Major Revision: 5/23/66

Identification

Save present dormant program.
SAVE, MYSAVE

Purpose

The user may preserve a currently-dormant program (e.g., just loaded, interrupted by the quit button, file system errors for which no error return was specified, or called DORMNT) and its machine conditions via the commands SAVE or MYSAVE. Execution may be either begun or continued at some later time by use of RESUME or CONTIN; the core image, et al., may be reinstated at some later time by use of RECALL or RESTOR (see AH.7.03).

Usage

```
SAVE -NAME1- -'T'-
MYSAVE -NAME1- -'T'-
```

SAVE In addition to the core image and machine conditions, SAVE will save the status of any active files so that they may be repositioned by RESUME and RESTOR. It will also save any command chain present.

MYSAVE In addition to the core image and machine conditions, MYSAVE will save the status of any active files in the current file directory, but will then switch to the user's file directory before creating the SAVED file. This is the version used by automatic logout. Resumption of the SAVED file from the user's file directory by RECALL or CONTIN will perform the necessary switch of directories.

NAME1 The created file is given the name NAME1 SAVED.

T The SAVED file will be created in temporary mode.

If no arguments are furnished, the file created representing the current state of the user's program will be given the first name "PROGN", the user's problem number, and the generic second name 'SAVED'.

If a file already exists which has the same name as will result from a SAVE or MYSAVE, the old version will be truncated; the old version's mode is, then, preserved.

Error Conditions

'MEMORY BOUND ZERO, NO SAVED FILE CREATED'.

means that the user was in DEAD status and there was no program to SAVE.

NAME1 'SAVED BUSY, SAVE NOT EXECUTED'.

means that another user was referencing the file NAME1 SAVED (or perhaps the user himself forgot to close it). The SAVE or MYSAVE must be repeated, perhaps with a new NAME1.

(END)

Identification

Saving and renaming temporary file generated by RUNCOM
SAVFIL, RERUN

Purpose

In order to preserve the user's core image and machine conditions as needed during a chain of commands, RUNCOM generates a series of temporary mode SAVED files with primary names of the special form ...00n when n=1,2,3, etc. The problem arises of preserving these files when a a RUNCOM is SAVED in midstream, and desired to be CONTINued at a later time - either in a subsequent LOGIN session, or after another RUNCOM. SAVFIL and RERUN are designed to preserve these files.

Usage

```
SAVFIL    NAME1

RERUN     NAME1
```

SAVFIL works on the unbroken chain of SAVED files with primary names of the form ...00i, i=n,n-1,...,2,1 where ...00(n+1) SAVED does not exist. Working in decreasing value of n, it renames the file ...00n SAVED to an unused name of the form $$$00j and makes it permanent mode. Finally, it appends the list of new names $$$00j to the file NAME1 SAVED.

REFUN restores these files to their original names and mode form the information continued in the file NAME1 SAVED. NAME1 SAVED is unchanged.

The recommended (and probably ONLY) way to use these commands is as follows:

```
MYSAVE  NAME1      to save a RUCOM job.
SAVFIL  NAME1
   .
   .
   .
RERUN   NAME1      to continue the RUNCOM at
CONTIN  NAME1         any future time
```

As automatic logout performs the MYSAVE but not the SAVFIL, a good practice would be to issue a SAVFIL LOGOUT immediately at one's next LOGIN, if it is desired to CONTINue the job at a later time.

Both SAVFIL and RERUN operate only in the user's file directory.

Revised: 5/31/66

Identification

Link to files in other U.F.D'S
LINK, UNLINK, PERMIT, REVOKE

Purpose

A user may allow files in his directory to be accessed by other users by means of a mechanism known as "linking". The users who have been allowed to form "links" or "link pointers" (U.F.D. entries which point to other U.F.D. entries instead of to the file itself) to a file need not, then, have a copy of the file in their own directories. It is also possible to establish links which have names other than those of the actual file. When he grants permission to link, the "owner" of a file specifies who will be permitted access and what apparent mode the accessors will have to treat the file in. *

Usage

If any of these commands is typed without arguments, the response will indicate the proper format.

1) Grant permission:

PERMIT NAME1 NAME2 MODE PROB PROG

NAME1 NAME2 is the name of the file in the current file directory to which the author is granting linking permission. The file NAME1 NAME2 need not exist, may exist in any mode, or may be a link pointer in the current file directory to a file or link pointer in some other directory. Linking permission, therefore, may be granted to any file to which the current file directory has access or may have access in the future. PERMIT does not actually establish a link. If NAME1 is *, all primary names are implied. If NAME2 is *, all secondary names are implied.

MODE is the mode which the author wishes to permit for the file. During the linking process, this mode will be 'or'ed with any other modes in the chain of links to determine the final mode. The mode may be octal or alphabetic with the following correspondence:

```
0   1   2   4   10   20   100                    *
0   T   S   R    W    V    P
```

PROB PROG specifies the problem number and programmer number of the user to whom the file NAME1 NAME2 is being permitted. If PROB is * all problem numbers are implied. If PROG is * all programmer numbers are implied.

2) Withdraw permission:

REVOKE NAME1 NAME2 PROB PROG

REVOKE withdraws the linking permission for file NAME1 NAME2 of the current file directory from the user PROB PROG. Note that REVOKE does not remove any links that have already been made.

3) Form a link:

LINK NAME1 NAME2 PROB PROG -NAME3- -NAME4-

LINK establishes a link in the current file directory to the file NAME1 NAME2 in the file directory of PROB PROG.

When NAME3 NAME4 is specified, NAME1 NAME2 is the name given to the file in the current directory and NAME3 NAME4 is the name of the file in the directory PROB PROG. If NAME4 is not specified, NAME2 will be used as the class name.

If permission has not been granted, the link cannot be established. Links may be established through a depth of file directories which is currently set by the file system to two.

4) Remove a link:

UNLINK NAME1 NAME2 ... NAME1n NAME2n

UNLINK will remove links to files so specified. If '*' is used as a primary name all files with given secondary name will be unlinked. If '*' is used as secondary name, all files with specified primary name will be unlinked. If UNLINK * * is typed, all links are removed.

This command in no way affects permission.

NAME1 NAME2 must be the name by which the file is known in the current file directory.

Method

The PERMIT command establishes a file named PERMIT FILE (VP mode) in the directory of the user giving permission. This file is line-marked, and may be printed out with the PRINT command. As the command is no longer privileged, "old" PERMIT FILEs (in Read-Only, Protected mode) will cause error returns when PERMITing, and should be changed to the new mode. In the case of problem numbers which have common file directories, a PERMIT FILE in a common file should probably be maintained by a designated member of the group. *

(END)

Revised: 5/9/66

Identification

Tape-handling commands
MOUNT, UMOUNT, VERIFY, LABEL, TAPFIL

Purpose

These commands have been added to CTSS to facilitate reading and writing of tape files using the standard file system calls. To use a tape, the Tape Strategy module must be told to mount it, its standard file header must be read and checked, or written, and the file system must be told that it is a tape file. (See also Section AG.5.05 for additional information about tape usage in foreground.)

Restriction *

To use foreground tapes, a user must have a administratively-assigned tape record quota. Because the use of tapes makes unusual demands on both the system and the operators, assignment of such quotas will be the exception rather than the rule.

Usage

1) Mount a tape:

MOUNT NAME LOGUNT -RING- -CHAN- -MESS-

NAME is the name or reel number of the reel to be mounted.

LOGUNT is a logical unit number by which the user wishes to refer to this tape. Any number (L.E. 2.P.18) will do, providing it is one that the user has not already used.

RING may be either 'RING' or 'NORING'. It specifies whether or not the reel should be file-protected. 'NORING' will be assumed, if not specified (i.e., file-protected).

CHAN may be '1' or '2' in the current system for channels A or B, respectively. If not specified, the supervisor will pick a channel.

MESS if present, must be either the characters 'MESS' or '(MESS)'. If the former, the supervisor will then type 'TYPE'. Up to 12 words of message to the operating staff will be accepted. If the latter, it must be followed by NAME1 NAME2, which refer to a card *

image (line-numbered) file containing the desired message.

The optional parameters may appear in any order.

The message sent to the operators is of the form:

```
FOR TAPE REFERRED TO IN  FOLLOWING MESSAGE, ASSIGNMENT IS  A8
USER  PROBNO=  M1416     USER PROGNO=   3
MOUNT TAPE 199 WITH NORING ... user comment ...
```

2) Dismount a tape:

UMOUNT LOGUNT -MESS- -RING-

where LOGUNT, MESS, RING are as above.

3) Verify the label on a previously-written tape:
(This must be done before opening the file.)

VERIFY LOGUNT -FILE- *

LOGUNT is a logical unit previously referred to by a 'MOUNT' command.

The program will say 'TYPE LABEL'. The 24 characters typed next must correspond to the label on the tape. If the tape cannot be mounted or the label does not match, a message will be printed. *

FILE refers to NAME1 NAME2 of a card image (line-numbered) file, from which the first 24 characters will be taken as the label. This option overrides the console typing just described.

4) Write a label on a tape:

LABEL LOGUNT -FILE- *

LOGUNT is a logical unit number referred to by a previous 'MOUNT' request.

The program will ask for a 24-character label, which will be written on the tape to provide the label which will be VERIFYed if the tape is to be read again. If the tape is bad or cannot be mounted, a comment will be printed.

FILE is the same as under the VERIFY description. *

5) Declare a file to be on tape:

TAPFIL NAME1 NAME2 LOGUNT -FILENO-

NAME1, NAME2 is the name of the file.

LOGUNT is a logical unit number, as in 'MOUNT', etc.

FILENO if specified, is the number of the file on the reel specified by LOGUNT. A FILENO of zero specifies the end of a set of files on a reel, so that this may be used to add to the end of a reel. FILENO is assumed zero if not specified.

(END)

Revised: 3/30/66

Identification

Context editor for 6-bit mode
EDL
J. H. Saltzer, X6039

Purpose

EDL is a context editor for line-marked, 6-bit BCD files. SQUASH SAVED is available in the public files to change currently-existing card-image (line-numbered) files to the line-marked format, which is currently acceptable to EDL, MAD, and FAP. (Files created by EDL itself are, of course, line-marked.) A significant saving of both space and time will be effected by the use of EDL instead of ED.

Usage

EDL NAME1 NAME2

NAME1 NAME2 is the name of the file to be edited.

Editing conventions are identical to those of the TYPSET command (Section AH.9.01), except that only the 6-bit character set may be used. In addition to the TYPSET erase (#) and kill (@) characters, EDL also accepts the standard CTSS erase (") and kill (?) characters. A backspace character will be set in the file as a colon. Tab characters will be inserted in the file wherever typed.

Error Condition

'INPUT FILE HAS IMPROPER FORMAT.' will be printed if EDL is being used on a file which is not correctly line-marked. In particular, this condition will occur if the file is of card-image format. To prevent damage to the file, quit out of the command (do not use the 'file' request) and either SQUASH the file or use ED.

(END)

Revised: 11/19/65

Identification

Combine seldom-used files
ARCHIV

Purpose

To combine files which are not frequently used so that the single archive file occupies fewer tracks than the many smaller files. The average saving is half a track per file. Individual files may be combined, listed, printed, deleted and recreated.

Restrictions

Files of any class or secondary name may be archived, however, only files of one class may be combined into a single archive file of the same class.

Files to be deleted are deleted by the standard convention of DELETE which are here restated under Method.

Usage

ARCHIV KEY NAME1 NAME2 FIL1 ... FILn

KEY=C: Combine files FIL1 NAME2 ... FILn NAME2 into an archive file NAME1 NAME2. Any old files NAME1 NAME2 will be deleted, if possible. FIL's are not deleted from the user's file directory.

KEY=P: Print file(s) FIL1 ... FILn which is (are) contained in archive file NAME1 NAME2. Card image and standard line-marked files may be printed. *

KEY=T: Print a table of contents of archive file NAME1 NAME2. If FIL1 ... FILn are specified only these will be listed. *

KEY=D: Delete FIL1 ... FILn from the archive file NAME1 NAME2. This involves creating a new archive file and deleting the old one with the standard hocus pocus of deleting.

KEY=X: Extract and copy FIL1 ... FILn from archive file NAME1 NAME2. The copy is named FILi NAME2 and any old copies are deleted.

KEY=XT: Same as X except that the file is extracted in 'temporary' mode. *

KEY=R: Replace each FILi in the archive file NAME1 NAME2 with a copy of the file FILi NAME2. This involves creating a new archive file and deleting the old one. If no FILi exists within the archive file. FILi NAME2 will be appended.

KEY=RD: Same as R except that after the new archive file has been successfully created and filed, all the files which were placed into the archive are deleted. *

KEY=U: The files specified are replaced in NAME1 NAME2 if the copy of the file in the user's directory has 'date and time last modified' greater than the date and time the corresponding entries were placed in NAME1 NAME2. If no FILi's are specified, all the entries in NAME1 NAME2 are updated in this manner. *

Whenever no FIL's are specified in the command call, unless KEY=D, the command is taken to be universal, i.e., the call includes every entry in the archive. *

Method

Each entry in the archive file consists of a header, in which file name, date and time last updated, and the number of words in the file are indicated, followed by a copy of the file. The word count in the header makes it unnecessary to pad the file, so that the file can be reproduced absolutely faithfully.

The header is 14 words long, consisting of four words of (777777000000)8, one word of (777777000011)8, and nine words of self-explanatory BCD information about the file. Thus a program which does not recognize line marks can still read the archive file (since the header is 14 words long), but programs which do recognize line-marks will see the header as four null records (carriage returns) plus the file entry information. If card image files are ARCHIVed, the header record will cause some programs to abort because of illegal file format since there will be a mixture of line marks and card images (e.g. disk editor).

Whenever ARCHIV creates a new file, it is first named 'prob prog', where prob is the user's problem number and prog is the user's programmer number. After this file is created, the file which it replaces, if any, is deleted and file 'prob prog is renamed and the mode changed to permanent or to the mode of the old file if it existed.

Files to be deleted are handled in the standard DELETE manner, i.e., verification is requested for protected, private, read-only, etc., files. If a file cannot be deleted, the new file may be found under the name 'prob prog'.

Identification

Compress BCD files
CRUNCH

Purpose

To compress a BCD file in such a way that it occupies less disk space and, incidentally, is in a form acceptable as input to BEFAP.

Usage

Crunch:

CRUNCH 'CR' NAME1 -NAME2- -'PUNCH'- -'72COLM'-

CR directs the crunching of file NAME1 NAME2 into a file NAME1 CRUNCH. If NAME2 is omitted it is assumed to be FAP.

PUNCH directs the crunching of file NAME1 NAME2 into a file NAME1 PUNCH which is in a form suitable for BPUNCH with RQUEST.

72COLM directs the crunching of only columns 1-72 of the source file. This results in additional space saving and the sequence numbers may be reconstructed during uncrunching.

The order and presence of PUNCH and 72COLM are optional.

Uncrunch:

CRUNCH 'UN' NAME1 -NAME2- -'PUNCH'- -'NUMBER'- -MAJ- -SEQ-

UN directs the reconstruction of the source file NAME1 NAME2 from the crunched file NAME1 CRUNCH. If NAME2 is omitted, it is assumed to be FAP.

PUNCH directs the uncrunching of NAME1 PUNCH rather than NAME1 CRUNCH.

NUMBER directs the resequencing of the source file NAME1 NAME2. In the absence of MAJ and/or SEQ, the first three non blank characters of NAME1 will be used in cols 73-75 and sequencing will begin with zero with increments of ten. The order of 'PUNCH' and 'NUMBER' is optional.

MAJ if specified in conjunction with 'NUMBER', the first three non blank characters are placed in

columns 73-75 of the source file NAME1 NAME2.

SEQ if specified in conjunction with 'NUMBER', causes sequencing to begin with SEQ. The fixed increment is ten.

Print:

CRUNCH 'PR' NAME1 -'PUNCH'- -'NUMBER'- -LABEL- -SEQ-

PR directs the printing of NAME1 CRUNCH

PUNCH directs the printing of NAME1 PUNCH rather than NAME1 CRUNCH.

SEQ is numeric to specify begin printing with card of sequence number SEQ.

NUMBER SEQ begins the printing with alter number SEQ

LABEL is alphanumeric to specify begin pointing with card containing LABEL in columns 1-6. The sequence numbers will appear on the left of the listing.

NUMBER LABEL begins the printing with the card with LABEL in cols. 1-6. The alter numbers will be printed on the left of the listing.

Revised: 1/3/66

Identification

List contents of file directory
LISTF

Purpose

To provide a command which lists the contents of a file directory, with numerous selectivity features if desired.

Description

LISTF enables the user to selectively list the contents of a file directory by permitting him to specify the

1. file directory
2. file names
3. authors
4. modes
5. range of dates last used
6. range of dates last modified
7. sorting process
8. output form

to be employed.

The user has the option to supress the search for linked files or to search only for linked files.

Usage

A. Basic

The basic call - LISTF - will first produce a one-line summary of the number of nonlinked files and the number of records in the user's current directory. This is followed by a table of nonlinked files in the form

NAME1 NAME2 MODE NREC DATE(last used)

sorted according to the date last used with the most recent date first. This is followed by a one-line summary of the number of linked files and a table of linked files in the form

NAME1 NAME2 MODE(in user's directory) PROBN PROGN LNAME1 LNAME2

alphabetically sorted with respect to the primary file name, where the last four items refer to the "other end of the link."

B. Options

The selectivity features and their usage are described on the pages which follow.

Conventions

1. Arguments are divided into four classes.

   a. meta-arguments (defined inductively from the Specifications Tables below)

   b. modifiers (defined inductively from the Specifications Tables below)

   c. file names - all arguments which cannot be identified as meta-arguments or modifiers

   d. special characters

   1) carriage return
   2) '
   3) (and)
   4) *

2. A request is a string of arguments terminated by a single quotation mark (') or by a carriage return.

3. A call is the command LISTF followed by a string of requests and terminated by a carriage return.

4. The order of the arguments is unimportant, aside from the following considerations:

   a. modifiers must immediately follow the meta-argument which they modify

   b. when sorting by dates, the list will begin with the first date specified

   c. two primary file names must be separated by a secondary name or by a meta-argument

5. Up to 19 arguments may be specified in one call to LISTF.

6. One interrupt level is set to enable the user to terminate the request being processed and begin the next. (WARNING: some output may be lost.)

7. If the user quits when he is listing linked files in a common file in the long form, his directory switching will probably not be restored. (This condition can, of course, be corrected by issuing a COMFIL 0 command.)

File Names

An asterisk (*) embedded in a file name specification refers to any and all characters in that position. A single * as a file name means any and all names.

EXAMPLES:

CTEST* means any name with "CTEST" as the first five characters, i.e., CTEST1, CTESTS, but not bCTEST, where "b" denotes blank.

TEST means any name with "TEST" as the 2-5 characters, i.e., CTEST1, bTESTS, but not bbTEST or TEST12.

** means any 1 or 2 character name.

If the secondary file name is omitted, * will be assumed.

If no file names are specified, * * will be assumed.

SPECIFICATIONS TABLES

SEARCH SPECIFICATIONS

META-ARG.	MODIFIERS	ACTION	DEFAULT	COMMENTS
(FILE)	NONE	ignores linked files		
(LINK)	NONE	ignores non-linked files		
			accepts both linked and nonlinked files	
(UFD)	name of a file linked to some file "U.F.D. (FILE)"	searches the files in the linked file directory	U.F.D. (FILE)	
(SYS)	NONE	searches the system files	current directory	
(CFIn)	NONE	searches the user's nth common file	current directory	
(AUTH)	authors number(s)	only accepts non-linked files written by user(s) specified	any and all authors	
(MODE)	1 or more descriptions, each comprising 1-4 of: O,T,S,R,W, V,L,P,* enclosed by parentheses	ignores files with modes not requested	any and all modes	e.g. (RP) means read-only, prot. (RP*) means at least read-only, prot. (R)b(P) means read-only or prot.
(USED)	up to 2 dates, each in the form MMDDYY, or '(OLD)', or '(OLD)', or '(NEW)'	only accepts non-linked files used between between dates, incl.	(NEW) (OLD)	(OLD) means earlies date (NEW) means (NEW) means present date
(MADE)	same as (USED)	only accepts non-linked files modified between dates, incl.	(NEW) (OLD)	same as (USED)

SORTING SPECIFICATIONS

META-ARG.	MODIFIERS	ACTION	DEFAULT	COMMENTS
(SDIR)	NONE	file directory order		
(SNA1)	NONE	alphabetic order NAME1		
(SNA2)	NONE	alphabetic order NAME2		
(SMOD)	NONE	sorted on(Octal) modes decreasing order		
(SREC)	NONE	number of records		nonlinked files *
(SUSE)	NONE	date last used		nonlinked files *
(SMAD)	NONE	date last modified		nonlinked files *
			(SUSE) for nonlinked files (SNA1) for linked files	
(REV)	NONE	reverses the order of sorting		

*
Listing of linked files will be suppressed in requests containing these meta-arguments.

OUTPUT SPECIFICATIONS

META-ARG.	MODIFIERS	ACTION	DEFAULT	COMMENTS
(LSUM)	NONE	summary lines only		
(LNAM)	NONE	NAME1-NAME2 table		
(LONG)	NONE	normal form plus TIME and DATE (last modified), AUTH.,DEV.,LOCK for non-linked files; normal form plus MODE ("or"ed),DATE last used), DATE (last modified), AUTH.,NREC,DEV. for linked files		if (UFD) was requested, linked files will be printed in the normal form
			normal form- (see USAGE)	

Example 1

```
              (1) (2)   (3)      (4)  (5) (6)  (7)
        LISTF  *  BSS  GAMMA*  (AUTH) 1   2  (LØNG)

 (8)   (9)  (10)  (11)  (12)   (13)   (14)  (15)
(MØDE) (RP) (W*) (AUTH) 99999 (CFL2) (USED) 080165

 (16)  (17)    (18)   (19)
(SDIR) (MADE) 080165 080165
```

(7)
would produce a table in the long form, in the order
(16)
of the file directory, of all files with the following properties

(4),(11),(14),(17)
1. non-linked

(1),(2)
2. secondary name "BSS" and/or primary
(3)
name "GAMMA (any character)"

(5),(6),(12)
3. written by user no.s 1, 2 or 99999

(9)
4. in read-only, protected mode or has write-only
(10)
bit set

(13)
5. in common file 2

(15)
6. last used on or before 9/1/65

(18),(19)
7. last modified on 8/1/65

where the superscripts are, of course, for reference only.

Example 2

If present date is 12/31/65 and all files in the directory were last used between 1/31/65 and 12/31/65, inclusive, then the following requests would produce identical tables (consisting of all the non-linked files used from 1/31/65 to 12/31/65, inclusive, in the normal form beginning with the file last used).

```
 1.   (USED)
 2.   (USED) (NEW)
 3.   (USED) 123165
 4.   (USED) (NEW) (OLD)
 5.   (USED) (OLD) (REV)
 6.   (USED) 013165 123165 (REV)
 7.   (USED) 123165 013165
 8.   (USED) (NEW) 013165
 9.   (USED) 013165 (NEW) (REV)
10.   (FILE)
11.   (AUTH)
12.   (MADE)
13.   (REV)(REV)(USED)
14.   (SUSE)
```

Identification

Print BCD card image files
PRINTF

Purpose

To print the contents of BCD card image files (line-numbered) either from the beginning of the file or from some specified line number.

Usage

PRINTF NAME1 NAME2 -SEQ-

PRINTF prints the contents of file NAME1 NAME2 by printing first characters 73-80 and then characters 1-72 so that the line numbers will appear on the left.

SEQ specifies the numeric portion of the columns 73-80 of the initial line to be printed. If SEQ is omitted, the beginning of the file is assumed. If SEQ does not match any line number, the next higher line number in the file will be used.

Major Revision: 1/3/66

Identification

Print a BCD file
PRINT

Purpose

To print the contents of a BCD file, which can be either line-numbered or line-marked, and either 6- or 12-bit mode. Specific lines and special format may be requested. Further, the command will function if the user is ATTACHed to another directory.

Restrictions

There is a limit of 22 words per record for 6-bit mode files, and a limit of 132 words per record for 12-bit mode files.

Usage

PRINT NAME1 NAME2 -LINES- -FIELDS- -'TAB'- ... -'TAB'- -'(FULL)'-

PRINT will normally print line-numbered files in the format: characters 73-80, blank, then characters 1-72. Line-marked files will be printed from character 1 through the last character, with 132 characters per line of type.

LINES (optional) may specify which lines or records should be printed if other than the initial line is desired. The specification may be one of three forms:

1) s — from s thru the end of file
2) s 'TO' e — from s thru e
3) s 'THRU' e — from s thru e

where s and e are decimal digits which are interpreted as line-numbers or record numbers. Line-numbers are matched against the right-most numeric field of card image files. Record numbers identify variable-length records by their numeric order, beginning with 1.

Line-numbers are assumed for card image files. The mode is switched to record number upon encountering any line-marked record. Using THRU instead of TO causes setting to the record number mode.

FIELDS may be specified only if LINES is not void (it may be 0 or 1). FIELDS comprises any number of pairs of decimal numbers from 1 to 132, of general form a1 b1 a2 b2....an bn. The PRINTed line will be a concatenation of every field specified by the position in the record read from the file, as from the ai character through the bi character.

Ai and bi may be in any order, and the fields are independent of each other. A field may be partly or entirely repeated and also printed in reverse order. If a specification field exceeds the length of a record, the outside characters will be set blank. If the last bn is omitted, it is assumed equal to an, defining a single character field.

TAB will cause tabular spacing to occur between each of the fields specified in the FIELDS list; each additional appearance of 'TAB' will cause additional "tab" to be inserted. On 1050 consoles, the left-hand margin should be set at 0 or 1, and the tab settings should be at every fifteenth position (i.e., 0, 14, 29... or 1, 15, 30 ...).

(FULL) causes the command to operate on files in 12-bit mode (e.g., '(MEMO)'-class files).

Title: A line of information will be printed to provide file name, date, and time if and only if the printing is to begin with the first record of the file and TO or THRU is not specified.

Break: An interrupt signal will stop the printing and terminate the command. The command terminates by calling CHNCOM.

Identification

Print contents of a file in octal
PRBIN

Purpose

Print on the user's console the contents of a file in octal. It may be used to examine SAVED or BSS files or BCD files which might contain illegal characters.

Usage

PRBIN NAME1 NAME2 -'O'- -N- -R-

O (optional) the presence of O indicates that the following arguments are expressed in octal rather than decimal.

N (optional) specifies the first word to be printed. N may be specified as 1 or omitted and assumed 1.

R (optional) Printing will be grouped into blocks of R words (5 words per line) where each block is labeled with its rank, i.e., N, N+R, N+2R etc. Naturally if R is to be specified, N may not be omitted. If R is omitted, it is assumed 10 (or 8 if O is specified). R may not exceed 40.

PRBIN terminates by calling CHNCOM.

Revised 11/19/65

Identification

Print summary of BSS files.
PRBSS

Purpose

To print a summary of information about the program in a BSS file or about the programs if the file is a library file.

Usage

PRBSS LIBE -ENTRY-

PRBSS prints a summary of information about the program(s) contained in file LIBE BSS. At least three lines are printed for each program:

1st line: Entry names and their relative locations
2nd line: Common break, program break, and transfer vector length
3rd line: Subroutine names in transfer vector (if any).

If LIBE is preceded by the parameter '(SQZ)', a summary of the file LIBE SQZBSS will be printed. (See Section AJ.4.04 for a description of SQZBSS files.) *

ENTRY (optional) specifies the program entry name at which printing should begin. If ENTRY is omitted, printing begins with the first program in the file.

BREAK A single interrupt signal will terminate the command by calling CHNCOM.

Revised: 9/24/65

Identification

Print SAVED file
SDUMP

Purpose

To print the machine conditions and/or locations within a SAVED file.

Usage

SDUMP NAME1

The machine conditions of file NAME1 SAVED will be printed on the user's console.

SDUMP NAME1 LOC -N-

The contents of N consecutive locations (decimal) beginning at octal location LOC, of the core image contained in file NAME1 SAVED will be printed on the user's console. All registers are typed in octal with nmemonics.

'-----' will be typed to indicate that one or more lines of all zero have been omitted. If N is not specified or is greater than 1000, 1000 locations will be dumped. Single break level is set to terminate printing and exit via CHNCOM.

Errors:

File not found.
Saved file has improper format.
Location not in saved file.
File system diagnostics from 'PRNTER'.

Identification

Combine files
COMBIN

Purpose

The COMBIN command combines several files of the same secondary name into a new file, also of the same secondary name. The format of the files is not significant.

Usage

COMBIN SEQ NAME1 NAME2 FIL1...FILn

COMBIN will combine files FIL1 through FILn of secondary name NAME2 into one file NAME1 NAME2 within the current file directory. If any FIL cannot be found, the NEED-USE convention will be followed (see Section AH.7.01). Within the USE process, an * for a corresponding FIL means that FIL should be ignored. The combining will not begin until all FIL's are accounted for. FIL's are not deleted.

SEQ is a decimal number of 1-4 digits. The numeric sequence field begins with SEQ x 10 with leading zeros to complete the numeric field or with the most significant digits lost if SEQ x 10 exceeds the numeric field width. Sequencing is done by incrementing the numeric field by 10. If SEQ = '*' or if NAME2 is 'SAVED', 'BSS' or 'CRUNCH', no sequencing will take place.

The sequence field (characters 73-80) may be composed of 2-5 numeric characters and 3-6 alphabetic characters. The numeric field width is determined by a scan of the first line of FIL1 from right to left, beginning with character 78, looking for the first nonnumeric character (blanks are treated as numeric zeros). The numeric field width and the alphabetic field width will remain fixed through the remainder of the command. The alphabetic information is obtained from each line of the FIL's. Note that the numeric field width will be at least 2 and not more than 5 characters wide.

EXAMPLES:

If characters 73-80 of the first line of FIL1 are ABC123GH and SEQ = 1, the new sequence for NAME1 NAME2 will begin with ABC00010.

If the first line contains Abbbbbbb and SEQ = 1, the new sequence will begin with Abb00010.

If the numeric field overflows, a message will be printed, "SEQUENCE FIELD OVERFLOW", and sequencing will continue from 0.

Line-marked files composed of 14-word lines may be sequenced. If a line of more than or fewer than 14 words is encountered, sequencing is stopped and not resumed during execution of the rest of the command. A message is printed, "SEQUENCING STOPPED AT xxxxx".

Identification

Subdivide files
SPLIT

Purpose

The SPLIT command divides or splits a specified file into one or more separate files of the same class. Either BCD or binary files may be SPLIT.

Usage

SPLIT NAME1 NAME2 MODE A1 S1 A2 S2 ... AN SN

NAME1 NAME2 is the file to be SPLIT. In case NAME1 NAME2 cannot be found, the NEED-USE convention if followed as in the LOAD command (Section AH.7.01).

Ai are the new files to be created, with the secondary name NAME2. All previous copies of new files are deleted, if possible. Any Ai may be replaced by "*" if the file delimited by S(i-1) and Si is not wanted. Any Ai may be NAME1. As the original file will not be deleted until all splitting is completed.

Si are the numerical dividers of the file in order of appearance as the file is scanned only once and are interpreted, depending on the mode, as line number, record number, or number of words. The Si (th) record (or words) belongs to file Ai unless Si falls between 2 sequence numbers, in which case the file is split between them.

e. g. If Nj .LE. Si .L. N(j+1) where N is sequence number in NAME1.

then file Ai ends with Nj and file A(i+1) begins with N(j+1)

Sn may be omitted if An is to go through the end of NAME1.

MODE:

There are three kinds of files which may be SPLIT:

1) Line-numbered - BCD card images (14 words) with numeric sequence number in column 76-80.

2) Line-marked or variable length records preceded by an extra word which contains the word count of the record.

3) String - no obvious record divisions. Records may be treated as 14 word records or by external word count.

MODE is an optional argument which may be inserted on either side of NAME1 NAME2.

Record number mode assumes 14 word records, unless they are line-marked, and numbers them sequentially starting with 1. This mode may be requested by the MODE argument (RCNO).

Word count mode splits strictly by a count of the words, including any line marks present. This mode may be requested by the MODE argument (WDCT).

If no mode is specified, it is assumed to be line numbered.

If, at any time, a record is encountered which does not appear to be a regular BCD card image (e.g. not 14 words long or non-numeric in columns 76-80) a change is attempted. If search is still being made for S1 (no splitting has taken place), the mode is changed to record number, if possible and the search continues. Otherwise, splitting is stopped, the rest of NAME1 is placed in a temporary file, and an appropriate comment is made. No other changes of mode can occur.

Revised: 8/30/65

Identification

Change the mode or the name or delete a file
CHMODE, RENAME, DELETE

Purpose

Commands to change the mode or the name of a file or to delete a file.

Usage

Delete:

DELETE NAME1 NAME2 ..... NAME1n NAME2n

DELETE calls the library subroutine DELETE to delete all files NAME1i NAME2i from the current file directory, if possible. Any of the library subroutine's messages may be printed. If for any reason a file cannot be deleted, a message is printed:

NAME1i NAME2i NOT DELETED

The command PRNTER may be issued to print the specific I/O error message

NAME1i is the primary name of a file to be deleted. If NAME1i is *, all files of secondary name NAME2i will be deleted. If NAME2 is also *, no files will be deleted and the message "NO FILE WITH SECOND NAME * FOUND" will be printed.

NAME2i is the secondary name of a file to be deleted. If NAME2i is *, all files of primary name NAME1i will be deleted.

Change mode:

CHMODE NAME1 NAME2 MODE1 ....NAME1n NAME2n MODEn

MODE arguments are:

- 0 - permanent
- T - temporary
- S - secondary
- R - read-only
- W - write-only
- V - private
- P - protected

Up to six of the letters can be concatenated to form combination modes; for example,

CHMODE NAME1 NAME2 RVP *

would make file NAME1 NAME2 read-only, private, and protected. *

If the mode of any file cannot be changed, a message is printed:

FILE NAME1i NAME2i MODE NOT CHANGED

The PRNTER command may be issued to obtain a complete error message.

NAME1i NAME2i - The same * convention is used as in the DELETE command.

Rename:

RENAME NAME1 NAME2 NAME3 NAME4 .... NAME1n NAME2n NAME3n NAME4n

RENAME changes the file name NAME1 NAME2 to the name NAME3 NAME4 by calling the supervisor entry CHFILE. All other files NAME3 NAME4 will be deleted before renaming NAME1 NAME2 by calling the library subroutine DELETE. The deleting of NAME3 NAME4, therefore, has the same options and messages as DELETE. If NAME3 NAME4 cannot be deleted, no names are changed. If the file cannot be renamed, a message is printed:

FILE NAME1i NAME2i NOT RENAMED

The PRNTER command may be issued to obtain a complete error message.

NAME1i is '*' the NAME3i must be '*' and all files of secondary name NAME2 are changed.

NAME2i is '*', then NAME4i must be '*' and all files of primary name NAME1 are changed.

NAME4n missing, is assumed to be NAME2n.

Revised: 1/3/66

Identification

Common files
COMFIL, COPY, UPDATE

Purpose

A group of "common" file directories (currently up to five * in number) is frequently assigned to programmers working on the same problem number. ("Common" is used in the sense of "accessible to all".) The COMFIL command allows the user to cause the currently atached to file directory to be one of the common file directories or to switch back to his own. The UPDATE command allows the user to transfer a file from the current file directory into one of the common file directories. The COPY command allows the user to copy a file from a common file directory into his current file directory.

Method

Both COPY and UPDATE create intermediate files whose names are a function of the current time of day. This method of generating unique names allows several users to be working in the same file directory without adverse interaction with each other.

If COPY or UPDATE is used to move OUTPUT RQUEST files, the resulting file will be an appended file rather than a replaced-by-deletion file, as is the standard procedure. If there is a temporary version of OUTPUT RQUEST in the receiving directory, it will be deleted before the COPY or UPDATE is performed.

Neither COPY nor UPDATE resets the current file directory switch, i.e., upon completion of the command the current file directory is the same as it was at the beginning of the command.

Usage

Comfil:

COMFIL -N-

N specifies the file directory desired as 0, 1, * 2, 3, 4, 5. 0 signifies the user's file directory. If N is omitted, it is assumed zero.

COMFIL switches the current file directory to N so that all subsequent commands will refer to directory N. Unlike the old file system, *

active files are now not reset when a directory switch occurs.

Copy:

COPY N NAME1 NAME2....NAME1n NAME2n

COPY transfers files NAME1 ..... NAME2n from common file directory N into the current file directory. Any files of the same name in the current directory will be deleted by the DELETE conventions after the successful copying of the new files. Files keep the same names but are always created in permanent mode.

N may be 0, 1, 2, 3, 4, 5, S or P. S and P are synonymous and allow copying from the public or system file directory. *

Update:

UPDATE N NAME1 NAME2....NAME1n NAME2n

N is the user's common file number 0, 1, 2, 3, 4, or 5. *

UPDATE transfers files NAME1...NAME2n from the current directory to the specified common file. Files keep their same name and mode. All previous versions in the receiving file directory are deleted by the DELETE conventions only after successful updating. The files in the current directory are unchanged.

Revised: 11/19/65

Identification

Library file
EXTBSS, UPDBSS

Purpose

A library file may be created by combining programs in BSS form. The program loaders can search this kind of file to find missing programs. The housekeeping of these files can be done by EXTBSS and UPDBSS.

Usage

Extract:

EXTBSS LIBE FILE1 ENTRY1 ... ... FILEn ENTRYn *

EXTBSS will extract from the library file LIBE BSS the first BSS routines with the entries ENTRY1 ... ENTRYn and create files FILE1 ... FILEn BSS. Older files of FILEi BSS are deleted, if possible. LIBE BSS is unchanged.

ENTRYi If an ENTRYi has the same name as FILEi, '=' * may be used in place of ENTRYi. ENTRYn (the last parameter on the line) may be omitted if it is identical to FILEn. If ENTRYi is '(MAIN)', the first main program will be extracted from LIBE BSS.

FILEi If FILEi is preceded by the parameter '(SQZ)', * the extracted file will be created in sqzbss format (See Section AJ.4.04). In this case, the name of the file will be FILEi SQZBSS. If the parameter '(SQZ)' precedes LIBE, extraction will take place from the file LIBE SQZBSS.

Update:

UPDBSS LIBE FILE1 ENTRY1 ... ... FILEn ENTRYn *

UPDBSS searches the library file LIBE BSS for the first BSS routines with entries ENTRY1 ... ENTRYn and replaces each routine with the corresponding file FILEi BSS. This is accomplished by creating a new file LIBE BSS and deleting the old, if possible. If LIBE BSS can not be deleted, no updating is accomplished. If an ENTRYi is not found in * LIBE, UPDBSS will print the following message: "ENTRYi NOT FOUND. DO YOU WISH TO APPEND IT," If the response is "YES", FILEi will be appended to LIBE.

ENTRYi The same conventions in EXTBSS with regard to the use of '=' and omission of the last parameter, ENTRYn, apply also to UPDBSS. If FILEi is '*', the first routine with entry name ENTRYi will be deleted from LIBE BSS.

FILEi If any FILEi is preceded by '(SQZ)' , the file FILEi SQZBSS will be inserted. Preceding LIBE by '(SQZ)' will cause LIBE SQZBSS to be updated.

If any FILEi cannot be found, the message * "FILEi BSS NOT FOUND." will be printed, and UPDBSS will exit to DORMNT. The user may then type "USE NEWFLi" to use a different FILEi, "USE *" to delete the entry from LIBE, or "START" to ignore the update of ENTRYi.

Revised: 5/30/66

Identification

Off-line processing
RQUEST

Purpose

Requests may be submitted to the dispatcher to print or punch current files, or send a current file to the other machine (MAC or Center) for reloading and updating. These requests may be submitted as punched control cards (see Section AE.1) or via the RQUEST command from the console, which will prepare a file called OUTPUT RQUEST in the user's directory. The control cards and the OUTPUT RQUEST files are processed several times a day by a background job called the disk editor.

Usage

RQUEST XX NAME1 NAME2 -OP-...NAME1n NAME2n -OPn-

XX='PRINT': The BCD file NAME1 NAME2 is printed off-line. If the file is not line marked, a blank word is inserted at the beginning of the line to insure single spacing and the first 84 characters of the record are printed. If the file is line-marked, the first character is the carriage control character and the next 131 characters are printed.

If the file is line-marked and the secondary name is FAP or MAD, the file will be effectively XPANDed to 80 columns for printing with tabs replaced by the appropriate number of blanks and null characters deleted. A blank word will be inserted in front of each line to insure single spacing. Sequence numbers will be inserted in columns 75-80. The file itself remains unchanged. If the secondary name is other than FAP or MAD, the file will be XPANDed to 132 characters by * inserting sufficient blanks so that tab stops come out at positions 10, 15, 30, (+15) ..., 120. Also, if the secondary name is ALGOL, LISP, or LSPOUT, a blank character will be inserted in front of each line to insure single spacing. However, an ALGOL file will be XPANDed to 132 characters by interpreting tabs for columns 11, 16, (+5) ..., 66.

XX='DPUNCH': The BCD file NAME1 NAME2 is punched off-line. If the file is line-marked, just the first 80 characters per line of data will be punched. Line-marked files will be XPANDed in the same way as described under PRINT.

XX='BPUNCH': The binary card image file NAME1 NAME2 will be punched off-line. The 7-9 punch and checksums should already be included in the card image file.

XX='7PUNCH': The file NAME1 NAME2 (of any format) will be punched off-line in a special card format which may be reloaded by the disk editor to reproduce the file exactly. The file is not deleted from the user's directory.

XX='DELETE': The file NAME1 NAME2 will be deleted from the * current file directory. PRIVATE or PROTECTED files may not be deleted. Deletion will not occur "through a link".

XX='CARRY': The file NAME1 NAME2 will be carried to the other computer and will be loaded onto the disk during the next load or update. It will be loaded as permanent mode, with the same name (NAME1 NAME2), within the same problem-programmer file directory. If a different problem-programmer specification is desired for the receiving file directory, OP may be PROB PROG, i.e., the desired problem programmer numbers. If a different file name is desired, OP may be PROB PROG NAM1 NAM2, where PROB PROG must be the problem programmer numbers for the receiving file directory and NAM1 NAM2 may be the name given to the input file.

The file NAME1 NAME2 on this machine is in no way changed. Any previous versions of the file on the receiving machine will be deleted unless they are PRIVATE or PROTECTED. Input * will not occur "through a link".

XX='PRNDEL', 'DPUDEL', 'BPUDEL', '7PUDEL':

The file(s) will be PRINTed, DPUNCHed, BPUNCHed, or 7PUNCHed and then the mode will be changed to temporary. PRIVATE or PROTECTED files will not be changed to temporary, nor * will files be changed "through a link". The next time the file is read or the user logs out, the file will be deleted. Note that any other request for the same file following a

"DEL" request will cause the file to be deleted.

OP refers to the options available for the CARRY * request.

Method

The RQUEST command creates or appends to a file in the user's file directory called OUTPUT RQUEST. This file contains control card images which will be processed by the disk editor program. Warning: words 13 and 14 of each card image are used for the requesting user identification. If ED is used to modify the OUTPUT RQUEST file, these identifying words are destroyed. After processing, the disk editor program will change the mode of OUTPUT RQUEST to temporary. This change to temporary allows the operations staff to rerun the disk editor if any difficulty was encountered in the first run. Note that OUTPUT RQUEST contains only the control cards which point to the actual files to be processed. The disk editor program, upon processing the request files, will generate three different tapes: printer, punch, and carry. These tapes are then the responsibility of the operations staff.

(END)

Revised: 1/3/66

Identification

General file system call
CALL

Purpose

CALL provides a single unprivileged command which may be used to call any one of various I/O system entries (subroutines) from command level.

Usage

CALL ENTRY ARG1 ARG2 ... ARGn

ENTRY may be any of the file system entries except RDFILE, RDWAIT, WRFILE, WRWAIT, BUFFER, FCHECK, or FWAIT (see AD.2, Summary). Note that privileged calls may be made only by users with appropriate privileges.

If tape labels cannot be specified in BCD as expected by VERIFY and LABEL, they may be specified in octal by

CALLing BVERFY or BLABEL.

ARGs are the arguments required by ENTRY. Optional arguments may be specified as * or -0. Trailing optional arguments may simply be omitted.

Responses

If ENTRY is FSTATE, one line of response will provide * length, mode, status, device, next-read, next-write, date-modified, time-modified, date used, author. (See AG.3.07.)

The ENTRY STORGE will furnish information in the general form aaaaaa uuuuuu, where the a's indicate the number of allotted records and the u's indicate the number of records used. (See AG.3.07)

The ENTRY ATTNAM will return PROBNO PROGNO. (See AG.7.04)

The ENTRY IODIAG will furnish the same information that the subroutine does, on one line. (See AG.4.06)

Revised: 5/19/66

Identification

Relocatable program loading
LOAD, LOADGO, VLOAD, NCLOAD, L, USE

Purpose

There are five different types of loading available for relocatable programs, i.e., BSS files. The first (LOAD) will load a program into core without destroying the loader or MOVIE) table, place the program in dormant status and return to the user for the next command. The second (LOADGO) is the result of the chain of commands LOAD and START. The third (VLOAD) will load the program; move all of the program and COMMON down in core to destroy the loader and MOVIE) table (thereby making the available core larger); place the program in dormant status and return to the user for the next command. The fourth (NCLOAD) is the same as VLOAD except that erasable COMMON is also destroyed so that no library routines which use erasable COMMON may be used. The fifth (L) is a separate command which allows any one of the previously mentioned four to be used with larger loading tables (see Restrictions).

Programs or files may be loaded (or searched as library files) from the user's file directory, from his common files and from several system files.

If needed routines cannot be found by the loader, the USE command may be used to specify which routines may be used instead.

Restrictions

Normal maximum table sizes are: MOVIE) table is 500 words and the table of missing entries is 100.

The tables for the L command are: MOVIE) table of 1200 words and missing entries of 250 words.

When several programs are loaded, the one using the most common should be loaded first. *

Usage

Any of the load commands (LOAD, LOADGO, VLOAD, NCLOAD) may be used in place of LOAD; all special arguments are optional and order is significant by meaning or where specified. Special arguments are those beginning and ending with parentheses as shown. They cause the loader to behave in a special manner. The non-special arguments are either file names or entry points, depending upon the preceding special

arguments.

Upon completion of loading, the current file directory is switched to its initial status.

LOAD (ORG) (CFLn) (LIBE) (SYS) (NEED) (NLIB) NAMES (MORE) *

(ORG) The presence of (ORG) instructs the loader to set the starting address to the entry name specified by the next non-special argument following (ORG).

(CONT) as the first argument, may be used for programs calling the loader through the command buffers in order to retain control in the event of a loading error. The next non-special argument is the name of the file which should be resumed in case of an error.

e.g.,
```
SAVE   X
LOAD   (CONT)  X  A  B  ...
SAVE   Y
RESUME  X
```

After this sequence, X can determine whether or not the load was successful by the existence of Y SAVED.

(NEED) The presence of (NEED) instructs the loader to treat the next non-special argument as a program entry point as though it had been an entry in a transfer vector.

(MORE) may be the last argument before the carriage return (because only one line can be interpreted by the command) to indicate that more arguments will be specified. In this case the loader will not print the NEED list; will restore the common file switching to the initial setting; and return to the user (by way of CHNCOM) so that the USE command may be used.

(CFLn) directs the loader to switch the current file directory to common file directory n which may be 0, 1, 2, 3, 4, or P. The current file directory is initially the user's file directory or a directory set by a COMFIL command. There may be any number of these switches in the argument list and each one supercedes the previous one.

(LIBE) directs the loader to use the next non-special argument as a file within the current file directory to be searched as a library file to find any missing routines.

(SYS) directs the loader to use the following non-special argument as a file from the system file directory to be searched as a library for any missing routines.

(NLIB) directs the loader not to search the system library (i.e., TSLIB1) for missing routines after the argument list has been processed.

(LIB) supercedes (NLIB).

NAMES may be the primary names of BSS files to be loaded or BSS files to be searched as libraries following certain special arguments or NAMES may be routine entry points as required by other special arguments.

NEED Following the processing of the argument list, the system library TSLIB1 will be searched for any missing routines (unless prohibited by (NLIB)). If routines are still missing, the current file directory is switched to the user's directory, a list of needed routines (by entry names) is typed by the loader and DORMNT is called so that the user may type the USE command. Upon completion of loading, the current file directory is switched to its initial status.

USE will reinstate the last common file switching and go back to the loader. All of the arguments available to the loader are therefore available to USE.

LOAD sets the origin of the first program at (5200)8. The MOVIE) table and the loader are left inviolate below this origin. COMMON addresses are relocated with the same parity as on the assembly listing. FAP coded subprograms, which contain the EVEN pseudo op, will be loaded with relative location 0 in an even core location. Upon completion, all loaders call CHNCOM with an available core image specified.

LOADGO is equivalent to the sequence of commands LOAD and START.

VLOAD After the entire program and library subroutines have been LOADed the program is moved down so that the origin is (30)8, covering the loader and the MOVIE) table. The (316)8 words of erasable COMMON are included with the program. The MOVIE) table will be preserved if MOVIE) occurs in the transfer vector of any routine loaded.

NCLOAD is the same as VLOAD except that the (316)8 words of erasable COMMON are not included and, therefore, if library subroutines which use erasable common are included, a COMMON assignment error message will be printed.

L The L command may be used if larger loading tables are needed (see Restrictions). The L precedes any one of the LOAD commands as: L LOAD ARGUMENTS. If the loader name is omitted, it is assumed to be LOAD. All of the regular loader arguments are available. The * program loading by LOAD starts at (7000)8 instead of (5200)8. There may be more than 250 missing entry names if this does not occur during a library search. L always calls CHNCOM, regardless of the outcome of the loading. No core image is kept if loading failed.

MOVIE) table is created by the loader to provide a storage map of all entry points of routines as they are loaded. It is always written as a file (MOVIE TABLE) in the user's directory in temporary mode. If the entry MOVIE) appears in the transfer vector of any routine loaded, by VLOAD or NCLOAD, the MOVIE) table will be preserved by moving it to the top of the load. The MOVIE) entry points to location (27)8 which contains the MOVIE keyword which contains the number of words in the movie table in the decrement and the location of the lowest word in the movie table in the address. The format of the MOVIE) table, starting with the lowest location, is:

1. fence
2. Lowest common break (address)
3. SVN prefix
4. Memory bound (address)
5. BCD entry name
6. Entry point for previous name (address)

.. Pairs of words 5 and 6 for each entry to the subprogram.

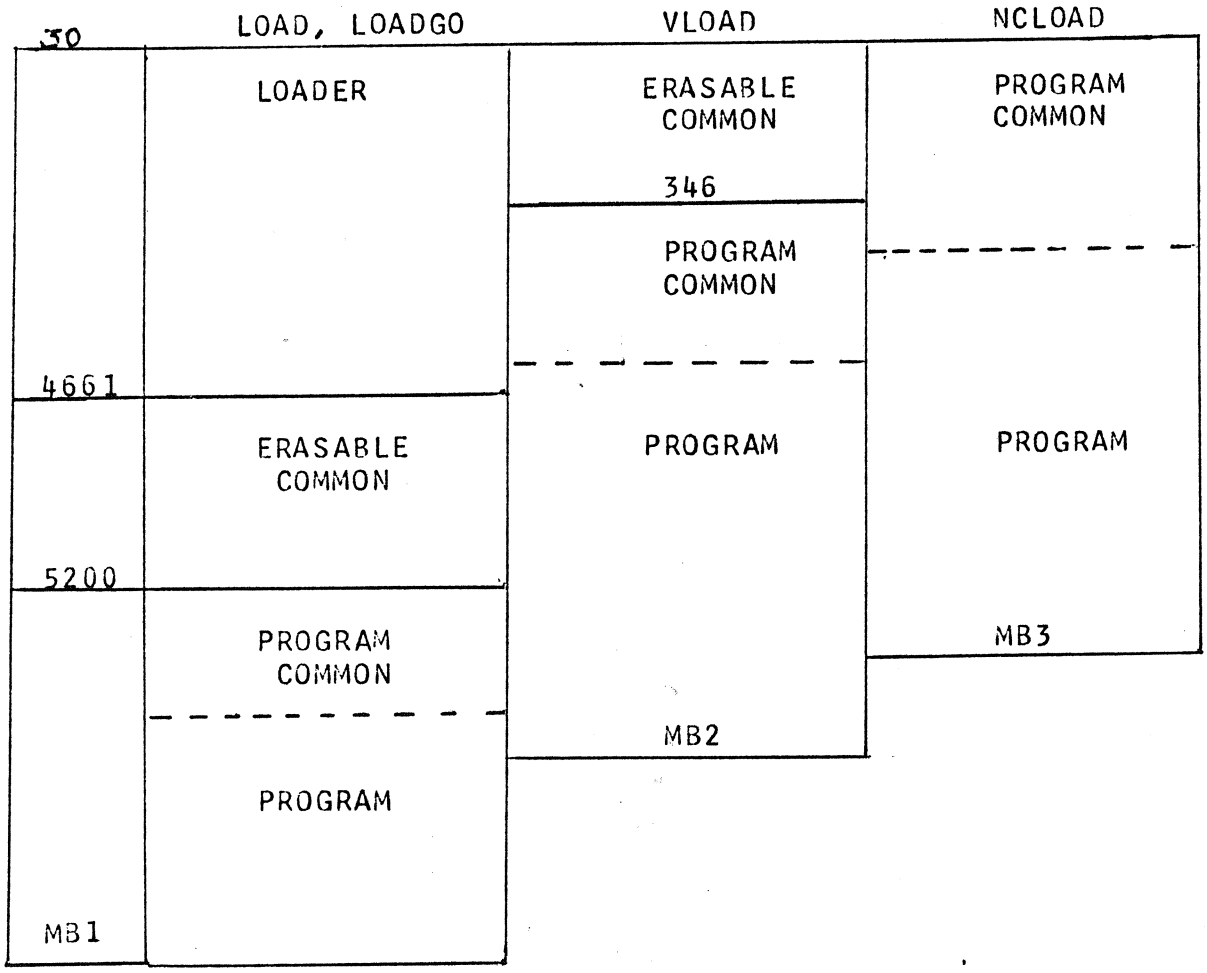
7. SVN prefix or PZE 0,,n, where there are n words in the transfer vector of this subprogram. *
8. Origin of this subprogram (address)
.. Repeat groups 5 thru 8 for each subprogram loaded.

CORE MAP
(All numbers octal)

*

Location

0-7	ZERO
10-23	BOOTSTRAP for NCLOAD and VLOAD or TSX LOAD,4
24-26	TSX (ORG),4 for NCLOAD and VLOAD or TSX LOAD,4
27	MOVIE) Keyword



Memory Bounds:

MB3= 30 + 77461 - COMMON BREAK + PROGRAM LENGTH
MB2 = 316 + MB3
MB1 = 4632 + MB2

(END)

Revised: 5/16/66

Identification

Absolute Program loading
LDABS

Purpose

To load a program from a file containing absolute column binary card images. A SAVED file may be created directly by LDABS, if desired.

Warning

Unlike the other loaders, LDABS will create the SAVED file representing the program it has loaded in seven-tag mode.

Usage

LDABS ANAME -SNAME- *

ANAME is the primary name of the file ANAME 'ABS' which contains absolute column binary card images with full word checksum.

SNAME is the primary name of SNAME SAVED which is (optionally) to be the SAVED file created by LDABS. Previous versions of SNAME SAVED are deleted using the DELETE conventions. *

LDABS will load a program into core with an upper limit of (77777)8 and a lower limit of 0. The memory bound is set, upon completion of the load, to the highest location loaded. Loading terminates with a transfer card and execution may be started at the transfer-location by issuing the START command.

Error Conditions:

a) If a check sum error occurs, the comment: CHECK SUM ERROR IN CARD XXXXX is printed, where XXXXX is the location in which the first word on the card is to be stored. After this comment is printed the card is ignored and loading continues.

b) If an attempt is made to store in a location greater than (77777)8, the comment: CARD YYYYY OUT OF BOUND is printed, the card is ignored, and loading continues.

c) If a transfer card is missing, i.e., an end of file is reached, the comment: TRANSFER CARD MISSING, TYPE OCTAL STARTING LOC, is printed. The characters typed are converted to an octal location and the transfer location for starting is set up.

d) Any card with other than a 7-9 punch in column 1 or a word count .GE. 23 will result in the message: CARD YYYYY ILLEGAL BINARY CARD. The card is then ignored and loading continues.

(END)

Revised: 5/19/66

Identification

Start or continue execution
START, RSTART, RESTOR, RECALL, RESUME, R, CONTIN

Purpose

Programs may have their execution interrupted (e.g., through use of the quit button or call to DORMNT) or delayed (e.g., LOAD as opposed to LOADGO, or the sequence LOAD, SAVE). The commands covered in this section give the ability to cause the execution of such programs. *

Usage

START -ARG1 ARG2 ... ARGn- *
RSTART

The START command may be used to begin a program which has been loaded by one of the LOAD commands, or it may be used to continue a dormant program from the place of the last interruption. The ARGi represent optional arguments, which will be placed in the command buffers; this technique is useful for programs which call DORMNT in anticipation of another "pass". *

RSTART is equivalent to START, except that it is transparent to (i.e., does not alter) the current command buffer and command location counter. It should be used when restarting a chain of commands.

RESTOR NAME1
RECALL NAME1

The RESTOR and RECALL commands will restore the core image from NAME1 SAVED complete with active files, if any. The program is placed in dormant status so that it may be (R)STARTed in order to continue from its last interruption.

In addition, RECALL restores the command list and common file switching from NAME1 SAVED, and preserves the command location counter and current command buffer in case of a subsequent RSTART. *

RESUME NAME1 -ARG1 ARG2 ... ARGn- *

(RESUME may be abbreviated by the letter R.) The RESUME command is effectively the same as RESTOR and START. The arguments are placed in the current command buffer so that it contains NAME1 ARG1 ARG2 ... ARGn. This is a technique for writing and checking out a new command.

CONTIN NAME1

The CONTIN command should be used to resume a program involving a chain of commands. It restores the program and machine conditions from NAME1 SAVED, together with any active files, the common file switching, and the contents of the command list, command location counter, and current command buffer. In other words, it is exactly equivalent to the chain of RECALL and RSTART.

Summary

RECALL RSTART CONTIN	RESTOR START RESUME
1. Restores command buffers.	1. Does not restore command buffers.
2. Restores chain, if any.	2. Does not restore chain.
3. Restores directory switching.	3. Does not restore directory switching.
4. Will not over-write command buffers and command list.	4. Will over-write command buffers.

(END)

Identification

Relocatable program loading
LAED, USE
C. Feldman, x5880

Purpose

LAED (Load AED) is a loader originally developed by the Electronic Systems Laboratory group for use with AED (see AH.2.01). It has several features which the standard loader (AH.7.01) does not.

Discussion

There are four different types of loading available for relocatable programs in either BSS or SQZBSS format. All four types are contained in the single LAED command. The type of loading desired is selected by typing one of four options following LAED. The first loading option LOAD will load a program into core without destroying the loader or MOVIE) table, place the program in dormant status and return to the user for the next command. The second option LOADGO is the result of a LOAD followed by the command START. The third option VLOAD will load the program, move all of the program and COMMON down in core to destroy the loader and MOVIE) table (thereby making the available core larger), place the program in dormant status and return to the user for the next command. The fourth option NCLOAD is the same as VLOAD except that erasable COMMON is also destroyed so that no library routines which use erasable COMMON may be used.

Programs or files may be loaded (or searched as library files) from the user's file directory, from his common files and from several system files.

If needed routines cannot be found by the loader, the USE command may be used to specify which routines may be used instead.

The list of files to be loaded, intermixed with the various loading options, may be placed into a separate disk file (second name LOAD). This LOAD file may then be referenced in the LAED command line, and the effect is the same as if the file contents had been typed by the user.

Two forms of MOVIE) table may be produced, the standard format or the format suitable for use with the LOADER/UNLOADER system (MAC-M-286). The latter contains all of the standard information, plus program size data. Regardless which of the two types of MOVIE) table or which of the four loading types is requested, a file named (MOVIE

TABLE) is written (in temporary mode 001) and added to the user's directory, which contains the MOVIE) information. This file is in a binary format which is not directly printable, but may be used by any of the available utility programs.

The loader may be used as a subroutine during execution of an object program for the purpose of resuming the loading process. The LOAD or LOADGO options cause the loader to insert the subroutine entry (LOAD) in the MOVIE) table for this purpose. Since the entry point (LOAD) is part of the loader itself, (LOAD) is not put in the MOVIE) table when VLOAD or NCLOAD is used.

Restrictions

Maximum table sizes are: MOVIE) table is 1080 words, the table of missing entries is 100, and the maximum size of a LOAD file is 44 lines. The "missing entry" table is not continually maintained, but is generated when needed (just before a library search or at the conclusion of loading). Therefore, it is possible during loading to temporarily build up more than 100 missing entry points without causing a fatal loading error.

Usage

The type of loading desired (LOAD, LOADGO, VLOAD, or NCLOAD) is typed immediately following LAED. If no option is typed, LOAD is assumed. The remainder of the command line is a series of special and nonspecial arguments. Special arguments are those beginning and ending with parentheses as shown below. Nonspecial arguments are either file names or entry points, depending upon the preceding special arguments.

Upon completion of loading, the user is left in the file directory he was in immediately preceding the LAED command.

The following paragraphs describe each of the special arguments recognized by LAED.

(ORG) The presence of the (ORG) option instructs the loader to set the starting address to the entry name specified by the next nonspecial argument following (ORG).

(NEED) The presence of (NEED) instructs the loader to treat the next nonspecial argument as a program entry point as though it had been an entry in a transfer vector of one of the binary files already loaded.

(MORE) may be the last argument before the carriage return (because only one line can be interpreted by the command) to indicate that more arguments will be specified. In this case the loader will not print the NEED list; will restore the common file switching to the initial setting; and return to the user (by way of CHNCOM) so that the USE command may be used.

(CFLn) directs the loader to switch the current file directory to common file directory n which may be 0 through 9 or P. The current file directory is initially the user's file directory or a directory set by a COMFIL command. There may be any number of switches in an argument list and each one supersedes the previous one.

(LIBE) directs the loader to use the next nonspecial argument as a file within the current file directory to be searched as a library file to find any missing routines. The file is searched repeatedly until one complete pass through the library is made in which no additional needed routines are found, or until all needed routines are loaded.

(SRCH) has the same effect as (LIBE) except that only one pass is made through the library. The argument (SRCH) thus assumes that the library is properly ordered so that no program references any program which occurs before it in the library, thus saving load time.

(SYS) directs the loader to use the nonspecial argument following (SYS) as a file from the system file directory to be searched as an ordered library for any missing routines. The argument (SYS) is exactly equivalent to the argument sequence (CFLP) (SRCH). The loader automatically performs a (SYS) TSLIB1 at the end of a LAED command whether or not any other libraries have been searched, and without any specific request by the user, if there are any missing entry points.

(NLIB) directs the loader not to search the system library TSLIB1 for missing routines after the argument list has been processed.

(LIB) supersedes (NLIB), thus restoring the automatic (SYS) TSLIB1 search at the conclusion of loading.

(SQZ) The argument (SQZ) instructs LAED that all indicated binary files following (SQZ) on the command line are in the SQZBSS format. Similarly, (BSS) returns LAED to the BSS mode. If neither argument is specified, (BSS) is assumed. The established mode also applies to all load files. LAED automatically switches to (BSS) mode whenever a new command line is typed (i.e. a USE or a START command after a NEED message).

(AEDP) This command searches the system library AEDLB1 for missing routines. The argument (AEDP) is exactly equivalent to the sequence (CFLP) (SRCH) AEDLB1.

(UNLD) causes LAED to produce the MOVIE) table and the file (MOVIE TABLE) in the proper format for the LOADER/ UNLOADER system. The argument (UNLD) must appear before any binary file names on the command line.

(GET) (NGET) instructs LAED that succeeding nonspecial argument file names are LOAD files, rather than BSS or SQZBSS files. (NGET) returns LAED to the normal mode (succeeding file names are of type BSS). A LOAD file consists of a sequence of standard 14-word card images, with the name of a BSS file, SQZBSS file or special loader argument appearing in columns 1-6 (one argument per line). The argument may appear anywhere within these six columns, and LAED will right-justify the word. Columns 7-72 are ignored, and may be used for comments. LAED also ignores any line containing blanks in columns 1-6 or an * in column 1.

The above is an exhaustive list of the LAED special arguments. If any word is typed in the LAED command line or LOAD file which is not one of the above, it is considered to be the primary name of a BSS, SQZBSS, or LOAD file, or an entry point, depending upon the special arguments preceding it.

The following is a list of the various on-line user and system typed statements used to communicate with LAED.

NEED Following the processing of the argument list, the system library TSLIB1 will be searched for any missing routines (unless prohibited by (NLIB)). If routines are still missing the current file directory is switched to the initial directory, a list of needed routines

(by entry names) is typed by the loader and DORMNT is called so that the user may type the USE command. Upon completion of loading, the current file directory is switched to its initial status.

USE — When USE is typed by the user, LAED reinstates the last common file switching and restarts the loading process. All of the arguments available to the loader are therefore available to USE. USE may be used to satisfy a NEED statement, or to load additional routines in an existing file originally created by LOAD or LOADGO types of loading.

LAED LOAD — Sets the origin of the first program at (7000)8. The MOVIE) table and the loader are left inviolate below this origin. COMMON addresses are relocated with the same parity as on the assembly listing. FAP coded subprograms which contain the EVEN pseudo op, will be loaded with relative location 0 in an even core location. Upon completion, all loaders call CHNCOM with an available core image specified.

LAED LOADGO — is equivalent to the sequence of a LAED LOAD followed by the command START.

LAED VLOAD — After the entire program and library subroutines have been LOADed the program is moved down so that the origin is (30)8, covering the loader and the MOVIE) table. The (316)8 words of erasable COMMON are included with the program. The MOVIE) table will be preserved if MOVIE) occurs in the transfer vector of any routine loaded.

LAED NCLOAD — is the same as VLOAD except that the (316) 8 words of erasable COMMON are not included and, therefore, if library subroutines which use erasable common are included, a COMMON assignment error message will be printed.

After the user has typed any of the above LAED or USE commands, LAED attempts to perform the indicated loading operations, and prints on-line alarms to report error conditions. These alarms are caused by three conditions:

1. Overflow of LAED tables or core memory (fatal).
2. Missing files or entry points (non-fatal).
3. More than 1 entry point with the same name (non-fatal).

If the loading is successful, the final operation performed by LAED is to produce the (MOVIE TABLE) file.

The MOVIE) table is created by the loader to provide a storage map of all entry points of routines as they are loaded. It is always written as a file (MOVIE TABLE) in the user's directory in temporary mode. If the entry MOVIE) appears in the transfer vector of any routine loaded, by VLOAD or NCLOAD, the MOVIE) table will be preserved by moving it to the top of the load. The MOVIE) entry points to location (278) which contains the MOVIE keyword which contains the number of words in the movie table in the decrement and the location of the lowest word in the movie table in the address. The format of the MOVIE) table, starting with the lowest location, is:

1. fence
2. Lowest common break (address)
3. SVN prefix
4. Memory bound (address)
5. BCD entry name
6. Entry point for previous name (address)

.. Pairs of word 5 and 6 for each entry to the subprogram.

7. SVN prefix or PZE 0,,n, where there are n words in this program's transfer vector.
8. Origin of this subprogram (address)

.. Repeat groups 5 through 8 for each subprogram loaded.

The format of the MOVIE) table created in conjunction with an (UNLD) loading argument is identical to the above format, except that item 7 is:

7. SVN prefix or PZE m, o, n, where there are m words in this program and n words in its transfer vector.

Provision has been made to allow the use of LAED as a subroutine during execution of an object program. The LOAD or LOADGO entries to LAED cause the loader to insert the subroutine entry (LOAD) in the MOVIE) table. Since the entry (LOAD) is part of the loader itself, (LOAD) is not put in the MOVIE) table when VLOAD or NCLOAD is used. For the same reason, a VLOAD or NCLOAD may not be initiated from an object program during execution.

The user calls the loader by issuing the instructions

```
TSX  (LOAD),4
***   LIST,,N
(error return)
(normal return)
```

LIST is the start of an array containing the file names to be loaded either right-justified or left-justified stored forwards in memory. N is the length of the list.

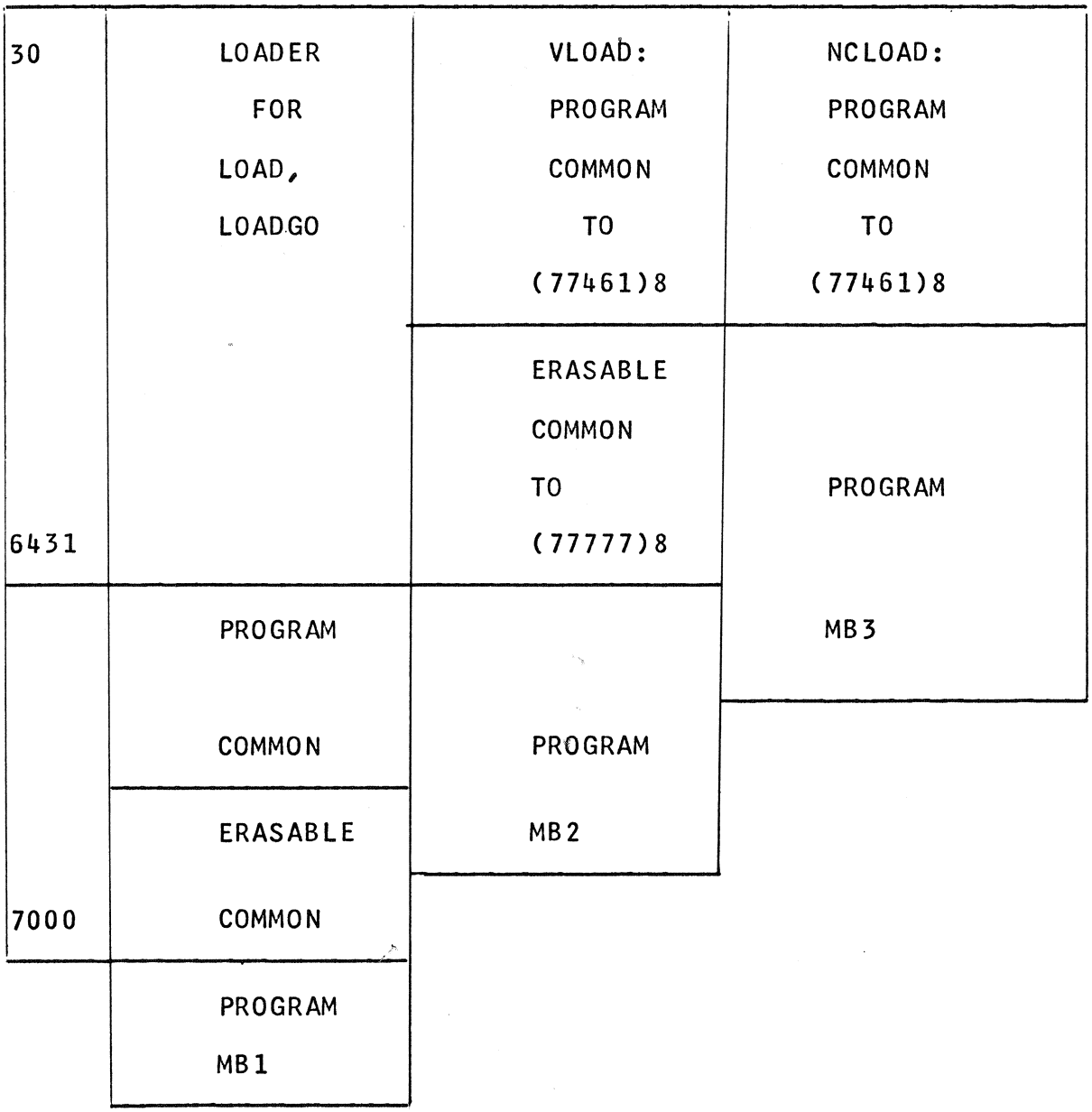
*** controls the printing of missing subroutines. If *** is PZE these will be listed by LAED. The message will be supressed if MZE is used.

Whether or not the on-line printout of missing subroutine names is requested, the error return is taken when one or more routines or files are missing. When this happens, the AC contains a pointer to the list of missing subprograms so that the user may use the information as he desires. The list terminates with a word of all zeroes. If only files are missing, the AC is zero.

LIST may contain any desired loader commands such as (LIB), (NLIB), etc. If the sequence (GET) BETA is used it should be at the end of the list. If it occurs elsewhere, the rest of the list will be ignored.

STORAGE MAP

(LOCATION)8	CONTENTS
0-7	ZERO
10-23	BOOTSTRAP for NCLOAD and VLOAD or TSX LOAD,4
24	TSX (ORG),4
25	TSX (ORG2),4
26	TSX (ORG3),4
27	MOVIE) Keyword



MB3 = 30 + 77461 - COMMON BREAK + PROGRAM LENGTH
MB2 = 316 + MB3
MB1 = 6431 + MB2

(END)

Identification

General discussion of debugging commands.

Method

There are three different kinds of commands within CTSS, one of which is of no importance in this discussion. The first kind is often referred to as a "disc-loaded" command. The distinctive property is that the supervisor loads the command from a core image SAVEd file and thereby eliminates any previous core image the user might have had. The second kind is often referred to as a "core-B transfer" command. Here the distinctive property is that the supervisor does not load the command, but instead, transfers to the relocating loader which is already in core-B. The loader then determines which command is specified and proceeds to load the command from a standard BSS library file (TSLIB2) into the area of core above the current core image. If the command has already been loaded, the loader merely transfers to the desired entry point.

Some of the present debugging commands are core-B transfer commands. The earliest routine available to CTSS was called FLEXPM which includes the commands PM, PATCH, STOPAT, and TRA. More sophisticated commands have been written more recently, such as FAPDBG and STRACE. These routines are able to make use of the tables created by the translators and the loader, such as the MOVIE) table and symbol table files. The use of these commands imposes some restrictions on the user, namely that the vanishing and absolute loaders not be used and that the symbol table files from the translators be available and of the proper format.

Programs which extend the memory bound during execution create some problems in connection with the debugging routines. Note that the core-B transfer commands are relocatable BSS subroutines with normal entry points. If the debugging routine is loaded after the program has started execution, there may be a conflict about the space acquired by expanding memory bound. Therefore, the solution is to force the debugging subroutines to be loaded with the program before execution. This may be accomplished either by placing one of the entry points in a transfer vector of one of the loaded programs or by use of the special arguments to the LOAD command.

The SP command is a disc-loaded command which may be used only by the system programs for patching core-A.
The SD command may generally be used for examination of locations in core-A.

The MADBUG command is a disc loaded command which serves as an intermediate supervisor between the user and the CTSS

supervisor. MADBUG allows the user to specify a MAD source file rather than BSS file. MADBUG manages all the calls to the MAD translator and the appropriate loader so that the restrictions implied by the core-B transfer routines are not as evident to the user.

Identification

FAPDBG - A symbolic debugging aid for FAP program
R. H. Campbell - x

Purpose

FAPDBG, as a symbolic debugging aid for FAP programs, was produced as an experiment with typing conventions and formats. FAPDBG acts upon requests typed by the user on the console and performs such functions as examining and typing or changing the contents of specified registers and allowing a subprogram to be run in controlled segments.

Reference

CC-216 FAPDBG, a symbolic debugging aid R. H. Campbell

Usage

```
 LOAD    NAME1  ARGUMENTS
FAPDBG  ALPHA
requests
```

The FAPDBG command can be issued anytime a program is dormant and the loader is available, i.e., may not have been loaded by a self-erasing loader. If the program extends memory bound or damages the loader, FAPDBG should be called before execution. The FAPDBG command calls the loader to load the FAPDBG subprogram from the debug library, "TSLIB2". FAPDBG uses the loader's symbol and loading tables to build its own symbol table (800 symbols maximum) for the subprograms which the user wishes to debug. FAPDBG is approximately (12400)8 locations in length.

If the line-numbered file ALPHA DEBUG can be found, requests are taken from there. When ALPHA DEBUG is exhausted, not found, or not specified, requests will be taken from the console.

Conventions:

1) A request is a single letter request name followed by arguments, all separated by blanks.
2) A blank is a string of any number (not zero) of spaces or tabulations.
3) Any number of requests may be concatenated on one line by typing an apostrophe or an equal sign between successive requests. Concatenation is recommended since FAPDBG will be brought into core less often and will generate less output.
4) If a request cannot be accomplished, FAPDBG will so inform the user and return to process the next request.

5) Syntax - The location, address, tag, and decrement parts of a request argument may consist of strings of symbols and octal numbers separated by plus and minus signs to denote the desired algebraic manipulation. The indicated operations are carried out, any negative result is converted to two's complement form and the right fifteen bits saved (in the case of the tag field, only the right three bits are saved). Symbols, which must be defined, may consist of any number of characters, at least one of which must be non-numeric (i.e., not 0 through 7), and none of which may be one of the special characters plus, minus , comma, space, or tabulate. If the number of characters is greater than six, only the last six will be used. Any string consisting only of the digits 0 through 7 will be considered an octal number of five digits, with left zeros if necessary. If more than five digits are typed, only the last five will be used. The line typed in is scanned from the left and each field is evaluated when encountered. If an undefined symbol is discovered, or a deviation from an understandable format is discovered, an appropriate comment is typed and processing of the request is terminated. If one or more requests cannot be interpreted, any go or proceed requests following them on the same line will be ignored.

There are four classes of requests: set up, register examination and modification, subprogram control, and FAPDBG control.

SET UP REQUESTS:

The set up requests are necessary to tell FAPDBG which subprograms are to be debugged and allow FAPDBG to build the necessary symbol tables. These requests are Load address, symbol Table, Work, and Equals.

LOAD ADDRESS: L ENTRY

ENTRY is an entry point of the subprogram to be debugged. The origin of the subprogram will be typed out and will be used as the relocation constant for all symbols within that subprogram.

SYMBOL TABLE: T -NAME1-

All the symbols from the file NAME1 SYMTB will be relocated by the origin printed from the last L request and placed in the FAPDBG symbol table. Note that this means absolute symbols and COMMON (except for the first-loaded) will be incorrect.

Successful completion is signaled by "SYMBOLS LOADED". If the FAPDBG symbol table becomes full, the last symbol entered will be typed out. Note that the symbols in the SYMTB file are in alphebetic order.

If NAME1 is omitted, all of the symbols will be deleted from the FAPDBG symbol table.

WORK: W ENTRY -NAME1-

W is the combination of L and T requests.

NAME1 need not be specified if ENTRY and NAME1 are the same.

EQUALS: E FE FS

FS is the symbol to be entered in the symbol table with the value of the expression FE.

FE is a FAP expression involving constants and/or symbols already entered in the symbol table (see convention 6.)

Register Examination and Modification

The register examination and modification requests permit the user to examine and change the contents of core locations as well as the live registers. They are look (floating point, Hollerith, full word integer, decrement integer, octal, symbolic), deposit, compare, signed and logical accumulator, and storage map.

LOOK: -request- -LOC1- -LOC2-

request sets the output conversion mode and if an argument is specified, prints the specified locations. Request may be one of the following:

F Floating point
H Hollerith
I Full word integer
J Decrement integer (Fortran)
O Octal
S Symbolic

LOC1 LOC2 are FAP symbolic expressions specifying a block of core from LOC1 through LOC2.

LOC1 specifies a single location.

The contents of a single location in the current output mode may be obtained by typing just the location expression without the look request with the restriction that the first symbol in the expression may not be a single letter. The contents of "* + 1" may be obtained by an empty request (just a carriage return or concatenation character).

DEPOSIT: D LOC FW

FW is the FAP word to replace the previous contents of location LOC.

This request may be abbreviated by omitting the request name, provided that the location expression does not begin with a single-letter symbol. The FAP word may be a symbolic machine instruction such as CAL ALPHA-10,4 or one of the data generating pseudo instructions OCT, BCD, FLO, INT (full word decimal integer), or JNT (decrement integer) followed by a blank and one word of data.

A symbolic machine instruction consists of a symbolic operation code, an optional asterisk to indicate indirect addressing, and an optional variable field in the same format as accepted by FAP, except that all numbers are interpreted as octal and that multiplication and division are not allowed. No blank may intervene between the operation code and the indirect flag; a blank must, however, precede the variable field. Note that since the address field is truncated to fifteen bits, the left three bits of the address part of type D instructions (left and right half indicator operations) will be considered by FAPDBG as the tag field, both for input and for output. Thus to insert the instruction

```
RFT  300105
```

it is necessary to type

```
RFT   105,3
```

The OCT pseudo instruction accepts a signed or unsigned octal integer of magnitude less than or equal to 377777777777. Thus, to insert the traditional fence, it is necessary to type

```
OCT -377777777777
```

The FLO pseudo instruction accepts a signed or unsigned floating point number with optional decimal point and optional E modifier to denote multiplication by the indicated power of ten. The B modifier is not allowed.

The INT and JNT pseudo instructions accept signed or unsigned decimal integers of sufficiently small magnitude to fit into the number of bits available (34359738367 for INT and 131071 for JNT).

The BCD pseudo instruction accepts any string of characters preceding the request terminator and assembles the last six into one word. If fewer than six characters are typed, spaces will be inserted on the left. Note that this pseudo instruction uses the input line image after FAPDBG has edited and "normalized" it. Therefore a string of spaces and tabulations will be interpreted as a single blank.

COMPARE and VERIFY: C ENTRY -NAME1-

ENTRY is the entry point of a subprogram already loaded in core.

NAME1 BSS is the name of the file which is to be compared with the core image of ENTRY. NAME1 need not be specified if it is the same as ENTRY.

C by using the origin value of the ENTRY subprogram, it will read and relocate each word in NAME1 BSS and compare it with the corresponding word in core. If a discrepency is found, FAPDBG will type in the current mode the location, the word from NAME1, and the contents of the memory location for which there is a discrepancy. "EXAMINATION CONCLUDED" will signal the completion of the request. The request may be terminated by a single interrupt; FAPDBG will close the BSS file and return to process the next request.

ACCUMULATOR: A -FW- or K -FW-

A places the FAP word 'FW' in the signed accumulator and clears the P and Q bits.

K places the FAP word 'FW' in the logical accumulator and clears the sign and Q bits.

A (or K) without argument types out, in the current mode, the contents of the signed (logical) accumulator followed by the P and Q (sign and Q) bits.

STORAGE MAP: M

M requests the typing of the storage map with subprograms listed in order of loading. The map includes the origin and entry points with their locations.

Subprogram Control

The requests which have to do with subprogram control allow the user to run his subprogram in controlled segments. They are break, go, and proceed.

BREAK: B -LOC-

Conditions FAPDBG to insert a "breakpoint" at location LOC. FAPDBG will save the location

and set an indicator to signal that a breakpoint instruction, specifically a transfer into FAPDBG, is to be inserted into that location. No subprogram modification occurs at this time. An examination of the breakpoint location will reveal its original contents and changing the contents (via a deposit request) will not remove the breakpoint. The breakpoint must not be placed at a subprogram-modified instruction or where it would be used for indirect addressing. Only one breakpoint at a time may be inserted.

The omission of LOC in the request causes the breakpoint to be removed.

GO: G LOC

Allows the user to start execution of the subprogram at location, LOC. FAPDBG will examine the breakpoint flag and, if a breakpoint exists, will save the contents of the break location and insert the necessary transfer instruction. It will then restore the machine conditions, and transfer to the specified location.

PROCEED: P

Allows the user to continue executing his subprogram from the state it was in just before control last entered FAPDBG. Upon encountering the breakpoint transfer instruction, control will be transferred to FAPDBG, which will save the machine conditions and restore the temporarily-removed instruction at the break location. FAPDBG will then type "BREAK." and wait for requests.

Proceed will cause FAPDBG to perform all the steps performed by go, except that after restoring the machine conditions, FAPDBG will execute the above-mentioned instruction and transfer to the appropriate location following its location as governed by any skipping which might occur. If the instruction is location-dependent, namely TSX, STR, STL, or XEC, FAPDBG will interpret it as if it were being executed from its normal location. Thus a breakpoint may be inserted at a subroutine call. A chain of XEC instructions will be interpreted to a maximum depth of ten. A subprogram in operation may be interrupted at

any time by pressing the interrupt button.

Internal Operation

The request which controls the internal operation allows the user to return to CTSS. It is quit.

QUIT: Q

Returns control to the Time Sharing Supervisor in such a way that a START command will transfer control to the place in the user's subprogram where it last entered dormant status.

Internal Symbols

The following symbols are permanently defined in FAPDBG as locations where the machine conditions are stored.

$MQ The multiplier-quotient register.
$A The signed accumulator
$K The logical accumulator
$SI The sense indicator register.
$X1 Index register one.
$X2 Index register two.
$X3 Index register three.
$X4 Index register four.
$X5 Index register five.
$X6 Index register six.
$X7 Index register seven.
* The current location.

This symbol is defined as the last location referred to by either the user or FAPDBG. It is redefined as the location of the next instruction to be executed in the user's subprogram by encountering a breakpoint or by a manual restart.

$LS Lights and switches.

This location contains the state of the machine conditions in the right-most eight octal digits as listed below; the off status is represented by zero, on status by one. Reading from left to right:

DIGIT	CONDITION
5	Floating point trap mode.
6	Divide check light.
7	Overflow light.
8	Multiple tagging light.
9	Sense light one.
10	Sense light two.
11	Sense light three.
12	Sense light four.

$IC The instruction location counter.

This location contains the address of the next instruction to be executed in the user's subprogram. It is set by encountering a breakpoint or by a manual restart. It is examined by the proceed request in order to determine the location to which to transfer control.

Summary of Requests in Alphabetic Order

Request	Meaning
A FW	CLA =FW
A	Type out signed AC with P and Q
B LOC	Insert breakpoint at LOC
B	Remove breakpoint
C EP FN	Compare and type discrepancies of subprogram EP and FN BSS.
C EP	C EP EP i.e., EP and FN are identical.
D LOC FW	Deposit =FW in LOC
E FE FS	Define symbol FS equal to expression FE
F	Set output mode to floating point
F LOC	Set floating point and type C(LOC)
F LOC1 LOC2	Set floating point and type C(LOC1 thru LOC2)
G LOC	Go to LOC
H	Set output mode to Hollerith
H LOC	Set Hollerith and type C(LOC)
H LOC1 LOC2	Set Hollerith and type C(LOC1 thru LOC2)
I	Set output mode to decimal integer
I LOC	Set integer and type C(LOC)
I LOC1 LOC2	Set integer and type C(LOC1 thru LOC2)
J	Set output mode to decrement integer
J LOC	Set decrement integer and type C(LOC)
J LOC1 LOC2	Set decrement integer and type C(LOC1 thru LOC2)
K FW	CAL =FW
K	Type logical AC with S and Q.
L EP	Find and type origin of subprogram EP
M	Type storage map
O	Set output mode to octal
O LOC	Set octal and type C(LOC)
O LOC1 LOC2	Set octal and type C(LOC1 thru LOC2)
P	Proceed (location in $IC) or interrupt after break
Q	Quit and return to CTSS
S	Set output to symbolic
S LOC	Set symbolic and type C(LOC)
S LOC1 LOC2	Set symbolic and type C(LOC1 thru LOC2)
T FN	Add symbols from FN SYMTB to symbol table (relocated by last origin typed out)
T	Remove all symbols from symbol table
W EP FN	L EP' T FN; work subprogram EP with symbols from FN SYMTB
W EP	L EP' T EP: FN is identical to EP

Identification

MADBUG - A MAD Debugging System
Robert S. Fabry

Purpose

MADBUG is a system under which the user can create and debug programs written in the MAD programming language. MADBUG allows the user to input and edit symbolic programs and to execute in a controlled way and interrogate the derived machine language programs. The most important consideration in the design of MADBUG was ease in learning and using, both for the beginner and for the advanced programmer. MADBUG is unusual in that it utilizes information which has been previously ignored. This information comes from: (1) the sequence in which the user types his requests, (2) the files available in the user's file directory, (3) the expanded information content of the new MAD symbol table files developed for MADBUG, and (4) the information inherent in the very limited, stylized set of coding sequences generated by a compiler. The use of this additional information manifests itself in two ways: (1) the user need provide very little information to accomplish a given task, and (2) the user does not have to understand assembly languages, machine languages, octal numbers, relative or absolute addresses, symbol tables, machine representations of constants, or any of a host of similar items. The MADBUG requests of CHANGE, DELETE, INSERT, and APPEND demonstrate the influence of the "Expensive Typewriter" program written for the PDP-1 by Steve Piner. The "DDT" program written for the PDP-1 by Robert Saunders and the "FLIT" program written for the TX-0 by Jack Dennis and Thomas Stockham have influenced the OPEN, VERIFY, BREAK, and KILL requests.

Reference

CC-247 and Mac-M-205 MADBUG: A MAD DEBUGGING SYSTEM R.S. Fabry

A DESCRIPTION OF MADBUG

MADBUG is instructed by requests, typed one per line. A request line is made up of the name of the request followed by its arguments, with one or more blanks for separation. Request names may be abbreviated by their first letter. In request lines, tabulation characters are equivalent to blanks. There may be blanks before the request name and after the last argument; blank request lines are ignored. Since blanks are used as delimiters, the arguments, which may be as complicated as "a(1)+1...b-3", must be typed without internal blanks. A request which operates on variables will operate on single variables or on blocks of variables, specified in the usual MAD manner as "alpha...beta"; a request which operates on cards will operate on single cards or on blocks of cards. For example, "verify alpha beta(1)...beta(3) k(1,1,1)" would verify, in a sense described later, the variables ALPHA, BETA(1), BETA(2), BETA(3), and K(1,1,1).

MADBUG requests can be classified into four groups: the edit requests which are PRINT, DELETE, INSERT, CHANGE, APPEND, MANIPULATE, and TRANSLATE; the core requests which are GO, OPEN, VERIFY, LINKAGE, BREAK, KILL, SAVE, and RESTORE; the requests for returning to CTSS which are QUIT and EXECUTE; and the declarations which are WORK, USE, and FORCE. These requests will be discussed in the next few sections.

The Work Request:

The MADBUG requests are carried out in the context of a single MAD subprogram. The WORK request allows the user to declare which subprogram is of interest. For example: "work prog" sets up MADBUG to work on the program in file PROG MAD. The file PROG MAD does not have to exist. As illustrated in the sample session, if the user adds lines to a non-existant file, MADBUG will create the file. Thus, if the user is working in the context of a subprogram PROG, and wishes to print a subprogram ROOT, he must first request "work root" and then may request "print".

Edit Requests:

MADBUG uses a different technique for editing than the CTSS EDIT command. Neither the user nor MADBUG supplies a line number for a card image. Instead of indicating a card image by giving its associated line number, the user has three options: (1) the statement label on the card, if any; (2) the card's position relative to another card which has a statement label (the third card before ALPHA is ALPHA-3; and (3) the number of the card in the deck (the 17th card in the deck is simply 17). In counting for (2) or (3), the user must count all physical card images including remark and continuation cards. MADBUG interprets the arguments of a

request before executing the request; thus, if a deck consisted of three cards, "delete 1 2" would leave the third card, but "delete 1" followed on another line by "delete 2" would leave the second card.

In unusual situations there may be a long section of program with no statement labels. The user is free to insert remark cards with statement labels in such a case. MADBUG, but not the MAD translator, will allow references to statement labels on remark cards.

Three special conventions exist for specifying statement labels: (1) the "*" is always taken to mean the previous card referred to by the user, so that a "print *+3" after a "print 6" would print the 9th card, and so that a "print alpha...*+2" would print three cards starting with ALPHA. (2) the "/" is always taken to mean the last card in the deck, so that, in a five card program, "print 1 3 5" is identical to "print 1 3 /". (3) Requests which operate on cards will operate on every card in the subprogram if no cards are specified, so that "print" is identical to "print 1.../".

MADBUG observes the standard conventions of horizontal spacing: the characters after a tab will be moved to column 12 and the characters after a tab-backspace will be moved to column 11.

The description of several of the editing requests will refer to input line blocks. An input line block consists of all the lines the user types before typing a blank line. The editing requests are defined as follows:

PRINT will print all cards mentioned as arguments. Thus, "print a(1)+1...b-3" would print a block of cards starting with the card after the card labeled A(1) and ending with the third card before the card labeled B.

DELETE will delete all cards mentioned as arguments. Thus, "delete" would delete all of the cards of the subprogram being worked, and "delete 1 3...6" would delete the first and the third through sixth cards.

INSERT will insert successive input line blocks before successive cards mentioned as arguments. Thus, one might see the following sequence:

```
U:print
M:ONE
M:
U:insert
```

```
U:zero
U:
U:print
M:ZERO
M:ONE
M:
U:insert 1 one
U:a
U:
U:b
U:
U:print
M:A
M:ZERO
M:B
M:ONE
M:
```

CHANGE will replace successive cards or blocks of cards, given as arguments, by successive input line blocks. A block containing any number of cards may be replaced by an input line block of any length.

APPEND with no arguments will append the input line block which follows the request line to the subprogram being worked. On the other hand, if the request has arguments, they are taken to refer to MAD subprograms which will be appended, in order, to the program being worked.

APPEND is also useful for creating a modified version of a subprogram while keeping the original. To do this, WORK the new name, APPEND the old name, and then make modifications.

MANIPULATE is a request for character manipulation within a card image. The first argument specifies the manipulation. Arguments after the first specify cards within which the manipulation will be performed. The first argument has the form: /***/***/ where the slash stands for any separation or delimiter character which must occur exactly three times, and the strings of asterisks stand for any pair of character strings. The manipulation consists of replacing all occurances of the first string by the second string. Any character except a tab or space may be used as the delimiter; it is recognized by its being the first character of the argument. The two character strings may include any characters except the

delimiter and the carriage return, and they may be of different lengths. If the first string is empty, it will be taken to match a null string before column one on the card, thus allowing a simple way of inserting a statement label on a card. As a confirmation to the user, MADBUG will print a list of cards on which the manipulation is performed. If the manipulation is performed more than once on a card, the card will be included in the list once for each time the manipulation occurs. MADBUG does not consider replacing a string by itself to change the symbolic program. Thus the user can replace a string by itself to locate all occurences of the string.

TRANSLATE has no arguments, and causes the subprogram being worked to be translated into machine language by the MAD compiler. From the user's point of view MADBUG is performing the translation. It is not necessary to translate any subprogram before using it. MADBUG will request any translations that are needed at load time. The TRANSLATE request is a convenience to the user who is changing several subprograms at one time, and who would like to catch any syntactic errors in one before turning his thoughts to another.

The Use Request:

The core requests, which will be discussed in the next section, operate in the context of a core image. MADBUG must have some way of knowing what subprograms to load when creating a core image. The arguments of the USE request are the subprograms to be used. Thus a user writing a subroutine ROOT and a test program MAIN might "use main root". There are provisions for using FAP programs, special libraries, and special loader parameters; these provisions are described later.

Core Image Requests:

Some core requests require cards for arguments, and their arguments observe the same conventions as those of the edit requests. A core request which refers to a declaration or remark card will operate on the first executable statement following the referenced card. Other core requests require variables for arguments. A variable is given as an argument in standard MAD notation, including multi-dimensional arrays and COMMON and ERASABLE variables, but not the dummy arguments of functions. Three special conventions exist for variables: (1) the "*" is always taken to mean the

previous variable refered to by the user; (2) if no variables are specified, the request will operate on every variable in the program; and (3) the block notation can be used to include several arrays or variables at once. Variables are taken to be ordered alphabetically (with a blank coming after R, alas.) and then by linear subscript.

The first time the user gives a core request, a core image must be created by MADBUG. This is accomplished by translating each of the needed subprograms into machine language, if necessary, loading the subprograms into core, and finally modifying some of the subprograms in order to intercept illegal references to an array. If an error is detected in this process, the core image will not be formed, and the core request will be terminated. The user should correct the error and try the core request again. The core image will be destroyed when the user issues the quit request or edits a program occuring in the core image. The core requests are defined as follows:

GO will start the user program. A single card given as an argument for GO will cause the user program to be started at the named card. If no argument is given, the user program will be started wherever it stopped last. A fresh core image will start at the beginning of the main program.

The user program will remain in control until (1) it terminates by calling DEAD, DORMNT, ENDJOB, ERROR, or EXIT; (EXIT can be implicitly called by letting control reach an END OF PROGRAM or END OF FUNCTION card.) (2) a "breakpoint" is encountered by the user program; (3) the user interrupts by pushing the break button once; or (4) an array is referenced with subscripts pointing outside of the dimensioned array. (Some array dimension violations are not caught; this is discussed in a later section.) On any of these occasions, control returns to MADBUG, and the user is informed of the reason.

Infrequently, the user program may have an error which causes control to return to CTSS. In this case, the user should type two CTSS commands, first "save (user)" to save his own core image and second "resume (mdbg)" to return control to the core image on which MADBUG saved itself. Even if the first of these commands results in an error comment from CTSS, the user should type the second. This procedure is called a manual restart.

OPEN will print the contents of variables mentioned as arguments, one by one, and after each, wait for the user to type a new value for the variable. If the user wishes the old value to remain, he just types a carriage return. In typing out the value of a variable, MADBUG makes use of the declared mode of the variable and of the current value to decide whether the value should be presented to the user in integer, alphabetic, floating-point, Boolean, statement label, or function mode. The user must type a constant for the new values in a form compatible with the declared mode of the variable. It is possible to change the input/output form associated with a declared mode permanently or to override the normal associations for a single request. This is discussed later.

One special note: because of the way the MAD compiler works, one may change the effect of a transfer statement by changing the value the variable which has the same name as the statement label to which the statement transfers. One may not, however, change the scope of a THROUGH loop in this fashion, even by changing the value of the variable with the same name as the THROUGH scope.

VERIFY will cause the values of variables mentioned as arguments to be compared with the values of the same variables in a fresh, unexecuted version of core. Each variable whose value has changed will be printed with its present value. Its value in the fresh version of core will also be printed if it is non-zero.

An option is available with verify; the user may specify any core image saved with the SAVE request to be used instead of the fresh copy of core discussed above. This is done by giving the name of the saved image following the request name and before the list of variables to be varified. As the user will discover below, this name must begin with an asterisk, and can thus be recognized by MADBUG.

The discussion of output forms used for the values of variables, which was given under the OPEN request, also holds for the VERIFY request.

LINKAGE causes MADBUG to tell the user which statement made the most recent call to the external function subprogram currently being worked.

BREAK will modify the machine language program in the current user core image so that control will return to MADBUG if one of the cards given as arguments is to be executed. When MADBUG regains control from the user program, the name of the statement which is about to be executed will be printed for the user. At this time the user will usually examine variables in his program to determine what his program is doing. "Breakpoints", as these points in the user core are called, belong to a given core image, and can vary from one saved core image to another. (See the SAVE request.)

KILL will remove any breakpoints at cards mentioned as arguments. It is not an error to insert a breakpoint where one already exists nor to remove one which does not exist. For example, to kill all the breakpoints in the subprogram being worked, "kill".

SAVE has a single name as its argument and causes a copy of the current user core image to be saved as a CTSS file with the primary name given as an argument and the secondary name SAVED. The name given by the user must begin with an asterisk. The current user core image was produced by loading, and has been modified by execution and by MADBUG requests. One may save the current core image under a name which has already been used for a save request. In this case, the current core image will replace the previous core image. All the core images saved using the SAVE request will be destroyed when the user's current core image is destroyed. This is because the saved files created by MADBUG are not normal CTSS saved files, and are useless out of the context of MADBUG.

RESTORE will replace the current user core image with a copy of the image whose name is given as an argument. The core image name must be a name under which the user has saved a core image using the SAVE request, or it must be *FRESH. *FRESH is a byproduct of the loading process. It is a completely unexecuted version of core with no breakpoints and with all variables at their initial values. Except for the special

way in which it is created, *FRESH is like any normal core image saved by the SAVE request.

Getting Back to CTSS:

When the user is finished with MADBUG, and desires to return to CTSS, he should use the QUIT request. The QUIT request will destroy all the files created during the session, except for the modified MAD programs and their associated BSS and SYMTAB files.

The EXECUTE request allows the user to return to CTSS for a single command, without ending his session with MADBUG. For example, the user could effect the CTSS command "listf aa mad" by requesting "execute listf aa mad". These commands are executed using the command chaining technique with the sequence: "save (mdbg)", the user's command, and "resume (mdbg)". No provision is made for saving a core image which might result from the user's command.

SPECIALIZED FEATURES AND TECHNIQUES

Two error comments that the user may get from MADBUG have special significance. One is "TRY AGAIN.", which always means that the current request has been terminated. The other is "CONSULT LISTINGS." which can only occur as a result of a bug in MADBUG. Any user getting this comment will please retain as much information in the way of output, files, etc. as he can and call Bob Fabry, x2524, so the bug can be removed promptly. The user can often continue with more requests in spite of a "CONSULT LISTINGS." error.

Two types of improper array references are not caught. First, references with a constant linear subscript are not checked. For example, one might DIMENSION A(10) and A(20)=100. Second, references to arrays which are given as arguments to functions are not checked. For example, one could have called for ROOT.(A(K)) where K is 20. This situation can sometimes be avoided by placing arrays in COMMON, and not passing them back and forth as arguments.

In unusual cases, the user core image may "blow-up" in such a way that the information about control and about the values of variables is gone or meaningless. In this case the user will still find MADBUG a useful tool, and may approach the problem by an exponential search through time for the point at which the blow-up occurs. Stated another way, this amounts to performing a series of tests in which each test is designed to cut by a half the uncertainty about when the blow-up occurs. When the user knows the exact point of the blow-up, he can then step through very cautiously, looking for clues. Such an approach relies heavily on BREAK, KILL, SAVE and RESTORE. At the start, the user moves a core image as close to the blow-up as he knows he can, SAVEs the core image, and guesses the half-way mark, in terms of opportunities for bugs, to a place by which the blow-up must have occured. He then uses BREAK and KILL to step his current core image to the half-way point he guessed. (1) If the core image blows-up in this process, he guesses a new half-way point, half way between his saved image and his old half-way mark, RESTOREs his saved core image, and trys his new guess. (2) If the core image doesn't blow-up in the process, he SAVEs his current core image for a new starting point, guesses a new half-way mark between his new core image and the blow-up, and trys this new guess. This process is fairly simple to carry out using MADBUG, and most blow-ups can be readily solved this way.

When loading is performed, MADBUG will normally load a program named (MDBG), which MADBUG provides, immediately following the files specified by the USE request. Then MADBUG will process the core images of all programs loaded into core before (MDBG) and insert patches, using an area reserved in (MDBG), to attempt to catch any user subprogram

when it accesses an array with an illegal subscript. If the user wishes to load programs which were written in FAP, MAD programs for which the symbolic programs are not available, debugged MAD programs which he does not wish to protect, or library files, he may specify the position of (MDBG) by typing (MDBG) in place of a file name in the USE request. All the files before this parameter will be treated normally, and all things after it will be ignored by MADBUG and just passed on to the loader. Any loader parameters, such as (CFLP) or (LIBE), can also be used after (MDBG). If the user needs more than eighty characters for his USE request, he may type a hypen as an argument of use. When the hyphen is encountered, MADBUG will immediately read the next input line for more arguments for the USE request. This may be done for several successive lines.

The FORCE request forces certain internal registers in MADBUG to new values, picked by the user. To FORCE a parameter, give the name of the parameter as the first argument of FORCE, and give remaining arguments as required by the parameter being forced:

FORCE PATCH will set the amount of patch space available in the user core images to the decimal number given as the argument. Initially PATCH is set to 500. The patch space is used during loading and whenever breakpoints are inserted. FORCE PATCH does not change the available patch space immediately, since the internal register is examined only during loading. A user would reduce the patch space if he was squeezed for core space. He would increase it if MADBUG complains, during loading, that there is not enough patch space, or if he exhausted the patch space inserting breakpoints. If the patch space is exhausted by breakpoints, however, it is usually sufficient to KILL some of the less neccessary breakpoints to get space for new ones.

FORCE FORMAT will set the normal input/output form associated with each of the possible modes for variables. After the word FORMAT, the arguments are taken in pairs, the first item of the pair indicates a mode and the second indicates a form. The modes are indicated by a digit from 0 to 7, standing for floating-point, integer, Boolean, function, statement label, mode 5, mode 6, and mode 7, in that order. The form designation is one of the following: "Gn" for floating point with n significant figures on output, "I" for integer, "A" for alphabetic, "P" for either

integer or alphabetic with MADBUG picking for output, "Ø" for octal, "B" for Boolean, "S" for statement label, and "F" for function. Initially, FORMAT is set to: 0 G3 1 P 2 B 3 F 4 S 5 Ø 6 Ø 7 Ø. (In this section, "Ø" is used to denote the letter "O".)

FORCE MODE alows the user to predetermine whether MADBUG saves itself as a permanant mode file or as a temporary mode file. The values of MODE are, correspondingly, "P" and "T". Mode is originally set to "P". The user will want to FORCE MODE to temporary if he is not interested in extreme reliability as much as in conserving his track allotment.

It is also possible to override all the normal I/O forms for the duration of one OPEN or VERIFY request. To do this, use one of the form designations listed above, but preceded by a slash. Insert it after VERIFY (and the saved file name, if present) or OPEN and before the arguments. For example, "open /o alpha".

MADBUG observes the convention that the first statement of a main program starts after the call to .SETUP which the compiler always inserts as the first executable machine instruction. Another convention at this level is imposed by the compiler. A breakpoint on an ENTRY TO statement will not be encountered when the entry is called, but will be encountered if control is transfered to the statement or falls to the statement.

MADBUG creates and destroys special files as it processes the user's requests. They are destroyed during the processing of the same request for which they are created. Normally, the user will not have to worry about them, but occasionally he may be made aware of their existance. (MDBG) SAVED is the name under which MADBUG saves itself when it chains to other commands. This file will vary in length during a session, but will be on the order of 30 tracks long. Its mode depends on the value of MODE, as described earlier. (TEMP) (MDBG) is used during file modification. When a word in a file must be modified, the modified file is first created as (TEMP) (MDBG), and then the original file is deleted and (TEMP) (MDBG) is renamed. The length of this file depends on the length of the file being modified. The file has permanent mode. (MDBG) BSS is created by MADBUG whenever loading is required. Its position in the new core image was discussed earlier. It contains the bootstrap for MADBUG and the patch area. It is one track long and has temporary mode. (MBGI) SAVED is a very short program which processes the input line blocks the user types while editing. It processes all the input line blocks associated with one edit request and reads in the

following request before chaining back to MADBUG. It is usually one track long and is permanent mode.

A user core image may use the command buffers. A call to CHNCOM will not return control to MADBUG. MADBUG saves the command buffers and counter initially and restores them when the user gives the QUIT request. MADBUG also treats the command buffers and counter as psuedo-machine conditions associated with each core image. The buffers are only lost on manual restart. A fresh core image has empty buffers.

By editing, the user modifies the MAD subprogram on which he is working. By inserting and removing breakpoints and by changing the values of variables, the user modifies the current user core image, (USER) SAVED. MADBUG does not change external files until the changes are logically needed. If the user uses EXECUTE to ask CTSS to process these files, he may want to insure that these logical modifications are made physically. To insure that the MAD subprogram being worked is modified physically, give a redundant WORK request using the name of the subprogram already being worked. Whenever a WORK request is given, the logical modifications associated with the subprogram previously being worked are made physically. To insure that the current user core image is modified physically, use a SAVE request. A user who cannot afford the added tracks can give an "execute delete" on the created SAVED file. This variation between the physical and logical modifications provides some degree of safety to the user who carelessly makes gross incorrect modifications to one of his programs. If the user should accidently type a "d" as a request line for example, he should quit by hitting the break button twice in succession. This will prevent MADBUG from actually deleting the file in question.

SUMMARY OF MADBUG REQUESTS

request	arguments	additional lines (3)
work	subprogram name	none
print	card names (1)	card images by MADBUG
delete	card names (1)	none
insert	card names (1)	card images by user
change	card names (1)	card images by user
append	none	card images by user
	(or) subprogram names	none
manipulate	special, then cards	card names by MADBUG
translate	none	comments by MADBUG
use	subprogram names	none
go	card name or none	comments by MADBUG (4)
open	variables (1,2)	values by both (4)
verify	variables (1,2,5)	values by MADBUG (4)
linkage	none	linkage by MADBUG (4)
break	card names (1)	none (4)
kill	card names (1)	none (4)
save	save-name	none (4)
restore	save-name	none (4)
quit	none	none
execute	command and arguments	depends on command
force	parameter, special	none

notes: (1) If none, all are implied.
(2) Optional form forcing first argument.
(3) Any request can get error comments from MADBUG.
(4) Comments by MADBUG if core image is created.
(5) There is an optional save-name argument.

Revised: 9/24/65

Identification

Post Mortem Debugging
PM

Purpose

Produce post-mortem information about the user's last dormant program (loaded by the relocatable program loader).

Restrictions

The program should be loaded by LOAD or LOADGO so that the loader and movie table are available.

Usage

The PM command may be followed by one of several requests.

PM 'ILC.' Gives the stop location or ILC (1 line). *

PM 'LIGHTS' Gives machine conditions and ILC (4 lines).

PM 'TRAPS.' Gives contents of trap location (1 line) *

PM 'STOP' Gives ILC and contents of two locations on either side of the stop (5 lines)

PM 'AUTO' Corresponds to LIGHTS plus STOP (9 lines.)

PM 'STOMAP' Gives origin and entry of all subprograms loaded.

PM NAME 'STOMAP' Gives the origin and entry of all * subprograms loaded beginning with NAME.

PM NAME Gives contents of four initial locations of * subprogram NAME (5 lines).

PM NAME LOC1 LOC2 -MODE- -DIRECTION-
Gives contents of all locations from relative location LOC1 through LOC2 of subprogram NAME in the specified mode and direction. NAME is '(MAIN)' for the main program. LOC is assumed to be decimal if the number is preceded by a slash, '/', it is taken as octal. MODE specifies the form of printed output and may be 'FIX', 'FLØ', 'DEC', 'ØCT', 'BCD', or 'ALL'. DIRECTION specifies the order of printing and may be 'FWD' or 'REV'. If MODE is ommitted 'ALL' is assumed if DIRECTION is omitted, 'FWD' is assumed. LOC1 and LOC2 may be replaced by 'ENTIRE' to cause printing of

the entire program.

PM LOC1 LOC2 -MODE- -DIRECTION-
Gives the contents of absolute locations LOC1 thru LOC2.

References to COMMON must be the high core locations which appear in the assembly listing, not the lower core area actually used for COMMON. (Caution: illegal requests, either outside the program range or improper requests for COMMON, cannot be interpreted correctly.)

Revised: 9/24/65

Identification

Relocatable program patching
PATCH, STOPAT, TRA

Purpose

To allow break points to be set in a program after it has been loaded, to allow transfer of control to a specified location, and to allow modification of the loaded program.

Restrictions

These service routines are normally loaded after the program is loaded and so the loader must be available in core. Therefore LOAD or LOADGO should be used for loading the program.

Usage

Set a break point:

STOPAT ENTRY RELLOC

ENTRY is an entry point in the desired subprogram. If ENTRY is omitted, the main program is assumed.

RELLOC is the relative octal location in the specified subprogram at which the break point stop is to occur.

STOPAT replaces the instruction at RELLOC with a transfer. When the transfer is executed, the original contents of RELLOC is restored and the program is placed in dormant status. The START command may then be used to continue with the execution of the original contents of RELLOC.

Transfer:

TRA ENTRY RELLOC

Same argument specifications as STOPAT. The issuance of the START command will cause a transfer to RELLOC. This may be used to restart the program from different locations during debugging sessions.

Modify the program:

PATCH ARG

ARG=entry: ARG may be the entry point of a subprogram which is to be patched by referring to relative locations within the subprogram. If ARG is omitted, (MAIN) is assumed.

ARG='(ABS)' allows patches to absolute locations.

ARG='(COM)' allows patches to relative locations within the COMMON region.

ARG='(PAT)' allows patches to be entered into locations above the user's current memory bound. This patch space is referenced by relative locations and is shared by all subprograms.

After a response from the PATCH command, the user enters lines of the form:

LOC, TYPE, VALUE, RELOC

LOC is the octal address to be patched. This octal number may be immediately followed by a special letter if it is desirable to override ARG for this response. The special letter may be A for absolute location, C for relative location in common, or P for a relative location in the patch space.

TYPE is the type of value to follow i.e.,
'OCT', octal word (used for instructions)
'FLO', fixed or floating-point number (E or F notation)
'INT', fortran integer
'DEC', MAD integer

VALUE is the number to be patched into LOC.

RELOC is the relocation specification for VALUE if TYPE is 'OCT'. It consists of two letters, the first for the decrement and the second for
A: absolute
R: relocatable
C: common
P: Patch space

If RELOC is omitted, AR is assumed. Successive VALUEs and appropriate RELOCs may be specified in any line.
Exit from PATCH by typing 'END'.

Identification

Absolute program patching
SPATCH

Purpose

Programs loaded with LDABS, NCLOAD, or VLOAD may be patched using some supervisor routines which do not require special loading and movie tables. This is accomplished by patching their SAVED file, rather than the core B program directly.

Usage

```
SPATCH  NAME1  LOC  A1  B1  A3  B2 ... An  Bn

    SPATCH  NAME2  ILC  L
```

SPATCH patches the file NAME1 SAVED beginning at absolute octal location LOC for n locations. If LOC is 'ILC', only the ILC of NAME1 SAVED will be patched, causing a transfer of control to absolute location L when NAME1 is RESUMED.

Ai Bi are the octal left and right half words respectively.

L is the location at which control should be RESUMED.

Identification

Supervisor debugging
SD, SP

Purpose

To allow for printing and patching the supervisor (core A).

Usage

The printing routine has several options:

SD ENTRY RELOC N

N consecutive locations starting at relative octal location RELOC in subprogram ENTRY in the supervisor will be typed on the user's console in unrelocated (i.e., relative) octal with operation code mnemonics. If N is omitted, it is assumed to be 1. If ENTRY is omitted, the request is taken to be absolute. Lines of zero are not supressed.

SD ENTRY 'TRACE'

The name of the calling subprogram and the relative location from which subprogram ENTRY was last called will be printed on the user's console. The user may continue tracing back by typing a carriage return. The trace may be terminated by the QUIT signal.

SD 'STOMAP'

A storage map of all subprograms loaded into the supervisor's core (core A) will be printed.

SD ENTRY

The contents of the specified entry will be printed on the user's console in appropriate form (BCD for LDNAME, all others in 5 octal digits).

Patching:

SP ENTRY RELOC A1 B1 C1 A2 B2 C2 ... AN BN CN

Patching will begin in relative octal location RELOC within the subprogram ENTRY. Al Bl are the relocatable octal left and right half-words, respectively. The Ci contain two

characters indicating how the left and right half-words are to be relocated. The characters may be A for absolute or R for relocatable. If a Ci is ommitted, it is assumed to be AR. If ENTRY and Ci are omitted, the patching is absolute.

Identification

STRACE - A trace debugging routine
B.L. Wolman, x6022

Purpose

STRACE (Subroutine TRACE) is a debugging program which allows the user to monitor the calls of selected subroutines. A set of conditions may be specified for each subroutine to be traced. At each call of the subroutine, STRACE checks to see if all the conditions are met. If they are, STRACE prints a message identifying the subprogram called, how many times it has been called, the absolute location of the call, the program in which the call occurred, and the relative location within the program making the call.

The user may request STRACE to STOP execution before executing a subroutine or to HALT after the subroutine has been called. If either of these options are used, STRACE will print an identifying message before going to dormant status. PM or OCTLK may be used to inspect the machine conditions. Issuing the START command will cause execution to continue.

The user may also specify a debugging subroutine which is to be called before executing a subroutine. This debugging subroutine may perform any function the user desires; the call issued by STRACE is of the form

DEBUG(LOC, ARG)

where DEBUG is the debugging subroutine name, LOC is the location of the call to the subroutine being traced, and ARG is a parameter previously specified by the user.

Options are also available which allow the user to obtain octal snapshot dumps of the machine registers, the subroutine calling sequence, and the value returned by the subroutine in the accumulator.

Usage

STRACE may be entered by issuing the CTSS command STRACE. Because of the method of implementation, the loader must be present in memory. The STRACE command may be issued immediately after loading, after a QUIT signal, or after a trace stop. (In general, STRACE may be entered any time the user's program is in dormant status). At the end of the input phase, STRACE will return to dormant in such a manner that the START command will cause execution to be resumed at the point where it was interrupted.

TRACE is an alternate entry which may be called as a subroutine. In this case, TRACE returns to 1,4 in the calling sequence. The calls are of the following form

```
AED          TRACE() $,
MAD          EXECUTE TRACE.
FORTRAN     CALL TRACE
FAP          TSX   $TRACE,4
```

When STRACE is ready for input or more input, it prints the word TYPE. and waits. After receiving this response, the user may enter a series of commands. Each command consists of a subroutine name followed by one or more requests. Within a command, blanks are used to separate requests and their parameters. Since a carriage return is completely equivalent to a blank, commands may be split across one or more lines of input. Each command is terminated by a comma. The last command is terminated by an asterisk which signifies the end of the input phase.

The following requests are currently recognized by STRACE (N and M are positive decimal integers less than 32768, DEBUG is the name of a subroutine).

AFTER N - Begin tracing after the Nth call of the subroutine.

EVERY N - Trace every Nth call. N should be non-zero.

UNTIL N - Trace until the Nth call. The AFTER condition should be less than the UNTIL condition.

STOP N - Go to dormant <u>before</u> every Nth call. If N is zero, the STOP condition will be removed.

HALT N - Go to dormant <u>after</u> every Nth call. Execution will be interrupted <u>after</u> the specified subroutine has been executed and <u>before</u> it has returned to the program making the call. This request should <u>not</u> be used if the subroutine being traced has an error return or does not always return to the same point in the calling sequence. If N is zero, the HALT condition will be removed.

ARGS N - Every N times it is called, print the arguments of the subroutine. Each word in the calling sequence is assumed to specify a single variable. The absolute and relative addresses of these variables and their contents will be printed in octal. The

relative location will be "****" whenever the specified location is in COMMON or is in turn an argument of the subroutine making the call. Whenever N is zero, the ARGS condition will be removed.

VALUE N - Print the value of the specified subroutine. The value of the subroutine will be obtained by interrupting execution in the same manner as the HALT request; the same restriction applies. The VALUE condition will be removed whenever N is zero.

PM N W1 W2 ... Wn - Every N times the specified subroutine is called, print an octal snapshot dump of the machine registers specified by the parameters W1 to Wn. The Wi's may be any of the following words.

AC	Accumulator, Q and P bits
MQ	Multiplier-quotient register
SI	Sense indicators
MB	Memory bound
X1	Index register 1
X2	Index register 2
X3	Index register 3
X4	Index register 4
X5	Index register 5
X6	Index register 6
X7	Index register 7
L1	First location in subroutine calling sequence
L2	Second location in calling sequence
L3	Third location in calling sequence
C1	First argument of subroutine
C2	Second argument of subroutine
C3	Third argument of subroutine
XS	Equivalent to the sequence X1 X2 X3 X4 X5 X6 X7
ALL	Equivalent to the sequence AC MQ SI MB XS

If any of the above words appears with an initial minus sign in the request, the PM of the corresponding register(s) will be removed. Because the PM request has a variable number of parameters, it must be the last request of any command. The PM print occurs after any call of a debugging subroutine and before any stop. The request PM 0 will suspend all PM requests for the particular subroutine.

CALL N DEBUG M - Before every Nth call, execute the debugging subroutine DEBUG with parameter M. If N is zero, the CALL condition will be removed; in this case the debugging subroutine name and the parameter M should not appear. If M is zero, the parameter used in the call of DEBUG will be the number of

times the subroutine being traced has been executed. If both the STOP condition and the CALL condition are simultaneously satisfied, the CALL of the debugging subroutine will occur before the STOP.

COUNT N - Reset the execution count of the subroutine to N. This request may be used to continue tracing after the UNTIL limit has been reached.

REMOVE - Remove the subroutine from the internal trace table. After this request has been given, STRACE will have no record of or control over calls to the subroutine.

OFF - Turn off tracing of this subroutine. All succeeding calls will be ignored until tracing is restored via the ON request.

ON - Restore tracing of this subroutine.

FIND - Print the entry point of the subroutine. Any requests after the FIND will be ignored. FIND should only be used if no tracing is desired, since entry points are automatically printed the first time a subroutine name is encountered during the input phase.

If no request is given following the subroutine name, the standard requests

```
AFTER 0 EVERY 1 UNTIL 32767 STOP 0
CALL 0 HALT 0 ARGS 0 VALUE 0 PM 0
```

are assumed. Any requests given by the user override the corresponding standard value. Any of the tracing parameters of a subroutine may be changed by the user in a later entry to STRACE.

Method

When STRACE is asked to trace a subroutine, it saves the name of the subroutine in an internal table. STRACE searches the MOVIE) table for the named subroutine. If it is found, STRACE obtains the entry point. STRACE then uses the MOVIE) table to find the origins of all programs in core. When it finds a program that has a transfer vector, it searches this transfer vector for a TTR to the subroutine entry point. If a TTR is found, it is changed to a TXL TRAP,,TABLE where TRAP is the address of the trace processing section of STRACE and TABLE is the index of the subroutine being traced in the internal trace table.

The REMOVE request causes essentially the inverse operation to be performed. All TXL TRAP,,TABLE instructions are changed to TTR ENTRY and the subroutine is removed from the internal table.

During execution of the user's program a call to a traced subroutine will result in a TSX to the TXL instruction in the transfer vector. The TXL instruction will transfer to the appropriate section of STRACE. Using the contents of index register 4, STRACE obtains the TXL instruction and checks to see if it is legal (i.e., does the table position indicated by the decrement actually correspond to a subroutine name?). If the TXL is legal, STRACE retrieves the tracing conditions for this subroutine and checks them. Depending on the conditions and the number of executions of the subroutine, STRACE may print the trace message before transferring to the subroutine.

When the HALT or VALUE requests have been specified, STRACE examines the subroutine calling sequence to determine where the subroutine will return. It then saves the instruction at the return point and the instruction immediately following in the trace table and replaces them with a transfer back to STRACE. When STRACE obtains control following the execution of the subroutine it restores the two instructions. If the subroutine does not return correctly the breakpoint will not be removed and the two instructions which were saved will be destroyed the next time the HALT or VALUE condition(s) are satisfied.

The call of the debugging subroutine and the execution stop occur just before the transfer to the traced subroutine. In both cases the user's machine conditions (with the exception of index register 4) are restored.

Restrictions

Only 20 subroutines may be traced at one time. This limit is somewhat arbitrary and may be increased in the future.

STRACE will correctly handle any subroutine that is called by an instruction of the form TSX SUB,4. A subroutine such as (IOH) which is entered by the instruction TRA* (IOH) cannot be traced. A subroutine should not be traced if there is any indirect reference to it through the transfer vector.

ERROR MESSAGES

The following error messages are currently implemented

TRACE TABLE FULL - No more subroutines can be traced until the REMOVE request is used.

NAME IS NOT IN TRACE TABLE - The user has attempted to use the ON, OFF, or REMOVE requests for subroutine NAME which is not in the internal trace table.

NAME IS NOT USED - Subroutine NAME has been loaded but is not called by any program. All requests for this subroutine are ignored.

NAME IS NOT IN MOVIE TABLE - Subroutine NAME has not been loaded. All requests pertaining to this subroutine will be ignored.

NAME IS NOT A REQUEST - STRACE does not recognize the request NAME. This word and the next word of input (most requests have a parameter) will be ignored. If the command line seems to be fouled up, the user can recover by typing a comma to terminate the command and then retype the entire command.

NAME PARAMETER MISSING, REQUEST IGNORED. - The user has typed a sequence such as AFTER, or UNTIL,. The parameter for the request NAME is missing, since the command was terminated by the comma, the user must enter another command. Note that the command

SIN AFTER UNTIL 2,

will result in the comment 2 IS NOT A REQUEST.

BAD CALL OF TRACE FROM LOC - There has been a spurious transfer into STRACE or else location LOC (the word pointed to by the instruction at 0,4) contains a TXL instruction which has an illegal decrement. The decrement of a legal TXL instruction should be less than 201 (for the current limit of 20 entries) and a multiple of 10. The user's machine conditions will be restored, and STRACE will go to dormant.

NAME IS NOT A LEGAL PM - STRACE does not recognize the word NAME as a legal PM parameter, it will be ignored.

NO DEBUGGING SUBROUTINE, CALL IGNORED. - The user has forgotten to supply the name of the debugging subroutine. The CALL condition will be removed.

Disclaimer

This version has been checked and debugged, although no claims are made in this respect. Any comments, suggestions, and/or modifications will be welcomed. In case of trouble, contact Barry Wolman, Room 532 T.S., Ext. 6022.

Revised: 7/30/66

Identification

Manuscript typing and editing
TYPSET, RUNOFF
J. Saltzer, X6039

Purpose

The command TYPSET is used to create and edit 12-bit BCD line-marked files. This command permits editing and revising by context, rather than by line number. The command RUNOFF will print out (in a format subject to control words placed in the file via TYPSET) a 12-bit BCD line-marked file in manuscript format. RUNOFF contains several special control features which were not available with the DITTO command, including type-justification.

References

This work represents one more iteration in the arduous task of creating an "ultimate" editing scheme. As such, it is primarily a synthesis of techniques which have been proven valuable in several separate problem areas. It is felt that this particular synthesis brings to bear on the editing problem an easy to use package of techniques, and might provide a model for an editor on a "next generation" time-sharing system. Here is a list of some of the sources of ideas for these commands:

J. McCarthy	(Colossal typewriter)
S. Piner	(Expensive Typewriter)
P. Samson	(Justify)
Comp. Center staff	(Input, Edit, and File)
M. L. Lowry	(Memo, Modify, and Ditto)
M. P. Barnett	(Photon)
V. H. Yngve	(Comit, Vedit)
R. S. Fabry	(Madbug)
A. L. Samuels	(Edits)
F. J. Corbató	(Revise)

An Edit-by-Context Program

Program Name: TYPSET

Description

TYPSET is a command program used to type in and edit memorandum files of English text. TYPSET, along with the command RUNOFF, is a replacement for the (old system) commands MEMO, MODIFY, and DITTO. Editing is specified by context, rather than line number, and input is accomplished at high speed since the program does not respond between lines.

Usage

TYPSET name

"name" specifies the primary name of a file to be edited, or of a file to be created; it may be absent, in which case a file is to be created, and must be named later by the "FILE" request.

When TYPSET is ready for typing to begin, the word "Input" or "Edit" is typed, and the user may begin. If he is creating a file, he begins in high-speed input mode; if he is editing a file, he begins in edit mode.

High-Speed Input Mode

In high speed input mode, the user may type lines of up to 360 characters in length (e.g., 120 underlined characters) separated by carriage returns. He does not wait for response from the program or the supervisor between lines, but may type as rapidly as desired. The full character set of his keyboard may be used.

The user leaves high-speed input mode and enters edit mode by typing an extra carriage return. When switching modes, the program acknowledges the switch by typing the name of the new mode, "Input" or "Edit".

Edit Mode

In Edit mode, the program recognizes "requests" of the form given below. All requests take effect immediately on a copy of the file being edited. Except where a request is expected to cause a response, such as "PRINT," successive requests may be entered immediately on successive lines without waiting for a response from the program. Each separate request must begin on a separate line. Program responses are typed in red, if you use a two-color ribbon.

Character Set

The standard 12-bit character set is available. (See Section AC.2.01) Note that upper-case comma and period * cannot be input from a 1050, but can be input from a model * 37 teletype. The preset erase character is # and the preset kill character is @ . The ¤ ¢ ± characters should not be used in typset memos.

Requests

Editing is done line by line. We may envision a pointer which at the beginning of editing is above the first line of the file. This pointer is moved down to different lines by some requests, while other requests specify some action to be done to the line next to the pointer. All requests except FILE may be abbreviated by giving only the first letter. Illegal or misspelled requests will be commented upon and ignored.

For purposes of description, the requests have been divided into two categories, those necessary for effective use of the command, and special-purpose requests which are not so generally useful. The first category includes eight requests:

LOCATE character string

This request moves the pointer down to the first line which contains the given character string. Only enough of the line need be specified to identify it uniquely. Since the pointer only moves down through the file the second occurrance of a line containing a given character string may be located by giving the LOCATE request twice. The line which has been found is printed in its entirety.

It is not necessary to count blank characters exactly. If one blank character appears at some point in the request string, any number of blank characters or tabs at the corresponding point in the file will be deemed to satisfy the request. If 2 blank characters appear together in the request string, there must be at least two blank characters or tabs at the corresponding point in the file, etc.

If the LOCATE request fails to find a line containing the given character string, a message is printed, and the pointer is set to point after the last line in the file. Any requests which were typed in between the LOCATE which failed and the message from the program about the failure are ignored. Another LOCATE request will move the pointer back to the top of the file to

begin another scan down through the file.

PRINT n

Starting at the pointer, n lines are printed on the typewriter console. The pointer is left at the last line printed. If n is absent, 1 line is printed and the pointer is not moved. If the pointer is not at a line (e.g., above or below the file, or at a line just deleted) only a carriage return is typed.

NEXT n

This request moves the pointer down "n" lines. If "n" is absent, the pointer is moved to the next line.

DELETE n

This request deletes "n" lines, starting with the line currently being pointed at. The pointer is left at the last deleted line. If "n" is absent, the current line is deleted and the pointer not moved.

INSERT new line

The line "new line" will be inserted after the line by the pointer. The first blank following the request word is part of the request word, and not part of the new line. The pointer is set to the new line. To insert more than one line, give several INSERT requests, or just type a carriage return to switch to high-speed input mode. All lines typed are inserted after the line being pointed at. When the user returns to edit mode by typing an extra return, the pointer is set to the last inserted line. If the very first edit request given is an INSERT, the inserted lines are placed at the beginning of the file. If an INSERT is given after the pointer has run off the bottom of the file, the inserted lines are placed at the end of the file.

CHANGE /string 1/string 2/ n G

In the line being pointed at, the string of characters "string 1" is replaced by the string of characters "string 2". If "string 1" is void, "string 2" will be inserted at the beginning of the line. Any character not appearing within either character string may be used in place of the "slash" character. If a number, "n", is present, the change request will affect "n" lines, starting with the one being pointed at. All lines in which a change was made are printed. The last line scanned is printed whether a change was made or

not. The pointer is left at the last line scanned. If the letter "G" is absent, only the first occurrence of "string 1" within a line will be changed. If "G" is present, all occurrences of "string 1" within a line will be changed. If "string 1" is void, "G" has no effect. Blanks in CHANGE-request strings must be counted exactly.

Example:

```
line:          It is a nice day in Boston.
request:       CHANGE /is/was/
new line:      It was a nice day in Boston.
request:       CHANGE xwasxisx
new line:      It is a nice day in Boston.
request:       CHANGE ' '.' g
new line:      It.is.a.nice.day.in.Boston.
request:       CHANGE '.''
new line:      Itis.a.nice.day.in.Boston.
request:       CHANGE "tis"t is"
request:       CHANGE '.' ' G
request:       CHANGE 'on 'on.'
new line:      It is a nice day in Boston.
```

FILE name

This request is used to terminate the editing process and to write the edited file on the disk. The edited file is filed as "name (MEMO)". If "name" is absent, the original name will be used, and the older file deleted. If no name was originally given, the request is ignored and a comment made. If "name" is given and a file of that name already exists, the user will be asked if he wishes to delete the old file. When this request is finished, the user returns to command level, and the supervisor will respond by typing "R" and the time used.

TOP

This request moves the pointer back to above the first line in a file.

The following seven requests are handy for special purposes, but will probably not be used as often as the ones previously described.

BOTTOM

This request moves the pointer to the end of the file and switches to input mode. All lines which are then typed are placed at the end of the file.

ERASE c

The erase character is changed from its preset "#" to character "c". (The erase character deletes the last character typed, recursively.) Do NOT type "erase e", because then you cannot change back to the "#" character again, nor can you "file" back to the supervisor level. You will have to QUIT and lose your core image.

KILL c

The kill character is changed from its preset "@" to character "c". (The kill character deletes the last line typed, if given before the carriage return.) Do NOT type "kill k", because then you cannot change back to the "@" character. Here, however, you can "file" to escape back to supervisor level.

VERIFY p

If the parameter, "p" is "OFF", the following program responses are not automatically typed:

"INPUT" or "EDIT" when the mode is changed.
Lines found by the FIND or LOCATE requests.
Lines changed by a CHANGE request.

If the parameter "p" is "ON", the responses are restored. The command begins in "ON" mode.

RETYPE new line

The line "new line" replaces the line being pointed at. The first blank following the request word is part of the request word and therefore is not part of the new line.

FIND character string

This request moves the pointer down to the first line which starts with the given character string.

SPLIT name

All the lines above the pointer are split into a file called "name (MEMO)". Any old copy of "name (MEMO)" is deleted. The remainder of the file may still be edited, and filed under another name. The SPLIT request may be used several times during a single edit, if desired. Unless at least one "TOP" request has been given, "name" must be different from the original name of the file being split.

Backspacing

The backspace key may be used to create overstuck or underlined characters. All overstruck characters are stored in a standard format, independent of the way they were typed in. CHANGE-, LOCATE- and FIND-request strings are also converted to this standard format, so it is not necessary to remember the order in which an overstruck character was typed in order to identify it. For example, suppose the line:

The NØRMAL MØDE statement of MAD

had been typed in by typing the letters NORMAL, five backspaces, a slash, and four forward spaces. The slashed Ø in NØRMAL can be changed to a standard O by typing

CHANGE 'Ø'O'

Restricted Names and Recovery Procedures

Two special names are used for intermediate files by TYPSET. They are:

(INPUT FILE) *
(INPT1 FILE) *

Following a QUIT sequence (or a CTSS system breakd$wn) one or both of these files may be found. (Whenever a QUIT sequence has been given, a SAVE command should be issued to save the status of all files.) Because the (INPT1 FILE) * generally contains a complete copy of the file since the * last TOP command, it may be renamed and used as a source * file, and may permit recovery of lost requests. The (INPUT * FILE) contains only that part of the file above the pointer, * and therefore contains only a partial record of the original * file. It should only be used if you also have a LOGOUT * SAVED file which was generated at the same time. The * original file is never deleted until the new, edited file has been successfully written and closed.

The user's disk status and file directory are updated at the end of each pass through the file, thereby providing additional insurance against accidental loss.

The intermediate files are normally written in permanent mode. If the user's track quota becomes exhausted while editing, TYPSET will switch to temporary mode intermediate files. If it is necessary to leave the edited file in temporary mode, a comment will be made.

If a new file name is to be created (including these intermediate files) and the user already has a file of the same name in his directory, he is first asked if he wishes to delete the old file.

Summary of TYPSET requests.

abbrev.	request	response
Basic requests:		
L	LOCATE string	line found *
		end-of-file
D	DELETE n	end-of-file
N	NEXT n	end-of-file
I	INSERT line	none
P	PRINT n	printed lines,
		end-of-file
C	CHANGE QxxQyyQ n G	changed lines *
T	TOP	none
	FILE name	Ready message
Special-purpose requests:		
B	BOTTOM	"Input" *
V	VERIFY ON (or OFF)	none
S	SPLIT name	no name given
R	RETYPE new line	none
E	ERASE x	none
K	KILL x	none
F	FIND string	line found *
		end-of-file

* These responses will not occur if VERIFY mode is off.

A Right-Justifying Type Out Program

Program Name: RUNOFF

Program Description

RUNOFF is a command used to type out memorandum files of English text, in manuscript format. Control words scattered in the text may be used to provide detailed control over the format. Input files may be prepared by the context editor, TYPSET.

Usage

RUNOFF NAME1 -P1- -P2- ... -Pn-

NAME1 is the primary name of a file "NAME1 (MEMO)" to be typed out.

P1,P2, etc., are any number of the following parameters, in any order:

STOP Pause between pages. *

NOWAIT Suppress the initial pause to load paper and the pause between pages.

PAGE n Begin printing with the page numbered "n".

BALL n Typewriter is using printing ball "n". If this parameter is omitted, Runoff assumes that the ball in use will properly print all CTSS characters in the file. The number "n" is engraved on top of the printing ball. CTSS characters not appearing on the ball being used will be printed as blanks, so that they may be drawn in.

Control Words

Input generally consists of English text, 360 or fewer characters to a line. Control words must begin a new line, and they begin with a period so that they may be distinguished from other text. RUNOFF does not print the control words.

.line length n

Set the line length to "n". The line length is preset to 60.

.indent n

Set the number of spaces to be inserted at the beginning of each line to "n". Indent is preset to 0.

.undent n

In an indented region, this control word causes a break and the next line only will be indented n spaces fewer than usual. This control word is useful for typing indented numbered paragraphs.

.paper length n

This control word is used for running off a memorandum file on non-standard paper. The number "n" is a line count, figured at 6 lines per inch. If this control word is not given, "n" is assumed to be 66, for 11-inch paper.

.single space

Copy is to be single spaced. This mode takes effect after the next line. (The normal mode is single space.)

.double space

Copy is to be double spaced. This mode takes effect after the next line.

.begin page

Print out this page, start next line on a new page.

.adjust

Right adjust lines to the right margin by inserting blanks in the line. The next line is the first one affected. (This is the normal mode.)

.nojust

Do not right-adjust lines.

.fill

Lengthen short lines by moving words from the following line; trim long lines by moving words to the following line. (This is the normal mode.) A line beginning with one or more blanks is taken to be a new paragraph, and is not run into the previous line.

.nofill

Print all lines exactly as they appear without right adjustment or filling out.

.page -n-

Print page numbers. (The first page is not given a page number. It has instead a two-inch top margin. See also "Manuscript Conventions", below.) If "n" is present, insert a page break and number the next page "n". Note that RUNOFF does not print completely empty pages.

.space -n-

Insert "n" vertical spaces (carriage returns) in the copy. If "n" carries spacing to the bottom of a page, spacing is stopped. If "n" is absent or 0, one space is inserted.

.header xxxxxxxxxxxxxxx

All of the line after the first blank is used as a header line, and appears at the top of each page, along with the page number, if specified.

.break

The lines before and after the ".break" control word will not be run together by the "fill" mode of operation.

.center

The following line is to be centered between the left and right margins.

.literal

The following line is not a control word, despite the fact that it begins with a period.

.heading mode P

This control sequence alters the mode of the running head to that specified by the parameter "P". Any of the following parameters are allowed:

CENTER The header will be centered on the page.

MARGIN The header will be adjusted against the right margin of the page.

FACING On even-numbered pages, the header will be adjusted against the left margin, on odd numbered pages against the right.

OPPOSED The header will be adjusted against the opposite margin from the page number.

In the absence of a .HEADING MODE control sequence, the default option is OPPOSED.

.odd page

This control word causes the current page to be printed out, and the next page to be numbered with the next higher odd page number.

.paging mode P1 P2 ... Pn

This control sequence alters the mode of page numbering to that specified by the parameter P1, P2, etc. The Pi's may be in any order, and selected from the following list:

MARGIN Page numbers will be adjusted against the right margin.

FACING Odd page numbers are adjusted against the right margin, even page numbers are adjusted against the left margin.

CENTER Page numbers are centered between the right and left margin.

TOP Page numbers are placed on the fourth line from the top of the page.

BOTTOM Page numbers are placed on the fourth line from the bottom of the page.

OFF Page numbers are discontinued.

PREFIX "string" The string of characters between quotation marks is prefixed to the page number. The quotation marks may be next to each other, in which case no prefix is used.

ROMANU Page numbers will be printed in upper case Roman numerals. *

ROMANL Page numbers will be printed in lower case Roman numerals. *

ARABIC Page numbers will be printed in Arabic. (This is the normal mode.) *

SET n Set the next page number to be "n". *

SKIP n Skip "n" page numbers. *

If in a single use of .PAGING MODE several pi's specify competing functions, the last one specified takes precedence. When the .PAGING MODE sequence appears in text at point A, all text up to A (and probably some text after A) will appear on a page controlled by the previous paging mode. The new paging mode will take effect on the next page. Then there is no danger of getting page numbers both at the top and bottom of the same page.

Use of the TOP parameter may conflict with the heading mode. If a heading and a page number should be printed in the same column, the page number will take precedence.

In the absence of a .PAGING MODE control sequence, the default options are: TOP MARGIN PREFIX "PAGE "

.append A

Take as the next input line the first line of A (MEMO). Note that the whole of A is appended, and that the appending is an irreversible process - that is, once RUNOFF encounters the .APPEND control word it will switch to file A (MEMO) and continue from its first line. Other text in the original file (which contained the control word) will not be processed by RUNOFF. The file A (MEMO) may, of course, itself call for appending of still another file, and so on.

All control words may be typed in either upper case or lower case. Illegal control words are ignored by the RUNOFF command. A comment may appear to the right of a control word, as long as it is on the same line.

Abbreviations

All control words may be abbreviated if desired. A list of abbreviations is given in the summary. In most cases, a single word is abbreviated by giving its first two letters; two words are abbreviated by giving the first letter of each word.

Manuscript Conventions

The RUNOFF program assumes a page length of 11 inches, with 6 vertical lines per inch. The top and bottom margins are 1 inch, except for the first page which has a 2-inch top margin. If a header is used, it will be placed 1/2 inch

from the top of the page. The first page is not numbered, nor is it given the header line, unless the control words ".header" and ".page 1" appear before the first line of text.

Customary margins are 1-1/2 inches on the left and 1 inch on the right, implying a 60-character line. This is the standard line length in the absence of margin control words.

Unless restrained from doing so (by STOP or NOWAIT), the program stops only before the first page for loading of paper. The paper should be loaded so that after the first carriage return typing would take place on line 1 of the paper. The left margin stop of the typewriter should be placed at the point typing will begin, and the right margin moved as far right as possible. Now, when you type the first carriage return, the program will start typing and continue to the end of the file.

Tabs

When performing right-adjustment, the RUNOFF command does not take special account of the tabulate characters. Therefore, tabs should not be used unless "fill" mode is off.

If a memo does use tabs in a section where "fill" is off, the mechanical tab stops on the typewriter must be set properly. The following conventions should be used in any memo which uses tabs: The first two lines of the memo should contain two comments, beginning with the words ".SET TABS AT", followedby a string of blanks and x's, with the x's positioned at the desired tab stop positions. The second comment should be ".TABS SET AT" followed by a string of tabs and x's. If the typewriter is correctly set up, the typset request "PRINT 3" will cause the two lines to be printed out with the x's lined up. (Do NOT have more than eight tab characters in a single line, or the program will add a extra carriage return that will appear in the runoff version.)

Backspacing

Underlining or overtyping may be accomplished with the aid of the backspace key, even in a line that is subject to right adjustment.

Summary of RUNOFF Control Words

abbrev.	control word	automatic break
.ap	.append A	no
.ll	.line length n	no *
.pl	.paper length n	no
.in	.indent n	no
.un	.undent n	yes
.ss	.single space	yes
.ds	.double space	yes
.bp	.begin page	yes
.ad	.adjust	yes
.fi	.fill	yes
.nf	.nofill	yes
.pa	.page (n)	yes, if n
.sp	.space (n)	yes
.he	.header xxxx	no
.br	.break	yes
.ce	.center	yes
.li	.literal	np
.hm	.heading mode P	no
.op	.odd page	yes
.pm	.paging mode P	no

If "automatic break" is yes, the lines before and after the control word will never be run together, and the previous line will be printed out in its entirety before the control word takes effect.

(END)

Identification

Documentation of program changes
LOG

Purpose

The LOG command has been designed primarily to provide a convenient way of bookkeeping information among users working on the same programs.

Another related usage is to supply any user with a permanent way of obtaining information about the latest modifications in CTSS.

Format

LOG -CF- -NAME1- -NAME2-

CF is a common file number, (0,1,2,3, or 4)

NAME1 and NAME2 are the names of a log file to which all new information is added, and kept in the specified common file CF. If NAME2 is missing it is assumed BCD. If NAME1 and NAME2 are missing, they are assumed SYSLOG BCD. If CF is also missing it is assumed 2.

Execution

The LOG command begins by printing the contents of the file NAME1 NAME2, if it exists, so that the user is reminded of the latest modifications which he or other users might have made in the same programs. An interrupt will terminate printing. LOG then chains to the INPUT command so that the user may type whatever information he wishes to add to the log file.

This information, along with a problem and user number, system code, date, and time, is recorded at the beginning of the previous log file. Due to this procedure, the latest information is always available at the beginning of the file.

CTSS Modification Bookkeeping

Modifications to the CTSS system are usually recorded in the file SYSLOG BCD kept in the system files. Any user may copy this file by using the following command

COPY S SYSLOG BCD

The file may then be printed.

LOG Structure

The LOG command uses other system commands, viz. COPY, UPDATE, EDIT, FILE, COMBIN, PRINTF. There is no restriction in using LOG in a chain of commands.

Some imperfections in the commands called on by LOG may result in meaningless messages, or in an abnormal exit breaking the chaining set up by LOG. Such flaws will be eliminated gradually by improvements in each particular system command.

Suggestion

Every new item recorded in the log file starts with *b*b*b heading the line containing the user's identification and date. It is suggested that no line typed in by users begin with this particular pattern. This would provide an easy way to process log files automatically for information retrieval, sorting by user, etc.

Identification

Users talk to GOD
REMARK

Purpose

Users may address themselves to "whom it may concern". The users' remarks file is printed off-line each day and the operations staff directs the printed copy to the appropriate members of the systems programming staff.

Usage

REMARK NAME1 NAME2

The 6-bit BCD file NAME1 NAME2 which contains the user's remarks is appended to a PUBLIC file called USER REMARK. This file is printed each day by the operations staff and delivered to the addressees. If NAME2 is omitted, it is assumed to be BCD. If NAME1 and NAME2 are omitted, instructions for using the command are printed.

Identification

INFO - An on-line documentation system
J. Winett - X81-301

Purpose

In order to better provide information about the programs associated with the Compatible Time-Sharing System an on-line system for documenting computer programs has been developed. The design of this on-line system attempts to satisfy the following objectives:

a) Have up-to-date information available to the user on request, thus eliminating the delays which occur in any memo distribution system.
b) Have the ability to get specific information on request, e.g. the author of a routine, as well as the complete documentation of a routine.
c) Give textual output in steps, i.e., printed according to information item types, and indicating the amount of printout that will result.
d) Provide the facility to search through the library of programs to determine the programs which satisfy particular conditions.
e) Standardize the format of the program description by requiring that when a new program is added to the system all information of interest be provided.
f) Permit editorial control of the program documentation that is to be available on-line.

The information describing a program is divided into information items. Each item of information is associated with an item name and is referred to as the item value of the associated item name. For example, the item value "WINETT" is associated with the item name "AUTHOR". The following items of information indicate what is required as documentation of a computer program:

1. Program NAME (N) - A single word.

2. Program TYPE (T) - One of the following: COMMAND, ENTRY, LIBRARY, or PUBLIC.

3. DESCRIPTORS (D) - Key words used to classify the programs in the information files.

4. PURPOSE (P) - A short abstract or sentence description indicating the context in which a program might be used.

5. USAGE (U) - The instructions of how to use the program, e.g., the calling sequence.

6. Programming LANGUAGE (L) - The language in which the program is written.

7. REFERENCE (R) - A bibliography of the program.

8. AUTHOR (A) - The name of the person who is responsible for the program.

9. DATE (DA) - The date the information was last entered or modified.

Additional information items may also be defined, e.g. program size, transfer vector, etc., but the above items are considered to be required to document any program.

Usage

A model of this information system has been implemented as a CTSS command program with command name INFO. The system may be initiated as a console command or chained to from another program. If, when the INFO system is called the NAME of a program is given as a command parameter, the documentation on that program will be printed after which the system will call CHNCOM. This procedure allows other command programs to access their own documentation. For example, when no parameters are supplied with a command which requires at least one parameter, the command should chain to the INFO command with the command name as a parameter. This technique provides a means of tying the documentation of a command program to the command itself.

If only the command name INFO is typed the system will respond

TYPE REQUEST, OR C.R. FOR INSTRUCTIONS..

whereupon a carriage return will initiate the request to describe the INFO command.

Alternatively, requests can be typed to the INFO system. There are three classes of requests: a) Retrieval requests to obtain information from the system, b) Storage requests for adding, changing, or deleting information from the system, and c) System requests which affect the operation of the system. The Retrieval requests - DESCRIBE (D), LIST (L), and FIND (F) are to be used by all CTSS users. The Storage requests - STORE (S), EDIT (E), ALTER (A), and REMOVE (R) - are to be used by the people responsible for the information stored within the system. The System request - QUIT - is used to terminate communications with the system, and the requests - END, TSSFIL, and USRFIL - are

used for changing the operation of the system. Whenever the INFO system prints a comment followed by two periods, it is the user's turn to type. After processing each request, the system types

OK..

To obtain a description of a Storage or Retrieval request the user types the request name only. A request to the INFO system indicates three types of information: 1) an imperative request to the system, 2) a list of single information words, and 3) information items specified by item names together with the item values associated with the item names. A request to the system is assumed to be indicated by one of the first few words typed. Other words following the request name may be item names and are added to a list of "information words" or may specify the values of information items and are added to a list of pairs consisting of an item name and its value. When the word "IS" or "ARE" is encountered in a request it is assumed that the previous word is an item name and that the following words form the item value. The input specifying the item value must be terminated by a comma (or the carriage return at the end of the request) and may be followed by other item names and their values or by item names alone. If the word "THEN" appears as an information word the input scanned so far is assumed to constitute a request. After the request is processed, the input following the word "THEN" is scanned for the specification of another request. Thus the word "THEN" indicates the termination of a request and allows multiple requests to be typed. Words other than item names or item values or the word "THEN" may be typed but are ignored.

Requests and item names may be abbreviated by their first letter (except the item name DATE which is abbreviated DA). If an item value is specified more than once in an input request the value last specified takes precedence. Thus, the on-line user may change or correct the specification of an item value by retyping the item name together with the item value in the same input request.

To continue the typing of a request on another line precede the carriage return (C.R.) by a dash (-). When in doubt of what to do, type a carriage return.

RETRIEVAL REQUESTS

1. The DESCRIBE (D) request:

DESCRIBE NAME IS name, i(1)...i(n)

This request is used to obtain the documentation of a program whose name is known. The input with this request gives the program name and the names of the desired items of information. If no item names are specified, the information on all items will be printed. For example,

DESCRIBE THE COMMAND WHOSE NAME IS INFO

produces all the documentation associated with the INFO command, and

D N IS INFO, USAGE

prints just the item USAGE for the INFO program.

When more than five lines of text are to be printed, the INFO system informs the user of the number of lines which follow. After realizing how much information will be printed, the on-line user may terminate the request by pressing the CTSS interrupt or quit button.

If the interrupt button is pressed the user may type "CONTINUE (C)" to resume printing or "RESTART (R)" to type another request. Printing will be resumed approximately ten lines after the line at which printing was interrupted. (This is due to the fact that the CTSS output buffers are cleared on interrupt.) Since a number of lines are lost on interrupt, the process of interrupting and continuing provides a means of skipping lines of documentation. Unfortunately, this procedure gets very poor response from CTSS.

If the quit button is pressed, the on-line user may type another command or type the CTSS command "START" to continue as described above. This procedure gets very much better response from CTSS.

2. The LIST (L) request:

LIST i(1)...i(n), TYPE IS type

This request is used to obtain a list of all available item names, a list of the available values of certain information items, or a list of the names of all CTSS programs of a particular type. The request may ask for the values of one or more of the following items to be listed:

ITEMS, AUTHORS, DESCRIPTORS,
LANGUAGES, TYPES, or NAMES

or may also request a list of all CTSS programs of a particular type by typing either or both of the types

COMMAND or ENTRY

after the words: 'TYPE IS' . The list of programs of a particular type is obtained directly from CTSS and thus is automatically provided the most up-to-date list of programs available. A request to

LIST NAMES

causes a list of commands and entries to be printed. A list of descriptors may be obtained by typing

LIST THE DESCRIPTORS

or just L D .

3. The FIND (F) request:

FIND i(1) IS v(1), . . . , i(n) IS v(n)

This request is used to perform a search for the program or programs which have particular information item values. The items to be matched are given by typing the item names together with their item values. Acceptable items for searching on are:

TYPE, DESCRIPTORS, AUTHOR, DATE, and LANGUAGE.

A date value must be given in the form "DATE IS mm/dd," where mm is a numerical month and dd is a numerical day. All programs whose date is greater than that given will be printed. Note that a year is not specified and hence January is less than December (This bug will soon be eliminated). Descriptors are single words typed in any order and separated by spaces or the word "AND".

For example, to find the commands which were documented since January 1 and have at least the descriptors UTILITY AND EDITING, type -

```
FIND TYPE IS COMMAND, DATE IS 1/01, DESCRIPTORS -
ARE UTILITY AND EDITING
```

or

```
F T IS C, DA IS 1/01, D ARE UTILITY EDITING .
```

(note the use of the dash to continue the input request on the next line.)

When a search results in more than twenty matching items the system asks whether the user wants to continue the search. The user may then type "YES" or "NO". For each twenty more matching items the user is given the option of continuing.

STORAGE REQUESTS

4. The STORE (S) request:

```
STORE NAME IS name, FILE IS file, i(1) IS v(1), -
                  . . . , i(n) IS v(n)
```

This request enables one to enter information about a new program into an information file. This request requires that information values be provided for each required item, as previously listed, in the form:

```
item name IS/ARE item value .
```

The NAME of new information items may be defined by typing the new item name and its value. When the INFO system prints an item name followed by two periods, the user is to type the value of that item. Item names and item values of other items may be supplied following the item value which was requested, by typing a comma after each item value and thus anticipating the required input and reducing on-line interaction.

If the word FILE is specified in the input specification, a file with primary name the same as the program name (if specified) and secondary name INFO is read. This file is assumed to contain item values for this program where each item value is preceded by a line giving the item name prefixed by a period and beginning in column one. If the primary name of this input file is not the same as the program name, the file name may be specified by typing the item

```
FILE IS file name .
```

If a file name is specified and a program NAME is not specified, the NAME of the program may be read from the input file. A program NAME is indicated in an input file by the presence of two periods before the program NAME. An input file may specify the documentation of many programs by preceding the documentation of each program with a line giving the program NAME prefixed by the two periods (for example, ..INFO). The priming of the command documentation was done from an input file (with name COMAND INFO) of this type by typing

```
STORE FILE IS COMAND .
```

5. The EDIT (E) request:

EDIT NAME IS name

This request re-creates a BCD file (as a line marked file) from the information in the system for use in making changes to information items using some CTSS editing procedure. The EDIT request requires that the program NAME be specified. The file created contains all information items except those items which can be used with the find request. Each item value is preceded by a line giving the item name prefixed by a period (e.g. - .PURPOSE) and consequently no line of an item value should begin with a period. The primary name of the file created is the same as the program name and the secondary name is INFO.

6. The ALTER (R) request:

ALTER NAME IS name, i(1) IS v(1), . . . , i(n) IS v(n)

This request allows one to change item values in the information documenting a program or to store additional information items. The ALTER request requires that the program name be specified and is used like the STORE request. The ALTER request is different from the STORE request in that it does not require that values for all information items be specified. That is, the user-system interaction is different.

7. The REMOVE (R) request:

REMOVE NAME IS name, D IS d, A IS a, ITEM IS i

This request is used to delete an AUTHOR, DESCRIPTOR, or ITEM name from the appropriate list, or to delete the documentation of a program from an information file when a program is deleted from CTSS. To REMOVE the documentation of a program give the program NAME. To REMOVE an AUTHOR from the list of AUTHORs or a DESCRIPTOR from the list of DESCRIPTORs, specify the item value to be removed. Verification of each request to remove the documentation of a program is required.

SYSTEM REQUESTS

8. The QUIT (QU) request:

This request causes the INFO system to call CHNCOM and may be used to terminate the INFO command or to chain to other commands.

9. The END (no abbreviation) request:

This request causes the INFO system to terminate through the standard COMIT termination sequence. (The INFO command has been written in the COMIT language.) The amount of unused free storage, i.e., the number of WORKSPACE registers, is printed.

10. The TSSFIL (no abbreviation) request:

This request causes the INFO files to be obtained from one of the CTSS system file directories and is required before the INFO system is included as a CTSS command.

11. The USRFIL (no abbreviation) request:

This request causes the INFO files to be obtained from the user's file directory rather than the system file directory. This request may be used by a user to indicate that the documentation files are to be obtained from the user's file directory. In this way a user may keep documentation on his private programs.

Summary of INFO requests

RETRIEVAL REQUESTS

1. DESCRIBE NAME IS name, i(1)...i(n)
2. LIST i(1)...i(n), TYPE IS type
3. FIND i(1) IS v(1), . . . , i(n) IS v(n)

STORAGE REQUESTS

4. STORE NAME IS name, FILE IS file, i(1) IS v(1), -
. . . , i(n) IS v(n)
5. EDIT NAME IS name
6. ALTER NAME IS name, i(1) IS v(1), . . . , i(n) IS v(n)
7. REMOVE NAME IS name, D IS d, A is a, ITEM IS i

SYSTEM REQUESTS

8. END
9. QUIT
10. TSSFIL
11. USRFIL

Required information items are:

NAME (N), TYPE (T), DESCRIPTORS (D), PURPOSE (R),
USAGE (U), LANGUAGE (L), REFERENCE (R),
AUTHOR (A), and DATE (DA).

Note: i stands for item name and v stands for item value.

Revised: 2/14/66

Identification

Mail command
MAIL

Purpose

To place a file containing a message to another user in his file directory, whether he is logged in or not.

Usage

```
MAIL  NAME1  NAME2 PROB1 PROG1 ... -PROBn- -PROGn-
MAIL  NAME1  NAME2  '(LIST)' LNAME1  LNAME2          *
```

NAME1 NAME2 Is the name of the file to be mailed. It must be line-marked, and no more than 1 record in length.

PROBi PROGi are the users to which mail will be sent.

(LIST) If the '(LIST)' option is given, the file * LNAME1 LNAME2 will be used as a "mailing list", and mail will be sent to all PROBi PROGi pairs in the file. The file may be card-image or line-marked; its format is free, except that items must be separated by spaces.

Mail will be placed in a file named MAIL BOX in the records * of user PROBi PROGi. If the file already exists, it will be appended to. Each piece of mail is prefaced with a message of the form "FROM USRPB USRPG DATE TIME" where USRPB is the sender's problem number, USRPG is the sender's programmer number, and DATE and TIME have the usual meanings. (To ascertain whether he has received mail, the user should periodically - daily, perhaps - issue the command 'PRINT MAIL BOX'. Because of the appending feature of the MAILing process, the command 'DELETE MAIL BOX' should be issued after a message has been PRINTed, to avoid having to run through previous mesages to get to the latest one.)

Any PROBi or PROGi may be '*', meaning "all";the command * will search the MFD and send mail to all users (but not to common files) satisfying the criterion. However, '* *' will not cause mail to be sent to all users.

Typing the command 'MAIL' without arguments is equivalent to * asking for instructions on how the command is to be used.

To avoid getting mail, one may place in his tracks a file of name MAIL BOX with PRIVATE mode.

If an addressee is over his record quota, 'MAIL BOX' will be written in temporary mode. *

Restriction

If the receiver's MAIL BOX is PRIVATE, PROTECTED or READ-ONLY, mail cannot be delivered.

(END)

Identification

Macro Command
RUNCOM, CHAIN

Purpose

Public and private commands may be linked or chained together in order that the chain may be executed by merely issuing one command. This is convenient if the same series of commands is to be executed more than once and the user does not wish to retype the series each time. Arguments to the commands may be specified at execution time.

Reference

Section AG.8 gives further information about macro command programs.

Usage

Command Chain:

The command chain, or macro-command, must first be prepared as a BCD line-marked or line-numbered file, with one command per line. Blank lines are ignored. Command arguments are separated by one or more spaces; if an argument is more than six characters long, it will be truncated from the left. Arguments may be command names, actual argument values or dummy symbols. If dummy symbols are used, there must be a list of the dummy symbols specified by the pseudo-command CHAIN somewhere before the first executable command.

```
Example of a macro-command:
CHAIN   ALPH  BET  TRANSL
ED      ALPH  TRANSL
PRINTF  ALPH  TRANSL
TRANSL  ALPH
LOAD    ALPH  BET  (LIBE)  OWNLIB
etc.
```

Comments may be included in the command chain as lines which have as the first character an '*' or a '$'. Comments introduced by '*' will be ignored during execution. Comments introduced by '$' will be printed on the user's console at the point of execution corresponding to their position in the chain.

Execution of Command Chain:

RUNCOM NAME1 ARG1 ARG2 ARGn

NAME1 is the primary name of the BCD command chain file NAME1 BCD.

ARGi are the arguments to be substituted for the dummy symbols (if any) in the same order as specified in the pseudo-command CHAIN. If any ARGi is '(NIL)', the corresponding dummy argument will be ignored; if it is substituted for a command name, the whole command is ignored. If any ARGi is '(END)', it will be replaced by a fence (all 7's). Any additional arguments will be ignored by commands in which this substitution is performed. If (END) is substituted as a command name, the chain is terminated at this point. If there are fewer ARGi than dummy symbols in the CHAIN specification, the rightmost dummies will retain their literal values.

RUNCOM will interpret the file NAME1 BCD, substitute the explicit arguments for dummy arguments, if any, and perform the execution of the specified commands by appropriate use of the supervisor command chain buffers and subroutines. RUNCOM contains a list of public commands indicating whether or not each command assumes a current core image; RUNCOM can then properly intersperse the SAVE and RESUME commands. Nesting and recursion are possible.

Core image management:

Some more details may be necessary to understand the mechanism whereby RUNCOM takes care of core images between commands.

As a general rule, a core image is kept over two consecutive commands if, and only if, the first one is supposed to leave a core image, and the second one is supposed to expect a core image.

e. g. LOAD - SAVE - FAPDBG
Use the same core image created by the LOAD command.

Whereas LOAD - SAVE - LISTF does not keep the core image from SAVE to LISTF. Commands which are supposed to leave a core image are:

```
CTEST1 to CTEST9
LOAD VLOAD NCLOAD LOADGO LDABS USE START
PM TRA STOPAT PATCH
FAPDBG STRACE L
SAVE RESUME R RESTOR
MYSAVE RECALL CONTIN RSTART
RUNCOM
```

Commands which are supposed to expect a core image are:

```
PM TRA STOPAT PATCH
USE START
SAVE MYSAVE
FAPDBG STRACE
```

(NIL) arguments as command names, and $ headed lines do not alter the saving of a core image.

As one may notice, RUNCOM itself may yield a core image, if the last command in the chain does. e.g.,

```
LOAD ALPHA BETA
SAVE ZETA
LISTF ZETA SAVED
RESTOR ZETA
```

may be used as a macro-command, and followed by a START command.

Common file switching:

The only commands which are allowed to begin and terminate in different file directories are:

COMFIL COPY UPDATE REMARK LOG

Indeed, COMFIL switches to whatever directory is specified, and the others switch to the user's file directory when completed.

Any other command must be initiated and terminated in the same file directory. On the other hand, there is no restriction on the various switching which may be performed during the execution of the commands, as long as the initial setting is restored before the end.

One should notice that the present implementation of the common files results in losing the temporary files created in the common files, as soon as one switches to some other directory. However, temporary files always stay in the user's file directory, regardless of the common files switching.

RUNCOM may be initiated in any common file, but the RUNCOM command will switch back to its initial file directory whenever it needs to load a new set of commands for execution. Consequently this may result in losing temporary files, if the execution was in a different common file.

It should be noted that a $ headed line produces a major break in the RUNCOM command. The following commands in the chain will then be loaded together in the supervisor's buffers, up to a maximum of 3 at a time. This peculiarity may be used to deal with temporary files in the common file directories.

```
COMFIL  3
LISTF  ALPHA  BETA
PRINTF  ALPHA  BETA
$  EDIT
EDIT  ALPHA  BETA
FILE  ALPHA  BETA
BETA  ALPHA
 ....
```

Will insure that the (EDIT FILE) is not lost between EDIT and FILE, because the previous line headed by a $ has automatically forced the beginning of a new sub-chain in the supervisor's buffers.

Some examples of macro-commands:

We shall assume here that the name of the BCD file containing the chain is MACRO BCD.

1.
```
CHAIN  FILE (NIL) (END)
ED  FILE  MAD
MAD  FILE  (NIL)
(END)  FILE  ...  (LIBE)  ...
```

   may be called in the following ways:

```
RUNCOM  MACRO  FILE
Whence:  ED  FILE  MAD
         MAD  FILE

RUNCOM  MACRO  FILE  (LIST)
Whence:  ED  FILE  MAD
         MAD  FILE  (LIST)

RUNCOM MACRO FILE (SYMB)  VLOAD
Whence:  ED  FILE  MAD
         MAD  FILE  (SYMTB)
         VLOAD  FILE  ...  (LIBE)  ...
```

2.
```
CHAIN FILE BCD FIL1 N1 N2
```

```
*  THIS MACRO INSERTS THE FILE FILE BCD
*  INTO THE FILE FIL1 BCD, AFTER LINE NUMBER N1
*  AND DELETES THE INITIAL PART OF FIL1 BCD
*  UNTIL AFTER N2.
SPLIT  FIL1  BCD  (A)  N1  *  N2  (B)
CHMODE  (A)  BCD  T  (B)  BCD  T
COMBIN  *  FIL1  BCD  (A)  FILE  (B)
```

May be called by:

RUNCOM MACRO ALPA FAP BETA 1030 1040
inserts ALPA FAP after line 1030, and deletes until after 1040

RUNCOM MACRO ALFA FAP BETA 1030 1030
same thing, but does not delete anything from BETA FAP

RUNCOM MACRO * FAP BETA 1030 1050
deletes in BETA FAP lines after 1030 until after 1050

3.
```
*THIS CHAIN ALLOWS STACKING COMMANDS TYPED ON THE
*CONSOLE, AND THEN STARTS THE EXECUTION
SPLIT MACRO BCD MACRO N
* N IS THE NUMBER OF THE LINE CONTAINING 'EXECUTION'
EDIT MACRO BCD
FILE MACRO BCD
$ EXECUTION
```

Revised: 8/30/65

Identification

Supply arguments in octal to any command
GENCOM

Purpose

If for some reason, the desired arguments for any command cannot be expressed in BCD, the command may be used with the arguments expressed as pairs of six-digit octal arguments.

Usage

GENCOM COMAND ARGU1 ARGU2 ... ARGUn *

COMAND is the BCD name of the desired command.

ARGUi are either the actual BCD arguments of COMAND * or pairs of arguments, OCTLHi OCTRHi (left and right half, respectively), which specify the octal equivalent of the desired argument. Leading zeros in the octal arguments may be omitted. Any argument which is pure numeric of digits 0 to 7 must be expressed as OCTLH OCTRH. If an OCTLH is not followed by its OCTRH, an error comment is printed.

GENCOM will combine the pairs of six-digit octal arguments, OCTLHi OCTRHi, into single twelve digit octal arguments, ARGi,and will initiate the command.

COMAND ARG1 ARG2...ARGn

Identification

Print I/O error diagnostics
PRNTER
3/30/66

Purpose

The PRNTER command calls the PRNTER subroutine (AG.4.06) to format and the print diagnostic information available from the IODIAG subroutine (also in AG.4.06).

Usage

PRNTER

prints one line of the user's console of the form:
-'I/O'- 'ERROR' n: diagnostic '--' subr 'AT' userloc '(F.S.'fsloc')'.

n = numeric value of file-system error code

diagnostic = BCD interpretation of 'n'

subr = entry in file-system in which the error was discovered.

userloc = location in user's program or command of call to 'subr'.

fsloc = location within file-system (F.S.) where the error was discovered. (This is generally of little interest to user).

Normal exit via CHNCOM.

Alternate Usage

For user programs and for command chains which contain individual commands which cannot continue execution when file-system errors are encountered, the PRNTER command may be called upon via the following alternate usage:

PRNTER MASK

MASK = binary argument used to control the printout of diagnostic information. Bits 28-35 correspond exactly to the bit positions used in specifying "MASK" to the PRNTER subroutine. In addition, the sign-bit (bit 0) controls command chaining: if the sign is negative (i.e. 1) control

passes immediately to CHNCOM; if the sign bit is positive (0) and the user was within a chain of commands, the comment

TYPE 'START' TO CONTINUE CHAIN

will be printed on the user's console, followed by a call to DORMNT, to allow the user to take any necessary corrective action. Typing START allows the chain to proceed via CHNCOM.

If 'MASK' equals 0 (+ or -), the PRNTER subroutine's default mask (375(8)) will be used.

The following examples are equivalent ways of setting up this usage from within a command or user program:

MAD: EXECUTE NCOM.($PRNTER$, MASK)

MASK is either a variable or a constant denoting the desired binary argument.

FAP:

```
CAL  =HPRNTER (name of command)
LDQ  MASK (argument of command)
TSX  NEXCOM,4 (optional TIA  =HNEXCOM)
```

MASK is either a literal reference (=Ø'n') or the address of a variable containing the desired binary value.

From the console, the appropriate binary configuaration may be generated via GENCOM or via judicious choice of a BCD argument.

(END)

Identification

Public File Subroutines

This section of the manual contains the documentation of user-submitted subroutines in the Public File. These routines must, of course, be loaded along with the programs which call them. The general procedure for this is:

```
LINK  NAME1  BSS  M1416  CMFL04
```

(This need only be done once, of course.) Then, for example,

```
LOADGO  PROG  NAME1
```

where PROG is the first name of a BSS file containing a program which calls a subroutine (or subroutines) contained in file NAME1 BSS.

The nature of the Public File and the procedure for entering programs in it are discussed in Section AD.4.

(END)

Identification

Simplified I/O package for MAD programs
MADIO
Peter J. Denning, T.S. 829, X6028
5/66

Purpose

MADIO is designed for use in the CTSS environment as a compact input-output subroutine package. Its reading facility features format free reading within one simple call. Its print facility incorporates the most commonly used MAD-type format specifications, a simplification of Hollerith field specifications, and the facility to print without carriage returns. A program may read from the console by means of a single call to READ, and print on the console with a call to PRINT. These calls are intended to replace the use of the READ FORMAT and PRINT FORMAT statements. MADIO is about (2400)8 locations long, half the size of the CTSS package, (IOH). Unlike (IOH), MADIO does not use program common; thus, it can be used in conjunction with the NCLOAD command, which can lead to very compact 'SAVED' files.

MADIO can be obtained by linking to 'MADIO BSS' in M1415 CMFL04. 'READ' or 'PRINT' may be extracted from 'MADIO BSS' by means of the 'EXTBSS' command.

Usage

FORMAT FREE READING

Program Name: READ.

Transfer Vector: RDFLX, WRFLX, WRFLXA.

Use: The call is:

READ.(A,B,...,L)

where A,B,...,L is the list of names into which values are to be read. Any or all of them may be in MAD block notation, i.e., A(J)...A(K), provided K is greater than J, and multiply-subscripted arrays are permissible. There is no restriction on the length of the list A,B,...,L.

The call to READ puts the user into input wait status under control of the READ program. The READ program counts the number of locations specified by the list A,B,...,L, including arrays. If there is any discrepancy between this count and the number of locations required for the items typed on the console, an error condition results (see

below).

The user types a line of the form

ITEM1 ITEM2 ... ITEMn

Each item is a data field, and one or more spaces separate each item.

(1) If "ITEMi" is a string of digits containing no decimal point, it is interpreted to be an integer. It may be preceded by a '-' or an (optional) '+' sign.

(2) If "ITEMi" is a string of digits including a decimal point the number is interpreted as floating point.

(3) If "ITEMi" is a string of digits 0-7 followed by a 'K', the string is interpreted as octal.

(4) If "ITEMi" is a string containing BCD characters other than the digits 0-9, '+', '-', 'K', '.' or ',' it is interpreted as a hollerith string. A hollerith string is entered six characters to the word, the final word left adjusted with trailing blanks.

(5) If 'ITEMi' is to be a hollerith string containing spaces then it is enclosed in dollar signs, '$'. If the final '$' is missing, the end of the line, which is assummed to be after the 14th word, is taken to be the end of the string. Thus an entire line can be read into a 14-word buffer by starting the line with '$'. The '$' is ignored so that in actuality 83 characters are read in, with a blank inserted as the 84th character.

(6) Number items and hollerith items may be mixed in any way on the line.

(7) If all the names A,B,...,L would require more than one line of typing (i.e., more than 84 characters are needed) as many items as desired may be entered on a line and remaining items entered in succeeding lines. The program gives the following comment:

*****TYPE k MORE ITEMS.

where k is the number of remaining locations in the list A,B,...,L to be filled. Hence the n arguments of the list could be entered on n console lines if desired.

Restrictions:

(1) No more than 36 items or 84 characters to a line, whichever comes first.

(2) No more than 12 digits to a number. Integers may not exceed in magnitude 2.P.36. If X.Y is a floating point number then the integer XY must not exceed 2.P.27. This latter restriction can be lifted if demand dictates.

Error Conditions:

(1) If the number of items typed is greater than the number of arguments in the list A,B,...,L then the following comment is printed:

*****YOU HAVE k EXTRA ITEMS, DO YOU WANT TO IGNORE THEM,

If 'yes' is typed the extra arguments are ignored, otherwise the program requests that the present line be retyped.

(2) If more than 12 digits in a number, READ requests retyping the line. If a comma appears in a string of digits, it is assumed to be a mis-typed decimal point and retyping is requested.

(3) Other miscellaneous errors are caught, and the following comment is printed:

*****ERROR AT ITEM NO. k. RETYPE LINE.

Special Calls to READ:

(1) The call

```
READ.($.TEXT.$,A,B,...,L)
```

or,

```
READ.(T,A,B,...,L)
VECTOR VALUES T = $.TEXT.$
```

Causes READ to read only in a BCD mode into the list A,B,...,L. This would be particularly useful for reading into a buffer:

```
READ.($.TEXT.$,BUFF(1)...BUFF(N))
```

If N is greater than 14, the remaining words may be entered on succeeding lines, as described above. When READ is called, and the first location in the calling sequence, T, contains the string ".TEXT.", READ enters BCD mode and ignores T. Hence the first item typed is entered into A, the second into B, etc.

Caution: Be careful of a situation like

READ.(A,B,...,L)

if you should enter the word ".TEXT." into A, the next call of this form may still have ".TEXT." in A; then the items typed will be treated as BCD and entered into B,...,L instead of A,B,...,L as intended.

(2) The message

*****TYPE k MORE ITEMS.

may occur frequently and may be annoying. The call

READ.($.OFF.$)

will cause the program to enter a mode in which this message is suppressed. The call

READ.($.ON.$)

will reset to normal mode. When in the OFF-mode, the first N items typed are entered normally, but the remaining k locations are filled in with 0's, and READ returns to the calling program after having read just one line.

NOTE: The READ FORMAT statement, if used, will cause incorporation of the standard CTSS input-output package at loading time, perhaps defeating the usefulness of the READ program.

CONSOLE PRINTING

Program Name: PRINT.

Transfer Vector: WRFLX, WRFLXA, RDFLX, EXIT.

Use: The call is:

PRINT.(FMT,A,B,...,L)

where FMT is a format statement, and A,B,...,L is the list of names to be printed according to FMT. No restriction is placed on the length of the list. Any of the names in the list may be in block notation, i.e., A(J)...A(K), provided K greater than J. Multiply-subscripted arrays are permitted.

FMT is a standard MAD format statement enclosed in dollar signs of the form:

VECTOR VALUES FMT=$ ... *$

or,

VECTOR VALUES FMT=$ ... N*$ (See below.)

If FMT specifies format for k locations and the name list specifies altogether m locations, with k≠m, then the minimum of k and m locations are actually printed. Each field specification is separated by a space or comma as desired.

Special Features

(1) It is no longer necessary to use H-formats. Hollerith strings are simply enclosed in parentheses, and no letter 'H' is used. However, since ')' is used to terminate a string, the convention '=)' is used to insert ')' into the output string, the '=' sign being ignored. To insert '=)' into the string, use '==)'.

Example: The format statement

VECTOR VALUES FMT=$(HOLLERITH STRING.)*$

results in

HOLLERITH STRING.

being printed.

Also,

VECTOR VALUES FMT=$(ARRAY(),I3,(=))*$

results in

ARRAY(k)

being printed, where k is the value of an integer variable named in the list.

(2) It is possible to print without a carriage return. The format statement is terminated with 'N*' instead of '*' (N = no return). This can be particularly useful for entering data into programs by means of single-line questions and answers.

Example: The format statement

VECTOR VALUES FMT=$(DO YOU WANT MORE,)N*)$

results in

DO YOU WANT MORE,

being printed without a carriage return.

(3) If an illegal format or some other error condition arises, the PRINT program gives an error description. Then it will allow the user to return to the calling program if he desires. With this feature, execution of a program need not be halted by a format error, as with other CTSS programs.

Restrictions:

(1) No E-formats are allowed.

(2) Field size must be less than or equal to 15.

(3) Integers must be less than 2.P.36 in magnitude.

(4) If X.Y is a floating point number then the integer XY must be less than 2.P.36.

(5) All C-formats are interpreted as C6.

(6) Only one level of nesting is allowed. Thus 5(6F8.2,S2,I5/) is allowed in the format statement, but not 5(6(F8.2),S2,I5/) or 3(F8.2,2(I5,S1,C6)).

(7) Of course use of the PRINT FORMAT statement defeats the use of PRINT.

Error Conditions:

(1) Illegal format results in the following:

```
*****TROUBLE AT FORMAT WORD 'word'.
*****PRESENT LINE IS..
*****  (output line up to error)
*****DO YOU WANT TO RETURN TO CALLING PROGRAM,
```

If 'yes' is typed, control is returned to calling program. Otherwise 'EXIT' is called. Note that with this feature, execution of a program is not halted by illegal format, as with regular CTSS library programs.

(2) Number exceeds specified field width. Signs and decimal points are included in the field width. The comment:

```
*****NUMBER TOO BIG FOR FIELD.
*****TROUBLE AT FORMAT WORD 'word'.
*****PRESENT LINE IS..
*****   (output line up to error)
```

```
*****DO YOU WANT TO RETURN TO CALLING PROGRAM,
```

(3) If the field width exceeds 15, the following is printed:

```
*****FIELD WIDTH EXCEEDS 15.
*****TROUBLE AT FORMAT WORD 'word'.
*****PRESENT LINE IS..
*****   (output line up to error)
*****DO YOU WANT TO RETURN TO CALLING PROGRAM,
```

(END

Revised: 3/30/66

Identification

Public File "Commands"

This section of the manual contains the documentation of those user-submitted programs in the Public File which are analogous to system commands. Each exists in SAVED file form, and is actuated by the RESUME command. The general procedure for using Public "commands" is to

LINK NAME1 SAVED M1416 CMFL04 (This need only be done once, of course.)

Then

RESUME NAME1 (with arguments as necessary)

The nature of the Public File and the procedure for entering programs in it are discussed in Section AD.4.

(END)

Revised: 1/3/66

Identification

Interface between user and CTSS.
. SAVED
N.I. Morris, T.S. 525, x6029

Purpose

. SAVED (read "dot - or point - saved") serves as an interface between the user and CTSS, allowing the user to decrease his typing load by giving him wide latitude in the abbreviation of command and parameter names. It also allows the chaining of commands, and offers the ability to communicate between consoles. Other convenient features available through "." are the ability to concatenate commands with the same arguments, automatic resumption of SAVED files (the R command need not be given), automatic file system CALLs, optional suppression of printing of W(ait) and R(eady) lines, and the ability to initiate execution of a SLEEPING program. *

Usage

"." may be employed through a link to the Public File, or it may be copied into the user's own files. Since "." modifies itself on many of its options, it is advantageous to maintain a personal version. The public version must be used with its options preset. Hence, only a private version which can be modified offers the full flexibility of the sub-system. *

1. Initiation: "RESUME ." Response is the character "R".

2. Chaining:

   a. A normal command line may be typed, with spaces delimiting the parameters as usual. However, additional commands may be typed on a given line (separated by commas) with a maximum of five commands comprising a single chain. These commands will be executed sequentially with "." attending to necessary linkages and resuming itself at the end of the chain. Possible form:

   FAP ALPHA (LIST) , LOAD ALPHA GAMMA , SAVE DELTA , STAR

   Note that the commas are parameters and therefore must be set off by blanks.

   b. There is no restriction on the number of characters used in a

given command line. More than one console line may be employed by making the last parameter on a line be '(ETC)' followed by the continuation on the next console line. The only restriction on command line length (not to be confused with chain length as measured in links) is that the total number of parameters (including commas, parentheses, and slashes) must be less than 50.

c. To prevent the resuming of "." at * the end of a chain, terminate the line with a slash ('/'). Note that this tactic allows the chaining of six commands. Commands which leave a desired core-image (e.g. RESTOR, LOAD, etc.) and are the last link of a chain should be followed by the slash to prevent destruction of the core-image by the 'R'.

d. Commands and parameters may also be * concatenated by the use of parentheses. E.g., the command line

```
( BEFAP NCLOAD SAVE ) PROG
```

will generate

```
BEFAP  PROG
NCLOAD PROG
SAVE   PROG
```

The command line

```
FAP ( ALPHA BETA GAMMA ) (LIST)
```

will generate

```
FAP ALPHA (LIST)
FAP BETA (LIST)
FAP GAMMA (LIST)
```

5. Abbreviations: a. Often-used commands and parameters * may be abbreviated in ". SAVED". These abbreviations may be determined through the use of the internal commands

```
ABBREV COM
ABBREV PAR
```

b. To define abbreviations (in a private version), the internal commands are 'DC' (for commands) and DP (for parameters). General form is

DC ab1 com1 ab2 com2 ... abn comn
DP ab1 par1 ab2 par2 ... abn parn
E.g. DC LO LOGOUT

c. To remove abbreviations (in a private version) the internal commands are 'RC' and 'RP'. General form is

RC ab1 ab2 ... abn
RP ab1 ab2 ... abn

d. The parameter abbreviation table can contain a maximum of 40 definitions. A parameter definition will take precedence over a command abbreviation, where "command" is understood to be the first item in a "command line".

e. Two types of parameter abbreviations are permanently defined; they are automatically provided for and do not appear in the parameter abbreviation table: *

0x is the abbreviation for COMFL0x.
.x is the abbreviation for (CFLx).

4. Private commands: 'SAVED' files may be resumed without explicit typing of the resume command. "." maintains an internal list of the primary names of 'SAVED' files, and it consults this list to determine if a 'RESUME' command should be generated before any command it receives. For a "private version" this internal list may be created and updated through the use of internal command 'UPDPRI', which has no argument and searches the user's file directory for SAVED files. Note that if a 'SAVED' file is created and the 'UPDPRI' not reissued, "." will not "know" that the subsequently issued command is a private one. This feature may be suppressed by the internal command 'SVLIST OFF' which causes "." to perform an FSTATE for each command it deals with, in order to check for a SAVED version;

order to check for a SAVED version; The public version now has SVLIST * OFF normally. Beware of LOGOUT * SAVED in this context as such a file would be resumed when a LOGOUT command is issued. 'SVLIST ON' returns to the list-checking state, which is considerably faster.

5. File System Calls: In like manner to the treatment of * RESUMEs ("private commands"), "." will also furnish the CALL command (see AH.6.07) for file system subroutine calls. The following subroutines can be CALLED by typing the name of the entry followed by the necessary arguments:

```
ALLOT    DELFIL   OPEN
ATTACH   FSTATE   SETPRI
ATTNAM   IODIAG   TRFILE
CHFILE   MOVFIL
```

For example,

```
     STORGE   2
```

will generate the command

```
     CALL STORGE   2
```

6. Interconsole communications:

a. When the user is in ". SAVED" either in waiting input or sleeping status, an interconsole message may be received. (Whenever the user enters a command line and "." is left, all communication is forbidden this prevents the user's getting garbage in his input buffer while editing, typsetting, etc.)

b. To send a message, use the internal * command 'WRITE' followed by PROBNO, PROGNO. This places the user in interconsole communication mode. Subsequent input lines will be received by PROGNO PROGNO if he desires to communicate.

c. To prohibit comminication (in a private version) use the internal * command 'FORBID'.

d. To permit a particular user to communicate (in a private version), use the internal command 'ALLOW'

with PROBNO, PROGNO as arguments. (To allow all members of either class - i.e. all programmers within a given problem number, or a given programmer number under any problem number - use 000000 for the appropriate argument.)

To permit all users to communicate, simply give the 'ALLOW' internal command with no parameters. (This is the pre-set state in the public version.)

e. The interconsole section of ". SAVED" may be left by giving a single "break" signal, which causes "." to resume a fresh version of itself.

7. Sleeping: The internal command 'RS' will initiate a sleeping program which wakes up every ten minutes and prints a comment. Monopolization of lines to the computer in this fashion is rather anti-social, and should not be done unless absolutely necessary.

8. Response suppression: Successive usages of the 'V.' (verify) internal command will alternately suppress and permit printing of the acknowledgement characters 'R' and 'W' and of the sleeping comment in the private version. *

9. Termination: The internal command 'Q' will return the user to dead status. (Logging out may, of course, be accomplished while in "."). "Q" will prohibit further interconsole communication before going to DEAD. Hence, it is wise not to quit out of "." but to use "Q". *

Identification

GPM - A General Purpose Macrogenerator
Christopher Strachey, x6010
2/14/66

Purpose

This macrogenerator is an on-line symbol string processor, both its input and its output being strings of symbols. It operates by a form of substitution which is completely general in its application, in that substitution is allowed anywhere. The result is a powerful system including such features as recursive functions and conditional expressions, which can be implemented with very few (but very rebarbative)instructions.

Reference

C. Strachey, "A General Purpose Macrogenerator," The Computer Journal, Vol. 8, No. 3, pp 225-241, October, 1965. (A limited number of xerox copies are available in the Project MAC Library.)

Usage

RESUME GPM

Modifications

A. Input/Output

All input/output in GPM is in 12-bit mode. Within macro-names, however, only the last 6 bits are effective, so that the name "defs" is equivalent to "deFS". The following symbols are substituted in the MAC implementation for the corresponding symbols in the reference:

substitute	$	for	∫
"	#	"	~
"	?	"	Q
"	(	"	≤
"	)	"	≥

The symbol ¢ is used to indicate a continued line. Return to the system for GPM is accomplished by the input symbol ¤, via an unmatched ≥ as in the reference.

B. Machine Macros

Four machine macros have been introduced into the MAC implementation which do not appear in the

reference:

$LOSE, name;
This macro causes the most recent definition of the macro-name "name" to be excised from the definition chain.

$ESS;
This macro causes the current size of the stack to be output. (The maximum allowable stack size is 5,000.)

$READ, name;
This macro switches the macrogenerator input from keyboard to the file name (memo).

1) If the file "name (memo)" does not exist, the result is a return to the macrogenerator via a monitor call.

2) Occurence of the READ macro within a memo file is prohibited and results in a monitor call, after which reading of the original file resumes. Input returns to the keyboard when the reading of a file is completed.

$UNSTRING, arg;
This macro inserts commas between the characters in the string referred to by "arg".

C. Memo Files

Memo files are created and edited by TYPSET. Conflict between symbol conventions in typset and the macrogenerator language must be avoided by beginning TYPSET with the commands

```
ERASE "
KILL  ?
```

which establish the erase and kill conventions of the macrogenrator.

Two public memo files have been prepared for the convenience of macrogenerator users:

1) File DEFS (MEMO) contains certain standard macrodefinitions for loading onto the stack.

2) Additional information about the current state of the macrogenerator language, including such data as a list of the macros defined in DEFS, may be obtained by linking to file GPMINF (MEMO).

(END)

Identification

Solution of equilibrium field problems
EPS SAVED
C. Tillman, Rm. 3-482, X4465, 3/15/66

Purpose

EPS is a console-oriented system intended primarily for the solution of equilibrium field (boundary-value) problems in two-dimensional continua. Implementation of this system has required developing extensive algebraic and input-output capabilities. Thus, while EPS does not have the generality of a complete programming system, it does provide a facility of considerable power and flexibility for on-line algebraic and numerical manipulations. Consequently, it may be applied to problems quite unrelated to those for which it was specifically designed.

Description

EPS treats systems of simultaneous, second-order partial differential equations by the method of finite differences. These equations are assumed to be representable in a standard linear form; however, since the coefficients for this standard form may be expressed not only as functions of position but also as functions of the unknown field and its derivatives, it is possible to use EPS in an iterative fashion to solve certain nonlinear problems.

Since the program obtains solutions by a finite-difference technique, problem definition requires specification of a finite-difference lattice. An important feature of EPS is that it permits use of irregular difference lattices, so lattice points may be precisely placed along boundary contours and may be concentrated in regions of special concern. Moreover, the positions of lattice points, even those on boundaries, may easily be caused to change during the solution process; thus, e.g., problems involving free boundaries may be treated.

Organization of EPS resembles that of CTSS in the sense that users cause various tasks to be performed by issuing commands followed by argument strings. However, unlike CTSS, EPS scans input in a manner similar to that employed by format-free compilers. Thus one may type several commands on a single line or continue a long command from one line to the next with complete freedom. It follows that a simple carriage return cannot be used with EPS to signal an "end of message". Rather, the user must denote the end of a command or sequence of commands by typing a "$" just before his final carriage return. Typical input lines for EPS are:

```
DEFINE r=SQRT(x*x+y*y) $
SET x=3, y=4     PRINT r$
```

EPS currently recognizes twenty commands. These include the commands "APPEND", "DELETE", "CLOSE" and "IMPOSE" for describing boundaries and boundary conditions, the commands "DEFINE" and "SET" for parameter specification, the commands "TALLY", "FORM" and "RELAX" for initiation of various specialized numerical procedures, plus "PRINT", "REVIEW", "LIST" and "EXPAND" for inspection of results or past input.

Reference

A discussion of EPS commands and usage conventions may be found in MAC-M-284, which may serve as a rudimentary user's manual. Further information and help in using EPS may be obtained from the author.

Usage

```
RESUME EPS
```

After the CTSS W(ait) line, the message 'PROCEED:' will be typed on the user's console. A command or sequence of commands may then be issued. Some commands produce output responses and some do not (most do); the user will, at any rate, be made aware of return of control to the EPS supervisor by a recurrence of the 'PROCEED:' message. More commands may be issued at this time--and so on.

(END)

Identification

Transmit file through a link
TRNSMT SAVED
C. Garman, 9/24/65

Purpose

Since the editing programs will not allow an edited file to replace the original file if the original is a linked file, TRNSMT may be used to send a copy of a file through a link.

Usage

R TRANSMT NAME1 NAME2 NAME3 -NAME4-

NAME1 NAME2 is the name of the file which is to be transmitted.

NAME3 NAME4 is the name of the link which designates the target file. If NAME4 is omitted, it is assumed to be NAME2.

A condensed instruction line will be printed if not enough arguments are supplied. File system errors result in a print out from PRNTER followed by a call to DORMNT. Typing START will cause an attempt to close all active files and exit via CHNCOM. Normal exit is also via CHNCOM.

Identification.

Binary file editing program (EDB)
J. H. Saltzer, x6039 2/14/66

Purpose.

EDB is a reincarnation of the TYPSET and EDL editing commands for use with arbitrary binary files.

Usage.

RESUME EDB name1 name2

will allow creation or editing of the file "name1 name2". The editing conventions of EDB are identical to those of EDL and TYPSET, as described in section AH.9.01. Each 36 bit word of the file being edited is a distinct line in the sense of the TYPSET description, and is represented for editing purposes as a 12-digit octal number. Care should be taken to insure that after input or editing, there are exactly 12 octal digits in a line. If there are more than 12 octal digits, the last 12 will be used; if fewer, leading zeros will be appended. Non-octal characters will be converted to octal by truncation of the high order bits.

(END)

Identification

Add information to card image sequence fields
SEQ SAVED
D. Widrig, X6005, 3/15/66

Purpose

Files in card image form occasionally need to have an identification field added to selected columns between 73-80. SEQ SAVED allows an user to add information quickly and easily to the sequence field.

Usage

R SEQ NAME1 NAME2

NAME1 NAME2 refers to the card image file to be modified. SEQ responds by typing "TYPE SEQUENCE FIELD". The user responds by typing 8 alphabetic or numeric characters indicating the new field desired. The user may use the character '*' to indicate that the original character in the card image is to be retained.

EXAMPLE:

To add the letters ABCDE to columns 73-76 of A MAD and to retain the original information in 77-80,

```
R SEQ A MAD
W 1416.0
   TYPE SEQUENCE FIELD
ABCDE***
```

Note... Typing R SEQ is equivalent to asking for a condensed set of operating instructions.

Errors

An intermediate file with name PROBN PROGN holds the re-sequenced card image file. If the usual RENAME conventions fail, then the user will be left with both PROBN PROGN and NAME1 NAME2.
Other standard file system errors may occur. In these cases, no PROBN PROGN will remain nor will NAME1 NAME2 be changed.
Linemarked files will produce the comment "FILES MUST BE CARD IMAGE" and the program terminates by calling CHNCOM.

(END)

Revised: 7/30/66

Identification

Compress or expand BCD files
SQZBCD SAVED, PADBCD SAVED
B. Wolman, T.S. 532, x6002, 7/30/66

Purpose

To compress card image (not line-marked) BCD files by removing blanks in order to decrease track usage, and to expand compressed files. *

Usage

To compress file 'ALPHA BETA' into file 'GAMMA DELTA':

R SQZBCD ALPHA BETA GAMMA DELTA

To expand file 'ALPHA BETA' into file 'GAMMA DELTA':

R PADBCD ALPHA BETA GAMMA DELTA

If DELTA is omitted, GAMMA BETA will be created. If GAMMA is also omitted, a new ALPHA BETA will be written.

Older copies of the output file (GAMMA DELTA) will be deleted by a call to the library subroutine DELETE.

PADBCD SAVED may be used to expand files which were compressed using SQZBCD.

(END)

Identification

File compression
SQUASH SAVED
J.H. Saltzer, Rm 518 T.S., x 6039, 8/30/65

Purpose

To convert a card-image BCD file to a 6-bit, line-marked file (with tabs re-inserted for FAP and MAD programs).

Usage

RESUME SQUASH NAME1 NAME2 NAME3 (NAME4)

At least three arguments must be given to SQUASH.

NAME1 NAME2 is the name of the file to be linemarked.

NAME3 is the primary name of the file to be created. NAME3 may be the same as NAME1, if desired, but it must be explicitly typed.

NAME4 is an optional secondary name for the created file. If NAME4 is omitted, NAME2 will be used as the secondary name.

Description

If NAME2 is "FAP", a tab will replace one or more blanks immediately ahead of columns 8, 16, and 30.

If NAME2 is "MAD", a tab will replace blanks appearing immediately before column 11 or 12; a character appearing in column 11 is proceeded by a colon (logical backspace).

If NAME2 is not "MAD" or "FAP" no tabs are inserted. In all cases, trailing blanks are stripped off and columns 73-80 are discarded.

Comments

Experience with a variety of FAP and MAD programs indicates that a saving of from 60% to 75% of storage space is typical.

Identification

File expansion for SQUASHed files
XPAND SAVED
J.H. Saltzer, rm 518 T.S., x 6039. 8/30/65

Purpose

To convert a BCD line-marked file to card image form interpreting tabs for FAP and MAD programs.

Usage

RESUME XPAND NAME1 NAME2 NAME3 -NAME4-

At least three arguments must be given to XPAND.

NAME1 NAME2 is the name of the file to be converted.

NAME3 is the primary name of the card-image file to be created. NAME3 may be the same as NAME1, if desired, but it must be explicitly typed.

NAME4 is the secondary name for the created file. If NAME4 is omitted, NAME2 will be used as the secondary name.

Description

Tab interpretation is based on the secondary name of the file to be created. If the secondary name is not FAP or MAD, tabs in the file are left uninterpreted.

If the secondary name is "FAP", tab stops are assumed at columns 8, 16, 30, and every four columns thereafter.

If the secondary name is "MAD", a tab stop is assumed at column 12 and every five columns thereafter. If a colon appears in column 12, it is discarded and the next character moved back to column 11 of the resulting card. Serialization by "ones" is placed in columns 75-80.

If tab interpretation results in a card image greater than 72 columns, the card will be truncated, and printed out with an appropriate comment.

Revised: 7/3/66

Identification

Compress and expand BSS files
PADBSS SAVED and SQZBSS SAVED
N.I. Morris, T.S. 502, x 6012

Purpose

To compact BSS files by a factor of 2 in order to save disk space.

Usage

```
R SQZBSS ALPHA -BETA-
R PADBSS ALPHA -BETA-
```

SQZBSS will create a file named BETA SQZBSS. All zero words and card sequencing will be stripped off the card images.

PADBSS will read file 'ALPHA SQZBSS' and recreate file 'BETA BSS'.

If BETA is omitted, ALPHA will be used.

Checksums are computed and compared against the checksums on the cards. If a discrepancy is found, an error comment will be printed.

SQZBSS decks may be loaded using the LAED loader by typing: *

```
LAED LOAD (SQZ) NAME1 ... NAME1n
```

where NAME1 ... NAME1n are the primary names of n SQZBSS files. To load SQZBSS and BSS decks intermixed use:

```
LAED LOAD (SQZ) NAME1 (BSS) NAME12 NAME13
```

(The commands LOADGO, NCLOAD, and VLOAD may be used in place of LOAD.) See also AH.7.04.

(END)

Revised: 9/24/65

Identification

Print U.F.D. (FILE)
LIST SAVED, LSTOFF SAVED *
B. Wolman, T.S. 532, x6022, 8/30/65

Purpose

'LIST SAVED' is offered as an alternative to the system command LISTF. LIST will print all or a portion of the user's (or another) file directory using the same format and conventions as LISTF. The output produced by LIST will be sorted alphabetically (primary name first) instead of by date. LSTOFF is the same as LIST except that it creates a * file which is suitable for off-line printing by way of the * RQUEST command.

Usage

R LIST -(CFNn)- NAME1 NAME2 NAME3 NAME4
-(SYS)-

LIST will give the name, device, mode, size and date of files NAME1 NAME2, NAME3 NAME4, ... The list of file names (up to 8 may be specified) may be preceded by an optional request to enter another common file. Any primary (secondary) name may be the character '*' meaning any file having the same secondary (primary) name. If no names are given, all files will be listed.

If LIST is used in a common file directory, the author number of each file will be inserted in the output listing immediately following the secondary file name.

LSTOFF will create a file of primary name UFDFOR. * The secondary name will depend on the common file being listed. The user's file directory has secondary name of the programmer number. (CFLn) directory has the secondary name of 'CMFLOn'. (SYS) directory has the secondary name of 'PUBLIC'. The file is always created in the file directory from which the command was issued. RQUEST may be used to process the file, off-line.

Options

LIST has several options which control file sorting, elimination of linked files, and usage of other file directories. The argument(s) which specify these options

may appear whenever LIST expects to find a Primary file name. More than one option may be used, and file names may be specified before and after option parameters.

The argument '$' requests LIST to sort files by secondary name first. Normally the primary names are sorted first.

The argument '$$' requests LIST to ignore all linked files. This option can save a great amount of time since an FSTATE must be done to obtain the date and author of linked files.

The arguments '(USE)' ALPHA BETA tell LIST to read file ALPHA BETA instead of 'U.F.D. (FILE)' which is the user's file directory. ALPHA BETA should be created by a link command such as

```
LINK ALPHA BETA PROB PROG U.F.D. (FILE)
```

This feature is especially useful when one cannot log in.

Method

LIST first checks to see if common file switching is desired and if necessary performs it. LIST then reads the argument list saving file names and options. The file U.F.D. (FILE) (or that specified by the (USE) option) is read by a single call to RDWAIT. One pass is made through the file looking for names matching those specified by the user. When a file is found it is added to the list of files to be printed. If necessary, FSTATE's are performed on linked files. After all files have been accumulated, LIST returns to the file directory in which it started, sets a single interrupt, and begins printing.

Major revision: 3/30/66

Identification

Text display on ESL console
DISPLY SAVED
H. Murray, X6020

Purpose

To display text on the ESL console (Room 908, 545 Tech Square).

Usage

RESUME DISPLY NAME1 NAME2 -LINE-

NAME1 NAME2 is the CTSS name of the file to be displayed.

LINE is the line number the picture is to begin with (if other than 1).

PUSH-BUTTON
8 EXIT BUT SAVE PICTURE
9 TO 'TURN' PAGE
10 TO FIND LINE
11 TO EXIT FROM PROGRAM
12 TO START OVER

Description

Typing "RESUME DISPLY NAME1 NAME2" will cause the first "page" of the file to appear on the screen next to the teletype. Any file may be displayed. Line-marked files will be displayed with one record on each line, as with the PRINT command. A file that is not text (e.g. BSS, SAVED) will be displayed with each word in the file interpreted as BCD.

Errors result in self-explanatory comments and calls to DORMNT.

Push-button number 9 on the console is used to step the program to the next page of text.

Push-button number 10 will command the program to find the line having the sequence number equal to or greater than the number in the decimal switches (above the toggle switch registers).

Push-button number 12 will start over at the beginning of the file.

When finished with the program, push-button 11 will sign off from the kludge and return control to the user via CHNCOM.

Push-button 8 also sends control to CHNCOM, but it causes the current picture to be retained.

If called with no arguments DISPLY signs off from the kludge and returns via CHNCOM. If called with the single argument '1' DISPLY signs the user onto one console and then goes to CHNCOM.

(END)

Revised: 7/30/66

Identification

List links in a file directory
LSTLNK SAVED
C. Garman, x5889

Purpose

LSTLNK will print a summary of the linkage information for some or all of the links in a file directory. The information printed is the link name, the directory in which the file resides, the mode, and the actual name if different from the link name.

Usage

R LSTLNK -CF- -USE- -OPT- -NAME1- -NAME2- *

CF may be used to specify common file switching and is of the form '(SYS)' or '(CFLn)' where n is any digit or 'P'. '(SYS)' and '(CFLP)' are synonymous and mean TSSFIL or M1416 CMFL04. The original common file switch is restored before termination of the command.

USE comprises three agruments which may specify a file (e.g., a link to another user's U.F.D.) to be searched instead of the current U.F.D. (FILE) and is of the form '(USE)' NAME3 NAME4. *

OPT if specified, may be either '(TO)' or '(NAME)', and modifies the effect of NAME1 and NAME2, below. *

NAME1 NAME2 specify files, directories or alternate names (compare with LNAMEi in AH.5.01) used to select the links to be printed: if OPT is null, the NAMEi refer to file names in the directory being searched; if OPT is '(TO)', then links pointing to files in the directory whose PROBNO PROGNO are expressed by NAME1 NAME2 will be printed; and finally, if OPT is '(NAME)', then the NAMEi refer to the names in the 'target' directory. Examples:

R LSTLNK A B

lists a link in the current directory.

R LSTLNK (NAME) A B

lists a link which is named A B in the directory in which the file resides.

R LSTLNK (TO) T099,9 9876

lists all links which point to files which reside in directory T999 9876. *

For each link encountered, the following information is printed:

NAME1 NAME2 PROBN PROGN MODE -NAME3- -NAME4-

PROBN PROGN is the file directory in which the file (or further link) resides.

MODE is 3 octal digit file mode.

NAME3 NAME4 is the name of the file in PROBN PROGN, if different from NAME1 NAME2. If NAME3 is the same as NAME1, NAME3 is printed as a single equals sign (=). If NAME4 is the same as NAME2, only NAME3 is printed.

Restrictions

Order of optional arguments when more than one are used must be as given in the general calling sequence line, above. When using the '(TO)' option, problem numbers must contain four digits, the first of which is zero (0). E.g., T0999, not T999.

Errors:

INVALID ARGUMENTS OF '(USE)'.
fence found for either NAME3 or NAME4

'LINK(S) NOT FOUND'
specified links not contained in directory.

'U.F.D. TOO LONG'.
entire file directory could not be read into available memory, most likely by mis-application of '(USE)'. Search will continue on contents as read.

file system errors - various result in call to PRNTER command.

(END)

Identification

Print file directory in octal
OCTLF SAVED
N.I. Morris, X6029
2/14/66

Purpose

OCTLF will print all seven words of file directory entry(s) in octal. Two lines are printed for each entry. The first contains the file name in BCD followed by the file name in octal. The remaining five words of the file directory entry are printed in octal on the second line. This routine is useful when the exact contents of a U.F.D. entry must be determined.

Usage

R OCTLF -CF- -USE- -NAME1- -NAME2-

CF is used to indicate common file switching. It is of the form '(CFLn)' where 'n' may be a single digit or the letter 'P' to indicate M1416 CMFL04.

USE consists of three parameters which specify a file directory to be listed in place of the user's 'U.F.D. (FILE)'. It is of the form '(USE)' FNAM1 FNAM2, where FNAM1, FNAM2 are the primary and secondary names of a link to the 'U.F.D. FILE' of the other file directory.

NAME1 NAME2 specify the file name(s) to be listed. If both are omitted, the complete file directory is printed. If either parameter is an asterisk ('*'), all files of given primary or secondary name are listed. If NAME2 is omitted, '*' is assumed.

(END)

Identification

Convert 6-bit to 12-bit files
6TO12 SAVED
J.H. Saltzer, T.S. 518, x 6039, 8/30/65

Purpose

To convert a card image file to (MEMO) form, for use with the RUNOFF command.

Usage

RESUME 6TO12 NAME1 NAME2 NAME3

NAME1 NAME2 is the name of a card-image file to be converted to 12-bit line-marked format.

NAME3 is the primary name to be used for the resulting output file. If NAME3 is omitted, NAME1 will be used. The secondary name of the output file is always (MEMØ).

The resulting 12-bit line-marked file may be edited with TYPSET or inserted into an already typed memo in at least two ways. The '.append' request of RUNOFF may be used, or the files may be combined by using the non-sequencing option of the COMBIN command.

Identification

Combine line-marked files
APPEND SAVED
C. Garman, T.S. 526, 8/30/65

Purpose

To combine line-marked files so that they can be printed by off-line request processing.

Usage

APPEND NAME1 NAME2 NAME3i NAME4i ... NAME3n NAME4n

NAME1 NAME2 is the name of the file to be created by combining the files NAME3i NAME4i ... NAME3n NAME4n.

APPEND is used to prepare a single line-marked file for off-line printing by use of a PRINT control card. If file NAME1 NAME2 does not exist, it will be created permanent mode; otherwise the file will be added to by using '.APEND'.

The files NAME3i NAME41 will be separated from each other in the combined file by a program-control page skip, identifying the file.

Line-marked files will be copied line-by-line; files which are not line-marked are assumed to be 14-word card images, and will be copied with a full word of blanks added at the beginning of the line, for single space program control.

If any of the NAME3i or NAME4i are '*' (single asterisk), the corresponding NAME3i-1 or NAME4i-1 will be used. NAME1 NAME2 may not be appended to itself.

EXAMPLE: (assume that ABCXYZ FAP has been assembled with (LIST))

APPEND OUTPUT BCD ABCXYZ * * FAP * SYMTB

which is equivalent to:

APPEND OUTPUT BCD ABCXYZ BCD ABCXYZ FAP ABCXYZ SYMTB

If all the names of the files the user wishes to append will not fit on one line, the user may type:

START NAME1 NAME2 NAME3i NAME4i ... etc.

after the 'READY' from the system; or if NAME1 NAME2 is the same, he need only type:

START * NAME3i NAME4i ...

In either of the last two cases '*' for NAME3i or NAME4i refers to the last NAME3 or NAME4 on the previous line.

All calls to system disc subroutines have been provided with the appropriate error returns, which all return to the system via 'CHNCOM'. If the user provides no arguments, or only NAME1, the comment 'INCORRECT FORMAT' will be printed. In case the command list was truncated, or there was a NAME3i without its following NAME4i, the comment 'NAME3i IGNORED' will be printed.

Identification

Enciphering, deciphering of files
GARBLE: ENCIPH SAVED, DECIPH SAVED
R. Fenichel, D. Edwards, 9/24/65

Purpose

In order to provide added security or locks for files, GARBLE will scramble and unscramble (encipher and decipher) files by using a key word which is not necessarily stored elsewhere within the system.

Method

GARBLE accepts a message from the user, and initiaizes a random-number generator with a value computed from the characters of the message. A new random number is then added to or subtracted from each character of the file, as it is being enciphered or deciphered, respectively.

Restrictions

The user had better remember the keys which he has used - no one else will. Also, it is poor cryptographic practice to use any given key on more than one file.

Usage

```
R ENCIPH NAME1 NAME2 -NAME3- -NAME4-
R DECIPH NAME1 NAME2 -NAME3- -NAME4-
```

to transform NAME1 NAME2 into NAME3 NAME4. If NAME3 is omitted, it is taken as NAME1; if NAME4 is omitted, it is taken as NAME2.

ENCIPH creates a file in PRIVATE, PROTECTED mode; DECIPH creates a file in PERMANENT mode.

Timing

About 2 seconds/record.

Identification

Compare two files
CMPARE SAVED
C. Garman, T.S. 526, x5989, 9/24/65

Purpose

CMPARE is a program to do a word-by-word comparison of the contents of two files.

Usage

'RESUME CMPARE' NAME1 NAME2 NAME3 -NAME4-

compares the contents of file NAME1 NAME2 with that of NAME3 NAME4. NAME4, if not given, is assumed the same as NAME2.

Results

For every word which is not identical in both files, a line is printed

wordno worda wordb

WORDNO decimal sequence number of comparison error, beginning at 1.

WORDA contents of word WORDNO in NAME1 NAME2.

WORDB contents of word WORDNO in NAME3 NAME4

Errors

Not enough arguments.

All I/O errors are currently reflected as 'DISK ERROR'. User may use 'PRNTER' command to determine nature of error. This will soon be changed to issue the PRNTER command automatically.

Limitations

Current version does not compare last ('n' .MOD. 54) words of files; this will be fixed shortly.

Normal exit via CHNCOM; if no comparison errors found, nothing will have been printed.

Identification

Reformat ARCHIV files
NWARCH SAVED
D. Wagner, T.S. 507, 11/19/65

Purpose

The ARCHIV command was rewritten in October 1965. The files created are no longer of the same format as those created by the old ARCHIV command. NWARCH enables old ARCHIV files to be converted to the new format.

Usage

R NWARCH NAME1 NAME2 -NAME3- -NAME4-

where NAME1 NAME2 is a file of old archive format. NAME3 NAME4 is created from it in new format. NAME3 and NAME4 have default values NAME1 and NAME2 respectively.

Identification

Character set changes
FIXMEM SAVED
A. Evans, T.S. 519A, 11/5/65

Purpose

On September 21, 1965, a character code table was installed in CTSS which contained a number of changes, mostly inconsequential. However, it included two serious errors. These errors do not affect users of the BCD character set. Since that time, users with memos containing semicolon or exclamation points have discovered that their old memos do not type out correctly. On November 22 1965, the code tables will be corrected (4 characters will change positions) and these old memos will again type out correctly. Unfortunately, memos typed in or corrected between September 21 and November 22 will contain incorrect codes for semicolon and exclamation points. FIXMEM SAVED in the public file has been written to take a (MEMO) file with incorrect codes and produce a new file with correct codes.

Usage

RESUME FIXMEM A B C D . . .

File A (MEMO) will be corrected and rewritten as B (MEMO) ; file C (MEMO) will be corrected and rewritten as D (MEMO), etc. A file which does not need correction will not be harmed by FIXMEM.

Following are the changes to the character code tables:

1. Semicolon goes from code 0573 to code 0503

2. Bell goes from 0553 to 0513 (this change is not done by FIXMEM.)

3. Open Square Paren goes from 0174 to 0553 (the 1050 exclamation point now maps into the Open Square Paren).

4. Close Square Paren goes from 0134 to 0501 (the 1050 "plus-or-minus sign" now maps into the Close Square Paren).

These changes insure that all old memos will continue to be usable without modification, even after the new MAC 1050 balls are installed.

A revision of CTSS Programmer's Guide, Section AC.2.01 which reflects these changes will be ready in November.

Identification

Print storage map
STOMAP SAVED
B. Wolman, T.S. 532, x6022, 8/30/65

Purpose

To print the storage map from the (MOVIE TABLE) file created by the standard loaders (i.e., every loader not using the '(OLD)' option).

Usage

R STOMAP
Prints the file '(MOVIE TABLE)' on the user's console.

R STOMAP ALPHA
creates a file 'ALPHA MAP' containing a numeric and an alphabetic storage map of the file '(MOVIE TABLE)'.

R STOMAP ALPHA BETA GAMMA
Creates a file 'ALPHA MAP' from the file 'BETA GAMMA'. If GAMMA is omitted, 'TABLE)' is assumed. If ALPHA IS '(ONL)', the storage map will be printed on-line.

Identification

Search a saved file
SRCH SAVED
M.I. Morris, T.S. 502, x 6012

Purpose

To search a SAVED file for a specific word.

Usage

R SRCH NAME1 LWORD RWORD LMASK RMASK

NAME1 SAVED is the name of the file to be examined. LWORD and RWORD are the left and right halves of the word being searched for. LMASK and RMASK are the left and right halves of a mask used to control the search. If no mask is specified, a mask of 777777 777777 is assumed.

The file specified is loaded into core from the disk. Then each word of the loaded core image is compared against the word specified in LWORD and RWORD with only those bits corresponding to 1 bits in the mask being compared. All occurrences of the word being searched for result in the printing of the absolute location of the word followed by the word itself. If no occurrence of the word is found, a message to that effect will be printed. After the search is completed, the program will go to DORMNT. To resume another search on the same file, type:

START LWORD RWORD LMASK RMASK

To finish a search, and continue a chain of commands, type 'START' followed by a carriage return.

Example:

To search the file 'PADBCD SAVED' for all LDQ instructions, type:

R SRCH PADBCD 056000 0 777700 0

Note that preceding zeroes may be omitted.

Identification

Generate dump of SAVED file for off-line printing
DUMPER
D. Widrig, X6005
2/14/66

Purpose

DUMPER can be used to generate dump files suitable for off-line printing. Complete machine conditions preserved in the SAVED files can be obtained.

Usage

R DUMPER NAME1 -'(CORE)'-

The machine conditions contained within NAME1 SAVED are re-formatted and written into a file called NAME1 (DUMP). If the optional argument, '(CORE)', is used, a complete FMS-like dump is also written into NAME1 (DUMP). Each word in the core dump will be interpreted as octal, BCD, and operation code.

Timing

If the '(CORE)' argument is not used, the creation of NAME1 (DUMP) takes 1.5 - 2.0 seconds and produces a 2-record file. If the '(CORE)' argument is used, an X-record SAVED file produces a 6X-record (DUMP) file taking 2.5 - 3.5 seconds per record of SAVED file.

(END)

Identification

Check success of RUNCOM
QUES SAVED
C. Garman, T.S. 526, 8/30/65

Purpose

May be used to check success of commands in a 'RUNCOM'

Usage

R QUES ALPHA BETA

QUES will check to see if file 'ALPHA BETA' exists. If it does exist, the chain will be continued immediately, without further ado. If 'ALPHA BETA' does not exist, program will print:
'FILE ALPHA BETA NOT FOUND.'
'DO YOU WISH TO PROCEED,
and will wait for input. An explicit 'YES' will cause program to continue the chain via a call to 'CHNCOM'. Anything else will case the program to abort the chain and return to the system via 'DEAD'.

R QUES ALPHA
is the same as 'R QUES ALPHA BSS'

R QUES
will cause program to pause unequivocally with 'DO YOU WISH TO PROCEED,' (same conditions on reply as before).

R QUES ALPHA BETA (NOT)
For protecting against the deletion of files, the appearance of a third argument, '(NOT)', reverses the sense of the question, ie. if 'ALPHA BETA' is not found, the chain will be continued immediately, with no typed response. If the file exists, the program will print:
'FILE ALPHA BETA ALREADY EXISTS... DO YOU WISH TO PROCEED,'. Waiting for a response (as above). Naturally, if 'BETA' is to be 'BSS', it must be stated explicitly.

NOTE ...

If QUES is used within a RUNCOM, and the question is not answered in the affirmative, files of the form '...NNN SAVED' may still remain in user's file directory, as would be the case in any other break in the RUNCOM chain.

Identification

Parameter identification within RUNCOM
RUNPRT SAVED
C. Garman, T.S. 526, 11/19/65

Purpose

If it is desirable to print a comment line which identifies the substituted parameters within a RUNCOM, RUNPRT may be used. It prints the contents of the current command buffer with excess blanks deleted.

Usage

RESUME RUNPRT ARG1 ... ARGn

RUNPRT will type one single line of text on the user's console, of the form

$ ARG1 ARG2 ... ARGn$

where all blanks in the parameters ARGi have been removed, and a single blank inserted between successive ARGi. ARGi may be any words to be used in constructing the comment and any (or none) of the ARGi may be sustitutable arguments within te RUNCOM.

Restrictions:

The full command buffer may be used, but only 14 words will be printed after conversion of the input parameters into the output image (null characters are not used in formatting the output line). Only the last six characters of any parameter will be printed.

Example:

Consider the following RUMCON, in file X BCD:

```
CHAIN ALPHA MAD
RESUME RUNPRT START 'X' FOR ALPHA MAD
MAD ALPHA (LIST)
```

The command

RUNCOM X BOOK FAP

would result in the following output on the user's console:

```
     Y STARTED
$ START 'X' FOR BOOK FAP
     LENGTH nnnnn
     ...
     X HAS BEEN RUN
```

Identification

Slave consoles
SLAVE SAVED
N.I. Morris, T.S. 507, x6012, 8/30/65

Purpose

To attach one or more remote consoles to serve as I/O devices for a user.

Usage

R SLAVE MODE ID1 ID2 ... IDN

MODE consists of any combination of the slave modes discussed in section AG.1.05. MODE may also be 'RELEAS' in order to release consoles that are already slaved.

ID1 ... IDN are the console identification numbers of the consoles to be attached or released.

Execution

The console(s) specified are slaved to the user in the mode specified. If MODE was 'RELEAS', the consoles are released from the user.

Identification

CTSS usage
WHO SAVED
N.I. Morris, T.S. 507, x 6012, 8/30/65

Purpose

To determine who is using CTSS at any given time.

Usage

R WHO
or R WHO N
or R WHO PROGN -PROGN- -N-

The name of the current system and the last time it was loaded are printed on the user's console, then the number of users currently logged in, and the current time and date.

Following this is printed the line, problem and programmer number, line multiplier, console identification number, time used since logging in, and login time for each user currently logged in.

If PROBN PROGN are specified, only statistics for PROBN PROGN will be printed. * PROGN will print all users with programmer number PROG. If PROGN is omitted PROBN * will be assumed.

N is an optional parameter giving the time in minutes that the program is to 'SLEEP' before again printing the number of users, date and time, and information about each user as explained above. However, instead of the time of login, the time used since last print out is printed. If N is omitted, control passes to CHNCOM. If N was given, this routine may be terminated at any time it is 'asleep' by typing a new command.

CTSS PROGRAMMER'S GUIDE INDEX

(5/31/66)

SUBROUTINES

COMMANDS

AED	AH.2.01
ARCHIV	AH.4 .01
BEFAP	AH.2 .02
BSS	AH.5 .05
CALL	AH.6 .07
CHMODE	AH.6 .03
COMBIN	AH.6 .01
COMFIL	AH.6 .04
COMIT	AH.2 .04
CONTIN	AH.7 .03
COPY	AH.6 .04
CRUNCH	AH.4 .02
DELETE	AH.6 .03
DELFIB	AH.1 .03
DYNAMO	AH.2 .05
ED	AH.3 .02
EDIT	AH.3 .01
EDL	AH.3 .07
ESL	AH.2 .06
EXTBSS	AH.6 .05
FAP	AH.2 .07
FAPDBG	AH.8 .01
FIB	AH.1 .03
FILE	AH.3 .01
GENCOM	AH.10.02
GPSS	AH.2 .08
INFO	AH.9 .04
INPUT	AH.3 .01
LABEL	AH.3 .06
LDABS	AH.7 .02
LINK	AH.3 .05
LISP	AH.2 .09
LISTF	AH.5 .01
LOAD	AH.7 .01
LOADGO	AH.7 .01
LOG	AH.9 .02
LOGIN	AH.1 .01
LOGOUT	AH.1 .02
MAD	AH.2 .10
MADBUG	AH.8 .02
MADTRN	AH.2 .11
MAIL	AH.9 .05
MOUNT	AH.3 .06
MOVIE)	AH.7 .01
MYSAVE	AH.3 .03
NCLOAD	AH.7 .01
OPS	AH.2 .15
PATCH	AH.8 .04
PERMIT	AH.3 .05
PM	AH.8 .03
PRBIN	AH.5 .04
PRBSS	AH.5 .05
PRFIB	AH.1 .03
PRINT	AH.5 .03
PRINTF	AH.5 .02
PRNTER	AH.11.01
RECALL	AH.7 .03
REMARK	AH.9 .03
RENAME	AH.6 .03
RERUN	AH.3 .04
RESTOR	AH.7 .03
RESUME	AH.7 .03
REVOKE	AH.3 .05
RQUEST	AH.6 .06
RSTART	AH.7 .03
RUNCOM	AH.10.01
RUNOFF	AH.9 .01
SAVE	AH.3 .03
SAVED	AH.5 .06
SAVFIL	AH.3 .04
SD	AH.8 .06
SDUMP	AH.5 .06
SNOBOL	AH.2 .12
SP	AH.8 .06
SPATCH	AH.8 .05
SPLIT	AH.6 .02
START	AH.7 .03
STOPAT	AH.8 .04
STRACE	AH.8 .07
STRUDL	AH.2 .13
TAPFIL	AH.3 .06
TFILE	AH.3 .01
TIP	AH.2 .16
TRA	AH.8 .04
TRACE	AH.8 .07
TTPEEK	AH.1 .04
TYPSET	AH.9 .01
UMOUNT	AH.3 .06
UNLINK	AH.3 .05
UPDATE	AH.6 .04
UPDBSS	AH.6 .05
USE	AH.7 .01
VERIFY	AH.3 .06
VLOAD	AH.7 .01

PUBLIC

6TO12	AJ.6 .01
APPEND	AJ.6 .02
CMPARE	AJ.6 .04
DECIPH	AJ.6 .03
DISPLY	AJ.5 .02
ENCIPH	AJ.6 .03
EPS	AJ.2 .03
ESL	AJ.5 .02
FIXMEM	AJ.6 .06
GARBLE	AJ.6 .03
LIST	AJ.5 .01
LSTLNK	AJ.5 .03
LSTOFF	AJ.5 .01
MADIO	AI.2 .01
NWARCH	AJ.6 .05
PADBCD	AJ.4 .01
PADBSS	AJ.4 .04
PRINT	AI.2 .01
QUES	AJ.10.01
READ	AI.2 .01
RUNPRT	AJ.10.03
SLAVE	AJ.11.01
SQUASH	AJ.4 .02
SQZBCD	AJ.4 .01
SQZBSS	AJ.4 .04
SRCH	AJ.8 .02
STOMAP	AJ.8 .01
TRNSMT	AJ.3 .01
WHO	AJ.11.02
XPAND	AJ.4 .03
.	AJ.2 .01

(END)