.

3. Purpose

The PUTR subroutine provides linkage to CPL WRITEB statements.

NOTE: The PUTR subroutine may be used to write data into a spanned Type C file.

4. Parameters

a. File-name is defined in a CPL FILE statement as a Class 2, indexed or spanned file. A record size must also be specified in the statement.

b. Record-name is defined in a CPL RECORD statement. The record may contain up to 2047 bytes.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entries:

GETR
PUTR

The PUTR subroutine provides the following entrypoint:

PUTR

6. Cautions

a. The number of bytes transferred by the PUTR subroutine is determined by the smaller of the record sizes defined in the:

1. The CPL RECORD statement

2. The CPL FILE statement

3. The RECSIZ specified in the .NEW JCL statement.

b. The CPL HOLD and FREE statements may be used with a file that is read by the PUTR subroutine.

c. PUTR will force an automatic expansion of the file if the key indicated a record beyond file boundaries.

7. See also:

In this manual:

NEWK GETK DELK GETR HLDR
FRER

In the CENTURION PROGRAMMING LANGUAGE manual:

WRITEB

```
***************
*             *
*    HLDR     *
*             *
***************
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF CPU 5-COMPATABLE APPLICATIONS. NOT FOR USE IN DEVELOPMENT OF NEW NON-CPU 5 COMPATABLE PROGRAMS!

1. General Form

CALL HLDR (file-name)

where

file-name	specifies the file containing the record that must be held for exclusive access.

2. Effect

The HLDR subroutine provides compatablility with CPU 5 CPL programs. STATUS reflects the success of the operation:

VALUE OF STATUS	MEANING
0	Sector(s) were free and are now held for the program's partition.
1	Sector(s) were already held by another partition.

3. Purpose

The HLDR subroutine provides linkage to CPL HOLD statements.

4. Parameters

File-name is defined in a CPL FILE statement as a Class 2, indexed or spanned file.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

HLDR

The HLDR subroutine provides the following entrypoint:

FRER

6. Cautions

a. The HLDR subroutine may be used only with Type I indexed files and uninitialized spanned files that are accessed with GETR and PUTR subroutines. HLDR is compatible with the CPL READB and WRITEB statements.

b. A HLDR executed on the same record twice without a FRER will result in an ABORT 14 - Invalid Sector Hold/Free.

7. See also:

In this manual:

NEWK GETK DELK PUTR GETR
FRER

In the CENTURION PROGRAMMING LANGUAGE manual:

HOLD

```
***************
*             *
*    FRER     *
*             *
***************
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF CPU 5-COMPATABLE APPLICATIONS. NOT FOR USE IN DEVELOPMENT OF NEW NON-CPU 5 COMPATABLE PROGRAMS!

1. General Form

CALL FRER (file-name)

where

file-name	specifies the file containing the record that should be released from exclusive access.

2. Effect

The FRER subroutine provides compatability with the CPL FREE statements. once even though more than one subroutine using it is called by the program.

3. Purpose

The FRER subroutine provides linkage with the CPU 5 CPL programs.

4. Parameters

File-name is defined in a CPL FILE statement as a Class 2, indexed file.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entries:

HLDR
FRER

6. Cautions

a. The FRER subroutine may be used only with Type I indexed files and spanned files that are accessed with GETR and PUTR subroutines.

b. The HLDR subroutine must be called at least once before the Linker Utility will add it and the FRER subroutine to the program.

c. The FRER subroutine is also compatible with the CPL READB and WRITEB statements.

d. A FRER of a record not held previously with a HLDR statement will cause an ABORT 14 - Invalid Sector Hold/Free.

7. See also:

In this manual:

NEWK GETK DELK PUTR GETR
HLDR

In the CENTURION PROGRAMMING LANGUAGE manual:

FREE

```
***************
*             *
*    ?EDIT    *
*             *
***************
```

1. General Form

CALL ?EDIT (integer-name, string-name-1, string-name-2)

where

integer-name	specifies the six-byte integer to be edited,
string-name-1	specifies the string that should receive the result of the edit, and
string-name-2	specifies the mask that should be used by the subroutine to edit the integer.

2. Effect

The ?EDIT subroutine converts a six-byte integer to a string, applying character editing (dollar signs, leading zeroes, commas, etc.) as specified by a mask.

Before editing, the subroutine fills the target string with the fill character specified in the mask (see "Parameters" below).

In the edit phase, the subroutine encodes the individual positions in the target string, moving from right to left, using digits from the source integer in the indicated digit positions; special characters assume the same relative positions they have in the mask. The encoding stops when the last digit of the integer has been used.

After editing, the subroutine compares the number of digits encoded from the integer with the number of significant digits specified in the mask. If the actual number of digits is less than the number specified, the remaining significant positions are filled with zeroes.

Finally, the floating character specified in the mask is encoded into the target string in the position to the left of the most significant digit in the string.

3. Purpose

The ?EDIT subroutine enables CPL programs to output numbers in a finished, or formatted, form.

4. Parameters

a. Integer-name is defined as a six-byte integer in a CPL SET or INTEGER statement, i.e., the name begins with a question mark (?).

b. String-name-1 is defined in a CPL STRING statement. The size of the target string is determined as follows:

```
  Number of digit position indicators in mask
+ number of special characters in the mask
+ 1
```

See example below.

c. String-name-2 is defined in a CPL DEFINE statement. This mask is constructed as follows:

General Form

```
'xy@######'
```

where x is the pre-edit fill character,
y is the post-edit floating character,
@ is the significance indicator, and
# is the digit position indicator.

The mask may also contain special characters, that is, characters other than @ or #. A special character is encoded into the target string in the position indicated if it is surrounded by digits from the integer.

The position of the significance indicator (@) in the mask string indicates the number of significant digits in the integer being edited. The digit positions to the right of the at-sign are considered to be significant, that is, numbers will always be written into those positions, even if they are zeroes. The positions to the left of the indicator contain numbers only when they are supplied by the integer; otherwise, they are left blank.

NOTE: Special characters to the right of digit indicators should be preceded by another at-sign, or they may not be be edited into the target string. If a plus (+) or a minus (-) is specified in the rightmost position of the mask, the subroutine will substitute a space in that position if the edited integer is positive, but it will use the specified sign regardless of type if the integer is negative.

Examples:

```
Date:            '0 @##/##/##'
Social Security: '0 @###-##-####'
Negative money:  ' (#,###,##@#.##@)'
Protected check: '*$,###,##@#.##'
```

For example, the mask:

```
DEFINE MASK:'0$###,###@.##'
```

requires a target string 11 characters long.

When the integer 123456 is edited using this mask, the resulting target string is:

```
0 $1,234.56
```

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

```
?EDIT
```

The ?EDIT subroutine has no entrypoints.

6. Cautions

a. The ?EDIT subroutine does not handle signs automatically. The CPL program must distinguish between positive and negative numbers and provide sign indicators in different masks.

b. After the ?EDIT subroutine has been executed, the target string contains the fill character followed by the edited integer (see example above). If the fill character should not be printed with the number, the string should be referenced as follows:

```
string-name+1
```

For example:

```
CALL MSG (TARGET+1)
WRITE (PRT,F01) TARGET+1
```

c. The ?EDIT subroutine is capable of editing six-byte integers only. Four-byte integers are edited by the EDIT subroutine.

7. See also:

EDIT

```
***************
*             *
*    CLREC    *
*             *
***************
```

1. General Form

   CALL CLREC (record-name)

   where

   record-name — specifies the data record area to be cleared.

2. Effect

   The CLREC subroutine initializes the record specified, i.e., binary zeroes are written into every byte of the record storage area in the partition in which the CPL program is running.

3. Purpose

   The CLREC subroutine clears a data record area so that it is ready to receive data and be written onto disk for the first time.

4. Parameters

   Record-name is defined in a CPL RECORD statement.

   See the examples in ?NKEY, NEWK, and NEWKEY.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   CLREC

6. Cautions

   The CLREC subroutine will initialize any record specified in the CALL statement, including one containing valid data.

```
***************
*             *
*    EDIT     *
*             *
***************
```

1. General Form

CALL EDIT (integer-name, string-name-1, string-name-2)

where

integer-name	specifies the four-byte integer to be edited,
string-name-1	specifies the string that should receive the result of the edit, and
string-name-2	specifies the mask that should be used by the subroutine to edit the integer.

2. Effect

The EDIT subroutine converts a four-byte integer to a string, applying character editing (dollar signs, leading zeroes, commas, etc.) as specified by a mask.

Before editing, the subroutine fills the target string with the fill character specified in the mask (see "Parameters" below).

In the edit phase, the subroutine encodes the individual positions in the target string, moving from right to left, using digits from the source integer in the indicated digit positions; special characters assume the same relative positions they have in the mask. The encoding stops when the last digit of the integer has been used.

After editing, the subroutine compares the number of digits encoded from the integer with the number of significant digits specified in the mask. If the actual number of digits is less than the number specified, the remaining significant positions are filled with zeroes.

Finally, the floating character specified in the mask is encoded into the target string in the position to the left of the most significant digit in the string.

3. Purpose

The EDIT subroutine enables CPL programs to output numbers in a finished, or formatted, form.

4. Parameters

a. Integer-name is defined as a four-byte integer in a CPL SET or INTEGER statement.

b. String-name-1 is defined in a CPL STRING statement. The size of the target string is determined as follows:

```
  Number of digit position indicators in mask
+ number of special characters in the mask
+ 1
```

See example below.

c. String-name-2 is defined in a CPL DEFINE statement. This mask is constructed as follows:

General Form

```
'xy@######'
```

where x is the pre-edit fill character,
y is the post-edit floating character,
@ is the significance indicator, and
# is the digit position indicator.

The mask may also contain special characters, that is, characters other than @ or #. A special character is encoded into the target string in the position indicated if it is surrounded by digits from the integer.

The position of the significance indicator (@) in the mask string indicates the number of significant digits in the integer being edited. The digit positions to the right of the at-sign are considered to be significant, that is, numbers will always be written into those positions, even if they are zeroes. The positions to the left of the indicator contain numbers only when they are supplied by the integer; otherwise, they are left blank.

NOTE: Special characters to the right of digit indicators should be preceded by another at-sign, or they may not be be edited into the target string. If a plus (+) or a minus (-) is specified in the rightmost position of the mask, the subroutine will substitute a space in that position if the edited integer is positive, but it will use the specified sign regardless of type if the integer is negative.

Examples:

```
Date:            '0 @##/##/##'
Social Security: '0 @###-##-####'
Negative money:  ' (#,###,##@#.##@)'
Protected check: '*$,###,##@#.##'
```

For example, the mask:

```
DEFINE MASK:'0$###,###@.##'
```

requires a target string 11 characters long.

When the integer 123456 is edited using this mask, the resulting target string is:

```
0 $1,234.56
```

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

```
EDIT
```

The EDIT subroutine has no entrypoints.

6. Cautions

a. The EDIT subroutine does not handle signs automatically. The CPL program must distinguish between positive and negative numbers and provide sign indicators in different masks.

b. After the EDIT subroutine has been executed, the target string contains the fill character followed by the edited integer (see example above). If the fill character should not be printed with the number, the string should be referenced as follows:

```
string-name+1
```

For example:

```
CALL MSG (TARGET+1)
WRITE (PRT,F01) TARGET+1
```

c. The EDIT subroutine is capable of editing four-byte integers only. Four-byte integers are edited by the EDIT subroutine.

7. See also:

?EDIT

```
***************
*             *
*    MVFILE   *
*             *
***************
```

1. General Form

   CALL MVFILE (file-name-1, file-name-2)

   where

file-name-1	specifies the source file definition, and
file-name-2	specifies the target file definition.

2. Effect

   The MVFILE subroutine moves the source file definition, set by a CPL FILE statement, into the target file definition, i.e., the file specified by file-name-2 assumes the Programmer Logical Unit and characteristics of file-name-1.

3. Purpose

   The MVFILE subroutine enables a CPL program to change files, particularly output devices, easily on operator selection.

4. Parameters

   a. File-name-1 is defined in a CPL FILE statement.

   b. File-name-2 is defined in a CPL FILE statement.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   MVFILE

   The MVFILE subroutine has no entrypoints.

6. Cautions

a. The MVFILE subroutine must be used before the file-2 is opened.

b. Once the file definition has been moved, file-2 may not be the target of the MVFILE subroutine until it has been closed.

c. The MVFILE subroutine will not work if either file specified has been defined as an indexed file (IND). If the target area is smaller, subsequent data areas will be destroyed.

```
***************
*             *
*    MVREC    *
*             *
***************
```

1. General Form

   CALL MVREC (record-name-1, record-name-2)

   where

   record-name-1 specifies the source record area, and

   record-name-2 specifies the target record area.

2. Effect

   The MVREC subroutine moves the data stored in the source record area into the target record area. The transfer takes place in the portion of main memory allocated to the partition in which the CPL program is running.

   EFFECT ON PROGRAM SIZE: 45 bytes

3. Purpose

   The MVREC subroutine enables a CPL program to move entire data records.

4. Parameters

   a. Record-name-1 is defined in a CPL RECORD statement.

   b. Record-name-2 is defined in a CPL RECORD statement.

5. Externals and Entrypoints

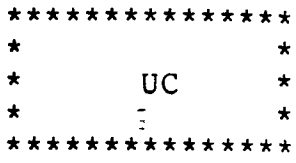
   An EXTERNAL statement in the CPL program must contain the following entry:

   MVREC

   The MVREC subroutine has no entrypoints.

6. Cautions

The target record area may not be smaller (in bytes) that the source record area; however, it may be larger.



1. General Form

   CALL UC (string-name)

   where

string-name	specifies the characters that should be converted.

2. Effect

   The UC subroutine converts the specified string from lowercase ASCII characters to uppercase ASCII characters.

3. Purpose

   The UC subroutine enables a CPL program to ensure that a string is in uppercase.

4. Parameters

   String-name is defined in a CPL STRING statement.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   UC

   The UC subroutine has no entrypoints.

6. Cautions

   The UC subroutine is normally not necessary since the operating system performs string comparisons as if all strings were in upper case.

7. See also:

   UC

```
***************
*             *
*      LC     *
*             *
***************
```

1. General Form

   CALL LC (string-name)

   where

string-name	is the name of a CPL string containing the characters to be converted.

2. Effect

   The LC subroutine converts the specified string from uppercase or mixed upper/lower case ASCII characters, to lowercase ASCII characters.

3. Purpose

   The LC subroutine allows CPL character strings to be converted to lowercase, for special display effects.

4. Parameters

   character-string is declared in a CPL DEFINE or STRING statement.

5. Externals

   An EXTERNAL statement in the CPL program must contain the following entry:

   LC

   The LC subroutine has no entrypoints.

6. Cautions

The LC subroutine will not give predictable results if attemped on an integer.

7. See Also:

UC

```
***************
*             *
*   BLTRUN    *
*             *
***************
```

1. General Form

   CALL BLTRUN (string-name)

   where

string-name	is the character string to be operated upon.

2. Effect

   The subroutine removes trailing blanks from a CPL character string.

3. BLTRUN is a pre-requisite for accurately determining the number of significant characters in the string, by the removal of trailing blanks.

4. Parameters

   String-name is declared in a CPL STRING or DEFINE statement.

5. Externals

   An EXTERNAL statement in the CPL program must contain the following entry:

   BLTRUN

   The subroutine BLTRUN has no entrypoints.

6. See also:

   STRLEN
   NOSIGN

```
***************
*             *
*   STRLEN    *
*             *
***************
```

1. General Form

   CALL STRLEN (string-name, integer-name)

   where

string-name	is the name of the CPL character string to be evaluated and
integer-name	is the name of a CPL integer to receive the result of the evaluation.

2. Effect

   The STRLEN subroutine counts the number of characters in a character string, and places that number in the integer.

3. Purpose

   The STRLEN subroutine may be used to determine string length for use in the variable field specifications of the CPL FORMAT statement.

4. Parameters

   string-name is declared in a CPL DEFINE or STRING statement

   integer-name is declared in a CPL INTEGER or SET statement

5. Externals

   An EXTERNAL statement in the CPL program must contain the following entry:

   STRLEN

   The subroutine STRLEN has no entrypoints.

6. Cautions

The STRLEN subroutine considers spaces, plus-signs, and minus-signs to be legitimate characters. To determine the number of characters in a string, excluding trailing blanks and/or sign characters, subroutines NOSIGN and BLTRUN should be used before subroutine STRLEN.

7. See also:

BLTRUN
NOSIGN

```
***************
*             *
*    FILL     *
*             *
***************
```

1. General Form

CALL FILL (string-name-1, number, string-name-2)

where

string-name-1	is the name of a CPL string to be filled with a specified character,
number	is either a literal or a 4-byte integer specifying the number of characters to be placed in the target string, and
string-name-2	is the name of a CPL string which declares the fill character.

2. Effect

The FILL subroutine places the specified number of fill characters in the target string.

3. Purpose

The FILL subroutine may be used to clear character strings by filling with blanks, or to move a variable number of characters into a string for printing horizontal bar-graphs.

4. Parameters

string-name-1 is declared in a CPL DEFINE or STRING statement

number is either a literal or a 4-byte integer declared in a CPL INTEGER or SET statement

string-name-2 is declared in a CPL DEFINE or STRING statement

5. Externals

An EXTERNAL statement in the CPL program must contain the following entry:

FILL

The FILL subroutine has no entrypoints.

6. Cautions

If there is more than one character defined in the string-name for the fill character, the system will use only the first character in the string for the fill character.

7. See also:

BLTRUN
STRLEN

```
***************
*             *
*   NOSIGN    *
*             *
***************
```

1. General Form

   CALL NOSIGN (string-name)

   where

string-name	is the name of a CPL string to be checked for trailing sign characters.

2. Effect

   If the last non-blank character in the character string is either a plus-sign (+) or minus-sign (-), the NOSIGN subroutine replaces that trailing sign with a blank. If the last non-blank character in the string is not a sign character, the subroutine has no effect on the string.

3. Purpose

   The NOSIGN subroutine is used to remove trailing signs from strings which may have been entered and terminated by the use of plus-return or minus-return instead of NEWLINE.

4. Parameters

   string-name is declared in a CPL DEFINE or STRING statement.

5. Externals

   An EXTERNAL statement in the CPL program must contain the following entry:

   NOSIGN

   The NOSIGN subroutine has no entrypoints.

6. Cautions

   a. Since any trailing sign characters will be replaced with a space, truncation of trailing blanks, subroutine BLTRUN, should not be done until subroutine NOSIGN has been run on the character string.

   b. Subroutine NOSIGN will only check the last non-blank character in a string for plus-sign or minus-sign. No other sign character within the string will be affected.

7. See Also:

   BLTRUN
   STRLEN

```
***************
*             *
*     GJP     *
*             *
***************
```

WARNING!! THIS SUBROUTINE IS FOR SUPPORT OF CPU 5-COMPATABLE APPLICATIONS. NOT FOR USE IN DEVELOPMENT OF NEW NON-CPU 5 COMPATABLE PROGRAMS!

1. General Form

CALL GJP (parameter-number, string-name)

where

parameter-number specifies a Job Control parameter, and

string-name specifies the string that should receive the value of the specified parameter.

2. Effect

The GJP subroutine moves the value of the Job Control Parameter whose number is specified into the string indicated by string-name.

3. Purpose

The GJP subroutine allows communication between jobstreams and CPL programs via the Job Control Parameters.

NOTE: Job Control Parameters are exclusive to the partition being used.

4. Parameters

a. Parameter-number may be either a numeric constant or an integer-name defined in a CPL SET or INTEGER statement.

b. String-name is defined as a six-character string in a CPL STRING statement.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

GJP

The GJP subroutine has no entrypoints.

6. Cautions

a. Job Control Parameters are maintained as ten-character strings but only the first 6 will be moved. The CPL string named in the subroutine call must be at least six characters long.

b. The CPL string should be equated to one of the following values before the GJP subroutine is called:

1. ' ' (six spaces) if the value moved into the string will be treated as a string.

2. '000000' (six zeroes) if the value moved into the string will be decoded into an integer before being used.

NOTE: The CPL DECODE statement may not move data from a left-justified string (see below) to an integer properly.

c. The justification of the characters within the Job Control parameter is determined by the manner in which the value was established:

1. If the value was set by a JCL .SETA statement, the parameter string is right-justified.

2. If the value was set by a JCL .SETC, .JOB, or .PARM statement, the parameter string is left-justified.

3. If the value was set when the jobstream was called up, the parameter string is left-justified.

NOTE: A parameter may be right-justified in a jobstream with the following JCL statement:

.SETA p = #p

where p is the parameter number.

7. See also:

PJP PUPSI GUPSI GETJP PUTJP

```
***************
*             *
*     PJP     *
*             *
***************
```

WARNING!! THIS SUBROUTINE IS FOR USE SUPPORT OF CPU 5-COMPATABLE APPLICATIONS. NOT FOR USE IN NEW NON-CPU 5 COMPATABLE PROGRAMS!

1. General Form

CALL PJP (number, string-name)

where

number	specifies the Job Control parameter that should receive the data, and
string-name	specifies the data that should be moved.

2. Effect

The PJP subroutine moves the value of the specified string into the Job Control parameter indicated by number.

3. Purpose

The PJP subroutine allows communication between jobstreams and CPL programs via the Job Control Parameters. NOTE: The Job Control Parameters are exclusive to the partition being used.

4. Parameters

a. Number may be an integer-name defined in a CPL SET or INTEGER statement or a numeric constant.

b. String-name is defined as a six-character string in a CPL STRING statement.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

PJP

The PJP subroutine has no entrypoints.

6. Cautions

The CPL string named in the subroutine call must be at least six characters long.

7. See also:

GJP GETJP PUTJP GUPSI PUPSI

```
***************
*             *
*    GETJP    *
*             *
***************
```

1. General Form

   CALL GETJP (parameter-number, string-name)

   where

   parameter-number specifies a Job Control parameter, and

   string-name specifies the string that should receive the value of the specified parameter.

2. Effect

   The GETJP subroutine moves the value of the Job Control Parameter whose number is specified into the string indicated by string-name.

3. Purpose

   The GETJP subroutine allows communication between jobstreams and CPL programs via the Job Control Parameters.

NOTE: Job Control Parameters are exclusive to the partition being used.

4. Parameters

   a. Parameter-number may be either a numeric constant or an integer-name defined in a CPL SET or INTEGER statement.

   b. String-name is defined as a ten-character string in a CPL STRING statement.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

GETJP

The GETJP subroutine has no entrypoints.

6. Cautions

a. Job Control Parameters are maintained as ten-character strings. Therefore, the CPL string named in the subroutine call must be at least ten characters long.

b. The CPL string should be equated to one of the following values before the GETJP subroutine is called:

1. ' ' (ten spaces) if the value moved into the string will be treated as a string.

2. '0000000000' (ten zeroes) if the value moved into the string will be decoded into an integer before being used.

NOTE: The CPL DECODE statement may not move data from a left-justified string (see below) to an integer properly.

c. The justification of the characters within the Job Control parameter is determined by the manner in which the value was established:

1. If the value was set by a JCL .SETA statement, the parameter string is right-justified.

2. If the value was set by a JCL .SETC, .JOB, or .PARM statement, the parameter string is left-justified.

3. If the value was set when the jobstream was called up, the parameter string is left-justified.

NOTE: A parameter may be right-justified in a jobstream with the following JCL statement:

.SETA p = #p

where p is the parameter number.

d. This subroutine may not be used on CPU 5-compatable

programs if it is to be expected to access right-justified strings.

7. See also:

PJP PUPSI GUPSI PUTJP GJP

```
***************
*             *
*    PUTJP    *
*             *
***************
```

1. General Form

   CALL PUTJP (number, string-name)

   where

number	specifies the Job Control parameter that should receive the data, and
string-name	specifies the data that should be moved.

2. Effect

   The PUTJP subroutine moves the value of the specified string into the Job Control parameter indicated by number.

3. Purpose

   The PUTJP subroutine allows communication between jobstreams and CPL programs via the Job Control Parameters. NOTE: The Job Control Parameters are exclusive to the partition being used.

4. Parameters

   a. Number may be an integer-name defined in a CPL SET or INTEGER statement or a numeric constant.

   b. String-name is defined as a ten-character string in a CPL STRING statement.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   PUTJP

   The PUTJP subroutine has no entrypoints.

6. Cautions

Job Control Parameters are maintained as ten-character strings. Therefore, the CPL string named in the subroutine call must be at least ten characters long.

7. See also:

GJP GETJP PJP GUPSI PUPSI

```
***************
*             *
*    GUPSI    *
*             *
***************
```

1. General Form

   CALL GUPSI (integer-name)

   where

   integer-name — specifies the integer that should receive the value of UPSI.

2. Effect

   The GUPSI subroutine moves the value of the User Programmable Switch Indicator (UPSI) into the specified integer.

3. Purpose

   The GUPSI subroutine allows communication between jobstreams and CPL programs via UPSI. NOTE: The value of UPSI is exclusive to the partition being used.

4. Parameters

   Integer-name is defined as a four-byte integer in a CPL SET or INTEGER statement.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   GUPSI

   The GUPSI subroutine has no entrypoints.

6. Cautions

   UPSI is an one-byte integer and therefore cannot contain values greater than 255 or less than 0.

7. See also:

PUPSI GJP PJP GETJP PUTJP

```
***************
*             *
*    PUPSI    *
*             *
***************
```

1. General Form

CALL PUPSI (number)

where

number — specifies the data that should be moved.

2 Effect

The PUPSI subroutine moves the value of number into the User Programmable Switch Indicator (UPSI).

3. Purpose

The PUPSI subroutine allows communication between jobstreams and CPL programs via UPSI.

NOTE: The value of UPSI is exclusive to the partition being used.

4. Parameters

Number may be either an integer-name defined in a CPL SET or INTEGER statement or a numeric constant.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

PUPSI

The PUPSI subroutine has no entrypoints.

6. Cautions

UPSI is an seven-bit integer and therefore cannot contain values greater than 255 or less than 0.

7. See also:

GUPSI GJP PJP GETJP PUTJP

```
***************
*             *
*   VOLNAM    *
*             *
***************
```

1. General Form

   CALL VOLNAM (file-name, string-name)

   where

file-name	specifies the file for which the disk volume name is required, and
string-name	is the string-name that is to receive the volume name.

2. Effect

   This subroutine obtains the volume name of the disk where the file specified is located, and puts it in a character string.

3. Purpose

   This subroutine is useful in such applications as report generators as a method of naming the disk that files are located on.

4. Parameters

   a. File-name is defined in a CPL FILE statement.

   b. String-name is defined in a CPL STRING or DEFINE statement.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   VOLNAM

   The VOLNAM subroutine has no entrypoints.

6. Cautions

   a. The resultant string must be at least ten characters in length.

   b. The volume name will be left-justified and space-filled to ten characters.

```
***************
*             *
*   GETTIB    *
*             *
***************
```

1. General Form

   CALL GETTIB (length,offset,result)

   where

   length is the number of bytes to obtain from the Task Information Block (TIB),

   offset is the number of bytes from the beginning of the TIB to where the requested data is stored in the TIB, and

   result is the location where the data is to be stored, depending on what information is requested.

2. Effect

   The GETTIB subroutine locates and moves data from the Task Information Block (TIB) for the executing partition, and stores the data for recall.

3. Purpose

   The GETTIB subroutine is used to get and store current partition parameters; such as UPSI, default disk of the partition, highest memory address in the partition, etc.

4. Parameters

   Both length and offset may be literals or the names of 4-byte integers.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   GETTIB

The GETTIB subroutine has no entrypoints.

6. Cautions

Some portions of the TIB are protected. If an attempt is made to get these portions of the TIB, an ABORT 21 - ILLEGAL SYSTEM BLOCK ACCESS ATTEMPTED will be generated.

7. See also:

PUTTIB

```
***************
*             *
*   PUTTIB    *
*             *
***************
```

1. General Form

   CALL PUTTIB (length,offset,result)

   where

   length is the number of bytes to be put in the Task Information Block (TIB),

   offset is the number of bytes from the beginning of the TIB to where the requested data is to be stored in the TIB, and

   result is the location where the data was stored, depending on what information is requested.

2. Effect

   The PUTTIB subroutine locates and moves data to the Task Information Block (TIB) for the executing partition, from where the data was stored.

3. Purpose

   The PUTTIB subroutine is used to modify information on current partition parameters in the TIB.

4. Parameters

   Both length and offset may be literals or the names of 4-byte integers.

5. Externals and Entrypoints

   An EXTERNAL statement in the CPL program must contain the following entry:

   PUTTIB

   The PUTTIB statement has no entrypoints.

6. Cautions

   a. Some portions of the TIB are protected. If an attempt is made to store into these portions of the TIB, an ABORT 21 - ILLEGAL SYSTEM BLOCK ACCESS ATTEMPTED will be generated.

   b. Although some TIB values may be accessed with GETTIB, these may be protected from modification. If modification is attempted, the result will be an ABORT 21.

7. See also:

   GETTIB

```
***************
*             *
*   GETPUB    *
*             *
***************
```

1. General Form

```
CALL GETPUB (file,length,offset,result)
```

where

file — is the file with which the Physical Unit Block (PUB) is associated,

length — is the number of bytes containing requested data from the PUB,

offset — is the number of bytes from the beginning of the PUB to where the requested data is stored in the PUB, and

result — is the location where the data is to be stored, depending on what information is requested.

2. Effect

The GETPUB subroutine locates and moves data from the Physical Unit Block (PUB) for the indicated file, and stores the data for recall.

3. Purpose

The GETPUB subroutine is used to locate and store information on the physical units associated with a file.

4. Parameters

Both length and offset may be literals or the names of 4-byte integers.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

```
GETPUB
```

The GETPUB subroutine has no entrypoints.

6. Cautions

Some portions of the PUB are protected. If an attempt is made to get these portions of the PUB, an ABORT 21 - ILLEGAL SYSTEM BLOCK ACCESS ATTEMPTED will be generated.

7. See also:

PUTPUB

```
***************
*             *
*   PUTPUB    *
*             *
***************
```

1. General Form

CALL PUTPUB (File,length,offset,result)

where

file is the file to which the devices are assigned,

length is the number of bytes to be put in the Physical Unit Block (PUB),

offset is the number of bytes from the beginning of the PUB to where the requested data is to be stored in the PUB, and

result is the location where the data was stored, depending on what information was requested.

2. Effect

The PUTPUB subroutine locates and moves data to the Physical Unit Block (PUB) on the devices assigned to the file, from where the data was stored.

3. Purpose

The PUTPUB subroutine is used to recall and put information on physical units assigned to the file, into the PUB.

4. Parameters

Both length and offset may be literals or the names of 4-byte integers.

5. Externals and Entrypoints

An EXTERNAL statement in the CPL program must contain the following entry:

PUTPUB

The PUTPUB subroutine has no entrypoints.

6. Cautions

   a. Some portions of the PUB are protected. If an attempt is made to store into these portions of the PUB, an ABORT 21 - ILLEGAL SYSTEM BLOCK ACCESS ATTEMPTED will be generated.

   b. Although some PUB values may be accessed by a GETPUB, these may be protected from modification. If these are accessed, an ABORT 21 will occur.

7. See also:

   GETP"B

Communications Module

APLIB Application

CPU-6

March 15, 1983

COMMUNICATIONS MODULE

OVERVIEW

The communications modules are independent modules which act as communications buffers; they also manage transmission/reception. A protocol, should the user decide to use one, must be defined and controlled by the utility or application program utilizing the communications modules. There is a set of CPL-compatible re-entrant subroutines (communications interface modules) implemented that perform the data buffering. The maximum transmission record size is 400 bytes.

Thus, by utilizing the communications modules, any application can perform data transmission and reception with any protocol desired. However, if the application is to interface with the XMIT/RECV utilities, it must honor the protocol.

The P.APLIB6 subroutines and their "call" formats are described independently on the following pages.

```
***************
*             *
*    OPCOM    *
*             *
***************
```

1. General Form

CALL OPCOM (file, sta)

where

file - name (address) of the RCB or CPL FILE statement assigned to the asynchronous communications port;

sta - name (address) of a four-byte integer to receive the open status when complete. Upon return, it contains one of the following values:

0 - open successful;
16 - file is of incorrect type for communications;
17 - illegal open;
18 - no memory available. There is not enough partition memory available to generate another buffer. The partition is at or close to the 32K maximum.

2. Effect

No actual data transmission or reception is performed. However, once completed, transmission or receptions can begin immediately and reception will begin if necessary.

3. Purpose

The OPCOM routine initializes buffers and parameters for a particular asynchronous port.

****
*   *
* GETCOM *
*   *
****

1. General Form

CALL GETCOM (file, rec, com, sta)

where

file - name (address) of the RCB or CPL FILE statement;

rec - name (address) of the CPL record area to receive the data. The first two bytes of the record should contain the record length;

com - name (address) of a four-byte integer to receive the communications flag. The communications flag is a four-bit value (0-15) that is transmitted/received as as the uppermost four bits of the transmission length. This value is ignored by the communications modules and its use, if any, is determined by the program or programs involved. For example, typical uses would be for protocol commands (as with XMIT/ /RECV). This value is also used by WAITC to acknowledge a successful transmission of a record to the transmission file; it is usually set to zero. If it is set to -1, then no acknowledgement will be transmitted.

sta - name (address) of a four-byte integer that will contain the status of the communications "read". It will contain one of the following values:

negative - reception not yet completed;
8 - port is not open or is of the wrong type;
22 - port is busy.

2. Effect

It returns immediately to the program whether a record is available in the receive buffer or not. It is the program's responsibility to monitor the "sta" for completion of the "read" and to use the WAITC routine to allow completion.

3. Purpose

The GETCOM routine performs a "read" of the communicatons port.

4. See also:

WAITC

```
***************
*             *
*   PUTCOM    *
*             *
***************
```

1. General

CALL PUTCOM (file, rec, com, sta)

where

file - name (address) of the RCB or CPL FILE statement;

rec - name (address) of the CPL record area to receive the data. The first two bytes of the record should contain the record length;

com - name (address) of a four-byte integer to receive the communications flag. The communica- flag is a four-bit value (0-15) that is transmitted/received as the uppermost four bits of the transmission length. This value is ignored by the communications module and its use, if any, is determined by the program or programs involved. For example, typical uses would be for protocol commands (as with XMIT/RECV). This value is also used by WAITC to acknowledge a successful transmission file; it is usually set zero (0). If it is set to -1, then no acknowledgement will be transmitted.

sta - name (address) of a four-byte integer that will contain the status of the communications "read". It will contain one of the following values:

Negative - transmission not complete;
0 - transmission not complete;
5 - line not opened;
6 - no response;
7 - output time out;
8 - line not opened;
22 - line busy.

2. Effect

The "call" parameters have the same meanings as with the GETCOM routine. Upon return, "sta" is set to the value of the third byte in ACK and will contain one of the values listed under "sta".

3. Purpose

The PUTCOM routine performs a "write" to the communications port and then reads a three-byte ACK back unless EOM is set to -1.

```
***************
*             *
*    WAITC    *
*             *
***************
```

1. General Form

   CALL WAITC

   where

   sta - contains the following values:

   -1 - reception is not complete;
    0 - reception completed with no errors;
    5 - device (line) not opened or assigned;
    6 - time out;
   19 - protocol error;
   20 - check sum error;
   23 - port is inactive.

2. Effect

   The WAITC automatically returns if none of the input requests have been completed. This allows the programmer to use only one input buffer for multiple input ports. It also provides the minimum turnaround time for receiving the next record which may be on its way. There are no "call" arguments. It is the responsibility of the program to check the various "sta" variables to determine which, if any, GETCOM requests have been completed.

3. Purpose

   The WAITC routine is used to complete any outstanding GETCOM requests. Its use is required and it should be used frequently if reception is occurring on multiple ports.

4. See also:

   GETCOM

```
***************
*             *
*   ENDCOM    *
*             *
***************
```

1. General Form

   CALL ENDCOM (file, sta, op)

   where

   file - name (address) of the RCB or CPL FILEEstatement;

   sta - name (address) of a four-byte integer to receive the ENDCOM status. Upon return, it will contain one of the following values:

   0 - ENDCOM successful;
   8 - port not opened or of the wrong type;
   22 - line busy or active;
   op - a literal zero/nonzero option flag. If zero, the asynchronous port is to remain assigned. If non-zero, it is to be released.

   NOTE: An ENDCOM status of 22 indicates that data is still awaiting reception and any outstanding GETCOM calls calls must be completed before closing of the port. This condition should clear once the WAITC call had completed.

2. Effect

   No actual data transmission or reception is performed. Optionally, the ENDCOM also releases (unassigns) the asychronous port.

3. Purpose

   The ENDCOM routine is used to close the communications port. This basically entails releasing buffers and other communications module resources so that the resources can be used by other future OPCOM "calls".

4. See also:

   GETCOM
   WAITC