Revised: 7/30/66

Identification

File status, change name or mode, or delete
CHFILE, DELFIL, FSTATE, STORGE, UPDATE

Purpose

With the new I/O system, as with the old, it is possible to change the mode or name of a file, to delete a file, or query the system about the status of a file. If the entry in the current file directory is a link, these routines refer to the actual file not the link entry.

Usage

Change:

CHFILE.($OLDNM1$,$OLDNM2$, NEWMOD , $NEWNM1$, $NEWNM2$)

OLDNM1 OLDNM2 is the name of the file which is to be changed (right adjusted, blank padded). This file may not be in active status at the time of the change.

NEWMOD is the desired mode of the file.

NEWNM1 NEWNM2 is the desired name of the file. NEWNM1 NEWNM2 may not be the same as OLDNM1 OLDNM2. To change just the mode, the new name must be specified as -0.

Error codes:

03. Attempt to change M.F.D. or U.F.D. file
04. File not found in U.F.D.
05. 'LINKED' file not found
06. Linking depth exceeded.
07. Attempt to change 'PRIVATE' file
08. Attempt to change 'PROTECTED' file of another user
09. Temporary file would overflow space allotted for device
10. File already exists with name 'NEWNM1 NEWNM2'
11. Machine or System error
12. File in active status

Delete:

DELFIL.($ NAME1$,$ NAME2$)

DELFIL will delete the file NAME1 NAME2 from the file directory and the space is immediately available for use within the record quota.

Error codes:

03. File not found in U.F.D.
04. 'LINKED' file not found
05. Linking depth exceeded
06. File is 'PROTECTED, PRIVATE, READ-ONLY, or WRITE-ONLY.
07. Machine or System error
08. File in active status

Status:

FSTATE.($ NAME1$,$ NAME2$,A(8)...8)

Upon return, the array A will contain the following information as integers:

A(8)= length of file in number of words
A(7)= MODE of file: MODE is negative and the 'OR' modes if the U.F.D. entry is a link.
A(6)= STATUS of file (1-4)
A(5)= DEVICE on which file resided (1-3)
A(4)= Address of next word to be read from file
A(3)= Address of next word to be written into file
A(2)= Date and time file was created or last modified, format of U.F.D.
A(1)= Date file was last referred to and 'AUTHOR' of file, format of U.F.D.

STATUS is 1 inactive
2 open for reading
3 open for writing
4 open for reading and writing

(N.B. "Open" means "opened by any user", not merely "opened by caller".) *

DEVICE is 1 Low speed drum
2 Disk
3 Tape

Error codes:

03. File not found in U.F.D.
04. 'LINKED' file not found
05. Linking depth exceeded

Size:

STORGE.(DEVICE,ALLOT,USED)

STORGE may be used to determine the number of records allotted and used on a particular device by the files of the current file directory.

ALLOT and USED are integer variables which, upon return, will contain the number of records allotted and used, respectively.

Error codes:

03. Illegal DEVICE specified
04. Machine or System error

Current UFD:

UPDATE.

UPDATE causes the I/O system to replace the user's U.F.D. (FILE) and the track usage table on the disk with the up-to-date versions which are maintained in core-A. The file system does this updating automatically and, therefore, UPDATE should not be called by the user.

(END)

Identification

Historic File System Error Procedure

Purpose

The historic supervisor disk control routine provided a standard error procedure as well as a handle by which the user may supply his own procedure.

Usage

Standard:

If a disk error occurs and the user has not specified an error return, the supervisor will type:

ILLEGAL CALL TO XXXXXX. NO ERROR RETURN SPECIFIED

and then call DORMNT so that debugging tools may be used.

User's option:

The user may add another argument to the calling sequence of any disk supervisor or library entry, in which he specifies the location of his error routine. If the prefix of this argument is PŻE, a diagnostic will be printed and control will be transferred to the specified location with an error code in the AC. If the prefix of the argument is MŻE, the diagnostic will not be printed but otherwise action will be the same as PŻE. The error codes are:

Illegal calling sequence	PŻE 1
Too many active files (.G. 10)	PŻE 2
User not found in Master File Directory	PŻE 3
Available space on module exhausted	PŻE 4
File not found	PŻE 5
Allotted track quota exhausted	PŻE 6

The error code of 1, "Illegal calling Sequence" may result from any of the following error conditions:

a. Illegal call to the .WRITE routine ; this occurs if the call to .WRITE references a file which is in active read status, or a file in relative read-write status where a relative address is not specified, or if a relative address is specified for a file not in relative read-write status or an R1 mode file in relative read-write status.

b. Illegal call to the .CLEAR routine; this occurs if the call references a file in active read status or relative read-write status.

c. Illegal call to the .FILE routine; this occurs if the call references a file in active read status.

d. Illegal call to the .READK routine; this occurs if the call references a file not in active read status, or if a relative address is specified for a file not in relative read-write status.

e. Illegal call to the .ENDRD routine; this occurs if the call references a file in neither active read nor relative read-write status.

f. Relative address too large for file; this occurs if an attempt is made to write into a relative address greater than the length of the file referred to.

g. File word count zero; this occurs on a call to .DUMP with a word count of zero, or a call to .FILE where no words have been written; the disk routine is so organized that a file with a zero word count may not exist.

h. Tried to rename read-only class 2.

i. Attempt to delete file in read-only mode.

j. File NAME1 NAME2 is not an active file; this occurs if a call to .WRITE, .FILE, .READK, or .ENDRD references a file not in active status.

Revised: 9/24/65

Identification

Library disk error procedure
SETERR, SNAP, RECOUP

Purpose

The library disk subroutines provide a standard error procedure as well as handles by which the user may provide his own error procedure.

Method

The library disk subroutines use a common routine which maintains an active file table. If an unexpected error occurs, the offended routine calls SNAP which prints an error message and calls RECOUP which in turn calls EXIT. EXIT is able by means of the active file table to properly * CLOSE any active write files and save core so that the user may then use debug facilities. RECOUP and SETERR are provided so that the user may supply his own error procedure.

Usage

SETERR:

```
      MAD:  EXECUTE SETERR. (-RETURN-,-ERROR-)
  FORTRAN:  CALL SETERR (-N-,-ERROR-)
      FAP:  TSX SETERR,4
            -PZE  RETURN-
            -PZE  ERROR-
```

SETERR modifies SNAP so that if SNAP is called, control will be transferred according to RETURN without disturbing any machine conditions.

RETURN is the error return location to which the library disk routines should transfer for unexpected errors. No message will be printed from SNAP.

ERROR is the location in which the logical accumulator will be stored i.e., the error code from the disk routine.

N Should be set by an ASSIGN statement in Fortran programs in order to provide the error return.

If only one argument is provided to SETERR, it will be used as the error return argument.

If no argument is provided to SETERR, the standard error procedure will be reinstated.

Every call to SETERR supercedes the previous one.

RECOUP:

CALL RECOUP (ERCODE, IR4,-IND-)

RECOUP may be supplied by the user if he wishes to provide his own procedure. If no user RECOUP is provided, the library version of RECOUP merely calls EXIT.

ERCODE contains the logical AC from the offended disk routine, or the error code from (IOH).

Error codes:
1 Illegal control character in format statement.
2 Illegal character in data field.
3 Illegal character encountered in octal input data.

IR4 (decrement) contains the contents of index register 4 at the time of the call to SNAP. It should be used to reset index register 4 before returning to the I/O routine.

IND contains the contents of the sense indicators at the time of the error in the disk routine. This argument is not present in the call from (IOH).

Sense indicators contain (decrement) the return location if processing is to be continued.

SNAP:

The library disk subroutines normally supply SNAP as the error exit to the supervisor disk routines. The call is,therefore, a TRA instead of a TSX and the AC contains the disk error code.

If SNAP has not been modified by SETERR, it will call PRNTER to print the standard error message, then print the following message and call RECOUP. *

XX CALLED SNAP FROM ABSOLUTE LOC NN. RECOUP CALLED. *

XX is the name of the disk routine in which the error occurred. *

NN is the absolute octal location of the call to SNAP. *

Identification

End-of-file procedure for library subroutines
EOFXIT, SETEOF, WRDCNT

Purpose

EOFXIT provides a common end-of-file procedure for all library subroutines which read tape or disk files. The user is supplied a handle whereby he may supply his own end-of-file procedure if he wishes.

Method

The standard library procedure is to call EOFXIT upon encountering an end-of-file. EOFXIT prints a message and calls EXIT. The user may call SETEOF before reading and thus modify EOFXIT to return to the user's eof procedure rather than calling EXIT.

Usage

EOFXIT:

The library routines call EOFXIT by:

```
        TSX   EOFXIT,4
        PZE   FILNAM
```

EOFXIT prints the message "END OF FILE READING NAME1 NAME2". It then calls EXIT, unless it has been modified by SETEOF.

SETEOF:

```
FAP:    TSX   SETEOF,4
       -PZE   EOF-
       -PZE   FILNM1-
       -PZE   FILNM2-

MAD:    EXECUTE SETEOF.(-EOF-,-FILNM1-,-FILNM2-)
FORTRAN: CALL SETEOF(-N-, -FILNM1-, -FILNM2-)
```

SETEOF will modify EOFXIT to return to location EOF in the user's program if an end-of-file is encountered. If there are no arguments, the standard eof procedure is restored. Each call to SETEOF supercedes any previous call.

EOF is the location of the user's end-of-file procedure.

N must be set by an ASSIGN statement in Fortran

```
i.e.  ASSIGN 1 TO N
      GO TO N, (1,2)
   1  ASSIGN 2 TO N
      .
      .
      .
   2  eof procedure
```

FILNM1,FILNM2 are the locations in which NAME1 and NAME2, respectively, will be stored by EOFXIT. If FILNM2 is missing, the logical tape number will be stored in FILNM1. If both FILNM1 and FILNM2 are missing, a single argument will be assumed to be EOF or N.

WRDCNT:

```
FAP:  TSX  WRDCNT,4    or   TSX WRDCNT,4
      PZE  LOC              STO  LOC
```

MAD or FORTRAN: CALL WRDCNT (LOC)

WRDCNT can be called only after an end of file was encountered by BREAD or VREAD.

LOC will contain the number of words transmitted by BREAD as a right adjusted integer. If WRDCNT is called by a FORTRAN program, the integer will be in the decrement of LOC.

Revised: 9/24/65

Identification

Terminal procedure.
EXIT, EXITM, CLKOUT, ENDJOB, DUMP, PDUMP

Purpose

To provide a common routine for the normal logical termination of all programs. The option is provided for placing the program in DORMNT status so that post mortem debugging may be used.

Usage

EXIT, CLKOUT and ENDJOB are synonomous.

```
EXECUTE EXIT.
EXECUTE CLKOUT.
EXECUTE ENDJOB.
END OF PROGRAM
END OF FUNCTION
```

The message "EXIT CALLED. PM MAY BE TAKEN" will be printed. EXIT calls CLOUT to close all active files. If no library routines calling the file system exist in the program, a dummy CLOUT will be loaded from the library with EXIT.

```
EXECUTE DUMP.
EXECUTE PDUMP.
```

The exit message will be printed with the name DUMP or PDUMP substitued for EXIT.

Any of the above calls cause all active files as defined by library subroutines to be properly closed and then a transfer to DORMNT.

```
EXECUTE EXITM.
```

The message "EXITM CALLED. GOODBYE" will be printed active files will not be closed transfer will be to DEAD.

Identification

Error Exit from Math Library Routines
LDUMP

Purpose

LDUMP is a subprogram to which some library math routines transfer upon encountering an error. The version of LDUMP which is in the library is a call to EXIT, but the user may provide his own version of LDUMP to provide recovery action.

Usage

The calling sequence to LDUMP which is used by the math routines is

```
FAP:        CLA   ARG1
            LDQ   ARG2
            TSX   LDUMP,4
            PZE   NAME
            TRA   IN          TO REPEAT ROUTINE
            TRA   OUT         TO EXIT FROM ROUTINE
      IN    LXD   IR4,4
            TRA   0,4
      OUT   LXD   IR4,4
            TRA   1,4
```

ARG1 contains the first argument to the math library subprogram.

ARG2 contains the second argument, if any, to the math library routine.

NAME contains the BCD name of the offending routine.

IN is the return of 2,4 which the programmer should use if he is writing his own LDUMP and wishes to repeat the offended subprogram after he has corrected the error.

OUT is the return of 3,4 which the programmer should use if he wishes to return from the offended routine without repeating its calculations.

Revised: 7/30/66

Identification

Current I/O system error procedures
IODIAG, FERRTN, PRNTER

Purpose

There are three different ways that errors from the I/O system can be handled: First, if the user does nothing, the I/O system will print a standard message and call DORMNT. Second, the user may call FERRTN to establish a single general error return for all I/O system errors. Third, every call to the I/O system will accept two additional arguments which specify an error return and a location into which the error code will be stored. These arguments apply only to the call in which they appear; that is, if a general return has been specified, it will be overridden for and only for calls in which error return arguments occur. The subroutines included in the I/O (or file) system are those listed in Section AD.2.

Usage

1. Standard:

   If an error is encountered by the I/O system and the user has not supplied an error return via FERRTN or via the optional additional arguments to the I/O system subroutine call, the I/O system will type a standard message and call DORMNT so that debugging tools may be used. The typed message will include the information available from IODIAG. Open files will not be closed.

2. Single return:

```
MAD:  OLDERR = FERRTN.(ERRLOC)

FAP:  TSX  FERRTN,4
      PZE  ERRLOC     (note PZE, not TXH)
      SLW  OLDER
```

   FERRTN sets the standard I/O system error return to be location ERRLOC.

   ERRLOC is the location to which control should be transferred if the I/O system detects an error. Upon entry to ERRLOC, index register 4 will contain the value set by the call to the I/O system that caused the error to be detected. To continue execution by ignoring

the I/O call, transfer to 1, 4. To continue execution by repeating the I/O call, transfer to 0, 4.

If ERRLOC is zero, the standard I/O error procedure will be reinstated.

OLDERR Upon return from FERRTN, the AC will contain the previous setting of the system error return. Each call to FERRTN supercedes any previous call.

3. Individual returns:

Each call to the I/O system entries will accept two additional arguments at the end of the call. The first is the location to which control is to be transferred if an error is encountered by the I/O system. The second, if specified, is the location into which the error code will be placed by the I/O system.

4. Diagnostic information:

IODIAG. (A(7)...7)

IODIAG may be called to obtain specific information about the I/O system error. Upon return, the array A will contain the following information:

A(7)= Location of call causing the error
A(6)= BCD name of entry resulting in error
A(5)= Error code
A(4)= Input/Output error code (1-7)
A(3)= NAME1 of file involved in error
A(2)= NAME2 of file involved in error
A(1)= Location of file system where error was found (of no use to user)

5. Printing of diagnostic:

A. Subroutine: PRNTER.(-MASK-,-FCN.-)

B. Command: PRNTER -MASK-

PRNTER The subroutine PRNTER may be called after an error in the I/O system in order to print the inf$rmation that is available from IODIAG. In other words, PRNTER is a routine which calls IODIAG and formats and prints the information. For usage of the command, see AH.11.01.

MASK If specified, bits in MASK call for the printing of different parts of the output message. The message parts and their corresponding bits are:

```
200  the word 'ERROR'
100  numeric error code
040  diagnostic
020  file name
010  routine name
004  location called from
002  file system location
001  carriage return
```

If MASK equals zero or is not given, default MASK of 375 is used.

FCN. If a function name is given, then instead of printing, PRNTER calls FCN. by

```
EXECUTE  FCN. (BUFF,Z)
```

where Z is the highest subscript of the array BUFF, and BUFF(Z)... BUFF(1) contains the (BCD) message which would otherwise have been printed. The called function could then, for example, write the error message into a file and continue execution.

For the benefit of FAP subroutines, the calling sequence is in fact

```
TSX  FCN,4
TXH  B,,'z'
TXH  =z
```

where z = message size, B = BES location of message buffer.

Error codes

Standard error codes:

There are a few standard error codes which may be returned from any of the I/O system calls.

001. Illegal calling sequence or Protection violation
002. Unauthorized use of priveleged call
100. Error reading or writing U.F.D. or M.F.D.
101. U.F.D. or M.F.D. not found, Machine error

Input/output error codes:

In many of the write-ups of the calls to the I/O system, one of the possible error codes is labeled Input/Output error. For the most part these errors are detected only after the I/O has been completed and will, therefore, be reported one call late. The actual error may be diagnosed by the value of A(4) after a call to IODIAG.

1. Parity error reading or writing file
2. Fatal error reading or writing file, cannot continue
3. Available space exhausted on this device
4. Tape file not mounted or not available
5. Illegal operation on this device
6. Physical end of tape sensed while writing *
   or
   Logical End of Tape of tape passed trying to open a file
   or
   End of tape file encountered unexpectedly.

(END)

Revised 8/30/65

Identification

Write BCD pseudo tape with format conversion
.PUNCH, .PNCHL, .TAPWR, (SCH), (STH), (STHM)

Purpose

The MAD and FORTRAN BCD tape and punch statements are compiled as calling sequences to library subroutines. These subroutines then simulate the writing of tape files by calling the library disk routines.

Usage

```
    MAD:  PUNCH FORMAT FMT, LIST          FAP:  TSX  .PUNCH,4
          PUNCH ONLINE FORMAT FMT, LIST         TSX  .PNCHL,4
          WRITE BCD TAPE N, FMT, LIST           TSX  .TAPWR,4

FORTRAN:  PUNCH FMT, LIST                       TSX  (SCH),4
          WRITE OUTPUT TAPE N, FMT, LIST        TSX  (STH),4
```

The FAP calling sequence compiled for MAD programs is of the form:

```
        TSX  .PUNCH,4  or  TSX  .TAPWR,4                    *
                           STR  N                           *
        STR  FMT,,DIR  or  STR  SYMTB,DIR,FMT               *
        OPS
        STR  LIST,,ENDLST
        OPS
        STR  0
```

The FAP calling sequence compiled for FORTRAN programs is of the form:

```
        CAL  N
        TSX  (STH),4
        PZE  FMT,,SWITCH
        OPS
        LDQ  LIST
        STR  SWT
        OPS
        TSX  (FIL),4
```

.PUNCH, .PNCHL, and (SCH) create or append to a pseudo tape line-marked file named .TAPE. 3

.TAPWR, (STH), and (STHM) create or append to a pseudo tape line-marked file named .TAPE. 'n'

N contains the number of the pseudo tape to be used (decrement for FORTRAN)

OPS may be indexing instructions.

SWITCH is zero if the format is stored backwards and non-zero if the format is stored forward.

LIST,,ENDLST are for standard list processing (see MOVE 1, 2, 3).

DIR If zero, the format is stored forward. If one, the format is stored backward.

SWT if zero with I format, the value is taken from the decrement of location LIST. If non zero with I format, the value is taken from the address of location LIST.

SYMTB in a MAD call, refers to the start (bottom) of symbol table for this routine. *

(FIL) provides blank padding; with (SCH) to 80 characters and with (STH) to 132 characters.

Disk errors will evoke the standard library disk error procedure and format errors call RECOUP.

Revised: 9/24/65

Identification

Read BCD pseudo tape with format conversion
.TAPRD, (TSH), (TSHM)

Purpose

MAD and FORTRAN BCD tape read statements compile as calling sequences to library subroutines which in turn call the library disk routines to read pseudo tape files from disk.

Usage

```
        MAD:  READ BCD TAPE N, FMT, LIST
           FAP:  TSX  .TAPRD,4
                 STR  N
                 STR  FMT,,DIR  or  STR  SYMTB,DIR,FMT        *
                 OPS
                 STR  LIST,,ENDLST
                 OPS
                 STR  0

    FORTRAN:  READ INPUT TAPE N, FMT, LIST
           FAP:  CAL  N
                 TSX  (TSH),4
                 PZE  FMT,,SWITCH
                 OPS
                 STR
                 STQ  LIST
                 OPS
                 TSX  (RTN),4
```

(TSH) and (TSHM) are synonymous.

(TSH), (TSHM), and .TAPRD read records from the disk file .TAPE. n according to the format and list. The file may be line-marked or fixed length of 14 words.

N contains the tape number (decrement for (TSH)).

OPS may be indexing instructions.

SWITCH of non-zero indicates the format is stored forward.

DIR If zero, the format is stored forward. If one, the format is stored backward.

LIST,,ENDLST are standard LIST processing (see MOVE1).

SYMTB in a MAD call refers to the start (bottom) of the symbol table for this routine. *

Identification

Read and write binary pseudo tape.
(STB), (TSB), (WLR), (RLR)

Purpose

FORTRAN programs which use binary tape statements may be compiled as background and run as foreground since the library subroutines will simulate the tapes as disk files.

Restrictions

The subroutine .RBIN called by binary tape statements in a MAD or MADTRAN translated program is not currently available in the library.

Usage

```
FORTRAN:  WRITE TAPE N, LIST
      FAP:  CAL   N
            TSX   (STB),4
            OPS
            LDQ   LIST
            STR
            OPS
            TSX   (WLR),4

FORTRAN:  READ TAPE N, LIST
      FAP:  CAL   N
            TSX   (TSB),4
            OPS
            STR
            STQ   LIST
            OPS
            TSX   (RLR),4
```

N contains in the decrement the number of pseudo tape.

OPS may be indexing instructions.

(TSB) and (STB) read or write the number of words specified in the LIST from the pseudo tape file .TAPE. 'n' by calling BREAD or BWRITE.

Identification

Pseudo tapes; backspace, write end of file, rewind
.BSF, .BSR, .EFT, .RWT, (BST), (EFT), (RWT)

Purpose

MAD and FORTRAN programs which refer to tapes are assigned disk space which is used to simulate the tape. These pseudo tape files may then be referred to by the standard MAD and FORTRAN statements which compile as calling sequences to the appropriate library subroutines. These library subroutines then simulate the functions as far as possible on the pseudo tape files.

Restrictions

The disk pseudo tape files may not be backspaced and therefore the backspacing subroutines do nothing but print a console message "BACKSPACE TAPE IGNORED".

Usage

```
      MAD:  BACKSPACE FILE OF TAPE N
            BACKSPACE RECORD OF TAPE N
            END OF FILE TAPE N
            REWIND TAPE N

   MADTRN:  BACKSPACE N
            ENDFILE N
            REWIND N

      FAP:  TSX  .BSF,4  or  TSX  .EFT,4  or  TSX  .RWT,4
            TXH  N

  FORTRAN:  BACKSPACE N
            END FILE N
            REWIND N

      FAP:  CAL  N        CAL  N        CAL  N
            TSX  (BST),4  TSX  (EFT),4  TSX  (RWT),4
```

.BSF and .BSR are synonymous and simply transfer to (BST).

(BST) does nothing but print the console message "BACKSPACE TAPE IGNORED" and return.

.EFT and (EFT) close the pseudo tape file .TAPE. 'n' by calling the library subroutine FILE.

.RWT and (RWT) close the pseudo tape file .TAPE. 'n' if it is active.

Major Revision: 3/30/66

Identification

Use of tapes in foreground
MOUNT, UMOUNT, VERIFY, LABEL, TAPFIL

Purpose

Tapes may be read and written by foreground users either with or without a console (FIB). The major difference between the user-I/O system interface for disks and tapes is that messages must be relayed to the machine operator to mount and unmount certain tape reels. Otherwise the calls are the same calls as described for the new I/O system.

Restrictions

Users wishing to use tapes must have an administrative-allotted tape quota. Unless otherwise specified (by user messages to the operator) reels will be mounted with write rings.

Usage

Mount:

MOUNT.(-CHAN-, UNIT, MESSAG(N)...N)

MOUNT must be used to direct the I/O system to mount a reel of tape on the unit to be subsequently referred to as UNIT.

CHAN specifies which channel is desired. '1' specifies channel A; '2' specifies channel B; '0' or '-0' indicates "no preference".

UNIT specifies a logical unit number (0 through 32767) by which the user will refer to this reel in other calls.

MESSAG is the BCD message which will be printed for the operator in conjunction with the I/O system's mounting instructions. The message should contain information about "file protection" (write ring or no write ring) and reel identification. It should be stored "forwards" in memory; that is, the first word of the message should be in the highest-subscripted location of a MAD array. (This is not the order which MAD's VECTOR VALUE's statement normally furnishes, and it must be provided for.)

N is the number of machine words in the message (N.LE.20).

error codes:

03. No tape unit available on specified channel.
04. Tape file already exists.

Unmount:

UMOUNT.(UNIT,MESSAG(N)...N)

UMOUNT is used to direct the I/O system to dismount a tape and free the corresponding tape drive for other use.

UNIT is the logical unit number as defined by MOUNT.

MESSAG is the BCD message which will be printed for the operator along with the I/O system unmounting message. It should include information about what to do with the reel. (See discussion under MOUNT.)

N is the number of machine words of MESSAG (N.LE.20).

error code:

03. Tape file in use.

Labeling:

LABEL.(UNIT, LABL(N)...N)

LABEL must be used to write a label on a new tape before it is opened for writing.

UNIT has previously been defined by a call to MOUNT.

LABL is the unique label for this reel which provides identification and verification by the user. (See discussion of array order under MOUNT.)

N is the length of LABL (N.LE.4).

Error codes:

03. Tape file does not exist.
04. Machine error or bad status.

05. Mount failed - illegal operation (key code 11).
06. Mount failed - operations difficulties (key code 12).

Label verification:

VERIFY.(UNIT, LABL(N)...N)

VERIFY must be called before opening a tape file for reading in order to check the LABL on the reel mounted on UNIT. This insures that the operator has mounted the correct reel. The file may not be opened until a correct verification has been made.

N is the length of LABL (N.LE.4).

Error codes:

03. Tape file does not exist.
04. Machine error or bad status.
05. Mount failed - illegal operation (key code 11).
06. Mount failed - operations difficulties (key code 12).
07. Labels do not match.

File names:

TAPFIL.($ NAME1$,$ NAME2$, UNIT, FILENO)

TAPFIL must be called to create an entry for the file in the U.F.D. When a tape file is created, its name, unit number, and file number are entered in the U.F.D. The file may then and later be OPENed for reading on the same UNIT number without a call to TAPFIL. If a tape file was created under a different file directory, TAPFIL may be used to enter it in the current file directory. If a tape file was created on one UNIT and is to be read on a different unit, it must be DELFILed from the U.F.D. and then reentered with the new UNIT number by a call to TAPFIL. Any number of files may exist on one reel. There is a restriction of one reel per file.

FILENO is a sequence number (integer or integer variable) used to specify which file on the reel will be referred to as NAME1 NAME2. If a user wishes to append a file to a reel, FILENO must be "0" or "-0". When the file is OPENed, the file system will assign the proper FILENO.

error codes:

03. File already exists.
04. Machine or system error.
05. User has no tape quota.

Additional Information

While the calls to the file system for tape usage may look like other file system calls, there are some differences between tape and disk/drum usage. The salient ones are listed here.

Mount-tape requests are not queued. Thus before any MOUNT request is considered, the tape operator must have complied with any previous MOUNT request. Since the operator is being allowed one minute to find and mount the tape, a user should protect himself against burning up time waiting for mounts to be completed.

If for some reason (e.g., no tape drive available) a tape-mount cannot be performed, the user is informed via an error return when he tries to LABEL VERIFY. Since certain tables are initialized during the mount process, these must always be cleared - even when the MOUNT does not succeed. The clearing occurs the first time LABEL or VERIFY, is called if the MOUNT did not succeed. If the user changes his mind and does not call LABEL or VERIFY after requesting a MOUNT, he then must call UMOUNT. UMOUNT is automatically called during LOGOUT.

Should a user Quit after a MOUNT request but before the required call to UMOUNT (a bad practice), a tape drive will be uselessly assigned. The tape operator can remedy this difficulty by depressing a certain set of console keys. The tape will then be dismounted automatically.

A tape file must be opened either for reading or for writing, not for both; record numbers must be consecutive during reading or writing. Attempts to rewrite a tape file will result in an error. When the physical end of tape is reached, the file being written must be closed. Moreover, the record being written is not retrievable from the tape. Consequently, the user must have the tape unloaded (call UMOUNT) and a fresh tape mounted (calls to MOUNT and LABEL). The writing can then be resumed in a new file by TAPFILing, OPENing the new file and then writing that last record again. Physical records on tape are in binary mode and are 433 (decimal) words long (except the last record of a file, which may be shorter). The first word contains the record number and, for the last record, the word count of the record.

Once a tape label has been successfully created or verified, subsequent calls to VERIFY or LABEL are ignored. This is an outgrowth of two provisions. First, it seemed a good idea to allow rapid successive calls to VERIFY in case the user wanted to search a list of label candidates. Second, to expedite file retrieval performed by the operations staff, it was necessary to allow superfluous calls to VERIFY, once a tape had been successfully verified.

Tape usage is not multi-programmed. Thus a user spins tape only when his program is running in core B. While this situation is not as bad as it could be (tape I/O is performed with interrupts), it obviously represents an inherent simplification in our first effort to incorporate tapes as foreground I/O devices.

Format of Tapes

BTL (header label)

End of File mark

Data File 1

End of File mark

EOFL (end of file label)

End of File mark

BTL (header)

End of File

Data File 2

.
.
.
.
.

Data File n

End of File mark

EOFL

End of File mark

EOLTL (end of logical tape label)

End of File mark

Format of BTL

Word(s)	Contents	Description
1	GEbb60	Words 1-2 constitute
2	ObBTLb	Beginning of Tape label
3	MITMAC	
4	000000	
5	xxxxxx	File number on tape (in binary)
6	000000	
7	xxxxxx	Date file created (file system format)
8	xxxxxx	Number of days file is to be retained (usually . . . 999)
9-10	xx...x	File name
11-14	xx...x	User supplied label

The only information currently read by Tape Strategy on a previously created tape file is words 1, 2 and 11-14. The rest may be appropriated.

Format of ELTL

1	bEOLTb
2-14	00...0

Format of EOFL

1	bbEOFb
2	Number of records in data file (binary integer)
3-14	000000

Format of Data File i

File i consists of 433-word records where

word 1 (Address of first word) = number of the record within this file.

(Decrement of first word) = 0 unless this is the last record of the file. Then it equals the number of words in this last record excluding word 1.

words 2-433 User supplied data.

(END)

Identification

Program status
DEAD, DORMNT, GETILC, FNRTN

Purpose

To remove a program from active status and place it in dead or dormant status and to be able to know the location of the last call to DORMNT.

Usage

DEAD:
as supervisor or library entry:

```
TSX  DEAD,4     optional (TIA  =HDEAD)
```

DEAD returns control to the supervisor and places the user in dead status. Machine conditions are not saved and memory bound is set to zero.

DORMNT: as supervisor or library entry:

```
TSX  DORMNT,4     optional (TIA  =HDORMNT)
```

DORMNT returns control to the supervisor and places the user in dormant status. Machine conditions, status, and memory bound are saved. If the START command is issued, control returns to 1,4. If a new program is read in, the machine conditions, status, and memory bound are overwritten.

GETILC: as supervisor entry:

```
TSX  GETILC,4    (TIA  =HGETILC)
```

Upon return, the AC will contain the value of the instruction location counter at the time when the user last entered dormant status.

FNRTN: as supervisor entry:

```
TSX  FNRTN,4    (TIA  =HFNRTN)
```

FNRTN returns the user to dormant status and resets the user's instruction location counter to the value it had when he last entered dormant.

Identification

Periodic Dormancy
SLEEP

Purpose

To allow the user to place his program in dormant status and have it be automatically restarted after a predetermined amount of time has elapsed.

Usage

As supervisor or library entry:

```
CAL  =n
TSX  SLEEP,4      optional (TIA   =HSLEEP)
```

n is the actual number of seconds the user wishes to wait before restarting his program at 1,4.

While the program is in dormant status, the user may turn-off the alarm clock by issuing a new command or using the QUIT sequence.

Identification

Interrupt levels
GETBRK, SETBRK, SAVBRK

Purpose

In order to allow a program to be interrupted from the console but continue running in some other section, programs may be organized to run on different interrupt levels.

Restrictions

Command level is 0. Levels may be dropped to the maximum depth of 3.

Method

Command level and a program initially placed in working status are at interrupt level 0. A program may drop the interrupt level and set the entry point for each level. During execution, the level may be raised either by a program call to the supervisor or by the user sending the interrupt signal. The interrupt signal causes the interrupt level to be raised by 1 and control to be transferred to the entry point previously specified by the program.

An interrupt at level 0 will be ignored, (i.e., an interrupt cannot be used to QUIT). Each interrupt will cause the supervisor to print INT.n. where n is the level to which control is to be transferred.

Usage

SETBRK:
as supervisor or library entry:

```
TSX  SETBRK,4     optional (TIA   =HSETBRK)
PZE  'loc'
```

SETBRK sets the interrupt entry point for the current level to the value of loc and drops the interrupt level by 1.

SAVBRK:
as supervisor or library entry:

```
TSX  SAVBRK,4     optional (TIA   =HSAVBRK)
```

SAVBRK raises the interrupt level by 1 and returns in the AC the entry point corresponding to the level just entered. If SAVBRK is called within level 0, the AC will be zero.

GETBRK:

as supervisor or library entry:

```
TSX  GETBRK,4     optional (TIA   =HGETBRK)
```

Upon return, the AC will contain the value of the instruction location counter at the time the user last "interrupted".

Identification

Storage Map
STOMAP

Purpose

To print a storage map giving the entry names and locations of all subprograms in core B.

Usage

As library subroutine:

```
TSX  STØMAP,4
```

The subprogram origin and the entry names and locations will be printed for all subprograms in core-B.

Identification

Floating Point Trap
.SETUP, (FPT), (EFTM), (LFTM)

Purpose

To provide a means of initializing for, interpreting, recovering from, or flushing the program because of floating-point overflow or underflow.

Method

When the 7094 is operating in floating-point trap mode, a floating point operation which causes overflow or underflow will also cause a machine trap. The subroutine (FPT) will interpret the trap and take appropriate action. Some initialization must be done before the trap occurs to enable (FPT) to interpret the traps. .SETUP and (EFTM) are used in the initialization.

Usage

Mad and Fortran both automatically compile a calling sequence to .SETUP at the beginning of each main program. It need be executed only once per program.

```
TSX  .SETUP,4
```

The multiple tag mode (3 index mode) is entered. Location 8 is set to TTR (FPT). The floating-point trap mode is established by a call to (EFTM).

A floating-point underflow will cause the execution of the TTR (FPT) which will then zero the offending register and return control to the instruction following the offending floating point instruction.

A floating-point overflow will cause the execution of the TTR (FPT) which will then print a message on-line giving absolute and relative locations of the offending floating-point instruction with the name of the subprogram and the machine spill code. (FPT) then calls ERROR which prints a back trace of the subprograms previously called, if possible, and then calls EXIT.

(EFTM) and (LFTM):
as supervisor or library entries:

```
TSX  (EFTM),4     optional(TIA   =H(EFTM) )
TSX  (LFTM),4     optional(TIA   =H(LFTM) )
```

(EFTM) enters floating-point trapping mode with trapping mode simulated in core B

(LFTM) leaves the floating-point trapping mode.

N.B. The LOAD command enters the multiple tag mode before completion. Consequently, a program loaded with the relocatable loader will be automatically initiated in 3 tag mode.

Identification

Memory allotment
GETMEM, SETMEM, GMEM, SMEM, EXMEM

Purpose

To provide a way of determining or expanding the current memory allotment.

Method

At load time the memory allotment is set by the number of words required by the program. Memory protection, however, can only be set in blocks of 256 words and is therefore set to the next highest block of 256. If, during execution, the user wishes to change his memory allotment and/or protection, SETMEM may be called.

Restrictions

Since memory protection is set in blocks of 256 words, it is possible that a program may store information beyond the memory allotment bound without causing a protection violation. However, swapping is done by memory allotment rather than memory protection, so that information thus stored is lost during swapping.

Usage

As supervisor or library entries:

```
        TSX  GETMEM,4      optional (TIA  =HGETMEM)

        CLA  ='n'
        TSX  SETMEM,4      optional (TIA  =HSETMEM)
```

GETMEM returns in the address portion of the AC the current memory allotment.

SETMEM sets the memory allotment to the value of n. If n is (77777)8, all of memory is allotted, including location (77777)8.

As library subroutines:
MAD or FORTRAN:

```
        A = GMEM.(I)
        A = SMEM.(J)
```

```
FAP:    TSX GMEM,4      TSX SMEM,4
        PZE I           PZE J
        STO A           STO A
```

A and I Upon return, will contain an integer giving the current memory bound.

J contains an integer giving the memory bound desired.

GMEM returns to the caller the current value of the memory bound.

SMEM sets the memory bound to the value desired.

To extend memory bound:

As library subroutine:

MAD, FORTRAN or FAP:

```
A = EXMEM.(INC)
```

INC contains an integer which will be used as an increment to extend the memory bound.

A Upon return, A will contain the new memory bound which is the sum of the old memory bound and the increment in INC. If the sum is greater than (77777)8 or if the prefix of the argument is not PZE, TSX or TXH, return is made with A and the AC set to zero and the memory bound is not extended.

Identification

Free or erasable storage management
FREE, FRER, FRET

Purpose

One technique of optimizing the amount of core space required by one program is to have each subprogram within the program take temporary storage from a common pool and put it back when it is no longer needed.

Usage

As a library subroutine:

```
AED:  X=FREE(N)$,    X=FRER(N)$,    X=FRET(N,X)$,

FAP:  TSX FREE,4     TSX FRER,4     TSX FRET,4
      PZE N          PZE N          PZE N
      STA X          STA X          STA X
```

N contains an integer specifying the size of the block of storage.

X contains (address) the address of the start or lowest location of the block of storage. If X is returned as zero by FRER, no block could be obtained.

FREE will find a block of storage either from free storage or by extending memory bound. If more space is requested than can be found, the following message will be printed, and EXIT is called:
'nnnnn LOCATIONS OF FREE STORAGE ARE UNAVAILABLE
(nnnnn is an octal number.)

FRER serves the same function as FREE except that if not enough space is available, return will be to the calling program with zero in the AC.

FRET returns storage to free storage. If a block of storage being returned overlaps memory bound or any block previously returned, the following message is printed and EXIT is called:
** ILLEGAL CALL OF FRET, BLOCK rrrrr SIZE nnnnn'
(rrrrr is a pointer to the block, nnnnn is size; both in octal)

Revised: 5/23/66

Identification

Reset file-wait return
TILOCK

Purpose

A field called ILOCK exists within the UFD entry for each file. This field contains the number of users who currently have the file open for reading. If a user tries to write a file when its ILOCK is greater than zero, he will automatically be placed in file-wait status until no more users are reading the file. If a user tries to open a file which is open for writing, he will also be placed in file-wait status. TILOCK is a routine which has been provided to allow the user to avoid file-wait. A call to TILOCK in a program sets a general return which applies until altered or removed to all I/O calls which would otherwise involve going into file-wait status. All background programs which use the file system must provide this call since any attempt to place background in file-wait status causes the background job to stop.

Usage

MAD:
```
OLDRTN = TILOCK. (RETURN)
```

FAP:
```
TSX  TILOCK,4
PZE  RETURN   (note PZE rather than TXH)
SLW  OLDRTN
```

RETURN is the location to which control will be transferred if an I/O call would normally result in file-wait. If RETURN is zero, the normal execution of file-wait will be reinstated.

OLDRTN upon return, the AC will contain the address of the previous return setting, if any.

(END)

Identification

Query or modify supervisor parameters
GETLOC, GLOC, SETLOC, SLOC, SYPAR

Purpose

To enable a user to examine a supervisor parameter. To allow the system programmers to modify an A-core parameter.

Restrictions

SLOC and SETLOC may be used only by M1416 programmers. GLOC, SLOC and SYPAR may not be called from FORTRAN programs unless the location is shifted to the address rather than the decrement of LOC (or CODE).

Usage

Get the contents of a location:
As supervisor or library entry:

```
FAP:  TSX  GETLOC,4     optional (TIA   =HGETLOC)
      PZE  LOC
      SLW  WORD
```

As library subroutine:

```
MAD:  WORD = GLOC.(LOC)
```

Upon return, WORD will contain the contents of the A-core location whose address is in LOC.

Set the contents of a location:
As supervisor or library entry:

```
FAP:  CAL  WORD
      TSX  SETLOC,4     optional (TIA   =HSETLOC)
      PZE  LOC
```

As library subroutine:

```
MAD:  EXECUTE SLOC.(WORD, LOC)
```

Upon return, the A-core location whose address is in LOC will be set equal to the contents of WORD.

Get a supervisor parameter:
As library subroutine:

```
FAP:  TSX  SYPAR,4
      PZE  CODE
      STO  PARAM

MAD:  PARAM = SYPAR.(CODE)
```

SYPAR returns a supervisor parameter in the AC.

CODE contains a right adjusted integer which specifies which parameter is desired.

0	nothing
1	Last or lowest COMMON location used
2	COMMON length
3	First location loaded
4	Program length (i.e., memory allocation)
5	System name
6-9	reserved
10+	Contents of A-core location

Revised: 8/30/65

Identification

Get common file number
GETCF, GETCFN

Purpose

GETCF will return the number of the common file directory to which the user is currently switched. *

Usage *

As a supervisor entry:

```
TSX GETCF,4     (TIA =HGETCF)
```

Upon return, the AC will be zero if the user is switched to his own file directory. Otherwise, the AC will contain the number of the common file directory to which he is switched.

As a library subroutine:

```
 FAP:  TSX   GETCFN,4
       PZE   CFN
       STO   CFS

FORTRAN:  CFS = GETCFN(CFN)

 MAD:  CFS = GETCFN.(CFN)
```

Both CFN and CFS will be set to the current common file directory number (0,1,2..). In Fortran, the file directory number is returned as a Fortran integer. This same value may be used later to call COMFL(CFN).

Revised: 5/6/66

Identification

Privileged users' calls to the I/O system
UPDMFD, DELMFD,ALLOT, ATTACH, MOVFIL, SETFIL, LINK, UNLINK

Purpose

Administrators and certain commands and utility programs are privileged to alter the supervisor and the accounting files. Certain calls to the I/O system may be invoked only by the privileged users or other users using the privileged commands.

Method

The accounting files contain the personal restriction codes for every user of the system. When a user logs in, his restriction codes are placed in a vector within the supervisor along with the other active users. When a user invokes a command, his personal restriction code is 'OR'ed together with the code of the command to make up the restriction code which becomes part of his machine conditions. The LOGIN command sets the low-order 6 octal digits of the user restriction code.

1	User may use common files
2	User may use privileged calls to the I/O system.
4	User may modify "PROTECTED" files of other users.
10	User may refer to "PRIVATE" files of other users.
20	User may modify the supervisor and I/O system.
40	User may use the ESL display routines.
100	User may use the 6.36 supervisor entries.
200	User may not use disk-loaded commands, except LOGIN and LOGOUT ("Restricted User", see Section AA.1).
1000000	User is background system.
2000000	User is foreground.
4000000	User is FIB.
10000000	User is incremental dumper.
20000000	User is privileged command.

A privileged command sets the seven low order bits "on". The bits which occupy the decrement may be moved left nine bit-positions to indicate the .not. condition, except in the case of the privileged command bit.

Usage

Update MFD:

UPDMFD.($ PROBN$,$ PROG$)

UPDMFD places a new user (problem number programmer number) in the master file directory. With this call it is possible to update the MFD during time sharing rather than having to wait for a disk editor run.

PROBN is the right adjusted problem number of the form ANNNN. A is an alpha character, and NNNN is a four digit number.

PROG is a one to four digit programmer number. Note the right adjustment and blank padding.

Error codes:

03. User already in M.F.D.
04. Machine or System error
05. Illegal PROBN (i.e., 0)
06. Illegal use of M.F.D.

Delete from MFD:

DELMFD.($ PROBN$,$ PROG$)

DELMFD will remove a user from the master file directory. The DELMFD will not be permitted if the user's record count is not zero.

Error codes:

03. User not found in M.F.D.
04. U.F.D. still in use.

Attach to UFD:

ATTACH.($ PROBN$,$ PROG$)

ATTACH will attach the user's program to the file directory of user PROBN PROG. The user now has full access to the files and file directory of PROBN PROG within the limits of his restriction code. Files which may have been opened while attached to PROBN PROG remain open even if the attachment is changed to a different file directory.

Error codes:

03. User not found in M.F.D.
04. Machine or system error

Quota allotment:

ALLOT.(DEVICE,QUOTA,USED)

ALLOT may be used to allot a quota of records for each user, for each device by first ATTACHing to the users' file directory and then calling ALLOT.

DEVICE is an integer or integer variable specifying the I/O device.

1. Low-speed drum
2. Disk
3. Tape

QUOTA is an integer or integer variable specifying the number of records to be allotted to the user on the specified device. A record is currently 432 words.

USED is normally not specified and should be used only to correct an error in the number of records used.

Error codes:

03. Illegal device specified

Move a file:

MOVFIL.($ NAME1$,$ NAME2$,$ PROBN$,$ PROG$)

MOVFIL is used to move the file NAME1 NAME2 from the current file directory to the file directory of PROBN PROG. Upon return from this call, the file no longer exists in the current file directory.

Error codes:

03. File not found in current U.F.D.
04. File is a 'LINKED' file
05. File is 'PROTECTED'
06. File already exists in 'PROGN PROG'
07. Machine or System error
08. File already active.
09. U.F.D./M.F.D. not found

Link to a file:

LINK.($NAME1$,$NAME2$,$PROBN$,$PROG$, $NAM3$, $NAM4$, MODE)

LINK establishes a link in the current file directory to a file in some other file directory. Links may be established to the maximum depth of two, as specified by the supervisor.

NAME1 NAME2 is the name which will be used to refer to the file in the current file directory.

PROBN PROG specifies the file directory to which the link is being made. This file directory may contain the actual file or it may contain a link to some other directory.

NAM3 NAM4 is the name by which the file is known in file directory PROBN PROG. If NAM3 NAM4 is not specified, it is assumed to be the same as NAME1 NAME2.

MODE is an integer or integer variable which will be 'OR'ed with all the modes through all the links to the actual file. The resulting 'OR'ed mode will be used as the mode in the current file directory.

Error codes:

03. File already in U.F.D.
04. Machine or system error
05. 'PROBN PROG' not found in M.F.D.
06. Illegal use of M.F.D.

Remove a link:

UNLINK.($ NAME1$,$ NAME2$)

UNLINK will remove the U.F.D. entry and the link associated with NAME1 NAME2, which was established by LINK. NAME1 NAME2 is the name used to refer to the file in the current file directory, as it is in LINK. *

Error codes:

03. File not found in U.F.D.
04. File is not a 'LINKED' file
05. Machine or system error

Date a file:

SETFIL.($ NAME1$,$ NAME2$,DAYTIM,DATELU,MODE,DEVICE)

SETFIL is used primarily by the file load and retrieval programs to create an entry in a file directory with a specific date and time.

DAYTIM is the date and time to be used as the date and time last modified in the format of the third word of a U.F.D.. (AD.2)

DATELU is to be used as the fourth word of a U.F.D. and contains the date last used and 'AUTHOR'.

Error codes:

03. Illegal device
04. Machine or system error

(END)

Identification

Get directory attached to
ATTNAM

Purpose

ATTNAM returns the problem number and programmer number (PROBNO,PROGNO) of the directory currently attached to by the file system. Cf. WHOAMI, AG.7.05.

Usage

As a supervisor or library entry:

```
MAD:  ATTNAM.(A(2)...2)

FAP:          TSX  ATTNAM,4
              PTH  A,,2          or:   PTW  A,,N
                                         .
                                         .
                                         .
                                    N  PZE  2
```

Optional:

```
      ATTNAM TIA   =HATTNAM
```

On return from the FAP calls, A is set to PROBNO and A+1 to PROGNO. On return from the MAD calls, A(2) is set to PROBNO and A(1) to PROGNO. (PROBNO is of the form ANNNN, where A is an alpha character and NNNN is a form-digit number; PROBNO is a one-to four-digit number. Both are right-adjusted.)

Only the standard error code 1 may be returned.

Identification

Obtain user status information from supervisor.
WHOAMI

Purpose

To provide commands and user programs with such pertinent system parameters as user identification, system name, and console identification. The subroutine operates at the level of "who is logged in and making the call," as opposed to "whose directory is the call coming from " - for which latter, see ATTNAM, AG.7.04.

Usage

As supervisor or library entry:

MAD:

```
WHOAMI.(A(N)...N)    (N.LE.4)
```

FAP:

```
TSX   WHOAMI,4
OPN   A,,'n'         (OPN=PZE or TXH; n .LE. 4)
```

Optional:

```
WHOAMI TIA   =HWHOAMI
```

On return, locations in array A will have been set as follows:

MAD	Contents	FAP
A(N)	PROBNO	A
A(N-1)	PROGNO	A+1
A(N-2)	SYSNAM	A+2
A(N-3)	IDCODE	A+3

where PROBNO is problem number, PROGNO is programmer number, SYSNAM is the six-character system name of the currently operating version of CTSS, and IDCODE is the console identification code.

Identification

General discussion of MACRO command programs

Purpose

It is sometimes desirable or convenient to be able to initiate one command which results in the automatic execution of several commands. Tools have been provided on several programming levels for initiating and controlling chains of commands.

Discussion

There are at least three levels of user interest in chain or macro command programs: 1) writing commands which may be used within chains, 2) initiating chains from within a high level programming language, 3) initiating chains at the machine language and supervisory call level of programming.

Commands may be thought of as being subroutines without the conventional subroutine linkage. A standard command linkage, however, has been provided within the supervisor so that command arguments will always be available and retrievable from a standard place. All commands should terminate with a call to CHNCOM rather than one of the conventional programming terminal routines. CHNCOM will continue a command chain, if there is one, or call DORMNT (or DEAD, depending on the memory bound) if there is no chain. Routines that will fetch the command arguments are COMARG, which is callable by MAD or FORTRAN programs, and GETCOM, which is the supervisor entry.

Two routines are available for executing single commands from the program level: NEXCOM is a limited-use supervisor entry and XECOM is a more flexible subroutine which may be called by MAD or FORTRAN programs.

Chains of commands may be constructed in a simple way as BCD line-marked or line-numbered disk files and executed by the MAD or FORTRAN callable subroutine SCHAIN or by the command RUNCOM. SCHAIN and RUNCOM do a lot of the housekeeping and set up calls to the appropriate supervisor entries.

On the more detailed level, chains may be constructed within the supervisor, the command location counter may be set or interrogated, and the chains may be executed and chained by calls to supervisor entries. On this programming level many of the housekeeping details must be handled by the user.

Identification

Single command
XECOM, NEXCOM, NCOM

Purpose

To allow the user to execute a single command from the program level rather than the command level.

Usage

NEXCOM:
as supervisor entry:

```
CAL  COMAND
LDQ  ARG1
TSX  NEXCOM,4      (TIA   =HNEXCOM)
```

as library subroutine:

```
NCOM.(COMAND,ARG1)
```

COMAND contains the BCD name, right justified, of the command to be executed.

ARG1 is stored as the first argument in the current command buffer. If there is to be no argument to COMAND, ARG1 should be the fence. If COMAND expects an argument list and ARG1 is not a fence, the previous contents of the current command buffer will be used with ARG1 as the first argument.

NEXCOM places the contents of the AC and MQ in the current command buffer and places the user in waiting-command status. Note that a fence is not placed in the command buffer following the argument. Control is not returned to the calling program except as may have been pre-arranged by CHNCOM.

XECOM:
as library subroutine:

MAD, FORTRAN, FAP:

```
K =  XECOM.  (COMAND,LIST)
EXECUTE XECOM.  (COMAND,LIST)
```

COMAND contains the BCD name of the desired command. Right justification is not necessary.

LIST is any legal list specifying locations which contain the BCD names of the arguments

appropriate to the command. Right justification is not necessary but the number of items in the list must be .LE. 18.

K will be zero if execution was successful; non zero if failure.

XECOM builds a chain of SAVE, COMAND, RESUME and calls CHNCOM. Thus control will be returned to the calling program after execution of COMAND, if COMAND called CHNCOM.

Identification

MACRO command
SCHAIN

Purpose

To allow the user to build a macro command program as a BCD disk file and call for its execution from the program level rather than command level. A macro command program is a chain of commands which can be executed by issuing just one command, with or without arguments.

Reference

SCHAIN is the subroutine call which is the equivalent of the RUNCOM command. For a complete explanation, see section AH.10.01 , RUNCOM.

Usage

MAD, FORTRAN or FAP:

```
A = SCHAIN. (FILNAM,-ARG1,ARG2....ARGN-)
EXECUTE SCHAIN. (FILNAM,-ARG1,ARG2....ARGN-)
```

FILNAM specifies the BCD file containing the chain of commands to be executed. The secondary name need not be BCD as is required for RUNCOM.

ARG'S are locations of BCD names of specific arguments to be substituted for the dummy arguments specified by the CHAIN pseudo-command. They may be single or list variables and the names need not be right justified.

A Upon return may contain a word of the form...XXX, which is not an error, but the primary name of a SAVED file representing the last dormant status yielded by the last command in the chain.

SCHAIN will intersperse SAVE's and RESTOR's or RESUME's so that the chain specified in FILNAM may be of any length. Control is returned to the calling program upon completion of the chain. The chain may include any number of RUNCOM specifications, since nesting and recursion are possible.

Identification

Chain control
CHNCOM; (GET,G,SET,S) CLS; (GET,G,SET,S) CLC

Purpose

To allow a user to set up and control chains of commands from the program level rather than command level. These routines are close to the supervisory level and require detailed control by the user.

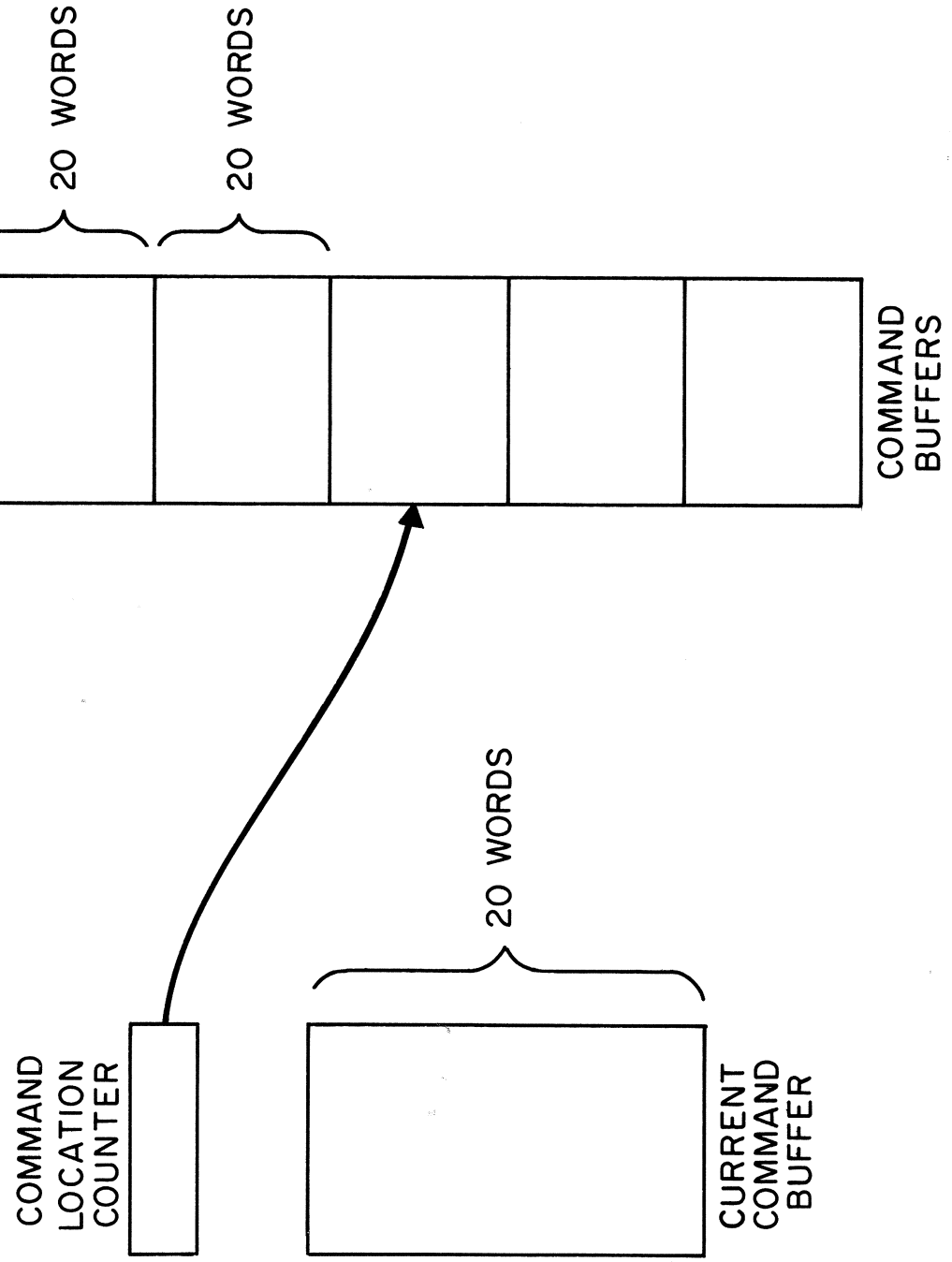
Method

In order to build a chain of commands, the BCD name of each command and its arguments must first exist in a fenced vector. The vector for each desired command is then moved into a command buffer within the supervisor and entered into its relative location within the command list (CLS) by the supervisor routine SETCLS. The relative location of the first command to be executed in the command list is entered into the command location counter (CLC) and the length of the command chain is entered into the supervisor by SETCLC.

Execution of the chain is initiated and continued by calls to CHNCOM. Commands can only be chained if each command terminates by calling CHNCOM so that the next command in the chain can be initiated. The calling sequence to CHNCOM specifies whether or not the calling program has a significant core image which might be useful to the next command in the chain. CHNCOM does some housekeeping before calling the next command in the chain: 1) sets memory bound to zero if no core image was specified in the calling sequence, 2) sets the instruction location counter to be the word following the calling sequence to CHNCOM, 3) increments CLC by 1, and 4) moves the next command buffer into the current command buffer or calls DEAD or DORMNT if no command remains in the chain.

Restrictions

A command list must be .LE. 5 commands.
Each command buffer with fence must be .LE. 20 words.



COMMAND MECHANISM

Usage

To enter a command in the command list or command buffer:
As supervisor or library entry:

```
        TSX  SETCLS,4      optional (TIA   =HSETCLS)
        PZE  TAB,,'n'
        ...
TAB     BCI  1,command
        BCI  1, arg1
        ...  ...
        OCT  777777777777
```

As library subroutine:
MAD or FORTRAN:

```
EXECUTE  SCLS. (TAB,N)
```

SETCLS moves 20 words from TAB into the Nth command buffer in the command list, or into the current command buffer if N is 0. A call to SETCLS with N = 0, does not initiate a command. A call to NEXCOM or XECOM is required to initiate the command.

SCLS interprets MAD and FORTRAN calling sequences which specify backward arrays and moves the words from TAB only to and including the fence into the command list.

TAB is the location of the fenced command table (.LE. 20 words) containing the command and its arguments in BCD(right justified and blank padded). The fence is interpreted by the command and SCLS not by SETCLS.

N & n specify the position within the command list (.LE. 5). N = 0 specifies the current command buffer.

To retrieve a command from the command list or current command buffer:
As supervisor or library entry:

```
        TSX  GETCLS,4      optional (TIA   =HGETCLS)
        PZE  BUFF,,'n'
```

As library subroutine:
MAD or FORTRAN:

```
EXECUTE GCLS. (BUFF,N)
```

GETCLS moves 20 words from the nth command buffer of the command list into locations beginning at BUFF.

GCLS interprets MAD or FORTRAN calling sequences, calls GETCLS and stores the command buffer backwards in BUFF. Only the words to and including the fence are moved into BUFF.

BUFF must be at least 20 words long for GETCLS.

To set the command location counter:
As a supervisor or library entry:

```
CLA  A
TSX  SETCLC,4      optional (TIA   =HSETCLC)
```

As a library subroutine:

MAD or FORTRAN:

```
EXECUTE  SCLC. (M,N)
```

A contains a word of the form PZE m,,n. Both SETCLC and SCLC set the command location counter to m and the number of the last command in the chain to n.

M or m is the number of the command in the command list which is the next to be executed. (m .LE. 5).

N or n is the number of the last command in the command list. (n .LE. 5).

To query the command location counter:
As supervisor or library entry:

```
TSX  GETCLC,4      optional (TIA   =HGETCLC)
STØ  A
```

As library subroutine:
MAD or FORTRAN

```
A = GCLC (M,N)
```

M will be set to the value of the command location counter i.e., the position within the command list of the next command to be executed.(m .LE. 5).

N will be set to the position of the last command in the command list.(n .LE. 5).

A will be set to a word of the form PZE m,,n.

To initiate or continue a chain:
As supervisor entry:

```
TSX  CHNCOM,4    (TIA   =HCHNCOM)
PZE  'j'
```

As library subroutine:
MAD or FORTRAN:

```
EXECUTE CHNCOM (J)
```

```
FAP:  CAL  = 'j'          or TSX  CHNCOM,4
      TSX  CHNCOM,4          PZE    'j'
```

J or j — j=0 specifies to CHNCOM that no core image is available for the next command. j=1 means that a core image is available and may be used by the next command.

CHNCOM — determines whether or not another command exists in the chain. If one exists, it is initiated. If no chain exists; DORMNT is called if j=1, DEAD is called if j=0.

Identification

Fetch a current command argument
GETCOM, COMARG

Purpose

To extract the Nth argument from the current command buffer.

Usage

As supervisor or library entry:

```
TSX  GETCOM,4      optional (TIA   =HGETCOM)
PZE  'n'
```

GETCOM returns, in the logical AC, the Nth argument of the user's latest command, i.e., of the current command buffer. The command itself is number 0. The arguments may be numbered 1-19, including the fence.

As library subroutine:
MAD, FORTRAN or FAP:

```
A  =  COMARG.(N)
A  =  COMARG.(N,B)
EXECUTE  COMARG.(N,B)
```

The Nth argument of the current command buffer is transferred to A and/or B.

Identification

Trace of Subroutine Calls.
ERROR

Purpose

ERROR is a subprogram which may be called by FAP, MAD, or FORTRAN programs in order to trace backwards to the main subprogram through the most recently executed chain of subroutine calls.

Restrictions

If FAP subprograms are used, they should include the linkage director and the instruction to save the contents of index register 4 must be included in the first twenty instructions of the subprogram.

Each subprogram executed must have at least one argument.

If ERROR is unable to complete the trace, the following message is printed and control is returned to the calling program.

```
TRACE FAILURE IN 'sub'
EXIT FROM ERROR
```

Usage

MAD, FORTRAN, or FAP:

```
ERROR.(MESS)
```

MESS is a BCD fenced message of .LE. 132 characters which will be printed on the user's console when ERROR is entered.

ERROR will trace back to the main program through the last subroutine calls and print comments of the following type and then return control to the calling program.

```
C(MESS)
ENTRY  ERROR   CALLED  BY 'sub1'
ENTRY  'sub1'  CALLED  BY 'sub2'
.
.
.
ENTRY  'subn'  CALLED  BY (MAIN)
EXIT   FROM    ERROR
```

Identification

BCD or spread-octal to binary
BCDEC, BCOCT

Purpose

To convert the BCD or spread-octal representation of an integer to the equivalent binary integer.

Usage

BCD to binary:
As library subroutine:

```
FORTRAN:  EQUIVALENCE (XNUM, NUM)
          XNUM = BCDEC (X)

   MAD:   NUM = BCDEC. (X)

   FAP:   TSX    BCDEC,4
          PZE    X
          STO    NUM
```

X is the location of the BCD word to be converted. X is assumed to be a BCD decimal integer and leading blanks and signs are ignored.

NUM and the AC will contain the right-justified binary integer equivalent to the absolute value of X.

Spread-octal to binary:
As library subroutine:

```
FORTRAN:  EQUIVALENCE(XNUM,NUM)
          XNUM = BCOCT(X)

   MAD:   NUM = BCOCT.(X)

   FAP:   TSX   BCOCT,4
          PZE   X
          STO   NUM
```

X is the location of the spread-octal word to be converted. X is assumed to be a BCD octal integer and leading blanks and sign are ignored.

NUM and the AC will contain the right-justified binary integer equivalent to the absolute value of X.

Identification

Binary to BCD
DEFBC, DELBC, DERBC

Purpose

To convert a binary integer to BCD with leading zeros.

Usage

As library subroutine:

MAD or FORTRAN:

```
A = DEFBC.(K)
A = DELBC.(K)
A = DERBC.(K)
```

A will contain a BCD decimal number (modulo 999999), right-justified and zero padded.

DEFBC converts the full 35 bit word (sign is ignored) K into a BCD decimal number.

DELBC converts the left half of K (sign is ignored) into a decimal BCD number.

DERBC converts the right half of K into a decimal BCD number.

Identification

Binary to spread-octal
OCABC, OCDBC, OCLBC, OCRBC

Purpose

To convert binary fields to spread-octal which is suitable for printing.

Usage

As library subroutine:

MAD or FORTRAN:

```
A = OCABC.(X)
A = OCDBC.(X)
A = OCLBC.(X)
A = OCRBC.(X)
```

X contains the binary number to be converted

A will contain the converted value in spread octal, i. e., six bits for each octal digit (0-7).

OCABC converts the address field of X to 5 digits with leading blank.

OCDBC converts the decrement field of X to 5 digits with leading blank.

OCLBC converts the left half of X to 6 digits.

OCRBC converts the right half of X to 6 digits.

Identification

Justification and padding
BŻEL,ZEL, LJUST, RJUST

Purpose

To allow the user to left or right justify and/or to interchange blanks and zeros.

Usage

Justification library subroutines:

```
    FAP:  TSX  LJUST,4          TSX  RJUST,4
          PŻE  WORD             PŻE  WORD
          STØ  X                STØ  X

    MAD:  X = LJUST.(WORD)      X = RJUST.(WORD)
FORTRAN:  I = LJUST (WORD)      I = RJUST (WORD)
```

WORD contains the word to be justified. Upon return the AC contains the adjusted word.

LJUST by left shifting, leading blanks are replaced by trailing blanks. Leading zeros are not replaced. If the word is all blanks, "bbbbb*" is returned.

RJUST by right shifting, trailing blanks are replaced by leading blanks. If the word is all blanks, "bbbbb*" is returned.

Interchange leading zero and blanks, library subroutine:

MAD, FORTRAN or FAP:

```
A = BŻEL (B)        A = ŻEL (B)
```

B contains the word to be modified. Upon return, the AC and A will contain the modified word.

BŻEL replaces leading zeros with blanks. If B is zero, "bbbbb0" will be returned.

ŻEL replaces leading blanks with zeros. If B is all blanks, "00000b" will be returned.

Identification

General purpose input/output conversion
(IOH), (RTN), (FIL), IOHSIZ, STQUO

Purpose

General purpose conversion of BCD to binary or binary to BCD for input or output, respectively, according to a format and data list.

Reference

CC 186 FORTRAN and MAD Format Specifications Spall

Method

A standard 22 word buffer is assumed to be located at (77742)8. Presetting of certain upper core locations indicates whether input or output conversion is desired. If input is indicated, the contents of the buffer is converted according to the specified format and stored in the locations specified by the list. If output is indicated, data from the list specification is converted according to the format and stored in the buffer.

The actual I/O data transmission to or from the buffer must be performed by an I/O routine. Appropriate calling sequences to the I/O routines and (IOH) are compiled by MAD and FORTRAN for any read/write statements which specify a format. Data or format errors cause (IOH) to call RECOUP.

Usage

Output, binary to BCD:

Fortran:

```
        TSX  USRSTH,4
        PZE  FORMAT,,SWITCH
RTN      .
         .
        LDQ  SYMBOL,t
        STR
         .
         .
        TSX  (FIL),4
         .
         .
USRSTH  Set upper
          core locs
        TRA* (IOH)
OUT      .
         .
        TRA  2,4
```

MAD:

```
        TSX  USRSTH,4
        PZE  FORMAT,,SWT
RTN      .
         .
        STR  FIRST,,LAST
         .
         .
         .
        STR  0
         .
         .
USRSTH Set upper
         core locs
       TRA*  (IOH)
OUT      .
         .
       TRA  2,4
```

Input, BCD to binary:

Fortran:

```
        TSX  USRTSH,4
        PZE  FORMAT,,SWITCH
RTN     .
        .
        .
        STR
        STQ  SYMBOL,t
        .
        .
        .
        TSX  (RTN),4
        .
        .
        .
USRTSH  Set upper core
        TRA* (IOH)
  IN    .
        .
        .
        TRA  1,4
```

MAD:

```
        TSX  USRTSH,4
        PZE  FORMAT,,SWT
RTN     .
        .
        .
        .
        STR  FIRST,,LAST
        .
        .
        .
        STR  0
        .
        .
        .
USRTSH  Set upper core
        TRA* (IOH)
  IN    .
        .
        .
        TRA  1,4
```

FORMAT is the beginning location of the desired format.

SWITCH is zero if the format is enclosed in parentheses and stored backwards in core. SWITCH is non zero if the format is enclosed in parentheses and stored forward in core (e.g. BCI).

SWT is zero if format is forward. SWT is one, if the format is stored backward.

SYMBOL,t locates the variable to be converted. A loop may be included here for arrays or a series of LDQ, STR. After each variable is converted by (IOH), return is made following the STR in order to find the next variable to be converted.

FIRST is the starting location of the list.

LAST is the final location of the list. LAST may be lower in core than FIRST. If the list is of length one, LAST is zero.

(FIL) is called to indicate that all the output data has been converted and the current buffer should be truncated.

STR 0 terminates the list in a MAD call.

(RTN) is called upon completion of the input data list. It restores the original (IOH) initialization (i.e., trap cells).

USRSTH is the user's output transmission program. It must initialize the appropriate upper core locations before calling (IOH). After each line image is completed in the buffer, (IOH) will return to OUT with index register 4 set in such a way that "CLA 1,4" will put into the address of the AC the location of the buffer and in the decrement of the AC the number of words in the buffer.

For MAD programs, USRSTH will be .TAPWR and for FORTRAN programs it will be (STH) or (STHM).

USRTSH is the user's input transmission program. It must initialize the appropriate upper core locations, read in the first buffer load and then call (IOH). Control is then returned to FIRST and the first data word is converted and placed in the MQ upon entry to (IOH) by way of the STR. Successive words are converted into the MQ by subsequent STR's.

An STR following depletion of the input buffer causes (IOH) to return control to IN in order to read the next record.

For MAD programs, USRTSH will be .TAPRD and for FORTRAN it will be (TSH) or (TSHM).

IOHSIZ:

MAD, FAP, or FORTRAN

```
TSX  IOHSIZ,4
PZE  N
```

N containing non-zero indicates to (IOH) that the diagnostic that "the field width of the format has been exceeded" should be suppressed. An N of zero resets the normal mode of printing the diagnostic.

STQUO:

MAD, FAP, or FORTRAN

```
TSX  STQUO,4
```

The next I/O statement will be initiated without resetting the buffer, that is, the line pointer is left where it was at the conclusion of the last I/O call. This is normally used in conjunction with the N modifier. (CC-186 for description of formats).

The following locations must be set before (IOH) is called for conversion:

(77737)8	address	Location of subroutine that (IOH) calls for input or output. This address corresponds to INPUT or OUTPUT.
	Tag	0
	decrement	+1 if format stored backwards -1 if format stored forwards
	prefix	TXL if FORTRAN type call TXH if MAD type call
(77740)8	address	location of first word of format statement.
	tag	0
	decrement	user's index register 4 on initial entry to the input-output subroutine.
	prefix	TXL for on-line printer TXH for all other I/O
(77741)8	address	scratch area for (IOH) to use for output. The number of words in the output record is stored here.
	tag	0
	decrement	maximum number of columns available in input or output record (may not exceed 132).
	prefix	TXL for output (binary to BCD). TXH for input (BCD to binary).
(77742)8		The beginning of a 22 word buffer from which BCD data is converted to binary or into which BCD data is placed after binary to BCD conversion.
(77771)8	address	location of symbol table (if any)
0	address	the address of RTN as RTN is the location to which programs should return after calling (IOH).

Identification

Pack and unpack, single characters to full words
PAKR, PAKL, UNPAKR, UNPAKL

Purpose

The characters from an array containing one 6-bit character per word may be packed into an array containing six 6-bit characters per word. Conversely the packed array may be unpacked.

Usage

As a library subroutine:

```
MAD:  NUM=   PAKR. (CR...CR(I), WD...WD(J) )
      NUM=   PAKL. (CL...CL(I), WD...WD(J) )
      NUMB=  UNPAKR. (WD...WD(J), CR...CR(I) )
      NUMB=  UNPAKL. (WD...WD(J), CL...CL(I) )

FAP:  TSX  PAKR,4          TSX  PAKL,4
      TIX  CR,,CR-'i'      TIX  CL,,CL-'i'
      TIX  WD,,WD-'j'      TIX  WD,,WD-'j'
      STO  NUM             STO  NUM

      TSX  UNPAKR,4        TSX  UNPAKL,4
      TIX  WD,,WD-'j'      TIX  WD,,WD-'j'
      TIX  CR,,CR-'i'      TIX  CL,,CL-'i'
      STO  NUMB            STO  NUMB
```

CR is an array of words, each containing one right adjusted 6-bit character.

CL is an array of words, each containing one left adjusted 6-bit character.

WD is an array of words containing six 6 bit characters.If the last word is not full it is left adjusted and blank padded.

NUM and the AC will contain the number of words entered into WD.

NUMB and the AC will contain the number of words (i.e., characters) entered into CR or CL.

PAKR or PAKL will take successive characters from (right adjusted) or CL (left adjusted), respectively, CR and pack them into successive words in WD.

UNPAKR or UNPAKL will unpack the words in WD as one character per word with blank padding into CR (right adjusted) or CL (left adjusted), respectively.

Identification

Fortran integers to/from full word integers.
FINT, MINT

Purpose

Fortran II integers occupy the decrement portion of a computer word. Most other systems, including MAD, use full word integers. These two routines will convert from decrement to full word or from full word to decrement.

Usage

As a library subroutine:

```
Fortran:  EQUIVALENCE (A,J)
          A = FINT (I)              I = MINT (J)

   MAD:   J= FINT. (I)              I = MINT. (J)
          INTEGER J, FINT., I       INTEGER I, MINT., J

   FAP:   TSX  FINT,4               TSX  MINT,4
          PZE  I                    PZE  J
          STO  J                    STO  I
```

I is a full word (MAD) integer.

J is a decrement (FORTRAN) integer.

A is equivalent to J.

FINT converts from full word to decrement integer. If the integer is too large, the following message will be printed and the integer will be taken modulo 32768.

```
MAD INTEGER EXCEEDS 32767
```

MINT converts from decrement integer to full word.

Identification

Complement, OR, and AND functions
COM, ORA, ANA

Purpose

COM executes the machine instruction COM, ORA executes ORA, and ANA executes ANA.

Usage

```
FORTRAN:  COMA = COM (A)
          ORABC = ORA (B,C)
          ANADE = ANA (D,E)

MAD:      COMA = COM.(A)
          ORABC = ORA.(B,C)
          ANADE = ANA.(D,E)
```

On return from COM, the arithmetic AC will contain the complement ('one's complement') of A.

On return from ORA, the arithmetic AC will contain the result of 'oring' B and C. On return from ANA, the arithmetic AC will contain the result of 'anding' D and E.

Identification

Internal conversion of stored data according to a format.
DECODE, ENCODE

Purpose

To encode (to BCD representation) or decode (from BCD representation) data in machine representation, according to a MAD/FORTRAN format statement.

Usage

As library subroutine:

```
FORTRAN:    A = DECODE (FMT, TEXT, LIST)
MAD:        A = DECODE. (FMT, TEXT, LIST)
FAP:    TSX   $DECODE,4
        PZE   FMT
        PZE   TEXT
        PZE   ARG1
         .
         .
        PZE   ARGN
        STO   A
```

The FAP call may also simulate FORTRAN and MAD calls:

FORTRAN

```
TSX   $DECODE,4
TSX   FMT
TSX   TEXT
TSX   ARG1
 .
 .
TSX   ARGN
STO   A
```

MAD

```
TSX   $DECODE,4
TXH   FMT
TXH   TEXT
TXH   ARG1
 .
 .
TSX   ARGK(j),,ARGK(m)
TXH   ARGN
STO   A
```

where

FMT refers to the format statement to be used in converting the data.

1) In a FORTRAN (or FORTRAN simulated) call, it may be a setting from a call to SETFMT. If SETFMT is not used it should be the H-specification of the format statement, e.g.,

```
A =  DECODE(5H(513),TEXT,list)
```

The format is expected stored in reverse order with FMT pointing to the first location (normal FORTRAN compilation).

2) In a MAD (or MAD simulated) call, FMT should point to the first location of the format statement, the format being stored in reverse order (normal MAD compilation).
3) In a FAP call, where the prefixes are PZE's, FMT should point to the first location of the format statement, the format being stored forwards.

TEXT is an array which contains the BCD to be decoded, or into which BCD information will be stored after encoding. If the call was from a FORTRAN or MAD program (or FORTRAN or MAD simulated program), the array is stored backwards. Otherwise the array is stored forwards.
NOTE: In calls to DECODE, each new item of TEXT must start in a new machine location. Due to the way records are transmitted, the memory bound should be at least 21 words past the start of the last record. This is ensured with normal loading procedure.

LIST is a list of arguments. It can be any length and may be single variables, subscripted or not, or MAD lists e.g. A(i)...A(n).
Not allowed are FORTRAN implied 'DO' loops, and FAP tagged variables.

A is a integer giving either:
1) For ENCODE, the length of the resultant text.
2) For DECODE, the number of words picked up from TEXT in order to fill the list.
"A" will be zero if the calling sequence is not recognized by COLT or if no arguments are specified in LIST.

(END)

entification

riable length calling sequence processor
LT, SELAR, MDL

rpose

provide one routine which general purpose subroutines ght call to interpret variable-length calling sequences nerated by MAD, FORTRAN or FAP. This routine will termine the type of calling-program and the number and pe of arguments in the calling-program.

age

cal definitions:
Program is the routine which is calling COLT.
Calling-program is the routine which is calling the program.

LT, as a library subroutine:

```
        TSX  COLT,4
        PZE  IR4
```

IR4 contains, in the decrement, the contents of index register 4 at the time the program was called.

AC upon return, will contain, in the decrement, the number of arguments in the calling sequence to the program and, in the address, a code specifying the type of the calling-program. The codes are:

0 unknown, or no arguments
1 FAP
2 FORTRAN
3 MAD

Index register 4 will contain the two's complement of the location in the calling-program to which the program should return, i.e., the location following the calling sequence.

SELAR; what type of argument:

```
          CAL*  COLT
          STA   SELAR
          CAL   ARG
          AXT   RETURN,1
   SELAR  TRA   **
           .
           .
  RETURN   ...
```

ARG is the argument from the calling-program which is to be examined.

RETURN is the location to which SELAR is to return.

SELAR will place a code in index register 1 indicating the type of argument

0 unknown
1 FAP
2 FORTRAN
3 MAD single argument
4 MAD list with TIX
5 MAD list with STR

AC upon return, will contain in the left half the significant part of the argument (TXH, TSX etc.)

MDL, MAD list processor:

```
          CAL*  COLT
          ARS   18
          STA   MDL
          CAL   ARG
    MDL   TSX   **,1
```

ARG is the MAD list argument from the calling-program to be examined.

AC upon return will contain:

address - number of words in the list
decrement - the increment to be used in indexing (+1 or -1)
prefix - TXH (plus) if the list is forward or TXL (minus) if the list is backward.

Identification

Determine type of calling program and FILNAM
GNAM

Purpose

To provide a routine which general purpose routines might call to determine the type of calling-program and a file name if one be requested.

Usage

Local definitions:
Program is the routine which is calling GNAM.
Calling-program is the routine which is calling the program.

As library subroutine:

```
      TSX   GNAM,4
      PZE   IR4
     -OPN   FILNAM-
```

OPN may be PZE, TXH, or TSX.

IR4 contains, in the decrement, the contents of index register 4 at the time the program was called.

FILNAM (optional) is the first of two consecutive locations in which the file name will be stored (forward if PZE, backward if TXH). The file name is assumed to be located by the first argument in the calling sequence to the program.

AC will contain a code, right-adjusted integer, specifying the type of the calling-program.

0 unknown
1 FAP
2 FORTRAN
3 MAD

Identification

List transmission
MOVE1, MOVE2, MOVE3

Purpose

To tramsmit data specified by an argument list from the calling program to the called program or transmit any list specified data from one place to another. The argument lists may be MAD, FORTRAN or FAP and the data arrays may be forward or backward.

Usage

As library subroutine:

```
        TSX   MOVE1,4
        OP    BGDATA,, -ENDATA-
        OPN
        TSX   MOVE2,4
        OP    BEGLST,,-ENDLST
ALPHA   OPN
        STR   DATOUT,,LSTOUT
 BETA   OPN
        TSX   MOVE3,4
```

OP may be TSX, TXH, PZE, TIX or STR. The decrement argument may be used only with TIX and STR.
TSX and TXH signify a single argument or backward array base.
PZE signifies a single argument or forward array base.
TIX and STR signify an argument list whose beginning location is specified in the address and whose ending location is specified in the decrement. Note that the list may be forward or backward depending on whether the address is less than or greater than the decrement.

BGDATA is the beginning location of a block of core in the program in which the data will be stored.

ENDATA (specified only when OP is TIX or STR) is the ending location of the data block.

BEGLST is the begining location of the list which specifies the data to be moved.

ENDLST (specified only when OP is TIX or STR) is the ending location of the argument list.

Identification

Name a format or file name
SETFMT, SETNAM

Purpose

To simplify FORTRAN calls to the library disk routines by providing formats and file names with labels which then may be used in calling sequences to library routines.

Usage

FORTRAN: CALL SETNAM (FILNAM,12H NAME1 NAME2)
CALL SETFMT (FORMAT, nH (.....))

FILNAM is the location which is to contain a pointer to the actual file name NAME1 NAME2.
NAME1 NAME2 are the actual primary and secondary names of the file, right-justified.

FORMAT is the location which is to contain a pointer to the actual format.

pointer is a word which contains in the address portion the address of the first word of either the format or file name. The left half will contain a TSX if the call was made by a Fortran or FAP program or a TXH if the call was made by a MAD program. Bit positions 12-17 will contain (77)8.

These two routines allow the library disk routines to be called with FILNAM and FORMAT as arguments instead of the actual BCD information.

i. e., CALL DWRITE (FILNAM, FORMAT, LIST)
instead of, CALL DWRITE (12H NAME1 NAME2, nH(....), LIST)

Revised: 5/9/66

Identification *

Get the date and time of day
GETIME, GETTM, GTDYTM

Purpose *

To provide the user with the current date and time of day. The formats in which information is returned differ; they are described under Usage.

Method

The time is computed by using values from the interval timer to update the last reading of the chronolog clock (last time someone logged in). The interval timer is incremented sixty times a second.

Usage

1) GETIME

As supervisor or library entry:

```
TSX  GETIME,4     optional (TIA   =HGETIME)
SLW  TIME
STQ  DATE
```

Upon return, the logical AC will contain the time of day as an integer in 60ths of a second. The MQ will contain the date in BCD as "MMDDYY".

2) GETTM

As library subroutine:

MAD, FORTRAN or FAP
CALL GETTM (DATE, TIME)

DATE is the location in which the date will be stored in the BCD form "MM/DDb".

TIME is the location in which the time will be stored in the BCD form "HHMM.M" . HH is the hour of the day (0-23) and MM.M is the minutes after the hour to one tenth of a minute (0-59.9).

3) GTDYTM

As supervisor or library entry (systems numbered MAC5A1, CTR5A1, and above):

```
MAD

    TIME = GTDYTM.(0)

FAP

    TSX    GTDYTM,4    optional (TIA =HGTDYTM)
    SLW    TIME
```

TIME is the location in which the date and time will be stored in (binary) "file system format". See Section AD.2 for the description of date and time last modified U.F.D./M.F.D. items.

(END)

Identification

Timer interrupt and stop watch
TIMER, JOBTM, RSCLCK, STOPCL, KILLTR, TIMLFT, RSTRTN

Purpose

To provide the user with the ability to time parts of a program and/or set a time limit on parts of a program.

Method

The foreground supervisor normally runs with the clock function turned off. A call to any of these time routines will turn the clock on. The interval timer is then used to time the function as specified by the user. The interval time is incremented sixty times a second so that all integer times will be in 60ths of a second.

Restrictions

The simulated clock (core B interval timer cell) may cause an interrupt only every 200 milliseconds because that is how often it is updated by the supervisor, but it will be incremented every 60th of a second. The execution of any command (e.g., MACRO or CHAIN) will turn the clock function off. The job time is initiated to 73 minutes upon the first call to the timer rather than at the actual beginning of the job. CLOCON and CLOCOF should not be used if the timer routines are being used.

Usage

All of the entries may be called by MAD, FAP or FORTRAN. If the prefix to the argument is non-zero (i.e., MAD or TXH in FAP) the integer variable will be full word integers. If the prefix is zero, the integers will be in the decrement.

To initialize or reset the stop watch to zero:

```
EXECUTE  RSCLCK.
```

To read the elapsed execution time since the last call to RSCLCK:

```
EXECUTE STOPCL. (J)
```

J is an integer variable which will contain the time used since the last call to RSCLCK in 60ths of a second.

To read the elapsed execution time since the first initialization of the clock:

EXECUTE JOBTM. (J)

J is an integer variable which will contain the elapsed execution time since the first call to one of the timer routines in 60ths of a second.

To initialize an elapsed time interrupt, i.e., an alarm clock:

```
FORTRAN:  ASSIGN  S TO N
          CALL  TIMER (J,N)

    MAD:  EXECUTE  TIMER. (J,S)

    FAP:  TSX  TIMER,4
          PZE  J
PZE  S
```

J is an integer variable specifying the length of time in 60ths of a second that the clock may run before interrupting.

S is the statement (location) to which control should transfer when the time, to the nearest 200 milleseconds, has elapsed.

TIMER Only nine calls to TIMER may be stacked. Any more than nine will be ignored.

To continue the instructions which were interrupted by the alarm clock:

EXECUTE RSTRTN.

To void the last setting of the alarm clock:

EXECUTE KILLTR.

To provide foreground/background compatibility to job time remaining:

EXECUTE TIMLFT. (J)

J is an integer variable which will contain the amount of time in 60ths of a second which the job has remaining to run. The first call to any of the timer routines will initialize the job run time to 72 hrs. The job run time for background jobs is taken from the identification card.

Revised: 9/24/65

Identification

CLOCK function
CLOCON, CLOCOF

Purpose

To cause the supervisor to simulate the interval timer for the user.

Restriction *

These routines should not be used if one of the following * routines is to be used:
TIMER, JOBTM, RSCLCK, STOPCL, KILLTR, TIMLFT, RSTRTN.

Method

If the clock function is on, the B core interval timer cell will be updated by the supervisor at each time burst (200 milliseconds). It will be updated by the elapsed time (running time, not real time) in 60ths of a second. Any B-core interval timer overflow trap will be interpreted at the time of the update. The system normally runs with the clock function off. Any new command will turn the clock * function off.

Usage

Turn the clock function on:
As supervisor or library entry:

```
TSX  CLOCON,4     optional (TIA  =HCLOCON)
```

Turn the clock function off:
As supervisor or library entry:

```
TSX  CLOCOF,4     optional (TIA  =HCLOCOF)
```

Revised: 2/14/66

Identification

List of miscellaneous library subroutines:

The following is a list of miscellaneous TSLIB1 subroutines. Further information or one page write-ups may be obtained from the consultants.

DFAD	DFSB	DFMP	DCEXIT	DFDP	SFDP
IOSET	IOPAR	IOEND	IOSCP	IOITR	
(SLO)	(SLI)				*
.01300	.01301	.01311	.03310	.03311	*
MAX0	MAX1	XMAX0	XMAX1		
MIN0	MIN1	XMIN0	XMIN1		
MOD	XMOD				
XSIGN	SIGN	XLOC			
DIM	XDIM	INT	XINT	XFIX	
EXP	EXP(1	EXP(2	EXP(3		
ACOS	ASIN	ATAN	ATN	COS	SIN
LOG	SQRT	SQR	TAN	COT	TANH
SIMCS	XSMEQ	XSIMEQ	XDETRM	XDTRM	DETCS
FLIP	RANNO	SETU	INDV	DPNV	

(END)

Identification

Floating-Point Overflow and Underflow
(FPT)

Purpose

To process the underflows and overflows which may occur during the execution of floating-point operations. Underflows are set to zero, the lowest possible absolute number, and overflows halt execution.

Method

An underflow or overflow automatically causes a transfer of control to location 8 with the location of the instruction following the offending instruction stored in the address of location 0. A spill code is stored in the decrement of 0. If an underflow condition exists, (FPT) places zero in the proper register and transfers back to the instruction following the floating-point instruction which caused the underflow.

If an overflow condition exists, (FPT) proceeds to do the following:

1. It prints on one line the comment:

   FLO-POINT OV-FLOW AT OCT LOC xxxxx ABS,
   or xxxxx RFL, PROG name SPILL xxxxx

2. It then calls the library subprogram ERROR, which prints an error traceback, if possible, enabling the user to determine the control path leading to the error.

3. After this information is complete, EXIT is called.

The spill codes are produced as follows:

Operation	AC	MQ	Decr. Portion Bits 12	14	15	16	17	Spill Code in octal
Add, Subtract		Underflow	0	0	0	0	1	01
Multiply, and Round	Underflow	Underflow	0	0	0	1	1	03
	Overflow		0	0	1	1	0	06
	Overflow	Overflow	0	0	1	1	1	07
Divide		Underflow	0	1	0	0	1	11
	Underflow		0	1	0	1	0	12

Underflow	Underflow	0	1	0	1	1		13
	Overflow	0	1	1	0	1		15
Effective Address of a Double-Precision Instruction is Odd, (except DST)		1						40

A transfer to (FPT) is placed in register 8 by .SETUP, which is automatically compiled into every FORTRAN and MAD main program compiled at the Computation Center.

Revised: 9/24/65

Identification

Log in
LOGIN

Purpose

To logout any previous user of this console, identify the new user, and initialize any system bookkeeping.

Usage

user:	LOGIN probno name
response:	W TIME,
	PASSWORD
user:	private password
response:	probno progno LOGGED IN date1 time1 from IDCODE
	message of the day
	CTSS BEING USED IS id
	LAST LOGOUT WAS date3 time3
probno	is the user's problem number.
name	is the user's last name of which only the last six characters are used.
TIME	is the current time of day. HHMM.M
password	after typing PASSWORD, the system turns off the printing mechanism, if possible, so that the user may type his password with privacy.
progno	is the programmer number equivalent to probno name.
date1	is today's date. MM/DD/YY
time1	is current time of day. HHMM.M
IDCODE	is the console identification code
message	of the day (optional) gives some pertinent information about the system.
id	is the name of the version of CTSS being used.
date2	time2 are date and time when used time began accumulating.
date3	time3 are date and time of user's last LOGOUT.

If the user has 'urgent mail' from the system, login prints

YOU HAVE URGENT MAIL

before the response:

probno progno LOGGED IN date1 time1

The user should print the file URGENT MAIL immediately after he is logged in.

Error messages

1. _____NOT PROBLEM NUMBER.
   LOGIN COMMAND INCORRECT.
   The user has typed a problem number which is not possible.

2. _____NOT FOUND IN DIRECTORY
   LOGIN COMMAND INCORRECT.
   The combination of the user's problem number, name, and password has not been found in accounting files. This printout contains what is not found (if it is the password, the printout says PASSWORD).

3. probno name ALREADY LOGGED IN.
   LOGIN COMMAND INCORRECT.
   User is already logged in on another console.

4. PARTY LINE GROUP NUMBER_____WRONG.
   LOGIN COMMAND INCORRECT.
   User's party line group, as found in accounting files, exceeds the maximum party group number (System or administrator error).

5. HANG UP
   This is the result of one of the following conditions.
   a. Primary lines in user's party line group are filled, and user is not allowed to be standby.

   b. User's party line group number is zero, and user is not allowed to be standby. In other words this user may not be logged in at all.

   c. Maximum number of users for time-sharing has been reached, the user's party line group is either zero, or the primary lines for that group have been filled.

   d. Maximum number of users for time-sharing has been reached, and there are no standby users to be logged out.

6. USER MAY NOT USE THIS CONSOLE.
   LOGIN COMMAND INCORRECT.
   According to the user's unit group, he is attempting to log in at a console he is not allowed to use.

7. _____UNIT GROUP NOT FOUND.
   User's unit group number is not found in accounting files.

8. ALLOTTED TIME EXCEEDED FOR THIS SHIFT - NO LOGIN.

The user's allotted time for this shift has been exceeded by his used time. *

9. IF YOU LOGIN YOUR FIB JOB WILL BE LOST.
DO YOU WANT TO LOGIN,
A FIB job belonging to this user is currently running. If he types YES, his FIB job will be logged out and he will be logged in. If he types NO, he will not be logged in, and his FIB job will continue to run.

10. USER NOT IN MFD. *
The problem number, programmer number combination is not in the master file directory.

11. FILE CANNOT BE OPENED. *
There is a disk error. See the operations staff.

12. DISK ERROR WHILE READING FILE _____. *
There is a disk error. See the operations staff.

13. DISK ERROR WHILE WRITING FILE _____. *
There is a disk error. See the operations staff.

14. FORMAT ERROR IN TIMUSD FILE. *
There is a disk error. See operations staff.

15. SYSTEM FILES MUST HAVE PROBLEM NUMBER FOR GROUP LEADER, SEE OPERATIONS STAFF. *
The file ALLOCT TIMACC must be in the system files. See the operations staff.

16. ERROR IN SYSTEM FILES. *
There is a disk error. See the operations staff.

17. TIME ACCOUNTING FILE LOCKED. *
One of the time accounting files is interlocked. See the operations staff.

Party Groups

The party line group is a way of insuring that at any time during time sharing, a specified number of people in each party group can be logged in.

Each user is assigned to a party line group each party line group is assigned a specified number of primary lines. A record of each user's party line group number is kept in his entry in the time-accounting file. A record of the number of lines assigned to each party line group is also kept in the time-accounting fle.

For a party line group having N primary lines, the first N people belonging to that party line group who log in will be assigned primary lines. The next person belonging to that party line group who attempts to log in is allowed to do so if the maximum number of CTSS users has not been reached. This user is logged in as a standby. Standby users may be logged out to allow a primary user to log in. An example

might clarify this.

Suppose there are 5 lines assigned to party group 1 and 3 lines assigned to party group 2. There may be a maximum of 8 users of CTSS. Four users belonging to party group 1 are logged in and one user belonging to group 2 is logged in. Users A,B, and E belong to party group 1. Users C, D, and F to 2. User A logs in and gets the last primary line of group 1, B logs in and is made standby. C logs in and gets second primary line of group 2. E tries to login. The number of users = 8, and all primary lines in group 1 are assigned so E is not allowed to login. F tries to login and the situation is the same as for E. D tries to login. The number of user = 8, but there are only 2 lines from D's party group assigned, so B is logged out to make room for D.

It should be noted that a user who is logged in as a standby can become primary while time-sharing, if a primary user in his group logs out.

Party line group zero has the notable line allotment of 0. That is, a user whose partyline group number is zero is always standby.

Revised: 7/30/66

Identification

Log out
LOGOUT, AUTOMATIC LOGOUT

Purpose

LOGOUT allows the user to terminate his use of the console without saving any machine conditions. The automatic logout * allows the system to terminate a given user or terminate all users for a system shutdown in such a way that each user may later RESUME his program from the point of interruption.

Usage

LOGOUT:

```
      user:  LOGOUT
  response:  W time
             probno progno LOGGED OUT date1 time1
             TOTAL TIME USED =  XX.X MIN.

      time   and time1 are the current time of day. HHMM.M

     date1   is today's date. MM/DD/YY

    probno   is the problem number

    progno   is the programmer number

      XX.X   is the time used in minutes.
```

AUTOMATIC LOGOUT:

```
  response:  WAIT
             AUTOMATIC LOGOUT
```

Automatic logout may be initiated by the system.
If the user is not in dead status, a saved file will be created called LOGOUT SAVED. The program may be resumed at some later time by:

```
RESUME LOGOUT

or CONTIN LOGOUT
```

(END)

Revised: 5/9/66

Identification

Foreground Initiated "Background" *
FIB, DELFIB, PRFIB

Purpose

The RUNCOM facility (AH.10.01) allows predescribed sequences of commands to be executed. The user of RUNCOM, however, must remain logged in and may not make any other use of his console until the completion of the sequence.

The FIB facility allows the user to specify files which are to be executed by RUNCOM when and only when the user is not logged in from a foreground console. The supervisor schedules a FIB job in the same scheduling queues as regular foreground jobs. (The FIB Monitor - a ficticious user - actually logs the FIB user in "over itself"; that is, on the line FIBMON had.)

Restrictions

The user must have a time quota allotted for FIB jobs (shift 5). A user's FIB job cannot be run while its donating user is logged in. A user who logs in during execution of one of his FIB jobs will cause that job to be automatically logged out. A user may have only one FIB job scheduled to run in any given two-hour period--but see "Batching", below. As one might expect, there is no way for FIB jobs to receive console input.

Usage

To initiate a FIB job:

FIB NAME1 -LIMIT- -TIME- -DAY-

NAME1 is the primary name of a file NAME1 BCD which is a list of the commands to be executed by RUNCOM as a "background" job.

LIMIT is the maximum execution time limit, in minutes, which the user wishes to place on the job. If LIMIT is not specified, a time limit will be set by FIB. No FIB job will be allowed to exceed a certain maximum time, which is currently set at 5 minutes (this is also the value used when no limit is specified). A FIB job which exceeds its time limit will be automatically logged out; it may be restarted by the user.

TIME and DAY specify a date and time (up to one month away) before which the job will not be run. TIME is expressed in "military time"(.GE. 0 .AND. .LE. 2359); DAY, of course, is .GE. 1 .AND. .LE. N, where N is the date of the last day of the month in which the command is issued. If a time earlier than "now" is specified, the command will assume that the next day (if TIME is less) or the next month (if DAY is less) is meant. A LIMIT must be given if a TIME is to be; a LIMIT and a TIME must be given if a DAY is to be; ordering of these arguments is fixed. If no pre-scheduling is specified, the current time will be used.

To delete a waiting FIB job:

DELFIB NAME1

To determine what FIB jobs the user has pending, for when they are scheduled, and what time limit has been placed on them: *

PRFIB

'FIBJOB FILE' will be searched for the user's jobs and the relevant information will be printed on the user's console.

Method

FIB jobs are run one at a time on a first-come-first-served basis. A FIB job is run in the same scheduling queues as foreground jobs but as the result of no console interaction, it moves to the lower priority queues. The donating user is logged in; the commands listed in the BCD file previously specified by the FIB command are executed by RUNCOM; and when the list is exhausted or the time limit is exceeded, the job (i.e., the donating user) is logged out and FIBMON logged back in. Calls to WRFLX(A) (which normally cause typing at the user's console) cause writing into a file, $$$FIB OUTPUT, in the user's file directory.

N.B. This file is in 12-bit mode and must be PRINTed accordingly.

Calls to DEAD or DORMNT will result in an automatic logout. Calls to the following subroutines will result in a Protection Mode Violation followed by an automatic logout:

ALLOW, ATTCON, FORBID, RDFLX, RDMESS, REDLIN, RELEAS, SET6 , SET12, SLAVE, SLEEP, SNDLIN, SNDLNA, and WRMESS.

If a FIB job is logged out for any reason, it must be restarted by the user. The FIB job running at system shutdown time will be run to completion or until it exceeds its time limit. If a FIB job is logged out because it exceeded its time limit it is logged out by ENDLOG so that as much as possible is saved.

The user cannot be logged in while his FIB job is running. If he is logged in when his FIB job's turn to run comes, the FIB job is passed over and the next FIB job is tried. The job that was passed over retains its relative position in the list of FIB jobs until it can be successfully logged in or until the user who initiated it deletes it. If the user's FIB job is running when he tries to log in, he will get this message:

```
IF YOU LOG IN YOUR FIB JOB WILL BE DELETED.
DO YOU WISH TO LOG IN,
```

If the user types 'yes', his FIB job will be automatically logged out, and LOGIN will continue to log him in. If he types 'no', he will not be logged in, and his FIB job will continue to run.

"Batching"

It is not desirable to allow any one user to monopolize FIB time by requesting several long jobs at once. However, if other jobs are not waiting it is not desirable to prohibit a user's running successive jobs. Therefore, the FIB command has been implemented as follows: a user may have only one job pre-scheduled to run in any given two-hour period in the system's FIBJOB FILE (which is written by the FIB command); but when FIBMON logs the user in for running a job, the entry corressponding to the job in FIBJOB FILE is removed before the job begins. If the job itself contains a non-pre-scheduled FIB command, then, that command would be acceptable, and, indeed, would be entered in FIBJOB FILE to be run after any pending jobs previously requested by other users. (Pre-scheduled jobs whose TIMEs have not yet arrived are skipped when the FIB Monitor looks for work.) Of course, if there are no other requests, the job would be run as soon as the current (calling) job terminates. The effect of all this is analogous the Background "express-run" batches, where only one job per user per batch is permitted; in FIB's case,however, the "next batch" is always starting.

(END)

Revised: 5/23/66

Identification

Query time and record quotas
TTPEEK

Purpose

TTPEEK will print the amount of time allotted and used for each shift and the number of records allotted and used for each storage device at the time the command is executed.

Usage

user: TTPEEK

response: W TIME

mo/date TIME 'TUSED'= tu *

SHIFT	MINUTES	*
ALLOTTED	USED	*
1 at1	ut1	
2 at2	ut2	
3 at3	ut3	
4 at4	ut4	
5 at5	ut5	

STORAGE

DEVICE	QUOTA	USED
DRUM	DRQ	DRU
DISK	DIQ	DIU
TAPE	TPQ	TPU

TIME is the current time of day. HHMM.M

mo/date are current month and date. *

tu is time used during current console session, * in minutes.

at1-5 is the time in minutes allotted to the user for shifts 1-5 this month. Shifts 1,2, and 3 are Monday through Friday; 9:00 A.M. - 5:00 P.M., 5:00 P.M., Midnight, Midnight - 9:00 A.M., respectively. Shift 4 is weekend time from Saturday 9:00 A.M. through Monday 9:00 A.M. Shift 5 is time used by FIB jobs.

utl-5	total time used in each shift.
DRQ	number of records on drum allotted to user.
DRU	number of records on drum used.
DIQ	number of records on disk allotted to user.
DIU	number of records on disk used.
TPQ	number of records on tape allotted to user.
TPU	number of records on tape used.

(END)

Revised: 2/14/66

Identification

AED - ALGOL Extended for Design
D. T. ROSS - X5880

Purpose

A general purpose programming system including a compiler, source language debugging facilities, and a library of subroutines. The compiler is especially suited to system programming, but includes algebraic statements, recursive functions, and mixed algebraic expressions for general purpose programming as well. The compiler language is an extended form of ALGOL-60, minus multi-dimensional arrays. Some of the syntactic forms of ALGOL are modified, such as procedure definition. Additional features include plex structure processing (a generalization of list processing), packing of data storage, and an input-string macro and synonym feature which includes conditional compilation. The subroutine library includes packages of routines for free-format input-output, for building of symbol tables for language processing, for plex dump and relocation, for "free storage" storage allocation, for use with the ESL display console, and for the "AED Jr." system, an experimental language processor. The AED command is the stable, tested version of the compiler. TAED is the experimental compiler, including new features in the checkout process. LAED is the special, extended version of the CTSS loader which contains additional features, such as loading a remote list of programs. The AED command contains additional options for source file conversion into extremely compressed or expanded block structured formats for ease of understanding.

References

MAC	146	AED-0 Programmer's Guide	Feldmann, Ross
MAC	154	Warnings & Restrictions in AED-0	Feldmann
MAC	169	"LOADER: A New Version of the BSS Loader"	Wolman
MAC	198	PLEX-DUMP & Relocation in AED-0	Fox
MAC	199	Stack manipulation in AED-0	Coe
MAC	207	"Internal Memos for AED Users"	Feldmann
MAC	208	"Flash No. 10 - New CTEST2 Command"	Feldmann
MAC	213	"Flash No. 11 - AEDBUG Usage"	Fox
MAC	225	Argument Checking for AED	Walsh
MAC	226	Availability of AED Jr. Systems	Ross
MAC	278	AED Bibliography	Ross *

(END)

Identification

BEFAP - Bell Laboratories' 7094 assembly language
O.C. Wright - x6004

Purpose

BEFAP is a version of FAP with a more powerful macro compiler and with the ability to handle compressed source decks directly (see CRUNCH). Its advantages are the abilities to edit larger files (via the alter feature with CRUNCH decks) and to produce more readable listing files. An immediate benefit is the ability to use and modify languages under CTSS which were developed and written in BEFAP. (e.g., BLODI,ALWAC, SNOBOL)

References

IBM C28-6235 FORTRAN II Assembly Program(FAP)

MAC 179 BEFAP command within CTSS R. U. Bayles

Usage

BEFAP NAME1 -'(CRUN)'- -'(LIST)'-

NAME1 FAP is the name of the source file to be translated. Files NAME1 BSS and NAME1 SYMTB will be created and any old versions will be deleted.

(CRUN) specifies that the crunched file, NAME1 CRUNCH, should be translated instead of NAME1 FAP.

(LIST) specifies that a listing file, NAME1 BCD, should also be created. It will be a line-marked BCD listing file which may be printed on-line by the PRINT command or off-line by RQUEST PRINT or PRINT control card.

If both (CRUN) and (LIST) are specified, they must be in that order.

Identification

COGO-90 - Coordinate Geometry Language
D. Roos - X5056

Purpose

COGO is a language and programming system for solving geometric problems in civil engineering.

References

Research Reports:

R64-12	COGO-90: Engineering User's Manual	Roos, Miller
R64-18	COGO-90: Time Sharing Version	Roos, Miller
R64-5	The Internal Structure of COGO-90	Roos, Miller

Usage

The system is activated by typing the time sharing command, COGO. Data may be read from the disk or typed in via the remote console. The same options are available for output.

Modifications

The format of several COGO commands has been changed since the publication of the above manuals. The revised formats are

READ/DISK NAME1 NAME2

Succeeding COGO commands are read from the disk file NAME1 NAME2.

DELAY/PRINT N

Succeeding output is written on the disk in file .TAPE . N, where N is any number from 0 to 9.

Identification

COMIT - Symbol manipulating and string processing
Bob Fabry - X2525

Purpose

COMIT is one of several available string processing languages. It is very powerful for performing string manipulation, such as substitution, rearrangement and duplication, on strings of alphanumeric characters e.g. natural language text. It is not so powerful on arithmetic facilities nor complex list structures.

References

MIT Press	Introduction to COMIT Programming	
MIT Press	COMIT Programmer's Reference Manual	
ACM Comm. Mar.1963	"COMIT"	V.H.YNGVE
MAC 156,CC237	COMIT operation in CTSS	V.H.YNGVE
CC 178	Availability of COMIT	V.H.YNGVE
CC 246	COMIT subroutines for generating Fortran programs	D.C.MATIATOF
CC 248	COMIT system under MIT's FMS	FABRY, YNGVE

Identification

DYNAMO - Model Simulation Language
A. L. Pugh III - x4426

Purpose

DYNAMO is a computer program for translating mathematical models from an easy-to-understand notation into tabulated and plotted results. The models may be modeled on any dynamic feedback system such as arises in business, economics, or engineering. The principal limitation on the model is that it be a continuous representation of the real world. As DYNAMO does not recognize individual items or events, models of job shops and the like cannot be tested. Persons familiar with both digital and analogue computers will find that DYNAMO in many ways behaves more like an analogue than a digital computer.

References

DYNAMO User's Manual A. L. Pugh III, M. I. T. Press

Industrial Dynamics Memo D-805 "Time Sharing DYNAMO User's Manual", A. L. Pugh III

Usage

DYNAMO NAME1 P R

where NAME1 is the name of the model to be run (with secondary name MADTRN), and P and R are optional (order is also optional). The effect of these letters is described below.

P-Page Skip

If the particular console being used has been adjusted so that the perforations are three lines above where the paper stops following a vertical form feed, this letter can be used to cause DYNAMO to skip to the top of a page rather than leaving four blank lines between pages.

R-Rerun

This letter cause DYNAMO to skip immediately to the rerun, even though there is a SPEC card included in the model.

After all the runs and reruns have been processed by DYNAMO, the console operator is given the opportunity to specify additional reruns by typing the normal rerun information, with one exception. The RUN card, instead of preceding the rerun, follows the rerun information and signals DYNAMO to start to process that rerun.

When DYNAMO is expecting this rerun information it will type out

PLEASE TYPE CHANGES IF RERUN DESIRED

The user types the cards for a rerun just as he would for a rerun with the regular version of DYNAMO. He does not have to specify the card number of the card he is changing. Nor does he have to wait for the computer to type a card number or M as he does when using the INPUT and EDIT commands. The tab signifies a skip to Column 7.

A feature of time sharing simplifies correcting typing errors. Should the user wish to delete a long line with several errors he may type a ? followed by a carriage return to start him at the beginning of a new line.

If the user does not wish to rerun his model he should type

QUIT

If while DYNAMO is either printing or plotting the results of a run the user decides that he does not want any further output but would like to skip on to the next rerun, he may press the break button once and DYNAMO will proceed immediately to the rerun.

Differences In Input

Basically the input to the Time Sharing DYNAMO is the same as the regular DYNAMO. There are several minor restrictions which are introduced by the time-sharing system while other restrictions have been removed.

1. As one has access to this model only through the console, the option to number the cards of a model now becomes a requirement.
2. A continuation card has a different card number rather than having the same number as the card it continues.
3. The contents of the identification card (the first card) are entirely optional. Columns 7 through 36 of this card are copied into the page heading.
4. The RUN card which is normally the second card is now optional.
5. The RUN number should be restricted to 5 instead 6 characters.
6. Because of the narrower page only nine columns are available for tabulating results instead of the former fourteen.

Identification

ESL display system (not a command)
C. Garman - X5889

Purpose

To provide a graphical input and output facility with a limited real-time capability. Two 18 inch CRT'S are provided for output. Input is from light pens, pushbuttons,, toggle switch banks, and other forms of analogue input. Real time rotation, translation, and magnification of appropriately constructed pictures is possible under program control.

References

MAC 122	DEMON: ESL Display Console Demonstration Program	Polansky
MAC 125	ESL Display console Time Studies	Polansky
MAC 166	B-core system for programming ESL in CTSS	Lang
MAC 201	ESL Display console system manual	Bayles
MAC 202	Proposal to improve rotation matrix of ESL	Stotz
MAC 217	Operating Manual for the ESL Display Console	Stotz, Ward

Identification

FAP - IBM 7094 machine language
Programming staff

Purpose

FAP is the IBM MACRO-FAP assembly program for the 7094 machine language code. It accepts all 7094 operation codes and the standard data defining pseudo-ops, as well as macro definitions. Input files may be line-marked or line-numbered. Four new pseudo-operations have been added to the time sharing version.

References

IBM C28-6235	Fortran II Assembly Program (FAP)	
CC 201	MIT version of FAP	
CC 217	Abbreviated FAP	Saltzer

Usage

FAP NAME1 -'(LIST)'-

NAME1 FAP is the name of the FAP source language file which is to be translated. The files NAME1 BSS and NAME1 SYMTB will be created if the assembly is successful. Any previous versions of these two files will be deleted.

(LIST) is an optional argument which instructs FAP to produce a line-marked listing file named NAME1 BCD. This file produces a listing similar to the one produced by FAP under FMS when it is printed off-line by RQUEST PRINT or print control card or on-line by the PRINT command.

INSERT pseudo-operation

The pseudo-operation INSERT NAME will cause the contents of the file NAME FAP to be inserted and assembled in place of the INSERT instruction. The entire file is inserted unless the END card, signalling the end of the input deck, is found. Only one level of nesting is allowed; i.e., an INSERT may not be used within a file which is itself being INSERTed.

LSTNG pseudo-operation

The pseudo-operation LSTNG controls console printing of the assembly listing. The first LSTNG card causes printing, beginning with the next line. Alternate appearances of LSTNG turn this feature on and off.

NOSEQ pseudo-operation
The pseudo-operation NOSEQ prevents sequence checking of serialization in columns 73-80; i.e. no "SOURCE ORDER ERROR" diagnostics appear in the NAME1 BCD listing file.

NOLNK pseudo-operation
The pseudo-operation NOLNK deletes the standard error procedure section of FAP. The linkage director, normally provided for all subprograms, is omitted, and calling sequences produced by CALL are shortened by two instructions.

Input files
Both line-marked and line-numbered files are acceptable as input, and files may be mixed (an INSERTed file need not contain the same type record as the main input file). All input files, regardless of type, have the secondary name FAP.

Records in line-marked files may contain tabs, which are interpreted by FAP. A maximum of 72 columns per card are assumed for line-marked records.

Identification

GPSS - General Purpose System Simulator
M. M. Jones - X5870

Purpose

GPSS is a simulation language that is easy to learn, use and debug. It automatically collects and prints many useful statistics. GPSS is particulary well suited for simulation of traffic flow models, such as communication nets, circuit models, computer systems, and queuing models.

References

MAC 140 On-line Version of GPSS II M.M. Jones
IBM B20-6346 General Purpose System Simulator II

Major Revision: 2/14/66

Identification

LISP - List Processing Language
J. Moses - X5867

Purpose

LISP is a high-level list processing language, mathematical in character. Programs specify computation by recursive functions. The time-sharing version contains functions which permit smooth interaction between LISP and the time-sharing environment. The language is used extensively in artificial intelligence work.

References

MIT Press	LISP Progammer's Manual	Levin
Information International	The Programming Language 'LISP'	Bobrow, Berkeley
MAC 134	LISP Exercises	Hart
MAC 153	Time Sharing LISP	Martin, Hart
MAC 206	CTSS LISP NOTICE	Hart
MAC 296	A New Version of CTSS LISP	Fenichel, Moses

Usage

LISP -NAME1-

NAME1 LISP is a BCD file containing pairs of S-expressions which will be initially read and executed by the LISP evalquote operator. If NAME1 is not specified. LISTEN NIL will be executed.

LISTEN NIL -- If the doublet LISTEN NIL is executed, subsequent S-,xpressions will be read from the console. When the atomic symbol STOP is typed, the system will normally enter the DORMANT state.

(END)

Revised: 8/30/65

Identification

MAD - Michigan Algorithm Decoder
University of Michigan; Barden, B. Galler, and R. Graham
Programming Staff

Purpose

MAD translates algebraic statements describing algorithms into the equivalent machine instructions. The MAD language was originally based on ALGOL 58 with certain extensions and adaptations. It allows some more powerful logical operations than Fortran II.

References

	MAD November 1963 (Reference Manual)	
	MAD December 1964 (Reference Manual)	
CC 186	Fortran & MAD format Specifications	Spall
CC 213	Abbreviated MAD	Corbato..etc.

RESTRICTIONS

The extended features in the appendix of the December 1964 manual have not been implemented.

Usage

The current compiler implements the language as described in the MAD Manual of November 1963. However, a few additions and modifications have been made.

MAD NAME1 -'(LIST)'- -'(SYMB)'-

NAME1 is the primary name of the source file NAME1 MAD which is to be translated.

(LIST) requests that MAD create a line-marked listing file called NAME1 BCD which may be PRINTed on-line or RQUEST PRINT for off-line printing.

(SYMB) requests that MAD produce a special symbol table named NAME1 SYMTAB which is used by MADBUG. (SYMB) also suppresses the normal on-line printing of length, entry point and transfer vector length.

CHANGES:

1. A new statement
   INSERT FILE ALPHA

will cause file ALPHA MAD to be inserted in the compilation after the INSERT FILE statement. Only one level of nesting depth of inserted files is allowed, although any number of INSERT statements may appear in the higher level program.

2. An addition has been made to the '...' block notation in MAD. Formerly only the form
A...B or A,...,B
was allowed, where A and B are variables. Now the second expression may be a constant, e.g.,
A ... 7.
See MAD Manual, November, 1963, page 16.

3. A change has been made in MAD for defined operators. (See MAD Manual, November 1963, pages 100-112.) This was needed due to the added feature of saving and restoring index registers 1,2 and 4 in functions. The change was made to the ..RTN. operator. This is now a unary operator, i.e. only a B operand. The function of the B operand remains the same, that is, the address of the value to be returned to the calling program. The A operand is internally set to the address of the index restoring code. This address is designated "FF". Note the example on pages 110-111 of the November 1963 manual. This should be changed to the following:

> ..RTN. This symbol, which is obviously invalid in a statement, stands for the operation of placing the appropriate value(s) in the arithmetic register(s) and then returning from a function to its calling program. It is analogous to the right hand side of a substitution statement (the B operand) and then a transfer to a given address (there is no designation for this address within the triple). As such, there is no result. As an example, if the result of a function were a double precision number, say mode 5, the following would be a reasonable definition.
>
> ```
> MODE  STRUCTURE  4..RTN.5
> JMP   *+3,BT,*+1
> CLA   B
> LDQ   B+1
> TRA   FF
> OUT   ACQ
> END
> ```

The address FF is the address of the index restoring code.

4. The input phase of MAD has been rewritten to use the new file system and accept line-marked (SQUASHed) files as well as card-image files as input. Any mixture of line-marked or card-image files may be used, e.g. INSERT FILE statement may insert the opposite kind of file from the main file. *

5. On a line-marked file, the tab and logical backspace will be interpreted as follows: *

   a. A logical backspace (colon) will imply a backspace to column 11 only if the colon occurs in column 12. (See b. below). All other colons will be treated as legal characters by the input routine.

   b. The first occurrence of a tab effectively indicates that the following characters must start at least in column 12. Should the first tab occur after column 12, one blank will be inserted.

   c. Further occurrences of tabs are interpreted to mean that the following characters are to be at least 5 columns away from the column reached by the last tab. This allows one to indent WHENEVER's, etc.

6. The input phase will construct a sequence number for internally generated card-images constructed for line-marked records. This number will be incremented by 1 for each line-marked record read. The sequence numbers should provide an aid to error checking and correction using EDL. *

Identification

MADTRN - Fortran II to MAD translator
Programming staff.

Purpose

Fortran II has not been implemented to operate with the time-sharing system. In order to allow users to operate with Fortran II programs, the MADTRN translator is provided. A Fortran II source language program may be translated to MAD and then translated to the equivalent machine instructions by the MAD compiler. MADTRN does not always produce perfect results and, therefore, should not be used unless absolutely necessary. MADTRN assumes a working Fortran program and therefore MADTRN diagnostics are minimal.

References

IBM		Fortran Reference Manual	
CC	188	MADTRN, A Fortran-To-Mad Language Translator	Korn
CC	186	Fortran and MAD Format Specifcations	Spall

Usage

MADTRN NAME1 OP

NAME1 is the primary name of the source language file named NAME1 MADTRN or NAME1 OP if OP is a class name.

OP=(LIST) The argument (LIST) will be passed on to the MAD compiler and the listing file named NAME1 BCD will be created by MAD.

OP=(SYMB) The argument (SYMB) will be passed on to the MAD compiler and the file NAME1 SYMTAB will be produced to be used by MADBUG.

Revised: 5/9/66

Identification

SNOBOL - A String Manipulation Language
D. Shea, X4107

Purpose

SNOBOL is a programming language for the manipulation of strings of symbols. A statement in the SNOBOL language consists of a rule which operates on symbolically named strings. The basic operations are: string formation, pattern matching, and replacements. Facilities for integer arithmetic, indirect referencing, input-output, debugging, and SNOBOL-coded and SNOBOL system functions are included.

Usage *

SNOBOL NAME1 -'ONLINE'-

will initiate action (compilation and execution) by the SNOBOL compiler on the line-marked file NAME1 SNOBOL. The SNOBOL program listing, and other compiler output, will be put in a file named NAME1 BCD.

ONLINE is an optional argument which causes all output to be printed out on the user's console. It may be abbreviated 'O'.

References

Journal of the ACM, January 1964, pp. 21-30. *
CC 235, MAC-M-307, CTSS SNOBOL User's Manual, Shea.

(END)

Major Revision: 5/23/66

Identification

STRUDL Command
R. Logcher, 1-142, x5326

Purpose

The Structural Design Language (STRUDL) is a computer-aided design system for bar structures. Until recently, the only aids to the design of this class of structures were ones which performed analysis. The STRESS language is one such impressive aid with a means for easy communication of analysis, data and generalized capabilities. Indeterminate analysis, however, is only one process of structural engineering; it is a necessary component without which sophisticated engineering can not be accomplished. The design-oriented system incorporates STRESS and most of its langugage under a command interpreter controlling many other processes needed for design.

References

MIT PRESS	STRESS: A User's Manual	Fenves, Logcher, Mauch, Reinschmidt
MAC-M-234	A User's Manual for On-Line Use of the Structural Design Language	Logcher, et al.

Usage

The subsystem is entered by typing (at command level):

STRUDL

The response is:

TYPE.

Details of usage are covered in the references cited.

(END)

Identification

BLODI - A BLOCK DIAGRAM Compiler
Bell Telephone Laboratories, Murray Hill, New Jersey
V.A. Vyssotsky, Carol Lochbaum, and J. L. Kelly, Jr.

Purpose

BLODI is a compiler intended for use in simulating sampled data processing schemes. It accepts as input a description of a circuit block diagram, written in BLODI language.

References

The Bell System Technical Journal, Vol 40, pp 669-676, May, 1961.

Bell Labs Technical Memorandum MM-61-123-4, February 10, 1961.

Bell Labs Technical Memorandum MA-1276, March 20, 1963.

Identification

On-line programming system
OPS
M. Jones, J. Morris, D. Ness, x5882

Purpose

OPS is a sub-system intended to facilitate on-line interaction between a computer and the general user. It allows loading, execution, and deletion of BSS subroutines by name and construction of FORTRAN-like procedures. A large repertoire of standard operators (subroutines) is available for data manipulation and simulation.

References

M. Greenberger, et. al., On-Line Computation and Simulation: The OPS-3 System, The MIT Press, December 1965.

MAC-M-277. OPS-3 Goes Public. Greenberger, Jones, Ness, Morris.

Usage

The sub-system is entered by typing (at command level)

OPS

CTSS will respond with a system W(ait) line; then OPS will respond with a line beginning "OK". Thereafter, typing the name of a subroutine causes its execution. In particular, typing

GUIDE INFORM

will introduce the new user to the guide files and their use in obtaining information about the system.

Identification

Searching Technical Information Project Bibliographic Files
TIP
W. D. Mathews, 14S-318, X5687, 3/15/66

Purpose

TIP is a command program to search and retrieve from the bibliographic data files maintained by the Technical Information Project. There are two sections to the TIP program, the input request interpreter and the data retrieval section. The user types several requests and signals that his input is finished; then the retrieval section answers his requests.

Usage

```
    user:  TIP
response:  TYPE YOUR REQUESTS
```

Or

```
    user:  TIP NAME1 NAME2
```

NAME1 and
NAME2 are the primary and secondary names of a 'read' file (see description of 'read' request below). When NAME1 NAME2 are specified, the program does not go into input wait in expectation of console interaction but instead reads its requests directly from a file.

```
response:  none
```

Input

Input looks very much like English sentences; it consists of words separated by blanks. The first word in each sentence is one of the request words (see below); punctuation and redundant or undefined words are usually ignored on the input line.

Description of the Data

The TIP data files consist of bibliographic information pertaining to recent issues of a selected library of physics journals. Currently the system contains data on 25 Physics journals and covers the period from January 1963 to date. The store of articles is updated weekly.

For each article in each issue of these journals, the following information has been extracted and recorded on the disc.

1. Identification---the journal, volume and page number on which the article begins.
2. Title---the title of the article.
3. Author---the names of the authors.
4. Location---the authors' institutional affiliation.
5. Citations---a listing of each journal article cited by the article being recorded.

The title, author and location fields are stored as BCD text. Due to limitations in the character set, greek symbols in the title are spelled out and superscript and subscript information is usually lost. Only the first ten authors of a given article are listed in the data, other authors are ignored. The location field is printed in reverse order, for example;

Cambridge, Massachusetts
Massachusetts Institute of Technology
Department of Physics

The identification and citation fields are stored in a coded and packed binary form, each consisting of a three part number of the form Jn, Vn, Pn. Code numbers have been assigned to the most frequently cited journals and only these journal numbers are stored, not the full name of the journal.

A typical article looks like this:

```
PHYSICAL REVIEW
VOLUME  135
J001 V135 P0960
VIBRATIONAL AND CENTRIFUGAL EFFECTS ON THE MAGNETIC
SUSCEPTIBILITY AND ROTATIONAL MAGNETIC MOMENT OF THE
HYDROGEN MOLECULE
  CHAN SUNNEY I.
    PASADENA, CALIFORNIA
    CALIFORNIA INSTITUTE OF TECHNOLOGY
    GATES AND CRELLIN LABORATORIES OF CHEMISTRY
  IKENBERRY DENNIS
  DAS T. P.
    RIVERSIDE, CALIFORNIA
    UNIVERSITY OF CALIFORNIA
    DEPARTMENT OF PHYSICS
      J001 V034 P0057   J001 V041 P0713   J001 V041 P0721
      J001 V058 P0310   J001 V078 P0711   J001 V080 P0476
      J001 V085 P0937   J001 V087 P1075   J001 V094 P0350
      J001 V094 P0893   J001 V103 P1254   J001 V111 P0203
      J001 V112 P1929   J001 V115 P0897   J001 V126 P0146
```

```
J002 V000 P0000   J002 V000 P0000   J003 V065 P0178
J012 V009 P0061   J012 V019 P1030   J012 V020 P0527
J012 V021 P2070   J012 V023 P1131   J012 V032 P0105
J012 V035 P1065   J012 V035 P1967   J012 V037 P0214
J012 V037 P1527   J012 V038 P1263   J027 V000 P0000
J027 V000 P0000   J030 V032 P0231   J030 V035 P0130
J046 V006 P0019   J055 V035 P0730   J160 V004 P0061
J311 V003 P0017   J311 V010 P0278
```

The articles are arranged in files, each file corresponding to a single volume of a given journal.

Description of Requests

Ten types of requests are recognized by the request interpreter. The ten requests, together with a brief description and an accepted abbreviation for each word, follow.

Abbreviation	Request-word	Description
L	Library	To find out what volumes are available.
S	Search	To establish the volumes to be searched.
F	Find	To specify the criterion of retrieval.
O	Output	To specify the type of output.
N	Name	To name the output files.
C	Cancel	To erase one of the request tables.
U	Unlock	To append to the previous request tables.
Q	Quit	To leave TIP and return to CTSS for any waiting commands.
R	Read	To designate some file as the source of input requests.
G	Go	To signal the input section that retrieval should commence.

Library Request

The library request gives an inventory of the volumes available for each journal in the TIP data library. For each journal, the full journal name is given, followed by the journal number code, a six-letter name code, an abbreviation, and the holdings. Sample entry:

```
ANNALS OF PHYSICS
 J384 - ANNPHY - ANN PHYS
  V 26 - 34
```

A journal may be referred to in the TIP program by using any one of the forms listed in the inventory entry. The numerical code 'J384', for example, is always equivalent to the code 'ANNPHY' or the abbreviation or the whole journal name.

Search Request

The search request is used to delimit the particular files to be scanned. Several forms of this request are permissible. The general forms (Gf.) and specific examples (Ex.) follow:

(a) To search a specific journal and volume
Gf. search Jn Vn1 Vn2 Vn3 ...
Ex. search annals of physics volumes 30, 32

(b) To search a continuous range of volumes in a journal
Gf. search Jn Vn1 to Vn2
Ex. search journal of chemical physics volumes 38 to 42

(c) To search all available volumes of a journal
Gf. search Jn1 Jn2 Jn3
Ex. search phys rev, annals of physics, nuovo cimento

(d) To search a special file not in the TIP data library
Gf. search alpha beta
Ex. search my file

(e) To search all files in the TIP data library
Ex. search all

Any of these request types, except type (e), may be combined into one search request. In combining search requests, the word search is specified only once, at the beginning of the line. Further, succeeding lines will be assumed to be part of the search request unless another request type is encountered.

Ex. search j chem phys 30, 31, phys letters,
my file, annals of physics 29 to 33

In this example, statement types (a), (b), (c) and (d) are combined.

Find Requests

The find request establishes the criterion of acceptance to be applied to each article. Only those articles within the search range which meet the requirements specified in the find request are considered to be part of the answer.

There are two basic types of find requests. These basic types answer the questions:

Type 1. Given a property, find articles having that property.
Type 2. Given an article, find other articles like it.

Variations of Find Request Type 1

Several variations of the find request are built up from the first basic find type.

Basic type 1: find field-specifier literalstring
Example: find title neutrino

This is the basic form of the find request. The field-specifier indicates what data field in the article is to be inspected and the literalstring determines what actual string of characters must be contained in the field for the find to be successful. For the fields identification and citation, the literalstring consists of a journal, volume and page specification:

Example: find citation j chem phys vol 41 page 395

Several criteria may be concatenated on one line. The vocabulary words and and or facilitate this concatenation.

Example: find title x-ray and title diffraction

This find request will be satisfied only if the title of an article contains both 'x-ray' and 'diffraction'. Ordering of the words is not considered important, i.e., the title could contain the word diffraction before x-ray. The word 'and' may be left out in a one line request. It is assumed if not explicitly stated. Further, if both literalstrings involved are to be found in the same data field, the field-specifier need not be repeated. All of the following are equivalent:

```
find title x-ray, title diffraction
find title x-ray and title diffraction
find title x-ray diffraction
find title diffraction x-ray
```

A find request may continue for several lines. The word find itself need not be repeated in this case but it is important to note that unless otherwise specified, the criteria found on succeeding lines of a find request are ored with preceeding ones. This is similar to the rule for anding of criteria and the two may be summarized: Unless otherwise stated, criteria on the same line will be anded, on different lines will be ored. Anding takes precedence over oring.

Anded criteria:

```
a. find title x-ray diffraction
b. find title x-ray
      and title diffraction
```

Ored criteria:
a. find title x-ray or title diffraction
b. find title x-ray
   title diffraction

A field-specifier must follow the explicit occurence of the vocabulary words and and or. Also, the field specifier must be stated whenever a new line is begun in the request.

Not Acceptable: (a) find title x-ray
diffraction

(b) find title x-ray and diffraction

Several field types may be mentioned in one request.

Examples:
find title neutrinos, author smith
find title superconduct, citation phys rev v 133 p 350

The vocabulary words not, but not, except are special words in the find request. They are all defined identically and have the ability of negating the sense of the next literalstring on the input line.

Example: find title not x-ray, not diffraction

will find the complementary set of papers found by the request 'find title x-ray diffraction'.

Several characters in the literalstring have special meanings. To understand their uses, a few comments about the scanning of the request line are appropriate. Each literalstring in the request is set off from the strings preceeding or following it by break characters, (usually a blank). When searching for titles, authors or locations, a leading blank is added to the literalstring that occurs on the input line and during the actual search, it is this augmented literalstring, with its leading blank, that is matched against the data field specified. The request 'find title magnet' will look for the string ' magnet' (with a leading blank) in the title and it will find titles containing magnet, magnetically, magnetic and so forth, but not paramagnetic, ferromagnetic etc.

To suppress the feature of augmentation which inserts a leading blank, the + sign may be used. The + sign allows any character to precede the literalstring and therefore will find all the above possibilities (paramagnetic, magnetic etc.). The request 'find title +magnet but not magnet' is a good way of finding only those instances where the literalstring 'magnet' occurs in the middle of a word (paramagnetic).

Sometimes it is useful to insert a blank into a literalstring as a real character to be found. Since the blank is ordinarily a break character, the asterisk has been supplied with special meaning for this purpose. The asterisk is an explicit blank. 'Find title magnet*' will only find the titles containing the word magnet. The asterisk insists that the word must terminate in a blank. The asterisk is also useful for putting phrases into one literalstring. 'Find title x-ray*diffraction' will take 'x-ray diffraction' as a phrase which must occur in the title. This is quite different from the request 'find title x-ray diffraction' (without the asterisk) since the latter only asks that the two words must co-occur in the title.

Finally, it may become necessary to unreserve some of the vocabulary or field-specifying words in order to use them in a literalstring. This may be done by putting single quotes around the word.

Example: find title 'title'

This request will look for the word <u>title</u> in the title field of an article.

Variations of Find Request Type 2.

Only one variation of the second basic find type is currently available. This involves the notion of bibliographic coupling. A paper is assumed to be similar to another paper if both contain similar citation items. If paper A cites paper X and paper B also cites paper X, the two papers are said to be coupled bibliographically. The strength of the coupling depends on the number of items commonly shared by the citation lists for papers A and B. In the TIP program, we may ask for papers having citation items similar to any other paper in the TIP data library.

Basic type 2: find share citations jvp
Example: find share citations with phys rev v 137, p 1.

The request reads: "find all papers which share citations with jvp", where jvp is the identification of a paper. Inasmuch as it is not necessary for the user to know precisely what the citations are that he is asking to find, we may look on this as an indirect request. The user specifies a paper he is interested in and the program inspects that paper's properties and finds other papers having similar properties. In this case, the similarity is found in the citation structures.

Share requests may be <u>anded</u> and <u>ored</u> with other share requests or with type 1 find requests.

Example: find title x-ray
or share citations with phys rev 137 1038.
Example: find share citations with phys rev 133 1
and share citations with phys rev v 133 p 350.

When two citation requests are anded together as in the last example above, it is required only that some citation item found in vol 133 page 1 be found in the object paper and that some citation item found on page 350 also be found in that object paper's citation list. It is not required that these be the same citation item. That is, in the example above, anding of the two citation requests is not the same as asking for papers which contain the citations common to both page 1 and page 350. It asks for papers which contain some citation common to page 1 and also some citation common to page 350.

Output Requests

There are three output modes - print, store, and save. The general form of the output request is the word 'output' followed by a mode designator and a field-specifier.

Examples:
output print title, author
output store citations
output save id, title

'Output print' specifies that the fields of data are to be unpacked and printed on the console.

'Output store' will unpack the information and write into a file the exact same information that would be printed on the console by the 'output print' request. This is useful for massive output since the user may then request off-line printing of the output file by use of the CTSS rquest command.

'Output save' specifies that the fields should not be unpacked but that, instead, a new file should be created in the same coded, searchable file format as the TIP data files. This is useful for the creation of sublibraries which may then be used for further searches.

The word 'all' is permissible as a field designator and is taken to mean all existing fields. Several modes may be specified on one line, several lines may be used to specify the output request, the word output need only be at the beginning of the first line but a mode designator must occur at the beginning of all succeeding lines.

Examples of legal output requests:

```
output print title author, save author, store citations

output
print title, author
store location
save citations
```

Unless otherwise specified, (by use of the 'name' request), the store request will write into a file called 'store file' and the save request will write into a file called 'save file'.

During a search for citations with either the 'find citations' or the 'find share citations' requests, an additional field is associated with each article which contains the citations found by the find request. This field may be printed out by using the word 'coupling'. 'Output print coupling' will print the citations under the heading SHARED LINKAGE TO Ja Vb Pc (where Ja Vb Pc was in a share citations request) and will print under the heading CITATION MATCH: (where the citations were found directly by the find citation request).

Name Request

The name request gives names other than 'save file' and 'store file' to the output files created by the TIP program. The format of the name request is:

```
name store file alpha beta
name save file gamma delta
```

It is not permissible to give both the store and save files the same name. Further, it is not possible to store or save a file which is being read or vice-versa. This is important when the save file itself is being searched.

Information is always appended to existing output files on the disc. It is possible to restore the original names of the store or save files by later saying 'name store file store file' or 'name save file save file'.

Cancel Request

The search, find, or output tables may be cancelled at any time during input. In addition, it is possible to cancel the various modes of output selectively instead of all output requests. Cancelling re-initializes the tables as though no requests had been made. Cancel expects the words 'search', 'find', or 'output' to follow it and the word output may further specify 'print', 'store' or 'save'. It is not possible to cancel one part of a request, the whole request table must be cancelled.

Example: cancel search, output print

The above example will cancel all search requests and also the output print requests. Output store and output save requests are unchanged.

Example: cancel find, output

All find and all output requests will be cancelled.

Example: cancel output, search, find.

All request tables will be cancelled.

Ordering of the words 'search', 'find', 'output' is unimportant. The word output may be followed by 'print', 'save' or 'store' in any order. If such words are present, only those modes of output will be cancelled; if they are not present, all output will be cancelled.

Unlock Request

After each retrieval, control is again returned to the request interpreter section. New requests may then be entered and they will override the old requests. Any request types not present in the new input, however, will be assumed to be the same as in the previous interaction. For example, if requests were made as follows:

```
search phys rev vol 136
find title plasma
output print id, title, author
go
```

Then for a succeeding interaction we might specify:

```
search phys rev vol 137
go
```

The new search request overrides the old one but the find and output requests will remain unchanged.

Specifically, the search, find and output tables are 'unlocked' during the request interpreting part of an interaction and these tables may be added to indefinitely. Once retrieval is started with the 'go' request, however, these tables become 'locked'. The rule for succeeding interactions is that a locked table cannot be added to; it must be deleted and a new, unlocked table created in its place. Tables which have been unlocked with the unlock request are free from this restriction and further requests will add on to these tables without deleting the old requests. In the above example, if we had said:

```
unlock search
search phys rev 137
go
```

The second interaction would then search both volumes 136 and 137 of the physical review. The conventions for the unlock request are exactly the same as for the cancel request

Examples:

```
unlock find, output
unlock output save, search
```

Quit Request

Quit is a request which exits to the system with an immediate call to CHNCOM. It is useful in interactions in which commands are waiting in the command buffers.

Read Request

It is possible to put all the requests which would ordinarily have to be typed on the console into a file on the disc. The request 'read alpha beta' will then cause the program to read input requests from file 'alpha beta' instead of from the console. When an end of file is reached input is again read from the console.

The file may be originally created by the CTSS Ed, Edl, or Input commands. Nesting of read requests is possible to a depth of 25. That is, a read file may itself contain another read request and so forth. Recursive read requests are not, however, legal --- (a file named 'alpha beta' may not contain the request 'read alpha beta'.

Go Request

For each retrieval, there must be at least a search, find, and output request. When the user has typed these requests and wishes retrieval to begin, he signals this by typing the request 'go'. The go request checks to see if search, find, and output requests are present, then it 'locks up' the tables containing these requests and transfers control to the searching routines.

Abbreviations *

In addition to the abbreviations for requests and names of journals, many other words may be abbreviated in input to the TIP program. A summary of these shortened forms follows. Each word is equivalent to other words on the same line.

Requests

S Search
F Find
O Output
N Name
C Cancel
U Unlock
Q Quit
R Read
G Go

Fields

I Id Ident Identification
T Title Titles
A Author Authors
L Location Locations
C Citation Citations B Bibliography
Link Linkage Coupling

Output Modes

P Print
St Store
Sa Save

For example, instead of:

```
Search Annals of Physics volume 30
Find title magnetic
Output print id title author
Go
```

We might say:

```
s j384 28
f t magnetic
o p i t a
g
```

Break Characters

Several characters serve as break characters in the input lines to the request interpreter. Several break characters in sucession will always be interpreted as only one break character. The break characters are: blank, tab, comma, period, equals sign, null (octal 57), slash, and carriage return. The following two strings are equivalent:

```
find author jones
find//author=jones.
```

(END)

Identification

FORTRAN IV Translator
FOR4
T. Burhoe, 868-9840

Purpose

To translate source programs written in the FORTRAN IV language to MAD and compile them; to provide source-level compatibility between CTSS and background FORTRAN IV; and to provide useful diagnostics on the source program.

FOR4 is an author-maintained CTSS command, offered on an experimental basis. As such, all inquiries, suggestions, or complaints should be addressed to Mr. Tom Burhoe, IBM Scientific Center, 545 Technology Square (4th floor), Cambridge, Mass. Telephone (617)-868-9840.

Usage

FOR4 NAME -'NOCOMP'- -'MADFIL'-

NAME is the primary name of file NAME MADTRN, the user's FORTRAN IV source program. It must be a line-numbered disk file. All source programs should be written in accordance with the FORTRAN IV language specifications as described in IBM Form C28-6390-2, available at the Coop. These are the specifications for FORTRAN IV as implemented under IBSYS, Version 13.

NOCOMP is an optional parameter indicating that the user does not wish the translated MAD program to be compiled into object code. (For instance, he may wish only to find the syntactic errors in his FORTRAN programs prior to submitting them for background runs.)

MADFIL is an optional parameter indicating that the generated MAD program should be left in permanent (0) mode in the user's directory, rather than in temporary mode as in the normal case. Also, detection of source errors will normally suspend the generation of further MAD code; the MADFIL option will create a MAD program regardless of errors in the source program.

In normal operation, FOR4 will generate an equivalent MAD program in temporary mode, compile it via MAD, and exit to CHNCOM. If source errors are detected, they are printed out along with the offending statements, and the final exit is to DEAD. If translation is interrupted via the BREAK key, an immediate exit to CHNCOM will occur.

There are nearly one hundred explicit one-line diagnostics in FOR4, thus permitting much flexibility and specificity in helping the user debug his programs. The diagnostics closely resemble those of IBSYS FORTRAN IV. One important objective of FOR4 is to permit an exact correspondence between programs which will compile and run under CTSS, and those which are run under batch processing. Thus, all IBSYS FORTRAN IV errors are diagnosed in FOR4, even though some may be "translatable" - e.g., mixed modes in aritimetic expressions.

The so-called "built-in functions" of FORTRAN IV may be used in FOR4 in external function form by including file 'F4LIBE BSS' at load time. This program contains entries for all the built-in functions which do not process "complex" or "double-precision" data exclusively. To link to this (1-track) library, do: LINK F4LIBE BSS (1410 CMFL04 .

Restrictions

Because of dependence on the IBJOB system, or lack of a MAD counterpart, the following FORTRAN IV facilities may not be used in FOR4:

'BLOCK DATA' subprograms
'DOUBLE PRECISION' and 'DOUBLE PRECISION FUNCTION'
'COMPLEX' and 'COMPLEX FUNCTION'
'NAMELIST'
Named-common conventions in 'COMMON' declarations
'ENTRY'
'RETURN i' (non-standard return)

Attempts to use these facilities will cause a special diagnostic to the user reminding him that they are legal in IBSYS FORTRAN IV, but cannot be processed by FOR4. In addition, the user should note the inconsistency in array storage between FORTRAN and MAD, the former storing by columns and the latter by rows. Thus, an attempt to equivalence a vector to a column of a matrix would be acceptable in FORTRAN, but the resulting MAD program would have the vector equivalenced to all or part of one or more rows of the matrix. Care is urged in this usage of FOR4.

All facilities of FORTRAN IV excepting those noted above are available to users in their full generality.

(END)

Identification

An Algebraic Desk Calculator version of the Formula Manipulation Compiler developed by the IBM Boston Programming Center.
FORMAC
R. Kenney, 491-0321

Purpose

To allow the user to manipulate a class of formal expressions and compute the values of arithmetic expressions. Included among the capabilities of the program are formal differentiation, substitution for one or more variables in an expression and expansion of expressions. After the expression has been manipulated in a way requested by the user, the results are simplified; like terms (i.e., terms which differ only by a constant factor) are combined, zero terms and unit factors are eliminated, etc. The result of the computation or manipulation is then available for further manipulation or for printing on the user's console.

FORMAC is an author-maintained CTSS command offered on an experimental basis. Inquiries, suggestions or comments should be addressed to Mr. Robert Kenney, IBM Boston Programming Center, 545 Technology Square (3rd Floor), Cambridge, Massachusetts, telephone (617) 491-0321.

Reference

CC-257. Description of Time-Shared FORMAC.

Method

The Desk Calculator statements are executed immediately, as they are typed in through the user's console. No provision is made for a stored program. That is, although results are saved and are usable from one computation to the next, the statements which caused the computations to take place are not saved. Thus, it is not possible to establish loop controls and various paths of program flow in the classical stored program sense.

The Desk Calculator accepts lines of up to 84 characters and prints 84 characters per line. The '$' is used as the end of statement marker. A statement may extend over any number of lines. FORMAC will respond 'READY' initially and between statements; do not type a new statement until the response has appeared. (Editor's note: If the response seems unduly delayed, check to see if you've forgotten to terminate the statement with a '$'.) Standard CTSS erase and kill characters apply during the typing of a line. However, there is no "context editing"; if an already typed line is

found to be incorrect, the entire line must be retyped.

Usage

FORMAC

Summary (see CC-257 for details):

Variables:

A variable name represents a fixed point variable if its initial character is one of the letters I-N. A variable is made either atomic or assigned from context , "upon its first appearance" in a FORMAC statement. It is considered "assigned" if its first appearance is on the left-hand side of an = sign. It is "atomic" (i.e., it stands for itself) if its first appearance is on the right hand side of an = sign. An atomic variable may not have an expression assigned to it.

Operators and Functions:

Expressions may be composed of the following operators and functions:

*,+,-,**,/, DERIV,SIN,COS,ATN,HTN,FAC,EXPON,LOG,DFC,COMB.

DERIV (A,B,N)	is the nth derivative of A with respect to B.
SIN (A)	is the SINE of expression A.
COS (A)	is the COSINE of expression A.
ATN (A)	is the ARC TANGENT of expression A.
HTN (A)	is the HYPERBOLIC TANGENT of expression A.
LOG (A)	is the NATURAL LOG of expression A.
EXPON (A)	is E**A.
FAC (A)	is the FACTORIAL of A.
DFC (A)	is the DOUBLE FACTORIAL of A (n(n-2)(n-4)...).
COMB (A,B)	is the COMBINATORIAL of A things taken B at a time.

Executable Commands:

The assignment statement assigns the variable name on the left of the equal sign as the name of the expression on the right. The expression may be composed of any of the operators and functions above.

Example: A = B+C+DERIV (X**2,X,1)$
Result: the variable A names the expression, B+C+X*2.

SUBST Substitution for variables in the expression. The list of parameters may be written explicitly or a label of a 'PARAM' statement may be used. Substitution proceeds from left to right.

```
Example 1:  D = SUBST A, (X,0)$
Example 2:  ABC = PARAM (X,0)$
            D = SUBST A,ABC$
```

Result: Both examples result in D naming the expression B+C.
(Assume A was the expression B+C+X*2 from the example above.)

EXPAND Performs multinomial expansion and the distributive law on the expression. The 'CODEM' option causes the result to be placed over a common denominator.

```
Example 1:  F = EXPAND D**2$
```
Result: F names the expression B**2+B*C2+C**¢
(Assume D was the expression B+C from above.)

```
Example 2:  F = EXPAND H/B+C, CODEM$
```
Result: F names the expression:
(B*C+H)*B**(-1)

PRINT Prints on the user's console the variables and their expressions which are specified.

```
Example:  PRINT A,D$
```
Result: Results in the following print-out on the console,
A = B+C$
D = B**2+B*C*2+C**2$

DUMP Prints on the user's console all assigned variables and their expressions.

```
Example:  DUMP$
```
Result: All assigned variables printed in the same format as the 'PRINT' command above.

ERASE Erase the expressions specified and return the storage they require to the storage pool.

```
Example:  ERASE A,D$
```
Result: The expressions which A and D names are erased.

CLEAR Clears the symbol table and reinitializes the storage pool. It has the same effect as reloading the Desk Calculator.

Example: CLEAR$
Result: All symbols removed from the symbol table and the storage pool reinitialized.

STOP Places the Desk Calculator in dead storage and returns to the CTSS supervisor.

Example: STOP$
Result: R XXX.X+XXX.X

Declarative Statements

DEPEND Declares dependence relationships between atomic variables for use with the differentiation operator.

Example: DEPEND (A,B/X,Y)$
Result: A depends on X and Y, B also depends on X and Y.
(NOTE: A does not depend on B nor X or Y or Y on A or B.)

Example: DEPEND (X/Y)
A =DERIV (X**2,Y,1)$
Result: A names the expression:
2*X*DERIV (X,(Y,1))

PARAM Sets up parameter pairs for use with the 'SUBST' command.

Example: ABC = PARAM (B,2), (C,X+Y)$
Result: When the label ABC is referenced in the 'SUBST' command, the number 2 will be substituted for every occurrence of B likewise X+Y will be substituted for C.

Interrupt Level:

The Desk Calculator has one interrupt level which returns to the input routine. When the Desk Calculator types 'READY', the next statement may be typed in. If the Desk Calculator is interrupted during execution of any command except 'PRINT' or 'DUMP' there can be no guarantee that further execution will give correct results. "USER BEWARE".

Errors

Error messages are tabulated in CC-257.

(END)

Identification

Generate line-numbered files
INPUT, EDIT, FILE, TFILE

Purpose

Allows the user to create a line-numbered, 14 words per line, BCD file and edit such a file by specifying the line numbers to be modified. These files are appropriate input for many of the card-oriented subsystems i.e., FAP, MAD, MADTRN, STRESS, etc.

Method

In the INPUT mode, the command generates and prints the line numbers (sequential and incremented by 10) so that the user need type only the data. The line number becomes part of the line as columns 75-80. In the EDIT mode, lines may be deleted, replaced or inserted by manual typing of the line numbers. At FILE time, the EDIT requests are sorted by line number and the later requests override any previous requests for the same line number.

Usage

Input:

```
     user:  INPUT
 response:  00010
```

The user then types a card image per line, according to the format appropriate to the programming language being used. Tabs may be used to specify fields. The carriage return terminates a line and the next line number will automatically be printed (incremented by 10).

A line consisting of only a carriage return will terminate the automatic or INPUT mode and enter the manual or EDIT mode. In the manual mode the following conventions may be followed:

1) A line number (all numeric) followed by a space or tab followed by the desired line allows the deletion, insertion, or replacement of a line of that sequence number. If the line number is followed by a tab, the first field is blank.
2) DELETE, 'n1'-',n2'- where n1 and n2 are previous line numbers. Lines n1 thru n2 will be deleted or just n1 will be

deleted if n2 is not specified. (Note that the commas in this request are optional.)

3) SEQUENCE, 'n1'-',delta'-. The automatic mode is resumed starting with line number n1 and subsequent incrementing by delta. If delta is omitted, the previous increment is retained. (Note that commas are optional)
4) A line consisting of only a carriage return resumes the automatic mode with the next line number after the last line of the file.
5) FILE NAME1 NAME2 terminates EDIT and initiates the FILE command.
6) QUIT signal terminates INPUT and EDIT and the input lines may be lost. (See next page.)

Edit:

user: EDIT NAME1 NAME2
response: XXXXX

XXXXX is the next line number following the last line of file NAME1 NAME2.

EDIT establishes the automatic or INPUT mode and appends to the file NAME1 NAME2. See conventions under INPUT.

File:

user: FILE NAME1 NAME2 -MODE-

NAME1 is the primary name of the file

NAME2 is the secondary name and should be the class of language used during INPUT.

MODE is the mode of the file: 0 is temporary, 1 is permanent, 2 or R1 is read only R1, 3 or R2 is read-only R2. If MODE is not specified, permanent is assumed.

FILE creates a sequenced line-numbered file with the line numbers as right-adjusted sequence numbers in the corresponding card images of the file. If an older file of the same name exists in the user's directory, it will be replaced by the new file. Any corresponding NAME1 SYMTB and BSS files will also be deleted. No message is given regarding this.

TFILE is also recognized as a command in the manual mode. TFILE is used by system programmers for checking-out new versions of FILE.

Error procedure for FILE:

In most cases the following procedure should succeed:

.Comments are printed explaining the cause of the error, and FILE calls DORMNT.

.Type a SAVE command. If track quota is exhausted, give the extra parameter 't' to create a temporary mode saved file.

.Fix up whatever was indicated as the cause of the error (file mode, track quota, etc.)

.CONTIN the SAVED file.

LOGOUT Peculiarities:

The FILE command deals with files which usually are of temporary mode. Consequently, they are lost on a LOGOUT. There is some possibility of losing both the old and the new files. This does not apply in the case of automatic LOGOUT where temporary files are saved.

The following tips may be of some help to restart a FILE command after a LOGOUT break.

.Before LOGOUT, if possible, change the mode of (EDIT FILE) and (FILE FILE) to permanent.

.If (INPUT FILE) is not found, all the input lines typed from the console have been lost, and must be retyped.

.If (EDIT FILE) is not found, rename the old file (EDIT FILE).

.If (FILE FILE) is not found, try a FILE command, without going through INPUT.

Revised: 8/02/65

Identification

Context editor for card image files
ED

Introduction

ED is a command for editing 14-word BCD card image files within CTSS. The command is based on TYPSET (CC-244, MAC-M-193 by J. H. Saltzer) and many of the conventions of TYPSET are used by ED. Tabs are automatically interpreted * for FAP, MAD, MADTRN, GPSS, COMIT, and ALGOL(i.e., AED) programs. Tabs may also be set by the user for other purposes. Although line numbers may be generated by the ED command, editing is done entirely by context. The ED command is offered as an alternate to the present INPUT, EDIT and FILE commands.

Usage

The ED command is initiated with the following CTSS command.

ED -NAME1- NAME2 -NAME3-

NAME2 is the secondary name of the file to be edited or created and must be provided. NAME1 is the primary name of the file to be edited. If NAME1 NAME2 is not specified, ED will assume that a new file is to be created and will start in the high-speed INPUT mode. If NAME1 is provided, the command will look for the file NAME1 NAME2. If the file is not found, the high-speed INPUT mode will be entered. If the file is found, the EDIT mode will be entered.

If NAME3 is specified and the file NAME1 NAME2 is found, the subsequent FILE will create a file NAME3 NAME2 and NAME1 NAME2 will remain unaltered. Any arguments to the FILE request, however, will take precedence.

New files will be created only on the disk (device 2) and * old copies resident on the drum will disappear if modified. The original file to be edited may be a linked file, however, any attempt to replace a linked file by the edited version will be rebuffed. The modified version must be filed under a different name. Linked files of the name (INPUT FILE) and (INPTI FILE) may not exist in the file directory or ED will not function.

HIGH-SPEED INPUT MODE:

When the user enters this mode, the ED command will type "INPUT:" on the user's console. While the user is operating in this mode, the ED command will accept input lines from the user's console. Tabs will be interpreted automatically for each input line. Backspace characters may also be used to move back one character position in the input line. No response is typed for input lines and as a result, the user may type successive lines as fast as he wishes. When the user types a line consisting only of a single carriage return, the ED command will place the user's console in the EDIT mode.

EDIT MODE:

When the user enters this mode the response "EDIT:" will be typed on the user's console. At this time the user may type requests to the ED command. All changes made to a file become effective immediately and as a result, the user is able to make recursive modifications to his file. We may think of a pointer which is positioned at a line in the edited file. When the user enters the EDIT mode from the INPUT mode, this pointer will be positioned at the last input line typed by the user. When the user starts the ED command in the EDIT mode, the pointer is positioned before the first line in the old file. If the end of file is reached by an EDIT request, the comment "END OF FILE REACHED BY:" is typed on the user's console followed by the request which caused the end of file to be reached. At this time the pointer will be positioned after the last line in the file. When in the EDIT mode, any line which is not a legitimate EDIT request will cause the comment "NOT A REQUEST:" to be typed on the user's console followed by the line which caused the error. In many cases it is possible for the user to stack EDIT requests. If one of the requests causes an error message to be typed, any stacked requests will be ignored. This is done in case one of the stacked requests depended on the successful completion of the request in error.

Any number of initial tabs or spaces (including 0) may occur in a request line. Arguments and the request must be separated by at least one space or any number of tabs or spaces. Wherever the argument is line image, however, tabs and spaces retain their normal significance.

Error messages

Some errors from the file system will result in a PRNTER type error message followed by a question to the user of

whether or not he wishes to continue. An answer of 'NO' will result in a call to DORMNT so that the user may SAVE the command, fix the problem, and RESUME the command.

EDIT REQUESTS:

```
REQUEST:        FIND line
ABBREVIATION:   F
RESPONSE:       none
ERRORS:         END OF FILE
```

The FIND request is used to move the pointer forward from its present position to the line specified by "line". "Line" is a normal input line and may contain tabs and backspaces. This line is used as a mask for selecting the desired line in the edited file. Matching is done only on the non-blank characters specified in LINE. For example, the request,

```
F (tab)-(tab)-ALPHA,1
```

might be used to find the line,

```
LOOP    TIX     ALPHA,1,4
```

```
REQUEST:        LOCATE string
ABBREVIATION:   L
RESPONSE:       none
ERRORS:         END OF FILE
```

The LOCATE request is used to move the pointer forward from its present position to the first line which contains the entire character string specified by "string". The full line of 84 characters is scanned, so that "string" may specify line numbers. It is recommended that "string" include the leading zeros of the line numbers to avoid any undesired match with program constants.

```
REQUEST:        NEXT I
ABBREVIATION:   N
RESPONSE:       none
ERRORS:         END OF FILE
```

This request is used to move the pointer forward from its present position in the file. "I" specifies the number of lines to be skipped over. If I is "0" or not specified, it is assumed to be "1" and the pointer will be moved to the next line in the file. If the NEXT request is given after the end of file has been reached, the pointer will be reset to the beginning of the file and moved "I" lines from there.

```
REQUEST:        DELETE I
ABBREVIATION:   D
```

RESPONSE: none
ERRORS: END OF FILE

The DELETE request will delete "I" lines from the file starting with the line at which the pointer is currently positioned. The pointer is left at the position vacated by the last line deleted by this request. If I is "0" or left unspecified, only the current line will be deleted.

REQUEST: PRINT I -L-
ABBREVIATION: P
RESPONSE: printed lines
ERRORS: END OF FILE

The PRINT request will print "I" lines from the file starting with the line at which the pointer is currently positioned. Upon completion of this request, the pointer will be left pointing to the last line printed. If I is "0" or left unspecified, one line will be printed. Normally lines are printed without line numbers. If the character "L" is present in the PRINT request, line numbers will be printed to the right of the printed lines.

REQUEST: RETYPE LINE
ABBREVIATION: R
RESPONSE: none
ERRORS: none

This request will cause the line at which the pointer is currently positioned to be replaced by LINE. LINE is a normal input line and may contain tabs and backspaces. The pointer is not moved by this request.

REQUEST: TOP
ABBREVIATION: T

This request will cause the pointer to be reset and positioned before the first line in the file. In addition, * an automatic TOP is performed by the FIND, LOCATE and NEXT if the pointer was positioned at the end of the file.

REQUEST: BOTTOM
ABBREVIATION: B
RESPONSE: INPUT:
ERRORS: none

This request will cause the pointer to be positioned after the last line in the file. Upon completion of this request the user's console will be placed in the high-speed INPUT mode. All subsequent lines will be treated as input and

added to the end of the file.

```
REQUEST:        INSERT or (C.R.)
ABBREVIATION:   I
RESPONSE:       INPUT:
ERRORS:         none
```

This request will cause the user's console to be placed in the high-speed INPUT mode. All subsequent lines will be treated as input and inserted after the line at which the pointer is currently positioned. If the INSERT request is given immediatly following a TOP request, the inserted lines will be placed at the beginning of the file.

```
REQUEST:         INSERT line
ABBREVIATION:   I
Response        none
Errors:         none
```

The INSERT request may be used to insert a single line without changing to the high-speed input mode. Line is a normal input line. It is inserted following the line at the present pointer position.

```
REQUEST:        CHANGE Qstring1Qstring2Q I G
ABBREVIATION:   C
RESPONSE:       none
ERRORS:         END OF FILE
```

This request will examine "I" lines starting at the line at which the pointer is currently positioned. Upon completion, the pointer will be left positioned at the last line examined by this request. If I is "0" or left unspecified, it is assumed to be "1" and only the current line will be examined. The character "Q" is taken to be the delineator or "Quote character" and may be any character in the 6-bit BCD set. "string1" and "string2" are arbitrary BCD character strings and may be of different lengths. If the character "G" (GLOBAL) is present, every occurrence of string1 will be replaced by string2. If "G" is not present, only the first occurence of string1 will be replaced by string2 in each examined line. EXAMPLES:

```
line:              ALPHA= ALPHA+ALPHA
request:    C *ALPHA*BETA*
new line:          BETA= ALPHA+ALPHA
request:    C *ALPHA*DELTA* 1 G
new line:          BETA= DELTA+DELTA
request:    C *DELTA+**
new line:          BETA= DELTA
```

```
REQUEST:          BLANK line
ABBREVIATION      BL
RESPONSE:         none
ERRORS:           none
```

The BLANK request will put blanks in the current line wherever non-blank characters appear in "line". For example 'BL ******' will clear the label field of a line in a FAP file.

```
REQUEST:          OVRLAY line
ABBREVIATION:     O
RESPONSE:         none
ERRORS:           none
```

The OVRLAY request will place the non-blank characters of "line" into the corresponding position of the current line. Notice that only non-blank characters of "line" replace what was in the current line. For example in a FAP file, if the current line is

```
                          TXI       *+1
```

then

```
          0  EOF(tab)   bbH  (tab)(tab)    comment
```

will produce

```
             EOF         TXH       *+1       comment
```

```
REQUEST:          VERIFY
ABBREVIATION:     VE
RESPONSE:         none
ERRORS:           none
```

The VERIFY request sets the verify mode. In the verify mode, completion of any of the requests FIND, NEXT, LOCATE, OVRLAY, BLANK and CHANGE will cause the printing of the current-pointer line. In addition, CHANGE will cause the printing of all changed lines. Requests may not be stacked while in the verify mode.

```
REQUEST:          BRIEF
ABBREVIATION:     BR
RESPONSE:         none
ERRORS:           none
```

The BRIEF request sets the brief or normal mode. Within the brief mode, the FIND, NEXT, LOCATE, OVRLAY, BLANK, CHANGE requests will not give the responses expected in the verify mode.

REQUEST: CLIP 'ON' or 'OFF'
ABBREVIATION: CL
RESPONSE: none
ERRORS: ILLEGAL ARGUMENT:

The request CLIP ON sets a mode such that any input line which exceeds column 72 will cause the message "TRUNCATED:" followed by the faulty line image. Any waiting input lines will have been deleted. Requests on which this may occur are FIND, INSERT, RETYPE, OVRLAY, BLANK and high-speed INPUT. The request CLIP OFF resets the mode . The normal mode is CLIP ON for all files except FAP files which are normally CLIP OFF.

REQUEST: SERIAL N
ABBREVIATION: S
RESPONSE: none
ERRORS: none

This request is used to change the increment between line numbers of successive lines to the increment specified by the decimal integer "N". Initially, this increment is set to 10 by the ED command. If N is "0" or not specified, it is assumed to be "10". Lines inserted after a line with the line number "L" will be sequenced L+N, L+2N, L+3N, etc. If the lines following the inserted lines have line numbers which are less than or equal to the line number of the last inserted line, as many lines as necessary will be resequenced to insure that all line numbers are unique and in ascending order. For example, assume that "N" is 2 and the user wishes to insert 9 lines after line 25 in a file that was previously sequenced by fives. The inserted lines would be numbered, 27, 29, 31 ... 43. The lines previously numbered, 30, 35, 40, 45 and 50 would be renumbered to, 45, 47, 49, 51 and 53 respectively. The remaining lines in the file would be unchanged.

REQUEST: COLON a
ABBREVIATION: CO
RESPONSE: none
ERRORS: ILLEGAL ARGUMENT

A colon (or backspace on 1050) is a logical backspace anywhere, eg., 'ABC ::: Db(C.R.) is interpreted as 'DbC'. The colon moves the character pointer back one but does not erase the characters over which it has moved. One should be careful in using this convention that the total number of characters does not exceed 84, as any extras will be added to the next line during INPUT, or result in a request during EDIT.

The COLON request allows the colon character to be inserted as text. (They may also be 'CHANGE'd in as desired.) If 'a' is T or TEXT, all ':' will be treated as text except for the ':' as the first character after a tab. If 'a' is B or BACKUP the normal mode will be reinstated and all ':' will be backspaces.

```
REQUEST:        TABSET T1 T2 ... TN
ABBREVIATION:   TA
RESPONSE:       none
ERRORS:         ILLEGAL TAB SETTING
```

Ti specify the columns at which tabs are to be set. Tabs must be set in ascending order and may not exceed column 72.

```
REQUEST:        FILE -NAME4-
ABBREVIATION:   FL
RESPONSE:       Ready message from CTSS
ERRORS:         NO FILE NAME GIVEN
          or    FILE WORD COUNT ZERO
                NOTHING IN FILE
                INPUT:
```

This request is used to terminate the editing process and write the new edited file on the disk. NAME4 specifies that the new file will be created as NAME4 NAME2. If NAME4 NAME2 is not specified, the old file will be replaced by the edited file or a new file NAME3 NAME2 will be created. If no name was given by the initial ED command or by the FILE request, an error message will be printed and the FILE request will be ignored.

If a file to be deleted is either READ-ONLY or PROTECTED, confirmation of deletion will be requested. If confirmation is denied or if file is LINKed, the EDIT mode will be reentered with the pointer at the top of the file. Any modes associated with the previous copy of the file will be transferred to the new copy.