The Compatible Time-Sharing System

A Programmer's Guide

The Compatible Time-Sharing System

A Programmer's Guide

SECOND EDITION

The M. I. T. Computation Center

P. A. Crisman, Editor

The M. I. T. Press
Massachusetts Institute of Technology
Cambridge, Massachusetts

Library of Congress Catalog Card Number: 65-25206
Printed in the United States of America

This second edition represents a major revision and extension of the first edition and is necessitated by the continuous evolution of the Compatible Time-Sharing System (CTSS) over the past two years of operation. As CTSS has been improved in reliability and capacity, since the summer and fall of 1963, it has been implemented at both the Computation Center and Project MAC. Both installations operate as a community service, seven days a week, twenty-four hours a day with the MAC computer being time-shared full time and the Computation Center computer being time-shared about half of the time. At present, over 110 consoles are scattered throughout the MIT campus, at New England colleges, and in the homes of several Project MAC participants. As a result, the two installations have had extensive experience with a broad spectrum of users. Therefore, it is no longer a question of the feasibility of a time-sharing system, but rather a question of how useful a system can be produced.

During these two years of growth, there have been frequent changes of hardware configuration. Over seven different varieties of terminals have been attached to the system (three are obsolete now) and several different drum and disk configurations have been used. Because of the programming interface design, most of these changes have been insulated from the average system user. Despite the numerous hardware changes it has become increasingly obvious that the essence of a useful time-sharing system lies in the programming, i.e., in the software, and not in the hardware.

The programming has grown from a skeletal form of perhaps 50,000 instructions to an estimated size of between 400,000 and 1,000,000 words of publicly-available system program. From the few languages which were first available, the system also has evolved to presently contain over a dozen languages. Much of this growth in both words and in languages is the work of many users rather than of system programmers. In fact, it has been a goal to enhance and simplify the process of sub-system writing by supplying a framework that is highly modular and which encourages division of responsibility and initiative.

Many of the ideas described in this manual were mentioned in the first edition but at that time had not been implemented. In addition, several key features have been introduced to make a more complete system. A brief list of some of these features, which are detailed more completely within this manual, are: password logic, introduction of more elaborate accounting procedures, inter-console message, public files, and macro commands. Further details of the system design and implementation are given in Project MAC Technical Report No. 16 by J. Saltzer. A summary of system operational experience is given by R. Fano in Project MAC Technical Report No. 12 (AD-609-296) and is also published as an article in the January 1965 issue of the IEEE Spectrum.

Two major features have been introduced into the system which deserve special comment. First, the entire secondary storage mechanism has been redesigned. This is considered to be the most significant and far reaching change because it improves the multi-programming capability of the system and the controlled sharing of files on the part of user. The design and implementation of this critical section has been led by Robert Daley.

The second major new feature is the improved message coordination with the typewriter terminals. This feature, while not obvious to users, has greatly improved the organization and operation of the supervisor program. The work in this important and critical area has been done by Stanley Dunten who also has been instrumental in maintaining effective system operation.

The present manual is considered a part of the system because it is maintained on-line within the system, and it represents an attempt to keep all system documentation continuously up to date. As system users know, documentation difficulties have been severe, with over 80 bulletins and numerous research memoranda prepared and circulated as amendments to the first edition of the manual.

The effect of the present manual is that an active system user can keep his manual updated. To do this, he should periodically inspect a special table of contents of the manual, which is maintained on-line within the system in reverse chronological order of changes that have been made to the various sections. From this special table of contents, he can quickly determine which sections have been revised since the last time he updated his copy, and then obtain on-line printouts of those sections he needs. Needless to say, the procedures of requesting appropriate sections by mail or in person will still be available. In any case, the need for maintaining a massive mailing list for amendments to the manual is eliminated.

Acknowledgements

In addition to the previously-mentioned critical work of preparing the present system by Robert Daley and Stanley Dunten, the system owes its present form to an ever increasing number of staff members and contributors. Other contributors to the system programming are, alphabetically: Janet Allen, Michael Bailey, Robert Creasy, Patricia Crisman, Marjorie Daggett, Daniel Edwards, Robert Fenichel, Charles Garman, Robert Graham, Thomas Hastings, Jessica Hellwig, Lyndalee Korn, Richard Orenstein, Louis Pouzin, Glenda Schroeder and Mary Wagner. In addition, contributions of some of the commands have been made by Margaret Child, Leola Odland, Don Oppert, and Jerome Saltzer. Many of the subroutine write-ups which served as reference documents for the present system were prepared by Edith Kliman, Judith Spall, and Susan Springer.

A great deal of the present system's impact upon users has been because of its reasonably continuous and reliable service. To a large extend, this has been due to the great zeal and perserverence of the Computation Center's operational staff, who have conscientiously dealt with the many problems which have arisen.

We wish to thank the Computation Center and Project MAC administration for contributing the proper environment and shouldering the many problems which have been generated. They have made possible the present system's high level of development.

Thanks are also due to the maintenance personnel of the International Business Machines Corporation and of the New England Telephone and Telegraph Company for their diligent efforts in maintaining a high level of system performance.

A special acknowledgement goes to the Advance Research Projects Agency of the Department of Defense, and the Office of Naval Research, the sponsors of Project MAC, and the National Science Foundation, for the support of some of the special equipment at the Computation Center.

F.J. Corbato
May 1965
Cambridge, Massachusetts

This handbook is an attempt to document the techniques of using a current version (model 13) of the compatible time-sharing-system (CTSS) which has been developed at the MIT Computation Center. It is primarily a manual of how to use the system, in contrast to many of the research memos, which have been more detailed in their documentation of the techniques of implementation. Because CTSS is basically a system which will allow an evolutionary development of time-sharing while continuing to allow more conventional background systems to operate, it is expected that the present manual will of necessity be revised many times before it reaches a final form. A good deal of the difficulty arises from, on the one hand, the rather drastic change in user operating techniques which time-sharing permits, and on the other hand the immense amount of programming required to fully implement the system.

The present work, although not highly polished, is being presented now to assist in this evolutionary process. It is expected to be a supplement to the Computation Center's Procedures Handbook which explains many of the general administrative details of the Center. Furthermore, a knowledge of programming is assumed of the reader. It has been our objective to present to an experienced programmer a reasonably complete manual which will allow him to use wisely the present version of the time-sharing system.

Because of the rapidity with which many of the features are being implemented, and the delays in distributing the inevitable revisions, some features are described here which are not yet accomplished. The reason for this is that it was felt to be important to indicate the intended scope and objectives of the system so that individual users could plan ahead in their applications. The features which are not implemented will be found listed in an appendix which will be revised periodically. In addition, each of the chapters can be expected to be periodically revised.

Since the present work is primarily a handbook, no attempt has been made to make any comparisons with the several other time-sharing and remote-console efforts which are being developed by groups else-where. The only other general purpose time-sharing system known to be operating presently, that of the Bolt, Beranek and Newman Corporation for the PDP-1 computer, was recently described by Professor John McCarthy at the 1963 Spring Joint Computer Conference. Other time-sharing developments are being made at the Carnegie Institute of Technology with a G20 computer, at the University of California at Berkeley with a 7090, at the Rand Corporation with Johnniac, and at MIT (by Professor Dennis)with a PDP-1. Several systems resemble our own in their logical organization; they include the independently developed BBN system for the PDP-1, the recently initiated work at IBM (by A. Kinslow) on the 7090 computer, and the plans of the System Development Corporation with the Q32 computer.

To establish the context of the present work, it is informative to trace the development of time-sharing at MIT. Shortly after the first paper on time-shared computers, by C. Strachey at the June 1959 UNESCO Information Processing Conference, H.M. Teager and J. McCarthy at MIT delivered an unpublished paper "Time-Shared Program Testing" at the August 1959 ACM Meeting. Evolving from this start, much of the time-sharing philosophy embodied in the CTSS system has been developed in conjunction with an MIT preliminary study committee (initiated in 1960), and a subsequent working committee. The work of the former committee resulted, in April 1961, in an unpublished (but widely circulated) internal report. Time-sharing was advocated by J. McCarthy in his lecture, given at MIT, contained in "Management and the Computer of the Future" (MIT, 1962). Further study of the design and implementation of man-computer interaction system is being continued by a recently organized institute-wide project under the direction of Professor Robert M. Fano. In November 1961 an experimental time-sharing system, which was an early version of CTSS, was demonstrated at MIT, and in May 1962 a paper describing it was delivered at the Spring Joint Computer Conference.

As might be expected, the detailed design and implementation of the present CTSS system is largely a team effort with the major portions of it being prepared by the following: Mrs. Majorie M. Daggett, Mr. Robert Daley, Mr. Robert Creasy, Mrs. Jessica Hellwig, Mr. Richard Orenstein, and Professor F.J. Corbato. Important contributions to some of the commands and the background system has been offered by Professor Jack Dennis, Mr. J.R. Steinberg, and members of the Computation Center Staff. Mrs. Leslie Lowry, Mr. Louis Pouzin, and Mrs. Evelyn Dow have contributed to the preparation of the commands.

Special credit is given to Professor Herbert Teager for the design and development of his Flexowriter control subchannel which allowed the original experimental version of the present system to be developed, tested, and evaluated; only with such an opportunity was it possible to have the confidence to make the present pilot development of the CTSS system.

We should also like to extend our thanks to the Computer Center of the University of Michigan where Professor Bernard Galler, Mr. Bruce Arden, and Mr. Robert Graham have been very helpful in advising us on the use of their Mad Compiler in our time-sharing system. In addition, Mr. Robert Rosin kindly made available the Madtran editing program for processing Fortran II subprograms to Mad subprograms.

We should further like to take this occasion to acknowledge partial support by the National Science Foundation, the Office of Naval Research, and the Ford Foundation, of the development of our present system. We also add our appreciation for the support provided the Computation Center by the IBM Corporation.

Finally, we should like to encourage the readers of this handbook to examine the present system with a view toward improvements and we shall welcome such criticisms.

F. J. Corbato
Cambridge, Massachusetts
May 1963

TABLE OF CONTENTS (12/31/69)

(* denotes file system section)

(END)

INDEX (12/31/69)

<u>SUBROUTINES</u>

COMMANDS

(END)

Identification

Introduction to Time-Sharing

Time-sharing is an ambiguous term. Some people use this term to describe concurrent operation of several parts of a single computer. This sort of operation, also called multiprogramming, generally is directed toward efficient utilization of hardware.

The time-sharing system described in this manual seeks to allow a somewhat different sort of efficiency. Although hardware utilization is still considered, the primary goal is concurrent, effective utilization of a single computer by several users.

The motivation for time-shared computer usage arises out of the slow man-computer interaction rate presently possible with the bigger, more advanced computers. This rate has changed little (and has become worse in some cases) in the last decade of widespread computer use.

In part, this effect has been due to the fact that, as elementary problems become mastered on the computer, more complex problems immediately become of interest. As a result, larger and more complicated programs are written to take advantage of larger and faster computers. This process inevitably leads to more programming errors and a longer period of time required for debugging. Using current batch processing techniques, as is done on most large computers, each program bug usually requires several hours to eliminate, if not a complete day. The only alternative available has been for the programmer to attempt to debug directly at the computer, a process which is grossly wasteful of computer time and hampered seriously by the poor console communication usually available. Even if a typewriter is available at the console, there are usually lacking the sophisticated query and response programs which are vitally necessary to allow effective interaction. Thus, what is desired is drastically to increase the rate of interaction between the programmer and the computer without large economic loss and also to make each interaction more meaningful by extensive and complex system programming to assist in the man-computer communication.

In addition to allowing the development of usable and sophisticated debugging techniques, an efficient time-sharing system should make feasible a number of relatively new computer applications which can be implemented only at great cost in a conventional system. Any problem requiring a high degree of intermixture of computation and communication on a real-time basis should readily lend itself to time-sharing techniques. Examples of this type of application include:

decision-tree problems; real-time management problems (airline reservations, hospital administration, etc.); gaming problems; sociological experiments; teaching machines; language learning problems; library retrieval; text-editing; algebra manipulators; and many more.

The Compatible Time-Sharing System (CTSS) is a general-purpose programming system which allows a new form of computer operation to evolve and yet allows most older programming systems to continue to be operated. CTSS is used from consoles which may be of several varieties, but which in essence are electric typewriters. Each console user controls the computer (i.e. as seen by him) by issuing standard commands, one at a time. The commands allow convenient performance of most of the routine programming operations such as input, translation, loading, execution, stopping, and inspection of programs. This command convenience, although it has a fixed format, causes no loss of generality since a command can also be used to start an arbitrary programming subsystem with its own control language.

The consoles of CTSS communicate with the "foreground" system, by which computation is performed for the active console users in variable length bursts, on a rotation basis, according to a scheduling algorithm. The "background" system is a conventional programming system (slightly edited for the time-sharing version) which, at the least, operates whenever the "foreground" system is inactive, but which may also be scheduled for a greater portion of the computer time. The entire operation of the computer is under the control of a supervisor program which remains permanently in the 32,768 word A-bank of core memory. When a user program is scheduled to be run, it is brought into the 32768-word B-bank of core memory (unless it is already there) from drum or disk memory.

Not only are the drum and disks used for swapping of active user programs, but all console users utilize the disk memory for semi-permanent storage of their active program and data files. Cards and magnetic tapes still serve in secondary roles as long-time and back-up storage devices.

(END)

Identification

General Description and Usage Techniques

The foreground system is organized around both "commands", which are system programs accessible to all users, and the user's private program files. Both types of programs are stored on the disk, along with files of data, documentation, etc. For convenience, the disk files have titles with name and class designators. Files can be entered from consoles or cards, and they may be punched out at disk editing time.

The Supervisor

The supervisor program remains in A-core at all times when CTSS is in operation. Its functions include: handling of all input and output; scheduling; handling of temporary storage and recovery of programs during the scheduled swapping; monitoring input and output performed by the background system; and performing the general role of monitor for all jobs. These tasks can be carried out by virtue of the supervisor's direct control of all trap interrupts, the most crucial of which is the one associated with the interval timer clock.

The interval timer clock is set for small bursts of time, currently 200ms. Every clock burst allows the supervisor to interrupt the program currently running in B-core in order to interpret input from the consoles or to issue output to the consoles. If the input from a console is other than a break character, it is left in the supervisor's core buffers. When a break character is encountered, the supervisor determines whether this is a line of input which has arrived early for one of the working programs or whether the status of one of the users should be changed; i.e., to working status or waiting command status. If the line was a command line, the user is placed in waiting command status so that the next time his turn arrives, the supervisor can load the command program as his working core image.

The user programs are run for periods of time determined by the scheduling algorithm. At the end of each program's allotted time or if it changes status, the supervisor determines which user is to be run next. It must then determine whether the program or programs currently in core must be dumped (to disk or drum), in part or entirely, to leave room in core for the next user. The next user program must then be retrieved from secondary storage together with the proper machine conditions.

In addition to maintaining input and output buffers for each user console, the supervisor keeps a record of the status of each user. The status of a user may be: "working", where a program is ready to continue running whenever it is next brought in ; "waiting command", where the user has just completed a command line at his console; "input-wait" or "output-wait", where the program is temporarily held up waiting for either a console line or a free output buffer; "file-wait", where the program is temporarily delayed until another user has finished using the requested file; "I/O queue wait", where a program is delayed because an I/O device (typically a tape) is busy or not yet ready; "timer-wait", where the program has requested that it be delayed for a specified time; "dormant", where the program has stopped running and returned control to the supervisor, but machine conditions and the status of memory are preserved for inspection, modification, or re-entry; and "dead", where the program has terminated, control has been returned to the supervisor, and machine conditions and the status of memory have been scrapped.

It should be noted that command programs are handled in exactly the same manner as the user's own programs, with respect to status and scheduling. The background system is also considered another user; at present it has a different place in the scheduling algorithm, with permanently lowest priority. In addition there is another type of background, consisting of background jobs initiated from consoles but left to run without console interaction; these jobs are run with exactly the same type of scheduling as normal foreground programs.

Command Format

Commands may be typed by dead or dormant users; they are interpreted by the time-sharing supervisor (not by the user programs). They can thus be initiated at any time, regardless of the particular program in memory. (It is for similar reasons of coordination, that the supervisor handles all input-output of the foreground system typewriters.) Commands are composed of fields separated by blanks; the first field is the command name, and the remaining fields are parameters pertinent to the command. Each field consists of the last 6 characters typed most recently since the last blank (initially an implicit 6 blanks). A carriage return is the signal which initiates action on the command. Whenever a command is received by the supervisor, "W t" is typed back. When the command is completed, "R t1 + t2" is typed back. "W" is the abbreviation for WAIT; "R" for READY; "t" is the current time of day; "t1" is seconds spent in execution; and "t2" is seconds spent in swapping. A command may be abandoned at any stage, including during the typing of the command line or during command output, by giving the "quit signal" peculiar to the console.

A "command line" which has a dollar sign ($) as its first character will be treated as a comment and will not be executed.

Command Initiation

At the completion of a command line at a user's console, that user is placed in waiting-command status. He is then set at the end of a scheduling queue which is chosen according to a rule assigning higher priority to shorter programs. When this user reaches the head of the highest-priority active queue, he will be placed into working status.

When the user first reaches working status, the supervisor searches its command directory for an entry giving information about the command. There are three types of commands:

1. A-CORE-TRANSFER - special supervisor functions, such as SAVE. A supervisor subroutine is executed in core A, and the user is restored to the state he was in before issuing the command.

2. B-CORE-TRANSFER - cause the user's program to be started at a given location. These commands (USE, START, etc.) cause the message

   "ILLEGAL SEQUENCE OF COMMANDS"

   to be typed if the user does not have a core image (i.e., if he is not in DORMANT status).

3. DISK-LOADED - these commands are by far the most numerous. The program which is associated with ("which performs") a given disk-loaded command resides in a disk file (of second name 'TSSDC.' for system commands, 'SAVED' for user commands), in the system file directory or the user's own files (see AH.10.04 concerning private commands). When it is executed, a disk-loaded command becomes the user's core image. Some disk-loaded commands are "PRIVILEGED" and may make supervisor calls which users are forbidden to make.

If the command name is found in the command directory, the supervisor either:

1. Executes the indicated A-core subroutine, and returns;

2. causes the user's location counter to be set to the correct value, and places the user in working status;

3. loads the indicated disk file as the user's program and starts the user at the beginning of

his new core image.

If the command name is not found in the directory, the supervisor assumes that the command is an unprivileged disk-loaded command, and attempts to load a command file with first name the same as the command name. If no such file exists, perhaps because the command name has been misspelled, the comment

```
'name' NOT FOUND.
```

will be typed. In such a case, the user's core image and machine status are preserved.

If the 200's bit in the user's restriction code is on, he is a "restricted user" and may not use any disk-loaded commands except LOGIN and LOGOUT. That is, he may use only

```
LOGIN, LOGOUT
RESUME, RESTOR, CONTIN, RECALL, R
SAVE, MYSAVE
START,RSTART
USE, PM, STOPAT, TRA, PATCH, STRACE, PAPDBG
```

All other commands issued by a restricted user will be "NOT FOUND".

(For all practical purposes, such a user may only resume SAVED files, and the particular SAVED files in his directory determine completely what use he may make of the system.)

If the 1000 bit in the user's restriction code is _not_ on, he is a "subsystem-restricted" user. Such a user may not alter his standard options or subsystem trap status; his subsystem will have been initialized by LOGIN. His ability to use CTSS is determined by the subsystem.

Program Termination

A foreground program terminates its activity by one of two means. It can re-enter the supervisor in a way which eliminates the core image and places the user in a _dead_ status; alternatively, by a different entry the program can be placed in a _dormant_ status (or be manually placed there by the user giving a quit signal). The dormant status differs from the dead status in that a dormant user may still restart or examine his program.

Input and Output Wait States

User input-output to each typewriter is via the supervisor, and even though the supervisor has a few lines of buffer space available, it is possible for a program to become input-output limited. Consequently there is an _input-wait_ status and an _output-wait_ status, into which the user program is automatically placed by the supervisor whenever input-output delays develop. When buffers become nearly empty on output or nearly full on input, the user program is automatically returned to working status; thus waste of computer time is avoided.

Scheduling

In order to optimize the response time to a user's command or program, the supervisor uses a multi-level scheduling algorithm. The basis of the algorithm is the assignment of each program as it enters working or waiting command status to an nth level priority queue. Programs are initially entered at a level which is a function of the program size (i.e., at present, programs of less than 4k words enter at level 2 and longer ones enter at level 3). There are currently 9 levels (0-8). The process starts with the supervisor operating the program which is first in the queue at the lowest occupied level, L. The program executes for a time limit = 2.P.L quanta; a quantum of time is one half second. If the program has not finished (left working status) by the end of the time limit, it is placed at the end of the next higher level queue. The program at the head of the lowest occupied level is then brought in. If a program P enters the system at a lower level than the program currently running, and if the current program P1 has run at least as long as P is allotted, then P1 will be returned to the head of its queue and P will be run.

There are several different time limits whose current values may be of interest to the users. If a data phone is dialed into the computer and the user does not log in within 2 minutes, there is an automatic hangup. If a user stays in any non-working status for one hour, he is automatically logged out. The clock burst which enables the supervisor to housekeep the console input and output and to change program status is currently set to 200 ms. The quantum of time used in the scheduling algorithm is one-half second.

Memory Protection and Relocation

To avoid fatal conflicts between the supervisor and multiple users, the CTSS IBM 7094 includes a special modification which behaves as follows:

Core memory is divided into 256-word blocks. There are two 7-bit protection registers which, when the computer is in its normal mode, can be set by program to any block numbers. Whenever a user program is run, the supervisor, as a final step just before transferring to the user program, switches the computer to a special mode such that if reference to any memory address outside the range of the protection register block numbers is attempted, the normal mode is restored and a trap occurs to the supervisor.

There is also a 7-bit relocation register which modifies every memory reference, during execution, by addition of the relocation register block number. Thus programs which have been interrupted by the supervisor may be moved about in memory, if necessary, with only the proper readjustment of the relocation register required.

Finally, if the user program, while in the special mode, should attempt to execute any instructions concerning input-output, changes in mode or core bank reference status, or resetting of the protection or relocation registers, the normal mode is restored and a trap occurs to the supervisor program in core bank A. Errors in this class are known generically as protection mode violations.

User Communication with the Supervisor

The supervisor performs a number of control functions which may be directly requested by the user. These include: all input and output (e.g., disk, drum, consoles, tapes); requests for information about or extension of the user program memory allocation; simulation of floating point trap; control of each user's status, interrupt level, and input mode; and other functions which involve communication with, or control by, the supervisor.

Since all protection violations cause a trap to the supervisor, users may conveniently communicate with the supervisor by means of such violations. Before rejecting a protection violation as a user error, the supervisor checks the possibility that it was caused by a user-program of the form

```
              TSX  NAME1,4
              ...  ...
              ...  ...
        NAME1 TIA  =HNAME
```

where NAME is the BCD name of a legitimate supervisor entry point. The details of each supervisor entry are described in section AG. The TIA instruction is described in IBM manual L22-6636; it may usefully (but inexactly) be read as Trap Into A core.

(END)

"TIME-SHARING PRIMER"

INTRODUCTION

Beginnings are most difficult. This is far more true than trite in regard to the use of the Compatible Time-Sharing System (CTSS), which involves techniques that are liable to seem rather obscure even to experienced programmers. This document was designed, then, in order to facilitate the new CTSS user's transition from batch-processing orientations to a time-sharing orientation. It does not pretend to offer "sophisticated" information. Rather, it is intended to relieve the reader of the necessity of having to worry about ferreting out -- usually by word of mouth -- the basic operational information which is prerequisite to sophistication. So, leaving only the details of becoming an accredited user through administrative channels, and of turning on and dialing in his particular console (see Section AC.3) to the reader, we attempt to present here a "toehold", a guide (including an annotated "script") to the new CTSS user for his first time-sharing console session.

(The material herein is based upon "Some Introductory Notes on Time-Sharing Console Usage Techniques for the Summer Programming Course," which was written as a reference for students taking the one week Basic Programming and FAP Courses, offered annually by the M.I.T. Computation Center; as such, it may have rather too pedantic a cast -- though one hopes that over-simplification is more informative than over-complication.)

A QUICK LOOK AT TIME-SHARING

System:

Time-sharing is a system which allows a number of users to make use of a computer "at the same time" for independent tasks. The technique is possible because of the large mismatch between computer speeds and human reaction times. Although the computer is actually sharing its attention among all of its users, it can be made to appear to each user as if he had control of the machine in its entirety. The program which regulates the process of co-ordinating activities (the CTSS Supervisor -- or "the system") resides in a separate bank of core memory, and actually causes the various users' programs to be brought into a second bank of memory from other storage devices.

Interaction:

Each user really has physical control only over some remote input-output terminal, usually a typewriter-like device ("console"). He issues basic instructions, called "commands", to the system by typing the name of the command and the arguments associated with it; the system will then bring in the program which is associated with (i.e., "which performs") the command and cause it to be executed. In general, the user types in lower case and the system's responses are in upper case (except on teletype devices, which operate only in upper case). When the system receives a command it acknowledges receipt by typing out on the user's console a line comprising the letter "W" (for Wait) followed by a five-digit number expressing the time of day. When the command has finished working, the system informs the user of this fact by typing a line comprising the letter "R" (for Ready) followed by two numbers separated by a plus sign, the first number expressing the number of seconds expended in executing the command and the second number expressing the number of seconds expended "swapping" the program(s) involved in and out of core.

The user is said to be at "command level" after receiving an "R" (Ready) line. When at command level, he may issue any system command desired. During the execution of a program, however, commands are not accepted; in particular, as commands themselves are programs they can not be over-ridden by the typing of new commands. In order to return to command level before the executing program has finished, then, the user must give a "quit signal" to the system. This quit signal is two pushes of the console's "break" button. The ability to quit is quite useful, especially when, for example, a user's program misbehaves or a command has furnished enough information for one's purposes but would continue to operate "longer" if not interrupted.

Files:

Most often, the arguments of commands are the names of "files" where a file is broadly defined as a logical set of information. A file may contain ("the information may represent") a source program, an object program, a set of data, text, lists, or almost anything definable by the user which is expressable in the symbols available. These files may be input from CTSS consoles or from punched cards (see Section AE.1) and are normally stored on the computer's magnetic disk storage devices; however, their actual location is of no importance to the programmer since he always refers to files by name. The system itself provides for references to actual locations internally and maintains a separate "file directory" for each user so that no conflicts arise in the assigning of names.

A master file directory (M.F.D.) is maintained, containing information about the location and contents of the several user file directories (U.F.D.). Each U.F.D.

contains information about the location and contents of the various files which the user has created. The U.F.D. is associated with a problem number and a programmer number. Also associated with certain problem numbers are "common files" -- file directories which contain files of common interest and are directly accessible to all users on the problem number.
Certain of the common files associated with the system programmers' problem number (M1416) contain information of general utility and are accessible to all users. (See Section AD for further information about files.)

Each file is required to have two names, a "primary" name and a "secondary" name, each of which consists of six or fewer characters. The primary name is almost always arbitrary and should have some mnemonic importance. The secondary name may or may not be arbitrary, depending on the contents of the file and the way in which they are to be used. For example, a file containing a MAD (Michigan Algorithm Decoder) source program may have the arbitrary primary name PROG1, but must have the secondary (class) name "MAD". Object program files have the secondary name "BSS" (Binary Symbolic Subroutine).

Applications:

Learning to use CTSS is similar to learning to play the guitar. Knowledge of a few basic chords enables the novice musician to play a rather large number of songs; knowledge of a few basic commands enables the novice CTSS user to write and execute an arbitrarily large number of programs in a rather large number of programming languages (Section AH.2). Beyond this basic area of application (which is the only one dealt with in detail here), however, are at least two other large areas of special application. In the first place, there exists a large number of special-purpose commands for such purposes as file manipulation, debugging, documentation, and interactive problem-solving (Section AH). In the second place, user programs may avail themselves of a wealth of library subroutines, both batch-processing and time-sharing in nature (Section AG). By taking advantage of these additional tools, the CTSS user may expand his repertoire of applications as necessary, and probably more rapidly than the guitar player expands his repertoire of songs.

Reference:

Further general information of interest may be found in Sections AA.0 and AA.1. Information about the use of the manual may be found in Section AB.

OVERVIEW OF BASIC CONSOLE TECHNIQUES USED IN PROGRAM CREATION

1. Typing errors in command lines and in input lines may be corrected by typing a commercial at sign (@) to cause the system to ignore ("kill") the entire line thus far, or by typing one or more sharp signs (#) to cause the system to ignore ("erase") one or more immediately preceding characters.

2. A console session is begun (after turning on and dialing in the console) by issuing the LOGIN command, identifying the user to the system and establishing that a line to the computer is available.

3. Source programs will be entered and modified or corrected using the text editing command EDL. (Other available editing commands are covered in Section AH.3.)

4. Compilation will be accomplished by the MAD command in this document -- although other compilers and assemblers are available in CTSS (Section AH.2.)

5. Once a program has been successfully compiled (or assembled), execution is effected by the LOADGO command-again for purposes of this documentation; loading of programs is covered generally in Section AH.7.01.

6. When a program has been satisfactorily run, it may be removed from the user's file directory by use of the DELETE command. Files not explicitly deleted will be left alone and will still reside in the disc storage units.

7. At the end of a console session, the LOGOUT command is given to inform the system that the user's line to the computer is free to accomodate someone else.

DESCRIPTION AND DISCUSSION OF COMMANDS

The LOGIN Command:

After the console has been turned on and dialed in, type a line of the following general form:

"login probno name",

where probno is an argument of the LOGIN command specifying the user's assigned problem number, and the second argument is the user's last name. _Commands and arguments must be separated by at least one blank_ ("_space_"). The system will respond with a W(ait) line, and then will type out "PASSWORD". At this point, the user must type his assigned private password, during which time the console's printing will be suppressed. Provided a line is available -- and the user has been allotted time and disk storage records on the system -- a message acknowledging the fact that the user has been "logged in" will follow. (Further details may be found in Section AH.1.01).

When there is no line available, the system will cause the console to be "hung up" (disconnect at the system's end of the connection), and the user should try to log in at a later time. If , on the other hand, no response is typed after the login command was given, CTSS is not in operation; information about when it is expected to be back in operation may be gotten from data-phone ext. 1300 (recorded message), or if the recorded message has not yet been updated, from the computer operator at MIT ext. 4127. Occasionally the system will not recognize "login" as a command; this means that the name of the login command has been temporarily altered so that the system programming staff can hold a test session.

The EDL Command for Input:

1. EDL is a CTSS command which is used for input and for "context editing" of files. We will take advantage of its input facility, to create the files which will later be edited. "Context editing" requires the unique specification and location of a line in terms of its contents by means of appropriate requests ("subcommands" of EDL) before the line can be edited. (This rather obscurely-stated point should be made clear by discussion below -- EDL for Editing, point 5 -- and by the Appendix). Requests to EDL may be abbreviated by their first letter, with the exception of the request "file" (see point 8), although the full request name may also be used; the abbreviated forms will be used herein.

2. A more complete description of the EDL command may be found in Sections AH.3.07 and AH.9.01.

3. To begin input: type, e.g., "edl abc123 madCR" or, in general, "edl name1 name2CR", where CR indicates Carriage Return.

4. Response: FILE ABC123 MAD NOT FOUND.

Input

"Input" is one mode of the EDL command; "Edit", the command's other mode, is discussed below.

5. For all lines which do not begin with statement labels, strike the "Tab" key, then type the line. For lines with labels: type the label, then strike Tab and type the rest of the line. For MAD continuation card indicators: tab, backspace, indicator, line. When finished with a line, strike Carriage Return (CR).

   N.B. Wherever CR is indicated, strike the appropriate key on the console; do not type the letters "CR".

6. To deal with typing errors while still working on the line in which they occur: The sharp sign (#) serves as an erase character and causes the ignoring of the immediately preceding character; more than one erase character may be used (e.g., XXY##YZ will be treated as XYZ by the computer). To kill the entire current input line, strike the at-sign (@). N.B. This also deletes tabs, e.g., "/tab/x=a+b@y/tab/x=a1*b" causes "y/tab/x=a1*b" to be treated as the input line. A kill character cannot be erased.

7. For typing errors discovered in prior input lines: follow the procedures discussed below under the EDL command for editing (beginning with point 4).

8. To file a program: strike an extra CR (i.e., CR after last line plus CR for an "empty" line); this action causes entry to the Edit mode in which the request "file" may be used. Response will be a system R(eady) line, and the user will be at "command level" again -- which he was not while using EDL. (It is important to distinguish between general system commands, on the one hand, and requests to a specific command, on the other.) A file named, e.g., "abc123 mad" will have been established in the user's file directory.

   N.B. EDL WILL ACCEPT REQUESTS IN THE EDIT MODE ONLY; in the Input mode, all material typed is treated as input.

9. To verify input (optional): type (general form) "print name1 name2". The PRINT command will cause the file to be typed back on the console.

<u>The MAD Command for Compilation</u>:

1. To cause the MAD (or the appropriate language's) compiler to operate on a program: the command is the name of the language and the argument is the primary name of the file; e.g., "mad abc123". The secondary name of the source file must be "MAD".

2. Response from successful attempt: A line beginning "LENGTH," followed by various other information. A file named , e.g., "abc123 bss" will have been created.

3. Error messages: These indicate "syntactic" mistakes; the source program file must be appropriately corrected.

4. Further details may be found in Section AH.2.10.

<u>The EDL Command for Editing</u>:
(CR Indicates Strike Carriage Return)

The following is excerpted from Section AH.9.01:

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.

The Appendix and the discussion below should clarify the importance of the "pointer". Requests which take arguments must be separated from the arguments by a space.

1. Type "edl name1 name2CR" (general form).

2. Response should be "Edit".

3. The EDL command will type back lines ("verify" them) after certain requests. The requests which will cause verification are "locate" and "change" (discussed below); wait for the response before issuing another request when one of these two has been given.

4. Type "tCR" ("t" is the abbreviation for "top"). (This is not strictly necessary for beginning to edit, but is required when the Edit mode has been entered from the Input mode, or when the pointer

must be moved "upwards".) The "pointer" is positioned "above" the first line of the file. Note that "top" is the <u>only</u> request to EDL which moves the pointer "upwards".

5. To position the pointer to a particular line, use "l" (for "locate"). The argument of this request (typed after a space which must follow the "l") is a string of characters which uniquely specifies a line amongst the lines "below" the pointer. The pointer will be moved to the line which contains the <u>first</u> occurrence of the string. E.g., if the "top" request had just been issued and the first two lines of a file were

```
A = B+C
D = A+X
```

the request "l aCR" would position the pointer at the first line, but "l a+CR" would have positioned it at the second line. (Note also that in the latter case the first line is then "above" the pointer, and if it is to be operated upon, the "t" request - "l" request sequence must be given again.)

6. To replace an entire line, the request is "r" (for "retype"). The argument (typed after a space which must follow the "r") is the entire new line itself (with appropriate tabs and terminal CR, as in Input). This request does not move the pointer.

7. To change a portion of a line, the request is "c" (for "change"). The argument (space as usual) is rather complex: Begin with an arbitrary character which does not appear in either the original string of characters to be changed or the new string ("q" is frequently useful); this character serves as a delimiter of the two strings. Between the delimiters, type the old and the new character strings, in that order. The first occurrence of the old string will be altered. For example, "c qabcqxyzqCR" will cause "abc" to be replaced by "xyz", and if the original line were "abcabc" the resulting line would be "xyzabc". <u>Blanks within the strings are significant</u>: "a bc" is not the same as "abc". (This request does not move the pointer.) "Global" changes are possible, but will not be dealt with here.

8. To insert one line after the line currently pointed at, type "i", followed by a space, followed by the line to be inserted. To insert

several lines, change mode from Edit to Input by giving an "extra" CR, or by typing "iCR". The response will be "Input". Type the line or lines, with appropriate tabs. When done inserting, return to Edit mode by giving an extra CR.

9. To delete a line or lines: position the pointer (with the "l" request) to the first line to be deleted, then type "d" (for "delete") followed by CR if only this one line is to be deleted, or by a space and a number (expressing the number of <u>consecutive</u> lines to be deleted) if more than one, then CR. (This request leaves the pointer positioned at the last line deleted.)

10. To move the pointer "downward" one or more lines, the request is "n" (for "next"); it takes a numerical argument, in the same fashion as "d".

11. To re-file under the original file name, simply type "fileCR" (<u>from the Edit mode</u>). This process replaces the older version with the edited version.

12. To file under a new file name, type "file xxxxxxCR" where xxxxxx represents the new primary name. This process preserves the older version, in the event that a comparison of both versions is desired for some reason (e.g., to determine which of two methods takes longer). Secondary names may not be changed when filing.

<u>The LOADGO Command for Execution</u>:

1. After a successful compilation or assembly (no syntactical errors) has been achieved, the command "loadgo name1" will cause the object program ("name1 bss") to be loaded and executed. Library search occurs during the loading process.

2. Shortly after the customary W(ait) response, the word "EXECUTION" will be typed by the system. This will be followed by the program's results, if all has gone well with the program. Then, provided there were no execution errors, an end-of-run message and a system R(eady) line will be typed out.

3. Further details may be found in Section AH.7.01.

<u>Program Logic "Debugging"</u>:

1. Wrong results imply errors in program logic. (See

CC Memo 182 for a list of common programming errors.)

2. When discovered, the errors can be corrected in the source file (name1 mad, e.g.,) with the EDL command.

3. After editing, the program must be recompiled with the appropriate language command (MAD).

4. The new program is executed with the LOADGO command.

5. If the results are still wrong, back to 1....

Housekeeping:

When a program is no longer desired, all files relating to it can be removed from the disk by typing (general form) "delete name1 *CR". The asterisk indicates to the DELETE command that it is to operate on all files with primary name "name1". Of course, individual files may be dealt with by "delete name1 name2". (Further details may be found in Section AH.6.03).

The LOGOUT Command:

At the end of a console session, give the command "logout". The system will respond with the present time, the date, and the total time used (in minutes). (Further details may be found in Section AH.1.02.)

APPENDIX: CONSOLE FAMILIARIZATION SESSION -- AN ANNOTATED SCRIPT

Introduction:

The program created in this script is deliberately simple-minded, so as not to distract from the basic point at issue -- console usage. (For demonstration purposes, some of the errors introduced are unique to the MAD language, but should be reasonably clear to the reader even if he is not familiar with MAD). The program is intended merely to compute and output the square root of the sum, and the product, of two numbers input from the console. (Data can, of course, be input to the program from files as well as from the console. Indeed, batch processing tape techniques are simulated on CTSS -see Section AG.5 -- and numerous subroutines are provided for direct disk file I/O -- see Section AG.2.)

Instructions:

1. Type the lines appearing in lower-case letters and wait for the system responses if a line in upper-case occurs next in the "script".

2. Hit Carriage Return at the end of each lower-case line.

3. Circled numbers to the left of the page refer to the Notes, which follow the "script".

4. Long-hand insertions are typing instructions (usually involving the Tab key) e.g., Tab .

5. The numbers in W(ait) and R(eady) lines are fictitious; expect different ones.

6. Before issuing the DELETE command, the LISTF command may be used to get a listing of the contents of your file directory (Section AH.5.01), and TTPEEK may be used to get a table of your time and track usage for the current month (Section AH.1.04). Neither command requires arguments.

Script:

```
login m1416 padlipsky
W 1315.1
Password
 STANDBY LINE HAS BEEN ASSIGNED
 M1416  3711 LOGGED IN  10/22/69 1315.6 FROM 800289
 LAST LOGOUT WAS  10/19/69 2247.1 FROM 800315
```

```
      CTSS BEING USED IS  MIT8A3
     R 6.783+.000

     edl simple mad
     W 1316.4
      FILE SIMPLE   MAD NOT FOUND.
     Input
     normal mode is integer
(1)  floating point a
          Tab  print comment$numbers,pleeuhz####ase$
          Tab  read data
          Tab  a=sqrt(b+c)
          Tab  d=bc
     end of a        Tab  end of program
(2)
     Edit
     t
     l mode
     NORMAL MODE IS INTEGER
(3)  r    Tab  normal mode is integer
     n
(4)  r    Tab  floating point a,d
     l a
(5)            PRINT COMMENT$NUMBERS,PLEASE$
     l a=
               A=SQRT(B+C)

(6)  c qtqt.q
               A=SQRT.(B+C)
     l d=
               D=BC
     i
     Input
(7)       Tab  print results a,d
          Tab  execute exit.

(8)
     Edit
     file
     *
     R 5.833+4.250
(9)
     print simple mad
     W  1321.3

     SIMPLE    MAD      01/10   1321.4

               NORMAL MODE IS INTEGER
               FLOATING POINT A,D
               PRINT COMMENT$NUMBERS,PLEASE$
               READ DATA
               A=SQRT. (B+C)
               D=BC
```

```
                PRINT RESULTS A,D
                EXECUTE EXIT.
                END OF PROGRAM
        R .616+416

        mad simple
        W 1321.9
         THE FOLLOWING NAMES HAVE OCCURRED ONLY ONCE IN THIS PROGRAM.
         THEY WILL ALL BE ASSIGNED TO THE SAME LOCATION, AND
         COMPILATION WILL CONTINUE.
               BC
               B
               C
        LENGTH 00072.  TV SIZE 00006.  ENTRY 00016
        R 2.766+.533

        edl simple mad
        W 1322.8
        Edit
        l bc
                 D=BC
(10)--  c qbqb*q
                 D=B*C
        file
        *
        R 3.516+1.450

        mad simple
        W 1323.7
        LENGTH 00071.  TV SIZE 00006.  ENTRY 00015
        R 2.216+.750

        loadgo simple
        W 1324.1
        EXECUTION.
        NUMBERS,PLEASE
        b=7,c=2*

(11)--
                 A = 2.707999E 26,   D = 14.000000
          EXIT CALLED. PM MAY BE TAKEN.
        R 6.166+1.050

        edl simple mad
        W 1325.5
        Edit
        l mode
(12)--           NORMAL MODE IS INTEGER
        d
        l read
                 READ DATA
(13)--  i Tab    whenever (b+c).l.0.,transfer to tag
        l exit
```

```
              EXECUTE EXIT.
c qqtag2q
TAG2          EXECUTE EXIT.
i
Input
tag   Tab     print comment$negative argument$
      Tab     transfer tc tag2
14
Edit
file
*
R 2.950+3.150

mad simple
W 1523.1
LENGTH 00107.  TV SIZE 00006.  ENTRY 00020
R 2.966+.900

loadgo simple
W 1523.7
EXECUTION.
15 NUMBERS,PLEASE
b=7.,c=2.*
16            A =    3.000000,      D =  14.000000
  EXIT CALLED. PM MAY BE TAKEN.
R 6.566+1.083

loadgo simple
W 1524.7
EXECUTION.
NUMBERS,PLEASE
b=-7.,c=,2.*
17 NEGATIVE ARGUMENT
  EXIT CALLED.  PM MAY BE TAKEN.
R 6.216+.816

delete simple *
W 1526.1
R 1.716+366

logut
W 1528.2
18 'LOGUT' NCT FOUND.
R .000+.C83

19 logout m1416#####
W 1528.4
 M1416  3711 LOGGED OUT 10/22/69 1536.3 FROM 800289
 TOTAL TIME USED =    .7 MIN.
```

Notes:

1. We decide pleeuhz isn't funny and use four erase characters.
2. The "empty line" takes us to Edit mode.
3. The line should have been tab'ed originally.
4. Same as 3, and we realize we want both answers floating.
5. These two locates demonstrate "context editing".
6. We remember that MAD subroutine calls require periods.
7. This insertion allows us to see the answers after execution and terminates the program in standard fashion.
8. The "empty line" again.
9. Verifying the typing.
10. The error message reminded us that we meant BC to be a product, not a name.
11. Whoops! A is 'way wrong. We have a bug.
12. We remember that the square root routine expects floating point arguments, and take the easiest route of getting them to be floating -- deleting the integer mode declaration.
13. We remember that the square root routine also expects positive arguments.
14. Still another "empty line".
15. N.B. the decimal points.
16. Success.
17. And success again.
18. The misspelled command is not findable.
19. Erase characters apply in command lines too.

(END)

Identification

Fixed File Names

Introduction

Unexpected file names appear in a user's file directory from time to time. The following is a partial annotated list of files generated:

1) by CTSS in performing system duties;
2) by one of the commands which makes a new file as part of its execution process; or
3) by another CTSS user.

Note, and be warned, that catastrophic conflicts will arise if several users are performing a command which generates a fixed file name at the same time in the same file directory (usually a common file). The obvious way to avoid such conflicts is to avoid performing such commands while attached to any directory other than one's "home directory".

Files With Both Names Fixed

C.O.D.	E.O.V.	AH.2.10:
M.T.X.	O.V.F.	AH.2.10:

These files are used for intermediate data by the MAD command.

(COMBI NFILE) AH.6.01:

This name is given to the intermediate file employed by the COMBIN command. It may be deleted if found.

MAIL BOX AH.9.05:

This file is created (or appended to) when a user gives the command MAIL with the problem number and program number of the addressee's file directory. When the recipient subsequently logs in the following message will appear on his console:

YOU HAVE MAIL BOX

(MOVIE TABLE) AH.7.01, AJ.8.01:

The MOVIE TABLE is created by the standard loaders. It is a temporary mode file and represents a map of the programs loaded.

OUTPUT RQUEST AH.6.06:

The RQUEST command for bulk I/O creates or appends to a file in the user's disk storage. When the file has been processed by the disk editor, it is set to temporary mode.

PERMIT   FILE   AH.3.05:

The PERMIT command establishes a line-marked file of private protected mode in the user's directory; PERMIT FILE contains information used in the linking process.

URGENT   MAIL   AH.1.01,
URGENT   POST   AH.1.01:

DAEMON can create this file in a user's directory so that his subsequent LOGIN will remind him TO PRINT the new file in order to get a message from the system. The alert message printed on his console is:

YOU HAVE URGENT MAIL
or
YOU HAVE URGENT POST

USER   PROFIL   AH.2.19:

This file is used by the '.' command to store the abbreviations and lists of SAVED files.

(BUG)   SAVED   AH.8.08:

This file is used by DEBUG to save the current core image when executing CTSS commands from within the program.

Files With Fixed Second Names

NAME1   ASCII   AH.3.09, AH.3.10:

EDA or QED (with the 'wa' instruction) creates a file with secondary name 'ASCII'. The ROFF command expects a file with the secondary name 'ASCII'.

NAME1   BCD   AH.2.07, AH.2.10,
AH.2.11:

A file of secondary name 'BCD' is produced by several of the language processors on request. Such files contain assembly/compilation listings; they are generated in response to the argument '(LIST)' in the language processor command.

NAME1 BSS e.g., AH.2.07, AH.2.10, AH.2.11:

A 'BSS' file contains an object program, produced by one of the language processors. 'BSS' is a 7094 term, documented elsewhere.

NAME1 (DUMP) AJ.8.03:

For details, see the DUMPER SAVED write-up.

NAME1 (MEMO) AH.9.01, AJ.6.01:

TYPSET creates a file with the secondary name '(MEMO)'. RUNOFF expects a file with the secondary name '(MEMO)'.

NAME1 RUNCOM AH.10.01:

A file of secondary name 'RUNCOM' or 'BCD' may be used to define a procedure consisting of a number of CTSS commands. These files may be executed at the console with the RUNCOM command or under FIB.

NAME1 RUNOFF AH.9.01, AH.9.06:

This file is created when using the 'PRINT' option with either the RUNOFF or ROFF commands. It contains the formatted version of the (MEMO) or ASCII file as it would normally appear on the console but it suitable form for offline printing via the RQASCI command.

NAME1 SAVED AH.3.03:

'SAVED' files contain machine conditions and core-images, for subsequent execution. For details, see the SAVE write-up.

progno SAVED AH.3.09, AH.10.03:

(Where progno is the user's programmer number.) This file is created by serveral commands (including SAVE and QED) and contains the user's machine conditions and core-image for later RESUMEing or CONTINeing.

progL SAVED AH.1.02:

(The user's programmer number followed by the letter "L".) At any time an automatic LOGOUT may be initiated by the system. The file may be RESUMEd at a later time.

NAME1 SQZBSS AH.4.04:

'SQZBSS' files contain compressed-form BSS "decks". For details, see the write-up on PADBSS SAVED and SQZBSS SAVED.

NAME1 SYMTAB AH.2.10, AH.2.11:

This is an optional file containing a symbol table, produced by the MAD (and, of course, MADTRN) language processor in response to the '(SYMB)' argument.

NAME1 SYMTB AH.2.07:

This is an automatically-generated file containing a symbol table, produced by the FAP language processor.

Files With Special or Fixed First Names

```
FAPBCD   progno
FAPBSS   progno
FAPSYM   progno
FAPTEM   progno    AH.2.07:
```

(where "progno" is the user's programmer number) These files are used by the FAP command in the assembling of the user's program.

```
(INPUT   progno    AH.9.01,
(INPT1   progno    AH.9.01:
```

These two names are used for intermediate files by TYPSET, ED, and EDL. Following a quit sequence (or an automatic LOGOUT) either one of these files may be found. It may be renamed and used as a source file (in the automatic LOGOUT case, editing may, of course, be continued when the prognoL SAVED file is resumed). When invoked, the editing commands will announce the presence of one of the intermediate files (if one is present); the user must either type 'yes' to the question about deleting it, or type 'no' and then RENAME it. The commands will not proceed unless the intermediate file is disposed of, one way or another.

probno progno AH.4.01:

This is an intermediate file used by the ARCHIV command (where probno, progno are a user's problem number and programmer number). For details, see the ARCHIV write-up.

.TAPE. 3 AG.5.01:

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

.TAPE.   n   AG.5.01:

The .TAPWR, (STH) and (STHM) subroutines create or append to a pseudo-tape line-marked file named .TAPE.   n, where n is specified in the calling program.

...xxx   SAVED   AG.8.02:

This is an intermediate file used in chaining commands. For details, see the SCHAIN write-up.

...00n   SAVED   AH.3.04:

This is an intermediate file used in chaining commands. For details, see the RUNCOM write-up.

(END)

Identification

Conventions of this manual

This CTSS Programmer's Guide will be divided into sections on a functional basis. The naming of the sections will be of the format MS.X.YY.

M	is the manual designation. Since the CTSS Programmer's Guide for the IBM 7094 is the first manual in a series, its designation will be "A".
S	is an alphabetic major section designation, e.g., this is section "B".
X	is the one or two digit subsection designation. This first publication will have subsections numbered from 1 to 13. Note that they will not be designated as 01 to 13.
YY	is the minor subsection designation. This is a two digit numeric designation (00,01,02....)

The manual was prepared by the CTSS commands QED and ROFF where each section is a separate file of the name MSXYY ASCII. Note the deletion of periods within the file name.

Users may request copies of complete manuals or any section thereof from the Information Processing Center's publications office. Or, at the user's convenience copies may be ROFFed on the user's 1050 or 2741 Selectric console or Model 37 Teletype. All of the files are linkable through file directory M1416 3212.

The table of contents will be maintained in two forms.

1) TABLE ASCII which may be ROFFed to produce the current table of contents in the form distributed with the manual (i.e., in sectional or functional order). The first line of TABLE will be dated to indicate the date of the latest change to the manual. Any revisions of the manual will be noted by date beside the section which was modified.

2) DATTOC ASCII which may be ROFFed to produce a table of contents in reverse chronological order of section modification. This will show rapidly the latest changes to the manual by section and date.

Within the text of the manual, areas of modifications will be noted by an asterisk or bar in the right hand margin. This will be done only on one level of revision, that is, the flags of any earlier revision will be removed before the later modifications are made.

Because the manual will be done as much as possible with the current limited character set and as little hand work as possible by the typist, the following conventions will be used.

1) The symbols designating "less than", "greater than", "less than or equal to", and "greater than or equal to", will be replaced by the MAD conventions of .L., .G., .LE., and .GE.

2) Octal notation is expressed as the octal number enclosed in parentheses, followed by an 8, e.g. (7777)8.

3) Exponentiation is expressed in the MAD notation of .P. (e.g., 2.P.9).

4) Optional arguments in calling sequences to subroutines will be enclosed within minus signs (e.g., -P&E BUFF-). This applies also to arguments to commands (e.g., -NAME2-).

5) Indication for a literal within a subroutine calling sequence will be typed in lower case and be enclosed within single quotation marks (e.g. 'j'). This means that the actual value should be used, rather than the location of the value.

6) Some command arguments must be literal values and these will be shown as uppercase characters enclosed in single quotation marks (e.g., 'REV'). This means that no substitution is possible, but the actual characters shown must be used.

(END)

Identification

Glossary and Conventions

Documentation Conventions

Within calling sequences, arguments written in upper case denote the location of a variable. Arguments in lower case denote the value itself. If literals are used, they are noted as such by the conventions of the language or as lower case letters enclosed in single quotation marks. Minus signs around an argument mean that argument is optional.

There are three possible kinds of calling sequences for subroutines. The statement "as supervisor entry:" means that the user must supply the TIA as noted beside the TSX. The statement "as supervisor or library entry:" means that the user may supply the TIA as noted, or he may use the external library name noted in the TSX in which case the library will supply the TIA. The statement "as library subroutine:" means that the subroutine is an external library routine. A MAD or Fortran calling sequence will usually be given but the routine may also be called by the equivalent FAP calling sequence.

Glossary

*	in front of an entry in the table of contents, indicates the new I/O system. An * in the right-hand margin, indicates a modification to the write-up.
AC	36-bit signed accumulator.
b	denotes a required blank in a character string.
C.R.	carriage return.
Console	In general, the word console means a typewriter console (e.g., 1050, 2741, teletype) rather than a special display console (e.g., ESL scope).
Current	File Directory is the file directory to which the user is currently switched. It is usually the user's file directory but may be switched to a common file directory by COMFIL or to another user's file directory by ATTACH.
External	Routines are subprograms (with entry points) which are called by other subprograms. The library entries and library subroutines are external routines. The FAP calling sequences

	give the entry point name. The FAP convention for calling external routines is: 1) EXTERN pseudo-op specification, or 2) preceding the name by $, or 3) CALL pseudo-op. All the FAP calling sequences in this documentation assume EXTERN specification so that the CALL and $ are not shown.
Fence	is a magic number used to designate the end of a variable-length string of parameters. The fence referred to in this documentation is a word of all octal sevens.
FILNAM	is used in calling sequences to indicate the initial location of 2 BCD words containing the name of a disk file (right justified and blank padded). In Fortran programs, FILNAM may be set by the subroutine SETNAM or it may be the file name in H specification form. In MAD programs FILNAM may be set in a Vector Values statement.
FMT	or FORMAT is used in calling sequences to indicate the beginning location of a format or a location containing a pointer to the beginning of the format, if SETFMT is used.
Library Entry	- The majority of the required TIA's for the supervisor entries have been placed in the library as library entries.
Line-Marked	Files are files composed of variable length records. Each logical record is preceded by a word containing binary ones in bit positions 0-17 and the number of words to follow in bits 18-35.
Line-Numbered	Files are files composed of 14 word logical records. Characters 73-80 are a sequence field (the leftmost 3-6 may be alphabetic and the rightmost 2-5 must be numeric).
LIST	is used in calling sequences to provide a list of parameters to the subroutine being called. It usually specifies parameters for input or output. A list may consist of a combination of single variables, dimensioned or subscripted variables, or block notation as described in the MAD manuals. In Fortran, the implied DO may be used only in I/O statements, not in calls to subroutines. In MAD, a LIST might be: A, B(1)...B(10), C(N)...C(1), G(J). The notation D(N)...N,

E(1)...10, is also available; this form in general is acceptable only to I/O system entries or associated library routines.

In FAP, a PZE prefix may be used with the location of a single variable.

The FAP equivalent of the above MAD LIST is:

```
TXH  A
TIX  B-1,,B-10
TIX  C-'n',,C-1
TXH  G-'j'

TIX  D-'n',,N
TIX  E-1,,L(10)     i.e., location of a 10
```

Memory bound or allotment is the number of core registers available to the program, counting register 0. Therefore, the first unavailable register is equal to the memory allotment, except in the special case of (77777)8 when the entire 32,768 words of memory are meant.

MODE With the previous file system, files could be one of four modes:

0. TEMPORARY - words are deleted as they are being read or skipped over.
1. PERMANENT - can be read or altered indefinitely.
2. READ-ONLY (class 1) - can be read but not altered until the mode is changed.
3. READ-ONLY (class 2) - can be read but not altered except by a control card submitted to the dispatcher.

With the current file system there are seven possible modes and the mode of a single file can be any combination of the seven, some of which are not meaningful.

000.	PERMANENT
001.	TEMPORARY
002.	SECONDARY
004.	READ-ONLY
010.	WRITE-ONLY
020.	PRIVATE
100.	PROTECTED

NAME1 NAME2 are used in calling sequences to indicate the actual name of a disk file. NAME2 is the secondary (class) name. The

	actual names are right adjusted, blank padded, BCD words.
String	Files - files having no logical record breaks. Processed as strings of words by externally specified word counts.
Supervisor	Entry - supervisor routines which reside in A core can be entered only by a special calling sequence convention.

```
                TSX  ROUTIN,4
                     ARGS
                       .
                       .
                       .
                       .
         ROUTIN TIA  =HROUTIN
```

If the name of the routine contains fewer than six characters, the BCD word referred to in the TIA must be left adjusted and blank padded. The TIA's for many of the entries have been placed in the library as library entries in order to save the user the inconvenience of supplying the TIA, and to allow for tracing supervisor entries if the standard debugging aids are used.

(END)

Identification

System Documentation

"Documentation", in the sense of assembly/compilation listings, of CTSS' supervisor, commands, and library subroutines can be made available to users interested in the fine details of system implementation. From the on-line source language files maintained by the system programmers, document tapes for off-line printing are prepared periodically. Although system listings are internal documentation of work by the system's group, there is a desire to make the system as widely understood as possible. For this reason, system listings are normally made available to those who indicate their interest. Users desiring to study large areas of the system (e.g., "the library") may request printing of the relevant document tape; the consultants will explain the details of the requesting procedure. Because these procedures are expensive of both machine and system programmers' time, casual requests for listings should be avoided.

Users desiring to study only a small area of the system (e.g., the SQRT subroutine) will probably not want the entire contents of document tape; to satisfy this type of need, the consultants will have listings of at least the library available for browsing.

(END)

Identification

Equipment Configuration

The primary terminals used with CTSS are modified Model 35 Teletypes, Model 37 Teletypes, and IBM 1050 and 2741 Selectric teletypewriters (adaptations of the "golfball" office typewriter). These terminals are located mostly, but not exclusively, within the M.I.T. campus. Several demonstrations have been conducted from such places as Europe, California, and South America. In addition, CTSS supports up to three ARDS storage tube display terminals via 1200 bit/second phone connections. Access may also be gained from the Telex or TWX' telegraph networks.

Although Teletypes and other typewriter-like terminals are adequate for most purposes, some applications demand a much more flexible form of graphical communication. The CTSS configuration includes for this purpose a multiple-display system developed by the M.I.T. Electronic Systems Laboratory for research in computer aided design. The system includes two oscilloscope displays with character and line generators and light pens, connected to a PDP-7 computer which maintains the display and performs such functions as rotation and translation. The PDP-7 commumnicates with the 7094 via the direct-data channel. The two displays can be operated independently of each other. Communication with the computer can be achieved by means of the light pen, and also through a variety of other devices (knobs, swithces, push buttons), as well as the normal typewriter terminal. The meaning of a signal from any of these inputs is entirely under program control. Because of cable length requirements, the display must be in a room adjacent to the 7094 installation; remote operation would require improved data transmission facilities.

All of these terminals can operate simultaneously by time-sharing the 7094 central processor. In order to assure reasonably prompt response, the maximum number of users is generally limited to about 30; however, this number is under control of the supervisory program, and is adjusted on the basis of system loading: CTSS has on occasion serviced as many as 38 normal users simultaneously.

The IBM 7094 central processor has been modified to operate with two 32,768-word banks of core memory and to provide facilities for memory protection and relocation. These features, together with an interrupt clock and a special operating mode (in which input-output operations and certain other instructions result in traps), were necessary to assure successful operation of independent programs coexisting in core memory. One of the memory banks is available to the users' programs; the other is reserved for the time-sharing system supervisory program. The second bank was added to avoid imposing severe memory restrictions on users because of the large supervisor program and to permit use of existing utility programs (compilers,etc.), many of which require all or most of a memory bank.

The central processor is equipped with six data channels, two of which are used as interfaces to conventional peripheral equipment such as magnetic tapes, printers, card readers, and card punches. A third data channel provides direct-data connection to terminals that require high-rate transfer of data, such as the special display system.

The fourth data channel provides communication with two disk units (IBM 2302) and a low speed drum (IBM 7320). The theoretical storage capacity of the disks is 76 million computer words and the capacity of the drum is 186,400 words. The time required to transfer 32K words in or out of core is approximately one second for both the disk and the drum.

The fifth data channel provides communication with two high speed drums (IBM 7320A). The capacity of a 7320A is the same as that of the 7320 but the transmission time for 32K words is one-quarter second.

The transmission control unit (IBM 7750) consists of a stored-program computer which serves as an interface between the sixth data channel and up to 112 communication terminals capable of telegraph-rate operation (up to 200 bits per second). An appropriate number of these terminals are connected by trunk lines to the M.I.T. private branch exchange and to the TWX' and Telex networks. Higher rate terminals can be readily substituted for groups of these low-rate terminals; for instance, to support ARDS terminals at high speed (on output), three 1200 bit/second terminals are installed. All of these terminals are compatible with Bell System data sets. Part of the core memory of the transmission control unit is used as output buffer, because the supervisor program and its necessary buffer space have grown in size to the point of occupying all of the A bank of core memory.

(END)

Identification

Clocks

Purpose

The CTSS IBM 7094 has an interval timer clock available as well as Chronolog clock. The interval timer clock is completely under control of the supervisor; its action is as follows: location 5, memory A, is incremented in the units position every 1/60 sec; whenever it overflows, an interrupt occurs which, if the clock is enabled, causes a trap to location 7 and the instruction location counter to be stored in location 6. The interval timer clock is more completely described in IBM Manual L22-6554.

The supervisor uses this clock both for interrupting programs and for time accounting. Base-time and day-of-the-month information are obtained from the Chronolog clock which is attached as a pseudo tape unit. The supervisor can also simulate the interrupt clock behavior for each user. By supervisor calls, the user can program for nested interrupts and computation time readings.

(END)

Identification

CTSS Character Set

Purpose

Two character sets, one a subset of the other, are standard on CTSS. The smaller set (the 6-bit or BCD set) is basically the 7094 standard BCD set of 6-bit character codes including 47 characters and blank, and augmented with four console control functions. (Carriage return, tabulate, form feed, and colon, which is used by some programs as a logical "backspace" character.) The larger set (the 12-bit or Full set) consists of 111 graphic and control characters, represented as 7-bit codes right-adjusted in a 12-bit field. This larger set includes both upper and lower case letters and a variety of special characters and console control functions.

Twelve-to-six bit mapping

All input from consoles is treated initially as 12-bit codes by the CTSS supervisor. These 12-bit codes will, however, normally be mapped into the six-bit subset by the supervisor unless special action is taken by the user program to prevent the mapping. Supervisor calls (SETBCD and SETFUL) are available for turning on and off the mapping.

In the CTSS Character Set table below, the 6-bit subset is contained in the upper half of the table. When a character from the lower half of the table appears in an input stream, it is mapped according to the following rules:

1. Characters in the table enclosed in parentheses are discarded.
2. All other characters except commercial at, number sign, question mark, and double quote are truncated to six bits by discarding the left six bits.
3. Number sign (#) is the "erase" character: the previous character is discarded. Double quote (") is also an "erase" character.
4. Commercial at (@) is the "kill" character: the entire line is discarded. Question mark (?) is also a "kill" character.

To simplify the job of a program which wishes to do its own 12-to-6 bit mapping, the supervisor on input inserts a flag bit (the fourth from the left) on those codes which are to be discarded upon mapping. For example, the 12-bit code for the percent sign, according to the table, is:

```
            000001000101          (0105 octal)
```

When using the RDFLXA supervisor call, the code which will be received by a user program will be:

000101000101 (0505 octal)

since this character is discarded when mapping to six-bit mode. The flag bit is optional on output characters. For example, to type out a percent sign, either code 0105 or 0505 is acceptable.

Device Code Tables

No one device is capable of input or output of the complete CTSS character set. For each device, a table is provided which lists the exceptions. In most cases, these tables indicate one of two mapping rules for exceptional characters. These rules are:

1. The character is discarded on output, or
2. The character prints as some graphic different from standard.

The fact that a different graphic is attached to a given code does not of course, imply that the code will be interpreted differently by the computer. This latter comment must be kept in mind when using a 1050 or 2741 console, which may have any of several slightly different sets of key caps and/or printing balls.

On the Model 35 Teletype and the Telex, the upper and lower case letters are mapped together as in the following example:

1. On input, a typed letter "A" will always produce the code for upper case "A", 0021.
2. On output, the code for lower case "A", 0121, will type an upper case A.

Character Code Tables

Unassigned positions in the CTSS character set table are reserved for future expansion. At present, these unassigned characters are discarded on output. In the individual device code tables, a lack of an entry implies that the corresponding entry in the CTSS character set table applies. The entry "ig" means that this character code is ignored on output to this device.

All codes are given in octal.

Abbreviations used in the character set tables:

ig	-	Ignored (see comment above)
WRU	-	Who are you
P-off	-	Printer off
P-on	-	Printer on
V.T.	-	Vertical tab
N.L.	-	New line (Carriage return and Line feed)
L.F.	-	Line feed
F.F.	-	Form feed
tab	-	Horizontal tabulation
hang	-	data phone disconnect
sngl	-	Single space carriage on return
dbl	-	Double space carriage on return
L.K.	-	Lock keyboard
U.K.	-	Unlock keyboard
back	-	Back space
BRS	-	Black ribbon shift
RRS	-	Red ribbon shift
CR$\cancel{L}\cancel{F}$	-	Carriage return without line feed
A.M.	-	Alternate mode
HLF	-	Half-line forward feed
HLR	-	Half-line reverse feed
ESC	-	Escape
ACK	-	Acknowledge
NAK	-	Negative acknowledge

CTSS Character Set

	0	1	2	3	4	5	6	7
0000	0	1	2	3	4	5	6	7
0010	8	9		=	'			
0020	+	A	B	C	D	E	F	G
0030	H	I		.	)	:		
0040	-	J	K	L	M	N	Ø	P
0050	Q	R	F.F.	$	*	N.L.		null
0060	blank	/	S	T	U	V	W	X
0070	Y	Z	tab	,	(			
0100	(|)	(])	(\)	(;)	(#)	(%)	(@)	(L.F.)
0110	(HLF)	(HLR)	(^)	(bell)	(!)	(WRU)	(hang)	(P-off)
0120	&	a	b	c	d	e	f	g
0130	h	i	(BRS)	(RRS)	(~)	back	(CR/LF)	"
0140	(_)	j	k	l	m	n	o	p
0150	q	r	(<)	([)	(ESC)	(>)	?	
0160	'	(L.K.)	s	t	u	v	w	x
0170	y	z	(V.T.)	({)	(})	(P-on)	(U.K.)	(A.M.)

NOTES:

1. Character codes in parentheses are discarded on input in 6-bit mode. In 12-bit mode these characters have (400)8 added to them, as a flag bit.
2. Character codes 0137 (double quote) and 0104 (number sign) are the erase characters in 6-bit mode.
3. Character codes 0156 (question mark) and 0106 (at sign) are the kill characters in 6-bit mode.
4. The codes 0017 (Interrupt), 0057 (Quit) and 0077 (Hang-up) on input are intercepted by the supervisor and are never sent through to the program.

Model 37 Teletype Character Set

Same as CTSS Character Set except as noted below:

	0	1	2	3	4	5	6	7
0000								
0010								
0020								
0030								
0040								
0050								
0060								
0070								
0100								
0110								
0120								
0130								
0140								
0150								
0160		(NAK)						
0170							(ACK)	ig

NOTES:

1. On early model 37's, codes 0107 (line feed), 0110 (HLF) and 0111 (HLR) are ignored on output.
2. Code 0107 (line feed) cannot be input.
3. Code 0117 (Printer-off) cannot be input.
4. Code 0175 (Printer-on) cannot be input.
5. Code 0017 (Interrupt) can be generated by one push of the "interrupt" button.
6. Code 0057 (Quit) can be generated by two pushes of the "interrupt" button.

1050/2741 Character Set

Same as CTSS Character Set except as noted below:

	0	1	2	3	4	5	6	7
0000								
0010								
0020								
0030								
0040								
0050			ig					
0060								
0070								
0100		ig	(¢)					
0110	ig	ig	(¬)	ig		ig		
0120								
0130					ig		ig	
0140								
0150				ig	(prefix)			
0160	ig	ig						
0170			ig	ig	ig		ig	ig

NOTES:

1. Interrupt and Quit signals are generated by the "Attn" key on 2741's and the "Reset Line" button on 1050's
2. Code 0107 (line feed) cannot be input from a 2741.
3. Code 0154 (prefix) cannot be input from a 2741.
4. Code 0117 (printer off) cannot be input.
5. Code 0132 (black ribbon shift) cannot be input.
6. Code 0133 (red ribbon shift) cannot be input.
7. Code 0175 (printer on) cannot be input.

Standard Model 35 Character Set

Same as CTSS Character Set except as noted below:

	0	1	2	3	4	5	6	7
0000								
0010								
0020								
0030								
0040								
0050								
0060								
0070								
0100								
0110	ig	ig	ig					
0120		A	B	C	D	E	F	G
0130	H	I	ig	ig	ig	ig		
0140		J	K	L	M	N	Ø	P
0150	Q	R			ig			
0160	ig		S	T	U	V	W	X
0170	Y	Z		ig	ig			

NOTES:

1. Some outside (i.e. not new-style MIT-modified) model 35's will not respond to code 0176 (Keyboard unlock).
2. On outside model 35's, code 0055 (Carriage return) will cause a Carriage Return _and_ a Line Feed on output. The computer will type a line feed whenever a carriage return is detected on input.
3. On outside model 35's, the tabulate character (0072) prints as a back slash and will not cause tab motion of the carriage.
4. Interrupt and Quit signals are generated by the "Break" button.

Telex Character Set

Same as CTSS Character Set except as noted below:

	0	1	2	3	4	5	6	7
0000								
0010				:				
0020	&							
0030						bell		
0040								
0050			ig		#			
0060								
0070			;					
0100	ig	ig	ig			ig	ig	
0110	ig	ig	ig		ig		ig	ig
0120		A	B	C	D	E	F	G
0130	H	I	ig	ig	ig	ig		
0140	ig	J	K	L	M	N	Ø	P
0150	Q	R	ig	ig	ig	ig		
0160	ig	ig	S	T	U	V	W	X
0170	Y	Z	ig	ig	ig	ig	ig	ig

NOTE:

1. Code 0115 (Who Are You) prints a Maltese Cross.
2. Either code 0035 or code 0113 will ring the Telex bell on output. On input, a bell produces code 0035.
3. Either code 0072 or code 0103 will print a semicolon on output. On input, a semicolon produces code 0072.
4. Either code 0020 or code 0120 will print an ampersand on output. On input, an ampersand produces code 0020.
5. Either code 0054 or code 0104 will print a number sign on output. On input, a number sign produces code 0054.

(END)

Identification

Special console characters

Purpose

When working at the console, there are several significant signals or characters which the user finds necessary. The "break character" is necessary to signal the end of a line so that the supervisor knows that it is time to analyze the line to determine whether or not action is required. The "interrupt signal" is useful for the user to signal his program that the pre-planned branching within the program should now be followed. This might be analogous to sense switch interruption during batch processing. The "Quit signal" signal is used to stop the current program (by placing it in dormant status) and return the user to the command level. The "erase character" is interpreted before the line is processed by the supervisor and it causes the immediately preceding character to be erased by moving the character pointer or counter back one. The "line-kill character" is also interpreted before the line is processed by the supervisor and it causes the deletion of the current line.

Break Character

The break character is a carriage return. Whenever a user types into his console, regardless of whether or not his program is running, the input character is received by the supervisor within 200 ms. The input character is added to the user's input message and if it is not a break character, no further action is taken. If the character is a break character, the message is called complete and one of several actions results.

If the user was at command level (i.e., the user was in dead or dormant status), he is placed in waiting command status. If the user's program was in input-wait status, it is returned to working status so that it may resume by reading the input message. If the user's program was already in working status, the message is merely considered early and is left in the buffer for subsequent reading by his program. (If early messages continue to arrive and the input buffer area becomes nearly filled, a message is typed out to the user requesting that he stop typing until his previous input is read.)

Quit and Interrupt Signals

When a program is first initiated or placed in working status it is said to be at interrupt level 0. This applies to both commands and user programs. The program continues execution until it terminates by entering dead or dormant status or until the user transmits the QUIT signal which places the program in dormant status immediately. This manual QUIT signal allows the user to change his mind, correct mistakes, etc.

Interrupt signals may be used by the user to externally direct or control certain pre-planned phases of his programs execution. These interrupt breakpoints may be recursively stacked to a maximum depth of 3. Whenever a console interrupt signal is received by the supervisor, control is returned (by means of a push down list) to the entry previously assigned. Interrupts are dealt with within a user's program by means of subroutines SETBRK, GETBRK, and SAVBRK (AG.6.03).

The interrupt signal is generated when the interrupt key is pushed once (ATTN on 2741, RESET LINE on 1050, BREAK on model 35, INTERRUPT on model 37). The quit signal is generated by pushing the button twice within two seconds.

Erase and Kill Characters

A console operating at command level is automatically set to the normal mode or 6-bit BCD code. (A program call to the supervisor is necessary in order to change to the 12-bit typing mode). While inputting in the normal mode, two special characters are recognized before the message is sent to the supervisor. The characters " (quote) and # (number sign) are interpreted as a single character eraser. This is accomplished by moving the character pointer back one space instead of forward, within the current line or message. Therefore, n quotes or number signs will erase n characters (not counting the quotes themselves as characters) back to, but not including, the previous carriage return or break character. The ? (question mark) and the @ (commercial at) are interpreted as a line-delete signal. The entire message back to the previous break character is erased.

(END)

Identification

Data phone extensions

Consoles may be connected with the 7094 via telephone lines through the data switch. Because of the differences in transmission rates between various types of consoles, there are several classes of lines:

1050/2741	Dial '0'
35ASR/KSR	Dial '9'
37KSR	Dial '1371'
ARDS display	Dial '1601'

All of these numbers are 'hunt groups', i.e. they cause the telephone exchange to search over a number of lines until one is found which is not busy.

Consoles have specific (although not necessarily unique) identification codes. These codes are to be used with the attached remote console supervisor entries; they are also checked by LOGIN for unit group restricted users. The console ID word consists of a type code (2 for 1050, 3 for TELEX, 4 for TWX', 5 for inktronic and 33KSR, 6 for 35ASR/KSR, 7 for 37KSR, 8 for 2741, 9 for ARDS), two to four BCD zeroes, and one to three BCD characters of identification, for a total of six characters.

Each data phone used with a console has a unique extension number which may be used for voice transmission. A data phone may be called from another data phone by dialing the 4-digit extension number, or from an MIT extension by dialing 818 followed by the data phone number. Note that data phone extensions are not regular MIT extensions.

If your console or data phone needs service, call MIT Ext. 4128, giving name, room number, console type, and nature of the trouble. The appropriate repairman will be notified; the record of the trouble is kept until the repair is made and reported back by the serviceman.

A recorded message giving the current status of CTSS is available at data phone ext. 1300. If an abnormal system comedown (crash) occurs, and CTSS will be down for more than 10 minutes, the operator will update the recording indicating expected comeup time and the nature of the trouble.

(END)

Identification

Historic file system

Purpose

The IBM 1301 disk served as the bulk storage for the time sharing system so that users files, system files and sub-system files could be quickly and randomly dumped and read. It was extremely important to have a flexible but efficient and usable central module which would handle all the disk input and output for all users. The following ideas were incorporated in the disk control subroutine which was used for about a year and a half. In August of 1965, the old disk control subroutine was replaced by a new module which incorporated many improvements, but also allowed for much upward compatibility for the old system. The old system will, therefore, be described here because of all the routines and write-ups which are still using the compatibility features.

Considerations

The following considerations went into the make-up of the file system and they might help in the understanding of the system.

1. The user should be able to write and maintain permanent programs and data files on the disk.
2. System and subsystem programs should be permanently recorded on the disk.
3. The user should have only symbolic reference to his files.
4. The user should be able to read and write many files simultaneously.
5. The user should not be able to reference any files not authorized to him.
6. The user should be able to initiate files in different modes such as temporary, permanent, or read-only.
7. In order to utilize the maximum storage capacity of the disk file the format of a single record per track should be used.

Protection

During time-sharing, all systems and users make use of the single standard input/output package. If a system does not use the standard routines, it can be run by itself with the disk inoperative or if it needs the disk, the contents of the disk can be dumped and later reloaded when time-sharing is restarted. During time-sharing, the standard package makes use of input/output trapping and memory protection to insure protection of user's programs and files. The user has access only to files which are authorized to him.

A further protection against loss of files is the operational procedure of dumping the contents of the disk files periodically onto tape. These dump tapes can be used by a retrieval program to reload the disk completely or selectively. These history tapes are kept on file by operations according to a schedule which is approximately: daily tapes for a week, weekly tapes for 4 months and yearly tapes forever. In case of a major unrecoverable catastrophe the entire system may be backed-up 24 hours by reloading the most recent dump tape. The user may recover any of his individual files from any of the tapes which contain them.

File Structure

Each user is assigned one or more tracks to serve as a directory of all his private files currently stored on the disk. A user does not have access to any other user's file directory. A group of users who may be working on the same problem may be assigned an extra set of file directories (called common files) to which all the users of the group have access.

The old system had two severe limitations: first, only one user could be working in a file directory at any one time, and second, that a reference to a single file could exist only in a single file directory. These limitations meant that in order to share routines or data, users had to copy files into and out of common files, so that there were multiple copies of the same file. Furthermore, whenever one user was using a common file, no one else had access to it. These limitations have been much alleviated with the new system.

The file directories contain the two BCD word names, the number of tracks used, the starting track address pointer, the date-last-used, and the mode of each file. A master file directory is maintained which contains a pointer to the file directory of each user in the system. A track usage table is also maintained which tells the system which tracks are already used and which are free. All the tracks of a single file are chained together by virtue of the first word of each track either pointing to the next track in this file or to the last word of this track if there are no more tracks. Whenever possible, the tracks for one file are assigned consecutively, in order to reduce the time lost in seeking. When the disk is reloaded from the dump tapes, the housekeeping is done to provide consecutive tracks for files which might previously have been scattered.

Usage

All files are referred to by a two word BCD name and no absolute track locations are known or needed. All calling sequences to the disk routines provide the facility of allowing the user to specify his own error procedure or accept the standard system error procedure. All of the calls and error procedures are described in section AG of this manual. Almost all of these calls will have write-around routines for the new I/O system so that they will behave in much the same way as they did before April 1965. Note that in the table of contents of this manual, the sections which refer to the new I/O system are preceded by an *.

(END)

Identification

The new file structure and Input/Output system

Purpose

The new file system was implemented, 1) in order to continue the basic philosophy of the previous file system and remove many of the weaknesses which had become evident in its years of exercise and 2) to provide and exercise a prototype of the file system which is proposed for the next time sharing system.

Some improvements to be found in the new system will be mentioned here, and it is assumed that the reader is familiar with the previous file system discussed in section AD.1. The I/O system can accomodate any configuration of I/O channels and/or devices and thereby provide a standard interface to all users. The back-up feature, of having files dumped onto tapes which can be saved for retrieval, will be accomplished by a DAEMON which is in constant operation during time sharing. In this way the amount of information which is dumped and the amount of time lost due to back-up will be greatly reduced. The I/O system can now deal with entries in file directories which are pointers (LINKs) to entries in other file directories rather than to the files themselves. This means that a user may permit other users to use any of his files without actually copying the desired files into other directories. Thus, many users may be referencing files within the same directory, simultaneously. Indeed, many users may be reading the same file. A lock does exist so that no one may reference a file which another user is altering. A further improvement is an increase in the number of modes which files may have. Additional entries have been added to the I/O system to allow the administrators to update the master file directory during time sharing operation so that new users can be placed in the system more quickly. The I/O system is modular for all machine dependent sections. By replacement of certain modules, different strategies for particular I/O devices, or I/O devices themselves, may be changed without affecting the overall I/O structure.

Structure of the I/O System

The I/O system presents a standard machine independent interface to all users. All calls to the I/O system are directed to the basic control module of the system called the File Coordinator. The File Coordinator then requests service from the Buffer Control Module, which in turn may request service from a particular Strategy Module. Attach Strategy Module is concerned only with a certain class of information storage. The Strategy Module may in turn request service from an I/O Adapter. The I/O Adapter is a module which processes input and output requests for specific I/O devices. All calls to the I/O system requesting input or output must follow this path of control, the File Coordinator- the Buffer Control Module- a Strategy Module- an I/O Adapter.

The File Coordinator:

The File Coordinator provides the interface between the file system and the user. It interprets the calling sequences, performs validity checking of the calls, and calls the appropriate module.

The Buffer Control Module:

The Buffer Control Module is called by the File Coordinator. Its functions are to maintain the user's active file status table parameters, to convert the user's calling sequences to appropriate I/O commands for the stategy modules, and to move the data words between the buffers and the user's data storage area. The Buffer Control Module in turn calls the appropriate Strategy Module when I/O is needed.

The Strategy Modules:

Each Strategy Module is responsible for a particular storage device. This module determines the strategy to be used in dealing with this storage device and its associated I/O Adapter. Requests are stacked in queues to be executed by the I/O adapter whenever the associated channel becomes free. In addition, the Strategy Module is responsible for keeping track of the number of available units of secondary storage for the device to which it is assigned. Requests are made to the Strategy Modules only through the Buffer Control Module.

The I/O Adapters:

The I/O Adapter is responsible for the operation of the hardware interface to a particular device or devices. The I/O adapter accepts requests for service from the Strategy Modules only. The I/O adapters are responsible for processing all traps associated with the devices to which they are assigned. The I/O adapters interrupt the appropriate Strategy Modules upon completion of previous requests.

Operation of the Buffer Control Module

The buffer control module (BCM) is called by the file coordinator and its function is twofold: 1) maintain the users active file status table parameters of file length, reading and writing status and pointers, buffer status and pending I/O, and 2) convert the user's calling sequence into appropriate calls to the I/O adapter for physical records and move data between the buffers and the user's data area on a word basis.

Whenever possible, data is moved directly from the I/O device into the user's data area without going through a buffer. In the general case, however, a buffer must be supplied for intermediate storage for those parts of the data which do not comprise a complete physical record on the I/O device. Some users may wish to devise more sophisticated I/O control when the system efficiency is considered unsatisfactory, so the following conditions are noted where files may be dealt with without providing a buffer. For example, a multiple buffers system may be built in the user's program without extra buffering by the system.

Reading without a buffer:

> If blocks of integral number of physical records are read or if reading goes through the end of file, no buffer will be used even if one is assigned.
>
> If no buffer is assigned and partial records are called for, the physical record will be read for each call in order to extract the logical or partial record from the physical.

Writing without a buffer:

> A complete new file of any length can be written by a single call without a buffer being assigned.
>
> An existing file may be written into without a buffer only from the beginning of a physical record through the end of a physical record or through the end of a file.
>
> Appending to a file or writing partial records requires a buffer.

Truncation without a buffer:

> Truncation without a buffer can only be accomplished if the truncation word is beyond the end of file or in front of the first word (file made empty).
>
> The BCM selects an appropriate strategy depending on whether a buffer has been assigned or not and returns an error if a buffer is mandatory where none was assigned. A user may switch a file from "no-buffer" mode to "buffer" mode or vice-versa by calls to BUFFER.

File Notation and Structure

The smallest piece of information which can be manipulated by the I/O system is an element. A file is an ordered sequence of elements. The file is the largest amount of information which can be manipulated by the I/O system.

Every file will have a unique name which is used to identify that file to the user. An element in a file is referenced by specifying the file name and the linear index. For example, the element "i" in file "a" is referred to as a(i). Files may be created, modified or destroyed by a CTSS program only through the use of the I/O system.

A file appears to the user to be a block of contiguous storage which may be referenced through normal sequential addressing conventions. However, the physical structure of the file is independent of the logical structure which the user experiences. The user may refer to a file only through the symbolic file name and should have no notion of where or how the file is stored. The number of elements which make up a file is arbitrary, and in fact a file may exist with no elements.

There are four basic operations for manipulating elements within files: opening, closing, reading and writing. To initiate a read and/or write operation, the file must first be opened for reading and/or writing. To terminate the reading and/or writing of a file, the file must be closed.

Modes:

A characteristic of every file is its mode. The mode of a file is specified by a 7-bit mask at the time it is created. (The mode may be changed later if desired.) Each bit in the mask indicates a different property of the file, and any combination of properties may be specified. The properties and the (octal) mask bit positions are shown below.

000. PERMANENT- If all bits in the mode mask are zero, the file can be read or written, and will be stored indefinitely.

001. TEMPORARY- Such a file will automatically be deleted the first time it is read. The deletion will not take place until the file is closed after reading.

002. SECONDARY- This property appears in directory entries for files which have been deleted by storage collection mechanisms. The entry is retained for purposes of identification.

004. READ-ONLY- The file can only be read. An attempt to write into or delete a file of this property will cause an error condition.

010. WRITE-ONLY- The file can only be appended to. An attempt to read from or delete a file with this property will cause an error condition.

020. PRIVATE- The file can only be referenced by the AUTHOR i.e. the user who created or last modified this file. An attempt to delete a file of this property will cause an error condition.

040. Unused mode bit.

100. PROTECTED- The mode of the file may only be changed by the AUTHOR of the file. Any attempt by another user to change the mode of this file will result in an error condition. A 'PROTECTED' file may not be renamed nor deleted, even by the AUTHOR.

File Directories:

The File Coordinator may service requests from a fixed number of active users. Requests from a specific user are in the form a(i), to reference the element "i" in the user's file "a". The File Coordinator however, manipulates information by use of an implicit address of the form c(b(a(i))). This address references the element "i" in the file "a", which is specified by the file "b", which in turn is specified by the file "c". The file "c" in this case is a specific Master File Directory and the file "b" is a specific User File Directory. After establishing a "c" and "b" pair, each successive call for a(i) will then be interpreted by the I/O system as c(b(a(i))), until another call is given specifing a new "c" or "b". By treating the user file directories and the master file directories as normal information files, multiple usage of single files can be accomplished in a general manner.

The formats of the Master File Directory and the User File Directories are shown on the next page. The groups of words 1-7 actually begin in the fourth word of the file and are repeated in the groups of seven for each entry in the file.

An entry in which both of the first two words are zero, means that an entry has been deleted.

The dates are of the format: bits S,1-8 contain the year -400 modulo 500, bits 9-12 contain the month, bits 13-17 contain the day, and bits 18-35 contain the number of seconds elapsed since midnight.

The AUTHOR is the programmer number of the user who created or last modified the file. The F is a 3-bit integer which specifies on which secondary storage device the file resides. If F is 0, the entry refers to a linked file. F is used by the Buffer Control Module to determine which strategy module should be called.

RCOUNT specifies the number of elements contained in a physical record of the file. NOREC specifies the number of physical records contained in the file. LCOUNT specifies the number of elements contained in the last physical record of the file. The highest element address in a file may be defined as (NORECS-1) * RCOUNT + LCOUNT. The 3-bit integer P is normally one. However, P=0 is equivalent to P=1.

ILOCK is used to allow multiple users to access the same file simultaneously. If a file is in read status, ILOCK contains a count of the number of users currently reading from that file. When the number of users reading from the file drops to zero, any user who wishes to modify that file will be allowed to proceed. When a file is opened for writing, the high order bit of ILOCK is set to 1. During the time that ILCCK indicates that a modification to a file is in progress, no new users will be allowed to reference that file.

If user "A" wishes to reference a file contained in some other user's file directory (user "B"), he can accomplish this by means of a "LINKED" file. A LINKED file is defined in a user's file directory as a file with a device specification of zero (F=0).

If a file in a user's file directory is a LINKED file (F=0), RCOUNT, NORECS and ILOCK are ignored. The problem and the programmer number of the user to which the link is made are in words 3 and 4. The name of the file being linked to is in words 6 and 7. A file may be linked in this manner through the file directories of several users. The depth of linkage is currently restricted to 2. The last entry must be a normal file directory entry which defines the file in a normal manner. Once this linking operation is completed, the file will be treated as a normal file. This operation will be repeated every time a user attempts to open a LINKED file.

The user may refer to his file directory as a file of the name "U.F.D. (FILE)" which is defined in his file directory as a normal file in READ-ONLY mode. The Master File Directory is defined as a User File Directory by the name "M.F.D. (FILE)" in the Master File Directory. This file is also referred to as "U.F.D. (FILE)" within the Master File Directory. To read the Master File Directory, first, ATTACH. ($M.F.D.$,$(FILE)$). The I/O system will never allow the Master File Directory or any User File Directory to be deleted.

MASTER FILE DIRECTORY, "M.F.D. (FILE)"

WORD .... CONTENTS ....

1. USER PROBLEM NUMBER (36 BITS)
2. USER PROGRAMMER NUMBER (36 BITS)
3. DATE AND TIME any file in U.F.D. LAST MODIFIED (36 BITS)
4. DATE LAST USED (18 BITS), AUTHOR (18 BITS)
5. --- (8 BITS), --- (10 BITS), F (3 BITS), RCOUNT (15 BITS)
6. --- (3 BITS), NORECS (15 BITS), P (3 BITS), LCOUNT (15 BITS)
7. The next "P" words contain specific information for a file of type "F".

USER FILE DIRECTORY, "U.F.D. (FILE)"

WORD .... CONTENTS ....

1. FILE NAME, PART 1 (36 BITS)
2. FILE NAME, PART 2 (36 BITS)
3. DATE AND TIME LAST MODIFIED (36 BITS)
4. DATE LAST USED (18 BITS), AUTHOR (18 BITS)
5. MODE (8 BITS), ILOCK (10 BITS), F (3 BITS), RCOUNT (15 BITS)
6. --- (3 BITS), NORECS (15 BITS), P (3 BITS), LCOUNT (15 BITS)
7. The next "P" words contain specific information for a file of type "F".

2302 Disk and 7320 Drum Strategy

The file directory entry for a 2302 or 7320 file contains pointers to the first and last tracks. For a file of this type, RCOUNT will be the number of data words in a single track. NORECS will be the total number of tracks in the file and LCOUNT will be the number of data words in the last track.

Each track in a file of this type will contain chain address pointers to the following and preceding tracks. In addition each track will contain a label in the following form:

```
I__LCOUNT__I__TRAKNO__I
S                    35
```

TRAKNO is a track sequence number. LCOUNT will be non-zero only in the last track of a file and will contain the count of the number of data words in that track. This count must match the value of LCOUNT in the user file directory for that file.

Tracks are assigned in a manner similar to that described in memo CC-196 (Disk Control Routine). All track usage tables will be files contained as entries in the Master File Directory. The file which defines the usage of disk tracks will be referred to as "DISKUT (FILE)". The track usage file for the 7320 drum will be referred to as "DRUMUT (FILE)".

2302 Disk and 7320 Drum I/O Adapter

The disk/drum Strategy Modules provide calls to the disk/drum I/O adapter specifying only logical track addresses. The I/O adapter is responsible for determining the actual channels which must be used. The adapter places all requests into a request queue and returns. The trap processor for the disk/drum I/O adapter empties the request queue on completion of previous requests for that channel. If a request is made requiring a channel not already in operation, a trap will be simulated for that channel.

Tape Strategy Module

Magnetic tapes will be treated as secondary storage in the same manner as disks or drums. Many files can be recorded on a single tape, but a single file may not consist of more than one tape. The first physical file of a tape file will be a BCD header label (see Section AG.5.05).

In a file directory entry for a tape file, RCOUNT will be 432 and B will be one. The seventh word of the file directory entry will contain an internal tape address known to the I/O supervisory systems only; this word contains a logical unit number and a file number. Other information in the file directory entry has the same meaning as described in the disk and drum Strategy Modules.

Each data record will contain 432 information words preceded by a control word in the following form.

```
PZE RECNO,,LCOUNT
```

RECNO will be the record sequence number. LCOUNT will be non-zero only in the last record of a file and will be the count of the number of words in that record. This word count must match the value of LCOUNT in the file directory entry for that file.

The I/O adapter for the tape Strategy Module will operate on request queues in the same manner as the disk and drum I/O adapters.

To use Tape Strategy, a user must have an 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

Note three things in particular about this I/O system. First, it is basically not a buffered system so that upon return from RDFILE or WRFILE it is safe to assume that the I/O has not actually been done yet. Before the specified data area may be referenced, a call to FCHECK and a "finished" return must be made. In other words, before a satisfactory delay has been made by FCHECK, the input data is not really there or the output data has not yet been transmitted so the user may not rewrite the data area. The second thing of note is that if an error return is specified, some errors are detected immediately and some are not detected until the next I/O call. Each RDFILE or WRFILE serves as an FCHECK on the preceding RDFILE or WRFILE on the same file. The third thing to note is that all of the I/O is considered to be by relative locations so that all files can be considered to be similar to addressable storage.

Calling Conventions:

Following is a list of calls to the new file system. The detailed write-ups of these calls can be found in section AG and in the table of contents their sections will be preceded by an *. Their calling sequences are given in MAD notation and the MAD compiler has been modified slightly to accept an integer or an integer-variable specifying the number of words in block notation rather than the last address of a block. The new file system is consistent in expecting the number of words rather than the last address in block notation. All arrays are stored forward so that the beginning address must be the lowest core location of the array. Also, all file names are specified by the locations of both BCD names rather than the location of the first name as FILNAM is used in the old file system. The file names are right adjusted and blank padded. For example:

```
     MAD:          FSTATE. ($ NAME1$, $ NAME2$,A(8)...8)
     FAP:          TSX FSTATE,4
                   TXH =H NAME1
                   TXH =H NAME2
                   TIX  A,,EIGHT      or    TXH  A,,8
                    .
                    .
                    .
           EIGHT   PZE  8
               A   BSS  8
```

In all of the calls, if an argument is not pertinent, a -0 may be specified (FAP: PTH = -0). All calls will accept two more arguments than shown. The first is the location of users' error return and the second, if supplied, specifies the location into which the error code will be stored.

Some of the arguments and information items are of special forms which might be noted here.

DEVICE =	1.	is low speed drum
	2.	is disk
	3.	is tape
File status =	1.	is inactive
	2.	is open for reading
	3.	is open for writing
	4.	is open for reading and writing

SUMMARY

Administrative and Privileged:

```
     UPDMFD.($  PROB$, $  PROG$)
     DELMFD.($  PROB$, $  PROG$)
     ATTACH.($  PROB$, $  PROG$)
     MOVFIL.($ NAME1$,$ NAME2$,$  PROB$,$  PROG$)
     SETFIL.($ NAME1$,$ NAME2$,DAYTIM,DATELU,MODE,DEVICE)
     LINK.($NAME1$,$NAME2$,$PROBN$,$PROG$, $NMI$ , $NM2$ , MODE )
     UNLINK.($ NAME1$,$ NAME2$)
     ALLOT.(DEVICE, ALLOT , USED )
     RSFILE.($ NAME1$,$ NAME2$)
```

Reading and Writing:

```
     OPEN.($STATUS$,$ NAME1$,$ NAME2$,-MODE-,-DEVICE-)
     BUFFER.($ NAME1$,$ NAME2$,BUFF(432)...432)
     RDFILE.($ NAME1$,$ NAME2$,RELLOC,A(N)...N,-EOF-,-EOFCT-)
     RDWAIT.($ NAME1$,$ NAME2$,RELLOC,A(N)...N,-EOF-,-EOFCT-)
     WRFILE.($ NAME1$,$ NAME2$,RELLOC,A(N)...N,-EOF-,-EOFCT-)
     WRWAIT.($ NAME1$.$ NAME2$,RELLOC,A(N)...N,-EOF-,-EOFCT-)
     TRFILE.($ NAME1$,$ NAME2$,RELLOC)
     FCHECK.($ NAME1$,$ NAME2$,FINISH)
     FWAIT.($ NAME1$,$ NAME2$)
     CLOSE.($ NAME1$,$ NAME2$)
```

others:

```
     UPDATE.
     SETPRI.(PRIOR)
     RESETF.
     CHFILE.($ NAME1$,$ NAME2$, NMODE , $NEWNM1$ , $NEWNM2$ )
     DELFIL.($ NAME1$,$ NAME2$)
     FSTATE.($ NAME1$,$ NAME2$,A(8)...8)
     STORGE.(DEVICE, ALLOT , USED )
     MOUNT.( CHAN ,UNIT,MESSAG(20)...20)
     UMOUNT.(UNITNO,MESSAG(20)...20)
     VERIFY.(UNITNO,LABEL(4)...4)
     LABEL.(UNITNO,LABEL(4)...4)
     TAPFIL.($ NAME1$,$ NAME2$,UNITNO,FILENO)
     IODIAG.(A(7)...7)
     TILOCK.(RETRN)
     FERRTN.(RETRN)
     ATTNAM (A(2)...2)
```

(END)

Identification

Library files

Organization

Library files are created by COMBINing BSS files into files which may then be searched for missing routines by the relocating loaders. Any user may create his own library files and, by use of the special arguments, direct the loader to search his library files instead of (or in addition to) the CTSS system library files. Subsystems of CTSS (e.g., AED) may have their own libraries and their own loaders. However, the ones being discussed here are the CTSS system library and loaders.

The system library is currently divided into files which reside in the system common file directory. TSLIB1 contains all of the standard routines described as library subroutines and library entries in this manual. The loader will normally search TSLIB1 for missing routines unless prohibited by special arguments. TSLIB2 contains the debugging subroutines and core-B transfer commands. The loader will search TSLIB2 automatically only when a core-B transfer command has been given. If the debugging routines are to be loaded with the program before execution the loader should be informed by (SYS) TSLIB2 or, for example, more completely by (NEED) FLEXPM (SYS) TSLIB2. A special library in the system file is KLULIB which contains subroutines for the "KLUDGE" (i.e., ESL scope console) and which may be searched if special arguments are given to the loader.

The library files may be improved by any user by following the maintenance procedure described in section AB.3. The library is maintained by the programming staff at the Computation Center.

(END)

Identification

Common Files and the Public File

Purpose

This section describes the nature of, and submission procedure for, programs in the "Public File"--a file directory accessible to all users of CTSS. To furnish perspective, the evolution of common files and the Public File is also discussed.

Development of Common Files

Within the former file system, a given file could be referenced from only one file directory and only one user could be attached to a file directory. In practice, a group of users could be working on one problem and, therefore, have need to access a common pool of programs and data. This conflict was partially resolved by implementing the concept of common files, where "common" implies some sort of "joint ownership". A group of users working on the same problem was assigned a single problem number. Each problem number could then have associated with it as many as four common file directories. Any user could switch from his own file directory to one of the common file directories associated with his problem number. With appropriate calls to the supervisor a user could copy any of his files into the common files or copy files from any of the common files into his own directory. Some restrictions still existed, namely, only one user could operate in a common file directory at any one time; to avoid locking users out of a common file, files had to be copied and, therefore, many copies of the same file existed; also, common files were rigidly associated with a problem number and therefore communication between problem numbers was impossible. (The current treatment of common files is covered in Sections AG.3.03 and AH.6.04.)

Development of System Files

The four common files associated with the system programmers' problem number took on the special function of servicing all users, regardless of problem number. Their common file 4 became known as the Public File and any user could put files there and copy files from there. In order to housekeep the system files, the Disk Editor, which was run at least once a day, deleted all files in the Public File which were in temporary or permanent mode. Only a system programmer could change a file in Public to the old file system's R1 or R2 mode (approximately Read-only and Read-only Protected). A further restriction was placed on the Public File, namely, only programs which were adequately documented could remain in Public. The documentation was available from the consultants. The system programmers' common file 2 became known as the System File, common file S, and any user could copy files from there. Common file S contained the binary files of all the commands and the BSS files of the libraries. The system programmers' common file 1 contained the source and binary files of the supervisor and common file 3 contained listing files of the supervisor.

Current Contents of the Public File

The Public File (M1416 CMFL04) is a file directory with a track quota of zero, the contents of which are available to all users. It contains nothing but linkage pointers to files which exist in other file directories. There are several reasons why these pointers must be placed in a Public File: 1) The Public File now also fulfills the role formerly played by the System File; hence, certain files must be made available through it to the programs which need them. For example, system libraries, TSLIBn BSS, are needed by the loaders. The actual BSS files reside in one of the other M1416 common files (accessible by system programmers only) but loaders can read them through the links in the Public File. 2) Many commands and their data files are maintained by their authors rather than by the programming staff. These command and data files may reside in the authors' file directories but are made available to all users of the system through links in the Public File. 3) Users have progams which are of general interest and usefulness but which have not been given command status. These programs are made available to all users through the links in the Public File.

Users' Programs

A major advantage of a time-sharing system stems from the ability it offers for users to share software as well as hardware. This "talent-sharing" can easily go far beyond the power offered by the range of compilers and library routines made available by batch-processing system programs; in some sense, every progaram on the disk could be thought of as a "system program". To facilitate exchange of users' programs, be they subroutines (for the documentation of which Section AI is reserved) or "commands" (the SAVED files which are documented in Section AJ), the Public File was instituted. Inclusion of a program in the Public File both guarantees its accessibility to all users and, indeed, publicizes its existence to all (studious) readers of the Programmer's Guide. However, inclusion of a program in the Public File also implies a degree of sanction by the administrators of the system. Because of this "sanction", then, programs which are submitted for inclusion in the Public File cannot automatically be accepted. Both the nature of the program and its documentation must be evaluated. To this end, the following submission procedure has been developed.

Submission Procedure

When a candidate for inclusion in the Public Files has been debugged, the author should send its documentation to the editor of this manual. There are two parts to the documentation. First, a typed (or TYPSET) write-up is required, in the general format of a section of this manual, with the following additions: The section on Purpose should be as extensive as possible, with emphasis on the areas of applicability of the program. If the program is fully documented elsewhere (e.g., MAC and/or CC memo), a full reference should be given. Examples of usage are extremely desirable. Second, information as to the directory and name(s) of the file(s) involved must be given, along with the names (and phone numbers) of at least two users, other than the author, who have used the program and who recommend its inclusion in the Public File. After favorable evaluation, the implementation considerations below apply, and the message of the day and the next set of revisions to the manual will herald the new arrival.

Implementation

One of the system programmers will LINK to the file containing the new Public program from his (M1416) common file 4. The author must, of course, have PERMITed the file to M1416 *. The system programmer, in turn, will PERMIT the link to all users. The mode of the link (the entry in the Public File) will normally be Read-only and Protected (RP) unless the author specifically requests a different mode.

A restriction on authors is implied by the fact that, at present, links may only be nested to a maximum depth of 2. (This limitation was made to allow efficient searching and to keep the file control system from executing an indefinite loop.) "Public" files, however, require two links to be reached and therefore, may not link further themselves.

The author's file directory is the only one which is charged for the records occupied by the file. There is no "free ride" for files "in" the Public File (as they are not actually there), while at the same time there need be only one copy of a file in the entire file system.

Usage of Public Programs

Once the Public program has been "hooked up" as described above any user may then LINK to the file entry in the Public File (M1416 CMFLC4) after which he may use the file it references as if it were one of his own files.

Through the LINK facility, it is, of course, not necessary to COPY into one's own files. Further, it is requested that users in general _not_ copy files listed in the Public File. The reasons for this request are to avoid proliferation of copies of files (thus conserving disk space) and to allow modifications made by the author to become immediately available to users of the file. Modifications are reflected immediately because the linkage information is kept completely in symbolic form. The chain of links is searched each time the file is opened or is referenced with the FSTATE supervisor entry.

(END)

Identification

Time-accounting files

Purpose

The time-accounting files keep all crucial user information such as password, time allotments, party group numbers, etc. These files are read and written by the commands LOGIN and LOGOUT and they can be updated by a few persons with special restriction codes.

Definitions

Each person who is permitted to use the time sharing system is assigned a unique _programmer number_ (4 digits). Depending on the number of jobs he undertakes, he will also be assigned one or more problem numbers (1 alpha and 3 or 4 numeric characters). Groups of people working on the same problem may be assigned the same problem number. When a user logs in, he types his problem number and last name. The combination of problem number and last six characters of the last name is neither unique nor secret. A six character secret _password_ is therefore requested by LOGIN so that a check can be made of the accounting files to see if such a unique combination exists. The unique combination defines a single user and a single file directory, with its associated time and space allotments, etc. An administrator allots a certain amount of computer time each month and a _quota_ of secondary storage space to each user. In addition, each user is placed in a _party group_. Each party group contains some number of users and some different number of slots or _lines_ which give access to the computer (see Section AH.1.01). Each user is also assigned to a _unit group_, which specifies the consoles the user may or may not use.

Description of files

There are five time-accounting files:

UACCNT	TIMACC
TIMUSD	TIMACC
PRTYGP	TIMACC
GRPUNI	TIMACC
TSSFIL	TIMACC

all of which are kept in the system files.

UACCNT TIMACC

The file UACCNT contains identifying information for each user. LOGIN searches UACCNT for the user's problem number, name, and password; this combination must be found before the person can be logged in.

Format of UACCNT TIMACC

Three kinds of entries are found:

1. Group header entry

wd 1	GRPXX
2-28	blank

This entry precedes an administrative group block, composed of one or more problem number blocks.

2. Problem-number header entry

wd 1	*
wd 2	probno, normalized, right-justified
3-28	blank

This entry heads a problem-number block, consisting of one or more user entries for this problem number. (A normalized problem number is of the form LDDDD.)

3. User entry

2 card images:

```
     word1 wcrd2 ...

1)   NAME  PROG   PARTY  STBY   UFD   UNIT  RCODE  FLAGS  PASS

2)   DRUM  DISK   TAPE   T1     T2    T3    T4     T5
```

FLAGS	and UNIT have blank right
RCODE	has leading zeroes
NAME	is left justified

PARTY	is party line group number
STBY	allow standby if non-zero
UFD	user's home file directory
UNIT	is unit group
RCODE	has leading zeroes
FLAGS	are binary indicators:
	001 out of funds
	002 account expired

DRUM,DISK,etc.	are quotas
T's	are in minutes

This entry corresponds to one authorized user or one common file. The following conventions are observed:

a. 28 word entry
b. each item in one word only
c. STBY always "S"
d. items right-justified except:

NAME	left-justified
FLAGS	one blank on right
UNIT	one blank on right

e. RCODE has leading zeroes
f. unused fields must be blank

A special entry type is distinguished, the <u>kludge user entry</u>. This entry follows a normal user entry for a user authorized to use the ESL display scope. It is identical to the preceding entry except in the following respects. The name has at least one asterisk (*) on the right, and is filled with asterisks to make 6 characters. For example:

```
SMITH   SMITH*,
COE     COE***,
LIPSKY  LIPSK*.
```

The party group is always "20" and the unit group always "2".

Entries for common files have only PROG, NAME and record quotas; name and programmer numbers are both CMFLXX.

Sort of UACCNT TIMACC:

major key: group

order is: 1, 2, 5, 3, 4, 6, 7, 8, ...

intermediate: problem number

numeric order

minor: programmer number

numeric order, common files last.

TIMUSD TIMACC

The file TIMUSD contains the following information for each user:

TUn	Time used for each shift.
DATE, TIME	Date and time of last logout.
UNIT	Console ID at last logout.
TL	Total time logged in since first of month.

LOGIN reads the TIMUSD file each time someone logs in. LOGOUT updates the time used information and re-writes that portion of the file containing information on the user logging out. If the user was not previously in the TIMUSD file, a new entry is appended to the end of the file.

Format of TIMUSD TIMACC

2 card images:

```
     word1   word2   ...

1)   PROB    PROG    NAME
2)   DATE    TIME    UNIT          T1   T2   T3   T4   T5   CTU
```

DATE	Last Logout	MMDDYY
TIME		HHMM.T
UNIT		20000. 800273 etc.
T1-5	Time used, shifts 1-5, in seconds	
CTU	Console time used, in minutes	

PRTYGP TIMACC

The file PRTYGP contains the party group information and the maximum number of users. The information contained is copied into the supervisor at system initialization time; the tables thus generated are later examined by LOGIN. Refer to section AH.1.01 for details about party groups.

Format of PRTYGP TIMACC

```
        word1   word2

1)      MXUSRS
```

```
2)...n)      GRP    MXGRP
```

MXUSRS	Maximum number of users permitted on CTSS
GRP	Party group number
MXGRP	Maximum primary lines for group

All items are right-adjusted in 6-character fields.

GRPUNI TIMACC

The file GRPUNI defines groups of consoles the user may or may not be allowed to use.

Format of GRPUNI TIMACC

Fixed field card images; one set for each unit group:

```
     word1   word2       ...

1)   UGN     NUMB
2)   FLAG    UNITID     UNITID     ...
```

UGN	Unit group number
NUMB	14* number of cards following
FLAG	Zero or blank indicates permitted consoles, otherwise indicated forbidden consoles.
UNITID	Console identification

TSSFIL TIMACC

The file TSSFIL defines those user file directories which are to be considered as 'public', and are to be made accessible via the supervisor entry TSSFIL. The information contained is copied into the supervisor at system initialization time.

Format of TSSFIL TIMACC

Fixed field card images; one card per directory:

```
     word1   word2

     PROBN   PROGN
```

PROBN	Problem number of this directory
PRCGN	Programmer number of this directory (e.g. CMFL01)

Both are right adjusted.

(END)

Identification

Bulk input and output

Purpose

Since the console is a relatively slow input/output device, it is necessary and desirable to have a means of entering programs and data into the disk files from card decks and conversely to be able to output disk files onto cards or the high speed printer. Files may be punched on cards in such a format that they may later be reentered into the system to duplicate exactly the original file. In this way, cards may serve as a permanent, inexpensive back-up. There exists a background program known as the "Disk Editor" to control these bulk input/output tasks.

Restrictions

Files of PRIVATE mode may in no way be output. Files of PRIVATE or PROTECTED mode may in no way be deleted by the Disk Editor; therefore, existing PRIVATE or PROTECTED files of the same name as new files may not be replaced by INPUT. None of the disk editor requests will alter (delete or input) a file "through a link".

Usage

A Disk Editor program is run several times a day by the operations staff. Request cards to the Disk Editor may be submitted to the dispatcher by the user, or the RQUEST command (AH.6.06) may be used to create a card image file called OUTPUT RQUEST, which will automatically be processed by the Disk Editor. (Each line within the OUTPUT RQUEST file is the equivalent of a control card and may, therefore, specify any of the following requests except INPUT. The format of each line is the same as a control card except that PROB PROG must not be specified. See Method, below.) Only the first 72 columns of a request card will be read by the Disk Editor.

The control cards for the Disk Editor are of the format:

```
XX PROB PROG NAME1 NAME2 OP ... NAME1n NAME2n
```

The fields are separated by one or more blanks, or by a comma, or by a comma and one or more blanks.

XX is the type of I/O operation desired. (See below.)

PROB is the user's problem number. (It must not be specified in an OUTPUT RQUEST file.)

PROG is the user's programmer number. (It must not be specified in an OUTPUT RQUEST file.) If a common file is specified, PROG is of the form CMFL0n.

NAME1 NAME2 is the file name. All requests except INPUT allow more than one file name per card with the restriction that the file name must be complete on one card, i.e., NAME2 may not be on a continuation card.

OP specifies an option (accepted by particular requests).

XX='C' Continuation card

XX='INPUT' This card must precede a card deck to be input to the disk as a single file, NAME1 NAME2. The deck may be in hollerith or column binary format. (The Disk Editor employs 28-word card images for column binary.) The last card of the deck must have '*EOF*' beginning in column 8. "Flip cards" may be included in the deck, between the INPUT card and the first card to be input. Only one file name may appear on the control card and OP may specify the desired mode, in octal, for the file. If OP is not specified, a permanent file will be created. If a PRIVATE or PROTECTED file of the same name already exists, the deck will <u>not</u> be input. Decks will not be input "through links". Any errors discovered within the deck will cause the entire deck not to be input. The authorship of the file created is the programmer number of the directory into which the file is being placed. If this directory is a common file, the authorship will be zero unless an additional option following the mode is used to specify the author. For example, the following card could be used to input a file into M1416 CMFL03 in PROTECTED/READ-ONLY mode with '3812' as the author:

```
INPUT M1416 CMFL03 TAPE FAP 104 3812
```

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 rest of the line, up to 131 characters, is 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 11, 21, 31, (+10) ..., 121. 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='SSPRNT' The BCD file NAME1 NAME2 is printed with a leading blank on each line to insure single space printing. Line numbered files are always printed single spaced.

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 (28-word card images) 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='PLCT' The file NAME1 NAME2 will be placed on the plct output tape to be processed on the CalComp plotter. (see APM-1)

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

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

The file(s) will be PRINTed, SSPRNTed, DPUNCHed, BPUNCHed, 7PUNCHed, or PLOTed, respectively, 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.

Method

The Disk Editor is a background job which is run several times a day by the operations staff. The users' file directories are searched for OUTPUT RQUEST files. When such a file is found, the editor ATTACHes to the user's file directory and processes the requests found in OUTPUT RQUEST. Because the editor "knows" who the user is, PROB PROG need not be specified in the OUTPUT RQUEST file. Due to the file system locks, the user will not be able to edit the OUTPUT RQUEST file while the Disk Editor is processing it. The OUTPUT RQUEST file will be changed to temporary mode by the Disk Editor after it is processed. After all OUTPUT RQUESTs have been processed, the editor may read cards from the background input tape. As a result of the requests, the editor may create three output tapes, namely punch tape, print tape and carry tape. These are then the responsibility of the operations staff.

7PUNCH Card Format

The 7PUNCH card format is peculiar to the CTSS system at M.I.T., so that it, perhaps, deserves description. The 7PUNCH cards are column binary cards which have punches in rows 12-11-0-7-9 of column one.

Word one	in octal = 7WW5WWTSSSSS
Word two =	full word logical checksum of all words on the card except the checksum itself (does not include columns 73-80).

Remaining words are data words.

wwww	is the word-count of the number of data words to be taken from the card. If wwww .LE. (26) 8, there are wwww words actually on the card (beginning with column 7). If wwww .G. (26) 8, there is only one data word on the card (columns 7,8,9) and it is to be repeated in core wwww times.
sssss	is a binary sequence number beginning with zero.
T	is zero, except on the last card where it is a one.

(END)

Identification

DAEMON: Disk Dump and Reload
M. J. Bailey

Purpose

For the purpose of user's file retrieval and catastrophe reloading of the disk, the contents of the disk must be written onto tape at some specified intervals. With the former file system, the entire content of the disk was written onto two sets of tapes at least once each day.

With the new file system a new approach is being taken to the problem of back-up tapes. A program called the DAEMON runs as a console-less foreground user continuously, except when a complete reload is being performed. The operation of the DAEMON will be controlled by the operator from the console keys under the guidance of on-line printer messages. The DAEMON can perform three separate functions. It may be instructed to perform a complete dump, at which time the entire contents of the disk will be written onto tape. This will normally be done once a week. The complete dump tapes will be divided into two sections, one for the system files (SDT) and another for the users' files (UDT). The DAEMON will be instructed to do incremental dumping as its normal continuous operation. The incremental dumping will consist of writing onto tapes (NFT) only those files which were modified or created since the last incremental dump tape was closed. The files will normally be written onto tape only after a user logs out. The volume of output to the incremental dump tapes should be considerably less than that of the complete dump tape. The third function of the DAEMON is to reload the system. An independent program will be used to reload the system files (including the DAEMON program) from the SDT tapes. As soon as the system files are loaded, the DAEMON will be called to complete the reloading from the remaining user dump tape (UDT) and incremental dump tapes (NFT). This final reloading will also be performed during time-sharing.

Retrieval of specific files can be requested by specifying the date of the last complete dump tape or specifying the date and time of the desired version from an incremental dump tape. Details of retrieval will be published at a later time.

(END)

Identification

Retrieval

Introduction

Files which have been lost (e.g., inadvertently deleted) from the disk may usually be retrieved from history tapes. Under the DAEMON, there are two sorts* of history tape: the Complete Dump Tape (CDT), which includes both System and User Dump Tapes (SDT, UDT); and the New File Tape (NFT), or the incremental dump tape. CDTs are created weekly by the DAEMON at the request of the Operations Staff. These tapes represent a dumping of the entire disk at a given point in time; and, in particular, of a user's entire directory. Alternate (i.e. every two weeks) CDTs are saved for one year. NFTs, on the other hand, represent a dumping of files which have been altered or created (not merely used) during users' console sessions. That is, when a user logs out, the DAEMON will determine whether any "new" files have appeared and will dump any such files it finds. (This process is usually performed within an hour after a given user logs out; therefore, barring unforseen circumstances, back-up is afforded to any user who does not log out, log in very shortly thereafter, and lose a file created during the last session.) NFTs are currently saved for only six weeks, due to tape library limitations.

Dump Maps

When the DAEMON performs dumping, it also produces listings of the files dumped. These "dump maps" contain time dumped, problem number/programmer number, file names, and other information. Binders containing print-outs of the listings are kept in the Dispatching Areas. The dump maps also specify which set of reels (within the dumping period) is involved in the right margin of the listing of files on the reels. For NFTs, the time of dumping is sufficient; however, note that the NFT dump maps are ordered by time of dumping only, and if a file was altered during several different console sessions the dump map must be searched carefully to find not merely an instance of a file's being dumped, but the instance of the file's being dumped which is specifically desired.

Scope of Requests

If several files are to be retrieved from a CDT, it is possible that a request for "entire directory" retrieval would be a good idea. The retrieval process will not disturb existing files (exception: secondary mode files which "exist" only as U.F.D. entries, but have been removed by the storage collection mechanism), so that only missing files will be replaced. This approach is desirable in that requests for too many individual files can over-fill the retrieve command's internal tables and necessitate a second scan of the tape.

Both NFT and CDT retrievals will accept an asterisk (*) as the first or second name of a file; the result will be retrieval of all files possessing the specified second or first name, respectively.

Submitting Requests

"Retrieval Request Forms" are available in the Dispatching Area. They are to be filled out, time stamped, and placed in the appropriate tray. The retrieval will be run by the Operations Staff as soon as possible.

Messages

Progress (or failure) reports on retrievals will be placed in the requestor's directory as files named 'URGENT MAIL' or 'URGENT POST'. They are headed with a row of asterisks, the words 'MAIL FROM DAEMON RETRIEVE', and the date the retrieval was run.

(END)

Identification

Restrictions for Background Systems

Purpose

Any programming system or program under such a system that is to be run as background under CTSS must observe certain conventions or restrictions. These conventions arise due to two main system requirements: that the background program be interruptible and that changes of machine state (such as enablement for traps) are a CTSS supervisor function illegal for the background to perform. The main area of a program affected is its input and output which __must__ be timing insensitive. (Of course, a background system may -- and most probably will -- place restrictions of its own on programs under its control. The MIT version of the Fortran Monitor System (FMS) is an interesting example of a background system, and is frequently used; its internal restrictions can be found through CC-255, a Computation Center Memorandum.)

Restrictions

Change of state:

All changes of state are trapped by the protection mode hardware but certain ones are processed by the supervisor and allowed, such as EFTM (enter floating-point mode).

The following instructions are not allowed and, if used, will cause an on-line diagnostic:

ECTM	LPI	TIB
ESNT	LRI	
ESTM	SEA	
ETM	SEB	

The instruction ENB (enable) is also not allowed, but if used it will be treated by CTSS (which processes the trap it causes) an an effective NOP (no operation) -- i.e., it will not be executed and control will be returned to the next instruction.

I/O timing:

Input and Output must be programmed so that they are not timing dependent; thus the LCHX (load channel) instruction is prohibited. An RCHX (reset and load channel) instruction, if given, must immediately follow the select instruction. An exception is made for the on-line printer and punch where up to 3 SPR's, SPU's and/or NOP's can come between the Select and RCHX instructions. If an RCHX is given that does not comply with these conventions, it will still be executed but its execution may turn on the I/O check light if it was not given "in time".

I/O flag:

All I/O commands (including TCH) must have a "1" in bit 20 (tag of 1 to FAP) to indicate that the information is to be transferred to or from B core. A diagnostic will be given if this condition is not met.

The FAP assembler accepts the pseduo-op, BCORE, which automatically includes this bit 20 in all I/O commands such as IOCD, IORT, TCH, etc., and flags any illegal instructions used.

I/O units:

Only the following I/O units are available for background systems:

a. card reader, card punch, and printer
b. tape units A1-A5, A10, B1-B5, B10
c. A7, the chronolog clock

Referencing of other units will cause a diagnostic.

Program stop:

Any intentional background system stop should be effected by an HTR instruction rather than an HPR. The instruction counter is set differently on the two instructions and due to this difference the HPR if interrupted (e.g. by data channel trap) does does not cause a genuine program stop. Example:

```
A       HTR     Instruction counter set to A;
                resumption after interrupt at A.
B       HPR     Instruction counter set to B+1.
                resumption after interrupt at B+1.
```

Any FAP program using the BCORE pseudo-op will automatically have all the HPR's flagged.

Console keys:

Operating procedures have been modified to limit operator intervention or interaction with a background system from the 7094 control console in such a way that no foreground user or the CTSS supervisor is affected. The address portion of the console keys (or "Panel Input Switches") is used by the CTSS supervisor for this function and therefore cannot be used by a background system. Operators can use the keys to simulate the following functions:

a. initiating "a standard error" procedure. (Octal key code 1)

b. depressing the "Load Cards" button (Octal key code 2)

c. depressing the "Clear & Load Cards" buttons, (Octal key code 3)

(The octal key codes are introduced by placing appropriate keys down in positions 30-35, and are called to the attention of the CTSS supervisor by placing key 21 down).

A "standard error" procedure is defined as: storing the instruction counter in a prearranged location and transferring control to another prearranged location (normally a transfer to a post-mortem routine or to the background system itself). The background system specifies these two locations to the CTSS supervisor by the following call:

```
          TSX     DEFERR,4
          PZE     ERRILC,,ERRTRA
```

where DEFERR contains: TIA =HDEFERR. ERRILC is the location where the instruction counter will be stored and ERRTRA is the location to which control will be transferred. The point of this procedure is that it allows the operator to take effective action in the event of some sort of "hang-up" in the background system, placing that system back into control if a program running under it "runs away" from it.

Independent operation:
If the background system is to be designed to operate independently of the CTSS supervisor, then the background system must be able to verify its mode of operation. A means of determining this so that a switch can be set is to execute the following instructions:

```
          TSX     TESTSS,4
           .                    (1,4) return if running
           .                          independently
           .                    (2,4) return if running
           .                          with CTSS

   TESTSS TIA     L
   L      TRA     1,4
```

If running under the CTSS supervisor, the TIA is interpreted as a regular supervisor call with a 2,4 return. If running independently, there is no "other core" to trap into and the TIA L is executed as a TRA L; thus the 1,4 return is the net result.

Timers:

The subroutines for determining the time operate properly whether the background system is running independently or not. The FMS subprogram GETTM can be used to read the date and time of day from the chronolog clock. The FMS subprogram TIMR can be used to determine elapsed time from the interval timer clock, although when running with CTSS the operation of the interval timer clock is simulated and incrementing takes place only every 200 ms. (as opposed to every 1/60 th of a second when running independently).

The simulated cell 5 interval timer can also be used as an alarm clock; this alarm clock is always enabled.

(END)

Supervisor Entries Reference List

Background System Only

AF.1	CHECK		check tape I/O
AF.1	DEFERR		define error procedure
AF.1	RSTIME		reset UTIME
AF.1	SELECT		does nothing
AF.1	TRA 1,4		does TRA 2,4

Privileged Commands Only

none	6.36AA	*	start tape write
none	6.36ZZ	*	finish tape write
none	636CHK	*	check tape I/O
none	CLOCIN		read chronolog
none	ENTLIN		enter input line for user
none	FINDSB		find standby user to kill
none	HNGUSR		hang up phone after LOGOUT
none	KILL		kill user
none	NOTIM		set user NOTIME code
none	PRINT		on-line print
none	PUNCH		on-line punch
none	RSSWB		reset user write buffer
none	SCHEDL		call scheduler
none	VACUUM		free all adopted consoles

Special Privilege

AG.7.01	SETLOC		modify supervisor
AH.2.06	DSCOPE		ESL scope

All Users

AG.6.05	(EFTM)		enter floating-trap mode
AG.6.05	(LFTM)		leave floating-trap mode
AG.8.03	CHNCOM		go to next command
AG.6.01	DEAD		program exit, dead status
AG.6.01	DORMNT		program exit, dorman status
AG.4.06	FERRTN		set file-error return
AG.6.01	FNRTN		go dormant, don't change ILC
AG.7.01	GETARY		examine block of supervisor
AG.8.03	GETCLC		get command location counter
AG.8.03	GETCLS		get command list
AG.8.04	GETCOM		get command parameter
AG.6.01	GETILC		get ILC at last call to DORMNT
AG.12.01	GETIME		get date, time
AG.7.01	GETLOC		examine supervisor
AG.6.06	GETMEM		get memory bound
AG.8.05	GETOPT		get option status
AG.8.05	GETSYS		get subsystem status
AG.7.07	GETWRD		get A-core variable

AG.12.01	GTDYTM		get file system date and time
AG.7.09	ISIN		get line no. of logged-in user
AG.8.05	LDOPT	S	load option bits
AG.8.01	NEXCOM		load new command
AG.8.05	RSOPT	S	turn off option bits
AG.8.03	SETCLC		set command location counter
AG.8.03	SETCLS		set command list
AG.6.06	SETMEM		set memory bound
AG.8.05	SETOPT	S	turn on option bits
AG.8.05	SETSYS	S	set subsystem status
AG.7.07	SETWRD		set A-core variable
AG.6.08	TILOCK		set file interlock return
AG.3.03	TSSFIL		attach to public directory
AG.12.03	UPCLOC		update simulated interval timer
AG.3.03	USRFIL		return from TSSFIL
AG.7.05	WHOAMI		get user identification parameters
AG.1.01	WRFLX		write on console with c.r.
AG.1.01	WRFLXA		write on console

Anyone_But_Background (FIB may use)

none	CHBALL	does nothing
AG.12.03	CLOCOF	turn off simulated interval timer
AG.12.03	CLOCON	turn on simulated interval timer
AG.3.03	COMFIL	attach to common file
AG.1.04	FORBID	forbid inter-program messages
AG.6.03	GETBRK	get ILC at last interrupt
AG.7.02	GETCF	get common file last attached to
AG.1.11	KILNBK	kill no-break mode
AG.12.04	RDYTIM	type ready message
AG.1.01	RSSRB	reset accumulated unread input
AG.6.03	SAVBRK	reset console interrupt handler
AG.1.02	SETBCD	put terminal in 6-bit mode
AG.6.03	SETBRK	set handler location for interrupt
AG.1.02	SETFUL	put terminal in 12-bit mode
AG.1.11	SETNBK	do not wait for "break" char (c.r.)
AG.1.02	SETNCV	turn off typewriter code conversion
none	WSCOPE	send graphical characters to ARDS

Foreground_Only (FIB may not use)

AG.1.04	ALLOW	allow inter-program message
AG.1.05	ATTCON	attach remote console
AG.7.08	GETBLP	get "blip"
AG.1.01	RDFLXA	read line from terminal
AG.1.05	RDLINA	read attached console
AG.1.04	RDMESS	read inter-program message
AG.1.05	REDLIN	read attached console
AG.1.05	RELEAS	release attached console
AG.1.05	SET12	set mode of attached console
AG.1.05	SET6	set mode of attached console
AG.7.08	SETBLP	set "blip"
AG.1.05	SLAVE	attach remote console as a slave

AG.6.02	SLEEP		go dormant, restart automatically
AG.1.05	SNDLIN		send line to attached console
AG.1.05	SNDLNA		send line to attached console
AG.6.02	WAIT		wait for timer or input
none	WRHIGH		write high-speed lines
AG.1.04	WRMESS		write inter-program message

File System

AG.7.03	ALLOT	*	set secondary storage allotment
AG.7.03	ATTACH	*	attach to other directory
AG.7.04	ATTNAM		find directory attached to
AG.2.08	BUFFER		provide file system with buffer
AG.3.07	CHFILE		change mode, name of file
AG.2.08	CLOSE		close file
AG.3.07	DELFIL		delete file
AG.7.03	DELMFD	*	delete MFD entry
AG.2.08	FCHECK		check on I/C completion
AG.3.07	FSTATE		get file status
AG.2.08	FWAIT		wait for I/C completion
AG.4.06	IODIAG		find out what went wrong
AG.5.05	LABEL	T	label tape
AG.7.03	LINK	*	establish link
AG.5.05	MOUNT	T	ask for tape to be mounted
AG.7.03	MOVFIL	*	move file directory entry
AG.2.08	OPEN		open a file
AG.2.08	RDFILE		read file
AG.2.08	RDWAIT		read file, wait until done
AG.3.06	RESETF		reset all open files
AG.7.03	RSFILE	*	reset locked file
AG.7.03	SETFIL	*	set file status
AG.2.08	SETPRI		set priority
AG.3.07	STORGE		get storage allotment and usage
AG.5.05	TAPFIL	T	create tape entry in UFD
AG.2.08	TRFILE		truncate file
AG.5.05	UMOUNT	T	ask for tape to be unmounted
AG.7.03	UNLINK	*	remove link
AG.3.07	UPDATE		update file directory
AG.7.03	UPDMFD	*	add MFD entry
AG.5.05	VERIFY	T	verify tape label
AG.2.08	WRFILE		write file
AG.2.08	WRWAIT		write file, wait until done

* Denotes privilege required
T Denotes tape call
S Denotes subsystem-restricted call

(END)

Identification

General I/O without format specification
RDFLXA, RDFLXB, RDFLXC, WRFLX, WRFLXA, RSSRB

Purpose

To read from or print on the console without format editing.

Usage

As supervisor or library entries:

```
          TSX   RDFLXA,4       optional(TIA   =HRDFLXA)
          PZE   LOC,,'n'   or PTW   LOC,,N
```

RDFLXA reads a line from the console and moves n words into core beginning at location LOC. On return, the AC will contain the value k, the number of (6-bit) characters read; that is, in 6-bit mode, the break character is the kth character; and in 12-bit mode, the break character is the k/2 character. The word containing the break character and subsequent words are padded with blanks. If the break character is not received before the supervisor's input buffer is full, bit 21 of the AC will be 1, indicating that another call to RDFLXA is required to continue reading the line. In this case, k will be a multiple of six.

To type out in the current mode:

```
          TSX   WRFLXA,4       optional(TIA   =HWRFLXA)
          PZE   LOC,,'n'   or PTW   LOC,,N

          TSX   WRFLX,4        optional(TIA   =HWRFLX)
          PZE   LOC,,'n'   or PTW   LOC,,N
```

To force 6-bit mode:

```
          TSX   WRFLXA,4       or TSX   WRFLX,4
          MZE   LOC,,'n'
```

To force 12-bit mode:

```
          TSX   WRFLXA,4       or TSX   WRFLX,4
          MON   LOC,,'n'
```

WRFLXA will print n words beginning at location LOC (n.LE.14 in 6-bit mode; n.LE.28 in 12-bit mode). It does not add a carriage return at the end of the line and does not delete trailing blanks.

WRFLX will print through the last non-blank character within the n words beginning at location LOC (n.LE.14 in 6-bit mode; n.LE.28 in 12-bit mode). Trailing blanks will be deleted and a carriage return inserted after the last non-blank character.

As library subroutines:

RDFLX:

```
        TSX   RDFLX,4
        PZE   LOC,,'n'
```

RDFLX will read a line from the console using RDFLXA. It will then strip the break character from the line, pad any remaining characters up to n words with blanks, and move the n words into core beginning at location LOC. If n is less than the number of words read, the characters in excess will be lost (n.LE.14).

RDFLXB, RDFLXC:

```
     MAD:     A= RDFLXB.(LOC,K)  ;  A= RDFLXC.(LOC,K)
 FORTRAN:     A= RDFLXB (LOC,K)  ;  A= RDFLXC (LOC,K)
     FAP:          TSX   RDFLXB,4   or    TSX  RDFLXC,4
                   PZE   LOC
                   PZE   K
                   STØ   A
```

LOC is the beginning location of an array into which information is to be stored. If called by MAD or FORTRAN, information will be stored backwards from LOC. If called by FAP (i.e., PZE prefix), information will be stored forward from LOC. The array LOC must be at least (k+5)/6 words long. A line of more than 14 words may be read with one call.

K contains the value k which is the number of 6-bit characters to be read.

A will contain a right adjusted integer equal to the number of 6-bit characters actually read.

RDFLXB using RDFLXA, moves k characters including the break character into LOC. Remaining characters up to k are blank padded.

RDFLXC   is the same as RDFLXB except that k and A do not include the break character.

To reset read-ahead:

```
        TSX   RSSRB,4       optional (TIA  =HRSSRB)
        PAR   =0
```

RSSRB will reset all input waiting for the user in the supervisor's input buffers.

The argument is unused at present, but should be specified as 0. Return is made to 2,4.

(END)

Identification

Set the console character mode switch.
SETFUL, SETBCD, SETNCV

Purpose

To set the console character mode switch.

Usage

As supervisor or library entry:

```
TSX  SETFUL,4      optional (TIA  =HSETFUL)
```

Sets the console character mode switch to "full" 12-bit mode.

```
TSX  SETBCD,4      optional (TIA  =HSETBCD)
```

Restores the console character mode switch to the "normal" 6-bit BCD mode.

```
TSX  SETNCV,4      optional (TIA  =HSETNCV)
```

Sets the console character mode switch to allow input to be transmitted to the user program without code conversion.

Upon return from any entry, the AC is zero if the previous setting was 6-bit mode, 1 if the previous setting was 12-bit mode, or 2 if the previous setting was no-convert mode.

All three library entries may be called by MAD or Fortran programs.

Restrictions

All input waiting in the supervisor's buffers is reset (lost) if any of these calls are made.

(END)

Identification

Console output
PRNTP, PRNTPA, PRNTPC

Purpose

To print a fenced message on the console with a routine which may be called by FORTRAN and MAD.

Usage

As library subroutine:

MAD:

```
EXECUTE  PRNTP. (MESS)
. . .
VECTOR VALUES MESS=$hollerith string$,777777777777K
```

FORTRAN:

```
CALL PRNTP (nH hollerith string)
```

PRNTP, the hollerith string up to the fence prints, on the user's console, 14 words per line. The string may be of any length. If the fence is (377777777777)8, there will be no carriage return at the end of the message. The fence which Fortran automatically supplies is (777777777777)8.

PRNTPA, instead of PRNTP, inserts a carriage return every 14th word, with no carriage return at the end of the message.

PRNTPC, instead of PRNTP, inserts no carriage returns at all. Users must supply what they wish in order to control the printing.

(END)

Identification

Inter-user communication
WRMESS, RDMESS, ALLOW, FORBID

Purpose

To provide the facility for users to communicate with each other directly, several routines have been added to the supervisor which allow the sending and receiving of messages by way of the console input buffers. Privacy screens have been provided which "allow" or "forbid" the sending of messages by specified users.

Method

1) Short messages may be sent to another user's console input buffer.
2) Selectively, short messages may be received in one's own console input buffer from other users.
3) The console input buffer may be read.

Usage

To send a message:
As supervisor entry:

```
          TSX   WRMESS,4           (TIA   =HWRMESS)
          PZE   =HPROBN
          PZE   =HPROGN
          PZE   LOC,,'n'
```

PROBN is the problem number of the receiver (5 character right adjusted with leading blank).

PROGN is the programmer number of the receiver (4 digits right adjusted, leading blanks).

LOC is the beginning location of the message to be sent (forward).

n is the number of words in the message beginning at LOC. If n is larger than 12, a value of 12 will be used.

Upon return, if the AC is non-zero, it contains an error code which indicates that the call was unsuccessful. The following error codes have been assigned.

1 - The specified receiver is not a current user of CTSS. (i.e. logged in).
2 - The receiver's input buffers are full.

3 - The receiver has not given permission for the sender to send messages to his input buffer.

If the AC is zero, the first word of the receiver's input buffer will then contain an octal 77 in character 1, and the sender's problem number in characters 2-6. The second word will contain the sender's programmer number, right adjusted and blank padded. The n words of the message will begin in the third word. If n is less than 12 the terminal words of the 14 word buffer will be blank padded.

To read a message from the input buffer:
As supervisor entry:

```
              TSX    RDMESS,4              (TIA    =HRDMESS)
              PZE    LOC,,'n'
       ALPHA  OPN    EMPTY
              Normal return
```

n words will be moved from the input buffer into locations beginning at LOC.

If the user's input buffer is empty at the time of this call and ALPHA contains a zero, the user is placed in input wait status. If, however, ALPHA does not contain a zero, control returns to ALPHA.

To be selective about who shall send messages to the user:
As supervisor entry:

```
              TSX     ALLOW,4              (TIA    =HALLOW)
              PZE     =HPROBN
              PZE     =HPROGN
```

PROBN is the problem number and PROGN is the programmer number of the programmer who may use WRMESS to send messages to the user's console input buffer. Each call to ALLOW overrides all previous calls.

If PROGN is zero, all programmers on problem number PROBN may send messages.
If PROBN is zero, programmer PROGN may send messages, whatever his problem number.
If both PROBN and PROGN are zero, any programmer may send messages.

To lock everyone out:
As supervisor entry:

```
          TSX     FORBID,4          (TIA     =HFORBID)
```

FORBID prevents any programs from sending lines to the user's console input buffer.

(END)

Identification

Slave remote consoles
ATTCON,RELEAS,SNDLIN,SNDLNA,REDLIN,RDLINA,SLAVE,SET6,SET12

Purpose

To allow multiple remote consoles simultaneously to serve as I/O devices for a single program.

Definitions and Conventions

The console at which a user logs in is his home console. Other consoles associated with a user have been attached by him, and they remain attached until he releases them.

A console attached to one user may not simultaneously be attached to any other user. An attached console may not simultaneously be the home console of any user.

An attached console which automatically transcribes into its output each character typed into the attacher's home console is an IO slave. Similarly, an attached console which imitates the home console's output is an OO slave. An attached console whose typed input appears as input at the home console is known as an II slave.

As described in AC.3, each console is permanently associated with a 6-character console identification word. These console I.D.'s are central to the present facilities.

To attach a console, dial into the computer, and when the ready message is typed, issue the command

```
          DIAL probn prog
```

where 'probn prog' is the user attaching the console. For details, refer to section AH.1.05.

A quit signal issued from an attached console causes it to be detached; in addition, if the console remains inactive for two minutes after being detached, it will be disconnected from the computer.

<u>Usage</u>

To attach a console:
As supervisor entry:

```
          TSX    ATTCON,4           (TIA    =HATTCON)
          PZE    CONSOL
```

CONSOL is the location containing the 6 character console identification of the console to be attached.

Upon return, the AC will be zero if the designated console is '(HOME)', attachable, or already attached to this user. The AC will be non-zero and no attachment made, if the designated console is attached to another, the home console of any user, or otherwise inaccessible.

To release a console:
As supervisor entry:

```
          TSX    RELEAS,4           (TIA    =HRELEAS)
          PZE    CONSOL
```

Upon return, the AC will be zero if the designated console was attached (and therefore is now released) or was '(HOME)'. In all other cases the AC will be non-zero and no action taken.

To send a line:
As supervisor entry:

```
          TSX    SNDLIN,4     (TIA  =HSNDLIN or  =HSNDLNA)
          PZE    CONSOL
          PZE    LOC,,'n'
    ALPHA OPN    FULL
          normal return
```

The line to be sent to the designated console's output buffer is n words long and begins at location LOC.

SNDLIN eliminates trailing blanks and adds the carriage return at the end of the line.

SNDLNA does not eliminate blanks and does not add the carriage return before sending the line.

CONSOL If CONSOL is '(HOME)', the line is sent to the user's home console output buffer. If the designated console is not attached to the user, return is to the normal return with the AC non-zero.

ALPHA If the output buffers at the designated console are full and ALPHA is zero, the user is placed in OUTPUT WAIT status. If ALPHA does not contain zero, control is immediately returned to ALPHA.

To read a line:
As supervisor entry:

```
              TSX    REDLIN,4        (TIA    =HREDLIN)
              PZE    CONSOL
              PZE    LOC,,'n'
       ALPHA  OPN    EMPTY
              normal return
```

REDLIN will move n words from the designated console's input buffer to core beginning at location LOC. If the move was successful, the AC is zero.

CONSOL If CONSOL is '(HOME)', the line will be moved from the home input buffers. If the designated console is not attached, no action is taken and the normal return is taken with the AC non-zero.

ALPHA If the designated console's input buffers are empty, and ALPHA is zero, the program is put into INPUT WAIT status. If the buffers are empty and ALPHA is not zero, control is returned immediately to ALPHA.

Alternate form:
As supervisor entry:

```
              TSX    RDLINA,4        (TIA    =HRDLINA)
              PZE    CONSOL
              PZE    LOC,,'n'   or   BLK    LOC,,N
              PZE    EMPTY
              PZE    ERROR
                                N    PZE    'n'
```

RDLINA will move n words from the input buffer to core storage beginning at LOC. The AC on return will contain a character count indicating the number of <u>6-bit</u> characters read, including the break character. If the line was incomplete (no break character), bit 21 will be on (40000 bit in the address field), and the character count will be a multiple of 6. (The character count returned is identical in format to that returned by RDFLXA. See section AG.1.01).

EMPTY Return will be made to location EMPTY if the input buffers do not contain a complete line. If EMPTY is 0, the program will be placed in input-wait status.

ERROR If CONSOL is not attached, return is made to ERROR. If ERROR is 0, normal return is made with the AC 0.

To create a slave:
As supervisor entry:

```
        TSX   SLAVE,4           (TIA    =HSLAVE)
        PZE   CONSOL
        PZE   MODE
        normal return
```

CONSOL If the designated console is attached, it is made a slave according to MODE and normal return is made with AC zero. If it is not attached, no action is taken and the normal return is taken with non-zero AC. If CONSOL is '(HOME)', this call is ignored and AC is zero.

MODE   There are three distinct slave modes (II,OO,IO) providing eight combinations for any single console. The word at MODE is interpreted as three pairs of letters. If any of the pairs is recognized, the console is made to slave accordingly. If MODE does not contain a recognizable pair, the console is unslaved.

To set the character mode:
As supervisor entry:

```
        TSX   SET,4        (TIA  =HSET6 or  =HSET12)
        PZE   CONSOL
```

SET6 sets the designated console in 6-bit mode.
SET12 sets the designated console in 12-bit mode. They both reset the input buffer unless the console is already in the specified mode.

If the designated console is '(HOME)', the user's console is mode-set. If the designated console is not attached, return is made with non-zero AC; otherwise, the AC is zero.

(END)

Identification

MAD, FORTRAN on-line input compatibility
(CSH), .READ, .READL, .LOOK, .SCRDS

Purpose

MAD and FORTRAN on-line input statements compile as calling sequences to library subroutines. These subroutines use the console as the input device instead of the card reader. A data list and format statement are required.

Usage

```
          MAD:   READ FORMAT FMT, LIST
          FAP:   TSX  .READ,4   or  TSX  .READL,4
                 STR  FMT,,DIR  or  STR  SYMTB,DIR,FMT
                 OPS
                 STR  LIST,,ENDLST
                 OPS
                 STR  0

      FORTRAN:   READ FMT, LIST
          FAP:   TSX  (CSH),4
                 PZE  FMT,,SWITCH
                 OPS
                 STR
                 STQ  LIST,t
                 OPS
                 TSX  (RTN),4

          MAD:   LOOK AT FORMAT FMT,LIST
          FAP:   TSX  .LOOK,4
                 STR  FMT,,DIR  or  STR SYMTB,DIR,FMT
                 OPS
                 STR  LIST,,ENDLST
                 OPS
                 STR  0

          FAP:   TSX  .SCRDS,4
                 PZE  BUF,,'n'
```

.READ	and (CSH) read lines from the console according to the format FMT and LIST.
SWITCH	if non-zero indicates that the format is enclosed in parentheses and stored forward.
OPS	may be indexing or other instructions.
LIST	is the beginning location of the LIST.
ENDLST	is the final location of the LIST.

DIR	if zero the format is stored forwards. If 1, the format is stored backwards.
SYMTB	in a MAD call refers to the start (bottom) of the symbol table for this routine.
BUF	is the first (lowest) location of an array into which data will be read.
n	is an integer indicating the number of words to be read into the array BUF.
.LOOK	reads one line from the console according to the format specified by FMT. The next time a read statement is encountered, the same input will be processed. If more than one line of input is requested by the format, the same line will be used.
.SCRDS	reads a line from the console and stores the number of words requested into the buffer.

(END)

Identification

MAD, FORTRAN on-line output compatibility
(SPH), (SPHM), .PRINT, .COMNT, .SPRNT

Purpose

MAD and FORTRAN on-line output statements compile as calling sequences to library subroutines. These subroutines use the console as the output device instead of the printer.

Usage

```
MAD:  PRINT FORMAT FMT, LIST           FAP:  TSX   .PRINT,4
      PRINT CNLINE FORMAT FMT, LIST          TSX   .COMNT,4
          FAP:  TSX  .PRINT,4     or    TSX   .COMNT,4
                STR  FMT,,DIR     or    STR   SYMTB,DIR,FMT
                CPS
                STR  LIST,,ENDLST
                CPS
                STR  0

FORTRAN:  PRINT FMT, LIST
          FAP:  TSX  (SPH),4
                PZE  FMT,,SWITCH
                CPS
                LDQ  LIST,t
                STR
                OPS
                TSX  (FIL),4

          FAP:  TSX  .SPRNT,4
                PZE  BUF,,'n'
```

(SPH)	and (SPHM) are synonymous.
.PRINT	and .COMNT are synonymous.
.PRINT	and (SPH) type on the console the output as requested by the format FMT and LIST. The maximum line length is 22 words.
SWITCH	if non zero indicates that the format is stored forward.
SYMTB	in a MAD call refers to the start (bottom) of the symbol table for this routine.
OPS	may be any indexing instructions.
LIST(,t)	is the beginning location of the list.

ENDLST	is the final location of the list.
DIR	if zero, the format is stored forwards. If 1, the format is stored backwards. If anything else, a symbol table is implied. See MAD manual for details.
BUF	is the first (lowest) location of an array containing BCD information.
n	is the number of words in the array BUF.

(END)

Identification

Print a comment
.PCOMT

Purpose

To print a comment from a MAD or FAP program on the user's console without a format statement.

Usage

```
MAD:   PRINT COMMENT $MESSAGE$
FAP:   TSX   $.PCOMT,4
       TXH   'n'
       BCI   'n',MESSAGE
```

MESSAGE is a string of no more than 132 Hollerith characters. The characters may not include dollar signs.

n is the number of BCD words to be printed.

(END)

<u>Identification</u>

Print variables without format
.PRSLT, .PRBCD, .PROCT

<u>Purpose</u>

To print a list of variables on the user's console from a MAD or FAP program without specifying a format statement.

<u>Reference</u>

MAD Manual, Chapter II, Section 2.16

<u>Usage</u>

```
MAD:   PRINT RESULTS list
       PRINT BCD RESULTS list
       PRINT OCTAL results list

FAP:   TSX   $.PRSLT,4 (or .PRBCD or .PROCT)
       TXH   SYMTB
       TXH   A
       TIX   LIST1,,LISTN
       TXH   0
```

SYMTB refers to the start (bottom) of the symbol table for this routine.

A refers to a single element.

LIST1 refers to the block of data.

LISTN refers to the end of a block of data.

TXH 0 marks the end of the list.

The values of the variables designated by the list are printed on the user's console preceded by the corresponding variable name and an equal sign, e.g.,

X = -12.4

Blocks are labeled as such and are printed using a block format. Elements of three and higher dimensions will be labeled with the equivalent linear subscript. If dummy variables are included, the specific values assigned to such variables and expressions during execution will be preceded by '...'.

PRINT RESULTS (.PRSLT) causes the output to be numeric (that is, integer or floating point).

PRINT BCD RESULTS (.PRBCD) causes the output to be printed as BCD information.

PRINT OCTAL RESULTS (.PROCT) causes the output to be printed as octal information.

(END)

Identification

Read without list or format
.RDATA, .RPDTA

Purpose

To read data from the console without specifying a list or a format statement. The data items are identified by their variable names as they are typed. The data may be read and printed with one statement.

Reference

MAD Manual, Chapter II, Section 2.16 and 1.1

Restrictions

An input line is limited to 72 characters. If character 72 is used, an implied comma is interpreted as the 73rd character. If more than 72 characters are input in one line, no error message will be printed, but errors will result in the input data.

Usage

```
MAD:     READ DATA
FAP:     TSX  $.RDATA,4
         TXH  SYMTB

MAD:     READ AND PRINT DATA
FAP:     TSX  $.RPDTA,4
         TXH  SYMTB
```

SYMTB is the start (bottom) of the symbol table for this routine

READ DATA reads information from lines typed on the user's console. The values to be read and the variable names are typed in a sequence of fields of the following form

V1 = n1, V2 = n2, ..., Vk = nk *

where the V are variable names and the ni are the corresponding values. Reading is continued from line to line until the terminating mark '*' is encountered.

READ AND PRINT DATA reads the data as explained above, and then immediately prints it out.

In case of an input error, a message is printed on the user's console. Included in this message are the type of input error, the line in which the error occurred, the column number in which the error was found, and the recovery procedure. If the user wishes, he may retype the offending line and all succeeding ones, in order to continue. Otherwise, he may terminate his program by the 'QUIT' signal. He may then use the PM or any other debugging command.

(END)

Identification

No-break mode
SETNBK, KILNBK

Purpose

As the CTSS supervisor receives input characters from a user, it normally waits to accumulate a whole line before signalling the user that he has input, so that the user program goes into input wait status until the break character (carriage return) is struck at the console. A special mode, _no-break mode_, is available for those applications where the user program wishes to be informed of input as soon as it arrives.

Usage

As a supervisor or library entry:

```
                  TSX  SETNBK,4     optional (TIA  =HSETNBK)
```

SETNBK will cause the supervisor to set no-break mode for the user. Subsequent calls to RDFLXA will return as soon as any characters have been typed.

```
                  TSX  KILNBK,4     optional (TIA  =HKILNBK)
```

KILNBK will restore the normal mode.

(END)

Identification

Print a message on the console
PRMESS, PRMESA

Purpose

PRMESS provides a convenient way for the MAD programmer to type output on his console.

Usage

```
          PRMESS.($LITERAL$,VAR,VAR(0)...N, ...)
```

PRMESS types the message which is the concatenation of all its arguments. Any number of arguments may be supplied. Note that MAD compiles the right code if a literal string of more than six characters is supplied as a single argument (it produces several arguments, one for each six-character chunk). PRMESS calls WRFLXA for each 14 words it accumulates and then calls WRFLX for the last 14 or fewer words specified.

PRMESA works like PRMESS, but does not end the line with a carriage return.

vectors may be specified in the form VAR(0)...N to type out a vector of N words running backwards in core, or VAR(N)...MINUSN (where MINUSN contains -N) to type vectors stored forwards in core.

Example

```
            V'S M1 = $THE ANSWER IS$
            PRMESS.(M1...3,BZEL.(DERBC.(I)),$ FURLONGS$)
```

Would type:

```
THE ANSWER IS     15 FURLONGS
```

If the value of I was 15.

(END)

Identification

Full-mode output from MAD programs
PR12, PR12A

Purpose

PR12 provides the MAD programmer with a convenient method of producing output in upper and lower case without sacrificing program readability.

Usage

```
          PR12.($LITERAL$, VAR, VEC...N, etc)
```

PR12 takes its BCD arguments and expands them according to the escape conventions described below. It calls WRFLXA for every 28 words it accumulates, and finishes with a call to WRFLX for the last 28 or fewer words.

PR12A works like PR12 except that it does not end the line with a carriage return.

Arguments to PR12 and PR12A may be specified like the arguments to PRMESS: that is, vectors running either forwards or backwards in core may be printed as well as single variables. PRMESS and PRMESA are secondary (negative) entry points to the program to save core space for programs which call both. Output is produced by calling WRFLX or WRFLXA with a prefix of MON, so that 12-bit mode is forced and the current character mode switch is unchanged.

Escape Conventions

The character-escape conventions have been chosen to save space and to have some mnemmonic value. The character which signals an escape is the apostrophe ('). Any character not preceded by an apostrophe prints as itself, except that letters are printed in lower case. The following table shows the mapping performed.

input	printed
A	a
'A	A
etc	etc
'.	!
'(	$
')	?
''	'
'=	"

'-	$\bar{}$
'+	$\bar{\varepsilon}$
',	;
'1	red shift
'2	black shift
'3	<
'4	>
'0	null
'5	%
'/	tab
'*	carriage return

The following special operators are defined:

'6	enter BCD mode
'7	return to full mode
'8	end of argument
'9	end of all text

After '6 is recognized, no escape sequence except '7 and '8 will be active, and all letters will be upper case. When '8 is seen, PR12 immediately goes to the next argument. When '9 is seen, PR12 dumps its buffer and returns.

Example

```
          V'S M2 = $'7 NOT FOUND. 'GO ON') '8$
          PR12A. ($'6'8$,N1,$ '8$,N2,M2...10)
```

Might produce the following:

```
     ALPHA OUTPUT not found. Go on?
```

With no carriage return at the end of the line.

(END)

Identification

Print 12-bit lines
PRFULL,PRFULA,PRTCHR

Purpose

To print a fenced or unfenced message containing 12-bit characters on the console with a routine that may be called by MAD.

Usage

```
     MAD:  PRFULL.(A,B...N)
           PRFULA.(A,B,C...N)
           PRTCHR.(SUBR.)

     FAP:  TSX  $PRFULL,4    or    TSX  $PRFULA,4
           PZE  A    or   A,TAG
           EFA  B    or   B,TAG
           PAR  C
           PAR  D,,N
           BLK  E,,M

           TSX  $PRTCHR,4
           PAR  SUBR
```

PRTCHR.(SUBR.) causes each 12-bit output character to be given to 'SUBR.' (by 'EXECUTE SUBR.(CHAR)') instead of printing it. This mode may be terminated at any time by

```
     TSX  $PRTCHR,4    or    PRTCHR.(0)
     PAR  0
```

PRFULL adds a carriage return at the end of the output string, PRFULA does not.

The calling sequence is of indefinite length, all arguments are concatenated to form one continuous character string. Blocks are normally processed backward from X(0) to X(N); if N is negative the block will be processed forward. 'X...0' is ignored. For 'PAR X,,N' arguments, if N>16383, it is considered negative and the count actually used is 32768-N. Any argument will be terminated if a fence (octal 77...77) is encountered within it. The setting of the FUL-mode switch (SETFUL, SETBCD) is not affected.

To facilitate the use of 12-bit characters from MAD and FAP programs the character apostrophe has been made to modify the character that follows it, usually into an otherwise inaccessable 12-bit character but sometimes into a control function. Upper case letters are normally converted into lower case. If an apostrophe is followed by a character that does not have a modification defined, the apostrophe is ignored.

A complete list of modifications is:

'ddd where ddd is octal:	the 12-bit character whose octal code is ddd is printed
'd where d is 0 thru 7:	the next d letters are printed in the case opposite from that in which they would otherwise have been printed
'8	ignore the remainder of the current word
'9	end of output string
'=	"
''	'
'+	&
'A	@
'B	black ribbon shift
'F	print-off
'H	hang-up
'.	!
')	>
':	backspace
'-	_
'L	print succeeding letters in lower case
'N	#
'O	print-on
'P	%
'Q	?
'R	red ribbon shift
'*	carriage return
'space	null
'/	backslash or cent sign
'S	$
'T	tab
'U	print succeeding letters in upper case
'V	|
'X	end this argument and print the next exactly as given
',	;
'(	<

(END)

Identification

Unbuffered disk string read and write
DSKDMP, .DUMP, .LOAD, DSKLOD

Purpose

To write or read a continuous block of core on (from) the disk as a file. These routines are usually used for large blocks of core, or short files.

Usage

Two routines are available as supervisor entries and library entries. An additional routine is available in the library which may be called by MAD and Fortran programs.

To write a file on the disk:
Core-B write around:

```
TSX  .DUMP,4
OPN  FILNAM
PZE  LOC,,'n'
```

OPN establishes the mode of the file{; PZE is temporary, PON is permanent, PTW and PTH are read-only, protected.

FILNAM refers to the file name which will be placed in the current file directory, deleting any older file of the same name.

LOC is the initial location from which n words will be written on the disk.

To read a file:
Core-B write around:

```
TSX  .LOAD,4
PZE  FILNAM
PZE  LOC,,'n'
SLW  M
```

n is the number of words to be read. It may be larger than the actual file size with the following restriction: LOC+n-1 must be less than the memory bound. FSTATE may be used to estimate n.

M will contain the number of words actually loaded, as a full word integer.

Corresponding library subroutine:

```
      MAD:  EXECUTE DSKDMP. (FILNAM,FIRST,N)
            EXECUTE DSKLOD. (FILNAM,FIRST,N)
            M = DSKLOD. (FILNAM,FIRST,N)
  FORTRAN:  CALL DSKLOD (FILNAM,FIRST,N)
            CALL DSKDMP (FILNAM,FIRST,N)
            A = DSKLOD (FILNAM,FIRST,N)
```

Core will be loaded or dumped from FIRST-n+1 through FIRST. If the number of words ,m, in the file is less than n, the file will load into the block of core through FIRST-n+m. Both DSKDMP and DSKLOD call the file system directly, i.e., they do not call the core-B write arounds.

(END)

Identification

Buffered disk input
SEEK, .SEEK, .READK, ENDRD, .ENDRD, BREAD, VREAD, DREAD

Purpose

To provide the facility to read fixed length, string or line-marked disk files in the buffered mode by calls from FAP, MAD or FORTRAN programs. Records may be converted according to a format statement or may be transmitted without conversion.

Method

Disk files to be read must be located in the current file directory, the hardware must be set in motion to locate the first track of the file, a buffer must be assigned to the file and the tracks must be read to fill up the buffers. All of this initial activity is accomplished by the user's call to SEEK in which he may specify buffer locations. If, however, the user doesn't care to specify a buffer, SEEK will assign available space by extending the memory bound.

Reading is accomplished by moving logical records out of the buffers into working core. When a buffer becomes empty, the supervisor fills it by reading the next track of the file into it. After sufficient data has been read from a file, the user may release the buffer and put the file in inactive status by a call to ENDRD.

Restrictions

The library subroutines maintain a list of active files and assigned buffers. There may be no more than 10 active files and no more than 20 automatically assigned buffers.

Reading by calls to the core-B write arounds instead of the library subroutines means that buffers are not automatically assigned, only one buffer can be used, errors cause execution of supervisor error procedures rather than the library error procedures, and the write-arounds to the new file system are used.

Usage

To open a file:
as core-B write around:

```
        TSX   .SEEK,4
        PZE   FILNAM
        PZE   BUF1
```

as library subroutine:
FAP, MAD, or FORTRAN,

```
        EXECUTE  SEEK.  (FILNAM,-BUF1-,-BUF2-)
```

BUF1, BUF2 are initial locations of 432 word blocks of core to be used as buffers. If no buffer is specified to the library subroutine, one buffer will be assigned by extending the memory bound if core space permits. If no buffer space is available, the library error procedure will be initiated. If two buffers are provided, reading will be more efficient, since I/O may be overlapped with processing.

SEEK calls SRCH which assigns a buffer, if needed, by calling FREE and maintains an active file table and buffer assignment table.

.SEEK does not call SRCH.

To read a record:
as core-B wrote around:

```
        TSX   .READK,4
        PZE   FILNAM
        PZE   LOC,t,n
        PZE   EØF
         .
         .
         .
    EOF SLW   WC
```

n words will be moved from the current buffer associated with FILNAM and stored in a block of core beginning at location LOC. n may be larger than the actual file size but LOC+n-1 must be less than the memory bound.

t of non zero means skip n words without transmission.

EOF If an attempt is made to read beyond the last word of the file FILNAM, control is transferred to location EOF.

WC upon end of file return, the AC will contain the number of words actually read, as a full word integer.

as a library subroutine:
FAP or MAD

```
EXECUTE BREAD. (FILNAM, LIST)
EXECUTE DREAD. (FILNAM, FORMAT, LIST)
WC=VREAD.(FILNAM, LIST)
```

LIST is any mixture of single variables and block notation vectors locating the variables to be read, if any.

FORMAT is the location of the format by which the variables in LIST will be edited by (IOH).

BREAD will read the n words specified by the LIST. n may be any size. No attention is paid to logical record breaks. If the input file is line-marked, the line-marks will be moved as data words.

DREAD reads logical records and edits them through (IOH). Each call to DREAD reads at least one logical record; however, the format may require the reading of more than one logical record. If the file is line-marked, the line marks delineate the logical records. If the file is not line-marked, the logical records are 14 words. If fewer words are requested than are available in the record, the excess of the record is lost. The format may specify the reading of more than one record; however, if more words are requested from a specific record than are available within that record, the library error procedure is initiated.

VREAD will read one logical record. A logical record is either delineated by line-marks, set by SETVBF, or assumed to be 14 words. The LIST may not exceed 22 words. If the LIST is longer than the logical record, the end of the list will be padded with blanks. If the LIST is shorter than the logical record, the remainder of the record will be lost. If the record was fixed length, the sign of WC will be minus. If the record was line-marked, WC will be positive. WC is a properly formatted integer but Fortran may have some difficulty because of the function naming conventions.

To close an input file:
as core-B write around:

```
        TSX  .ENDRD,4
        PZE  FILNAM
```

as library subroutine:
FAP, MAD, or FORTRAN

```
        EXECUTE ENDRD. (FILNAM)
```

ENDRD will delete the file from the active file table and release the buffer.

(END)

Identification

Buffered Disk Output
ASSIGN,.ASIGN,APPEND,.APEND,.WRITE,FILE,.FILE,B-D-V-FWRITE

Purpose

To provide the facility to write fixed length, string or line-marked disk files in the buffered mode. Records may be converted according to a format statement or they may be transmitted without conversion.

Method

The file must be defined and placed in an active file table and buffers must be assigned. This initialization is accomplished by ASSIGN or APPEND. Writing then causes data to be moved from working core into the buffers. When a buffer is full, it is written on a track of the disk by the supervisor. A file in write status must be closed by FILE in order to assure that the last buffer has been written on the disk and the file name is entered into the file directory.

Restrictions

If the library subroutines are used, an active file table and assigned buffer table are maintained. There may be no more than 10 active files and 20 automatically assigned buffers. If the program is terminated by any terminal library routine, all files in write status will be properly closed. Any disk errors will initiate the library disk error procedures.

If the core-B write arounds (.ASSIGN, .APEND and .WRITE) are used and the program is terminated without going to .FILE or EXIT, the file will be lost. EXIT has been modified to include a CLOSE.(ALL). Any disk errors initiate I/O system error procedures. Only one buffer can be used with calls to the core-B write arounds.

For any given file, calls to the library subroutines may not be intermixed with calls to the core-B write arounds or I/O system entries. That is, buffers may not be assigned by .ASIGN with reading being done by BWRITE, etc.

Usage

To open a new file:
core-B write around:

```
                TSX  .ASIGN, 4
                OPN  FILNAM
                PZE  BUF1
```

as a library subroutine:

```
          EXECUTE ASSIGN. (FILNAM, -BUF1-,-BUF2-,-BUF3-)
```

OPN defines the mode: PZE is temporary, PON is permanent, PIW and PTH are read-only protected. The library subroutine will define the mode as permanent.

BUF1,BUF2,BUF3 are the initial locations of 432 word blocks of core to be used as buffers. If no buffer is specified for the library subroutine call, two buffers will be assigned by extending the memory bound, if core space permits. If no buffer space is available, the library error procedure will be initiated. Writing with only one buffer is extremely inefficient since it forces the use of WRWAIT. Two buffers greatly increase efficiency because this allows use of the core-B buffering routine BFWRIT. Three buffers make it possible to overlap I/O with processing.

ASSIGN calls SRCH which assigns two buffers if necessary by calling FREE, and maintains an active file table and buffer assignment table. This allows terminal subroutines to close active files properly.

.ASIGN does not call SRCH.

ASSIGN and .ASIGN If a file already exists named FILNAM, it is deleted.

To open an old file in order to add information:
core-B write around:

```
                TSX  .APEND,4
                PZE  FILNAM
                PZE  BUF1
```

as a library subroutine:

FAP, MAD or FORTRAN

```
        EXECUTE APPEND. (FILNAM,-BUF1-,-BUF2-,-BUF3-)
```

APPEND is the same as ASSIGN except the file name is located in the file directory and data to be added to the file will be written at the end of the existing file.

To write a file:

core-B write around:

```
        TSX  .WRITE,4
        PZE  FILNAM
        PZE  LOC,,'n'
```

n is the number of words to be written into file FILNAM beginning at location LOC.

as a library subroutine:

FAP, MAD or FORTRAN

```
        EXECUTE  BWRITE. (FILNAM, LIST)
        EXECUTE  DWRITE. (FILNAM, FORMAT, LIST)
        WC  =   VWRITE. (FILNAM, LIST)
        WC  =   FWRITE. (FILNAM, LIST)
```

LIST is any mixture of single variables and block notation vectors locating the variables to be output.

FORMAT is the format by which the variables in LIST will be edited through (IOH).

BWRITE will write the n words specified by the LIST as a record without line marks. LIST may be any length.

DWRITE will write the n words specified by LIST as a line-marked record after they have been edited by (IOH). (3 .LE. n .LE. 22). If n .L. 3, blanks will be filled in until the record is 3 words long. If the combination of FORMAT and LIST specify a line longer than 22 words, (IOH) will type an error message and then call RECOUP.

VWRITE will write the n words specified by LIST as a line-marked record. 3 .LE. n .LE. 22 (same convention as DWRITE). WC will contain an integer equal to the number of words written (not including the line-mark). The actual record length is WC+1..

FWRITE will write a fixed length record without line-marks. If the LIST is shorter than the fixed length, blanks will be filled in. If the LIST is longer than the fixed length, only the first words are written and the excess is lost. The fixed length is assumed to be 14, unless set by SETVB(F). WC will contain an integer equal to the number of words written, the sign will be minus.

WC when WC is returned, it is the proper integer format for the language of the calling program. Fortran, however may have some difficulty as a result of the mode of the function convention. Fortran users should equivalence WC with an integer variable.

To close an output file:
core-B write around:

```
TSX  .FILE,4
PZE  FILNAM
```

as a library subroutine:
FAP, MAD, or FORTRAN

```
EXECUTE FILE. (FILNAM)
```

FILE will cause any active buffers to be written on the disk, FILNAM will be entered into the current file directory, the buffers will be set free, and the file removed from active status. If the library subroutines have been used to write the file, a call to any terminal subroutine (EXIT, DUMP, etc.) will cause the calling of FILE for all active files.

.FILE should be used only if the file was written by the .WRITE write around.

(END)

Identification

Addressable disk files
.RELRW

Purpose

To allow disk files to be treated as addressable secondary memory. Relative locations within a disk file may be specified for reading or writing.

Usage

To open an addressable file:
core-B write around:

```
          TSX  .RELRW,4
          PZE  FILNAM
          PZE  BUF1
```

.RELRW will open an addressable file which may be read or written. If writing, the mode is permanent.

BUF1 is the initial location of a buffer whose size should be at least 432 words.

To read or write an addressable file:
core-B write around:

```
          TSX  .READK,4                 TSX  .WRITE,4
          PZE  FILNAM,,reladr
          PZE  LOC,,'n'
          PZE  EOF
```

reladr is the relative location within the disk file where the reading or writing will begin. The first word is number 1. If reladr is outside the limit of the file, the normal end-of-file procedure will be followed for reading or the supervisor error procedure will be followed if writing.

LOC,,'n' n words of core beginning at location LOC will be read from or written in the disk file FILNAM.

EOF Location to which control will be transferred upon encountering an end of file.

Identification

Set the length of fixed length records
SETVBF, SETVB

Purpose

Records which are read or written by FWRITE or VREAD may be fixed length. The normal fixed length is 14 words. If a different length is desired, SETVBF may be used to specify the length.

Usage

As a library subroutine:

MAD, FAP, or FORTRAN

```
B = SETVBF.(N)
```

SETVBF and SETVB are synonymous. Both names are provided because of the Fortran function naming convention.

N is (location of) the number of words to be considered for fixed length records by FWRITE or VREAD. N may not be greater than 22. If N .GE. 22, the record length is set to 22.

B will contain the previous setting of the fixed record length.

(END)

Identification

Service to library disk routines
SRCH, BLK, FLK, ENDF, CLOUT

Purpose

Service routines are available to the library disk subroutines to assign buffers, find files, maintain the active file and buffer tables, and close out files.

Usage

To search active file table:

```
        TSX  SRCH,4
        PZE  FILNAM
             not found
             found
```

not found return means that FILNAM was not found in the active file table.

found returns with the status of FILNAM in the address of the AC and a buffer number (1-20) in the decrement of the AC. If the file is not using an assigned buffer, the buffer number is zero. Write status is 1; read status is 2. The sign is + if enough buffers are assigned to use core-B buffering routines (BFREAD, etc;). The sign is - if supervisor I/O must be used.

To assign a buffer:

```
        TSX  BLK,4
             error return
             normal return
```

BLK searches the buffer assignment table. If there are no free buffers and there are fewer than 20 assigned buffers, an attempt is made to extend the memory bound by a call to FRER.

error return is taken if there are already 20 buffers assigned or the attempt to extend the memory bound was unsuccessful.

normal return is taken with the address of the buffer in the address of the AC and the number of the buffer (1-20) in the decrement of the AC.

To enter a file in the active file table:

```
TSX  FLK,4
PZE  FILNAM
PFX  status,,PTR1
PZE  ,,PRT2
     error return
     normal return
```

status	is 1 if writing, 2 if reading. The status word is stored in the first free space in the active file table.
PTR1,PTR2	is the buffer number. If number is non-zero, a pointer to the file in the active file table is placed in the assigned buffer table.
PFX	is PZE if enough buffers are assigned to use core-B routines (BFREAD,BFWRITE); otherwise, it is MZE.
error	return is taken if there are 10 active files already.

To remove a file from the active tables:

```
TSX  ENDF,4
PZE  FILNAM
```

The buffer is freed, and the file is removed from the active table. The file is not closed.

To remove a file from the active tables:

```
TSX  CLOUT,4
```

All the files are closed by calls to CLOSE and BFCLOSE. All buffers are freed and returned to "free storage".

(END)

Identification

Generate file of zeros
.CLEAR

Purpose

To create a new file which contains n zeros.

Usage

Core-B write around:

```
        TSX  .CLEAR,4
        OPN  FILNAM,,'n'
```

.CLEAR will create a file of the name specified in FILNAM which will contain n zeros. The opening and closing of the file are accomplished by .CLEAR so that .ASIGN and .FILE should not be called.

OPN specifies the mode of the file: PZE is temporary, PON is permanent, PTW and PTH are read-only and protected.

(END)

Identification

Input and output
OPEN,BUFFER,RDFILE,RDWAIT,WRFILE,WRWAIT,TRFILE
FCHECK,FWAIT,CLOSE,SETPRI

Purpose

Files may be opened on any I/O storage device for reading, writing or reading and writing. A file which has been successfully OPENed is said to be "active". A buffer may be assigned if needed and priorities may be set for different files.

Method

It is assumed that the user is familiar with section AD.2 and AG.4.06 of this manual. In order to read or write a file, the file must first be opened and in most cases a buffer should be assigned. Calls to RDFILE or WRFILE initiate the I/O for a relative location within the file. The actual data transmission is not completed upon return from the call. A subsequent RDFILE, WRFILE, FCHECK, or CLOSE is necessary to complete the data transmission and I/O error checking. All calling sequences will accept the two extra arguments for the error procedure. Any arguments which are not pertinent may be specified as -0.

Usage

OPEN:

```
OPEN.($STATUS$,$ NAME1$,$ NAME2$,MODE,DEVICE)
```

STATUS	may be 'R' for read, 'W' for write or 'RW' for read-write. (justification is not significant).
MODE	specifies the mode of a new file to be created and may be the inclusive logical or of any of the following octal values. If MODE is not specified, a permanent file will be created. 000 - Permanent 001 - Temporary 002 - Secondary 004 - Read-only 010 - Write-only 020 - Private 100 - Protected
DEVICE	is pertinent only when creating a new file and it specifies which I/O device is desired. If DEVICE is not specified, the system will

assign a device.

1 - Low speed drum
2 - disk
3 - Tape

Error codes:

03.	File is already in active status
04.	More than ten active files
05.	$STATUS$ is illegal
07.	Linking depth exceeded
08.	File in 'PRIVATE' mode (different author)
09.	Attempt to write a 'READ-ONLY' file
10.	Attempt to read a 'WRITE-ONLY' file
11.	Machine or System error
12.	File not found in U.F.D.
13.	Illegal device specified
14.	No space allotted for this device
15.	Space exhausted for this device
16.	File currently being restored from tape
17.	Input/Output error, see AG.4.06
18.	Illegal use of M.F.D.
19.	U.F.D. not found (i.e., OPEN through a link).
20.	Attempt to read secondary mode file.

Assign a buffer:

```
BUFFER.($ NAME1$,$ NAME2$,BUF(N)...N)
```

BUFFER	In general a buffer should be assigned to an open file for reading or writing.
BUF	The buffer space should be specified in block notation as the beginning location of the buffer and the size. The size must be large enough to accomodate a physical record from the I/O device.
N	is the buffer size and 432 seems to be the going size.

Error codes:

03.	File is not an active file
04.	Previous I/O out of bounds (membnd changed)
05.	Buffer too small
06.	Input/Output error, see AG.4.06

Set priority:

```
SETPRI.(PRIOR)
```

SETPRI is used to assign priorities to certain tasks which would otherwise be processed in the order in which they were received. When files are opened for reading and/or writing, they are assigned the priority set by the last call to SETPRI. If there was no previous call to SETPRI, all files will be treated with equal priority.

PRIOR is an integer from 1 to 7. The higher the value the lower the priority.

Error codes:

Standard error codes. See section AG.4.06

Read: RDFILE.($ NAME1$,$ NAME2$,RELLOC,A(N)...N,EOF,EOFCT)

RDWAIT.($ NAME1$,$ NAME2$,RELLOC,A(N)...N,EOF,EOFCT)

RDFILE initiates the I/O necessary to move N words of data into location A(N) through A(1) from file NAME1 NAME2.

RDWAIT is a single call which incorporates RDFILE and FCHECK so that upon return, the data has all been moved and all of the error checking has been done.

RELLOC specifies the initial location within the file from which reading is to begin. If RELLOC is zero, reading continues from the word following the last word <u>read</u> from the file. On the first call to RDFILE either 0 or 1 specifies the first word. Note that in a file which is open for reading and writing, there are two separate pointers (i.e., the last word read and the last word written).

EOF is the location to which control will be transferred if the end of the file is encountered before N words are available to transmit into A. If RDFILE was called the words have not actually been transmitted to A so that FCHECK or CLOSE is necessary if data from A is to be used. The file is not closed by encountering an end of file.

EOFCT is an integer variable which will contain the number of words to be transmitted by the call to RDFILE when the end of file was encountered.

Error codes:

03. File is not an active file
04. File is not in read status
05. No buffer assigned to this file
06. Previous I/O out of bounds (membnd changed)
07. Input/Output error, see AG.4.06
08. U.F.D. has been deleted

Write:

```
WRFILE.($ NAME1$,$ NAME2$,RELLOC,A(N)...N,EOF,EOFCT)

WRWAIT.($ NAME1$,$ NAME2$,RELLOC,A(N)...N,EOF,EOFCT)
```

WRFILE	initiates the I/O necessary to move N words from the array A(N) thru A(1) into the file NAME1 NAME2.
WRWAIT	is a single call which incorporates WRFILE and FCHECK so that upon return, the data has been moved and error checking has been done.
RELLOC	is the relative location within the file where writing is to begin. If RELLOC is zero, writing will begin after the last word <u>written</u> in the file. If RELLOC is zero on the first call, writing will begin at the location following the last word of the file. RELLOC may not be larger than the current length of the file.
EOF	is the location to which control will be transferred if the N words to be written would have to be written through the end of file (i.e., if part of the record could be contained within the file and the other part would extend to outside the file). This does not occur when appending to the file with a RELLOC of zero where entire records are placed at the end of the file.
EOFCT	is an integer variable into which the I/O system will store the number of words actually to be written when control was transferred to EOF. An FCHECK is necessary as with any WRFILE.

Error codes:

03. File is not an active file
04. File is not in write status
05. No buffer assigned to this file
06. Allotted space exhausted for this device

07. Previcus I/O out of bounds (membnd changed)
08. Input/Output error, see AG.4.06
09. Illegal use of write-only file (ncn-zero 'RELLOC')
10. Max file length exceeded

Truncate:

TRFILE.($ NAME1$,$ NAME2$,RELLOC)

TRFILE The file NAME1 NAME2, which was previously opened for writing, will be truncated (i.e., cut-off) immediately before the relative location RELLOC. If RELLOC is less than the read or write pointers, they will be reset to their original places, (i.e., the read to the first word of the file and the write to after the last word of the file).

Errcr ccdes:

03. File is nct an active file
04. File is nct in write status
05. No buffer assigned to this file
06. Previcus I/O out of bounds (membnd changed)
07. RELLOC larger than file length
08. Input/Output error, see AG.4.06
09. Illegal use of write-only file (ncn-zero 'RELLOC')

Check:

FCHECK.($ NAME1$,$ NAME2$,FINISH)

FWAIT.($ NAME1$,$ NAME2$)

FCHECK is used to check to see if a previous read or write of a specific file has been ccmpleted and checked for errors. Note that RDFILE, WRFILE , TRFILE, and CLOSE incorporate an automatic FCHECK at the beginning so that if FCHECK is not called explicitly, any I/O errors are detected one call later than the call that caused the error.

FWAIT is the same as FCHECK except that control will not be returned to the user until all I/O has been ccmpleted and checked.

FINISH is the location to which FCHECK will return control if the I/O is completed and checked. If the I/O is not completed, FCHECK will take the normal return.

Error codes:

03. File is not an active file
04. Previous I/O out of bounds (membnd changed)
05. Input/Output error, see AG.4.06

Close:

```
CLOSE.($ NAME1$,-$ NAME2$-)
```

CLOSE is used to close an active file and return it to inactive status. CLOSE incorporates an FCHECK for the last I/O call and initiates and FCHECKs the I/O necessary to empty any waiting output buffer.

NAME1 may be 'ALL' and NAME2 not specified for all active files to be closed.

Error codes:

03. File is not an active file
04. Previous I/O out of bounds (membnd changed)
05. Input/Output error, see AG.4.06
06. Machine or System error

(END)

Identification

Load a file into a free area of core
LDFIL

Purpose

To load a file into a free area of core, and then pass control to a specified function, giving information as to where the file has been loaded and how long it is.

Usage

```
        FAP:   TSX    LDFIL,4
               PZE    =H NAME1
               PZE    =H NAME2
               PZE    FUNCT
             - PZE    ARG1 -
             - PZE    ARG2 -

        MAD:    LDFIL. ($ NAME1$ NAME2$,FUNCT.,-ARG1-,-ARG2-)
```

LDFIL loads the file NAME1 NAME2 and calls FUNCT with the following call

```
        FAP:   TSX    FUNCT,4
               PZE    LODAD
             - PZE    ARG1  -
             - PZE    ARG2 -

        MAD:   FUNCT. (LODAD,-ARG1-,-ARG2-)
```

LODAD contains the exact word count (WC, as an integer) of the file NAME1 NAME2. The file is loaded into locations LODAD+1,...,LODAD+WC.

ARG1 ARG2 are optional arguments which LDFIL will transmit, if present, to FUNCT.

A return from FUNCT will automatically mean a return to the program which called LDFIL with all registers except index register 4 preserved.

LDFIL uses FRER, FRET and CCLT in addition to the I/O system routines.

If sufficient space is not available to load NAME1 NAME2, LDFIL will cause a comment to be printed (by FRER) and call EXIT.

(END)

Identification

Buffered Input and Output
BFOPEN, BFREAD, BFWRIT, BFCLOS, BFCODE

Purpose

Because entries to core-A and the file system involve quite a bit of overhead, it is advisable to provide for buffering and for all blocking and unblocking of buffers in core-B routines and to call the file system only to transmit full records. These ("BF-package") library routines are available to provide single or double buffering in core-B. Double buffering is definitely advantageous to programs which are "compute-limited" because it allows overlapping of CPU time with I/O time.

Method

The file system is used for all actual I/C. In order to read or write a file, the file must be opened with one or two buffers specified. In the case of writing a file, one extra buffer is always needed to assign to the file system; new files files opened by BFOPEN will be in the permanent mode. Calls to BFREAD and BFWRIT cause words to be moved from (to) a buffer to (from) the user's work area. When a buffer is empty (full) it is refilled (emptied) using RDFILE (WRFILE) with RELLOC=0. If a second buffer were assigned (third in the case of a write file) it will then be used, otherwise a call to FCHECK will be made in order to reuse the single buffer. Actual data transmission to or from a file is initiated each time one of its buffers is empty (full). I/O error checking is completed by a call to FCHECK in the case of a single buffered file or on a subsequent call to RDFILE or WRFILE for double buffered files.

Restrictions

Every call is a fixed length calling sequence so that each argument must be specified, either explicitly or by specifying -0. Only those arguments specifically stated as optional, by the minus (-) convention, may be specified by -0.

All buffers must be 432 words long and the location specified in the calling sequence must specify the lowest core location of the block because the data are loaded into the buffers in the forward direction.

A maximum of ten files may be open at any one time.

Usage

OPEN:

```
MAD: BFOPEN.(STAT,NAME1,NAME2,BUF1(432),-BUF2(432)-,-BUF3(432)-,ERR)

                    FAP:      TSX   BFOPEN,4
                              TXH   STAT
                              TXH   NAME1
                              TXH   NAME2
                              TXH   BUF1
                              TXH   -BUF2-
                              TXH   -BUF3-
                              TXH   ERR
```

STAT may be 'R' for read, 'W' for write - where 'R' or 'W' is left justified in the word. (Any status other than 'W' will be interpreted by BFOPEN to be the same as 'R' and passed to the file system as given in the call. Thus, a status of 'RW' will enable the user to read <u>and</u> write the file, using BFREAD for reading and WRFILE for writing. Because the BF-package considers the file open for reading only, calls to BFWRIT would result in an error return.

NAME1 NAME2 are the two locations containing the BCD name of the file.

BUFn is the beginning location of a 432 word buffer. Reading requires one buffer for single buffering and two for double. Writing requires two buffers for single buffering and three for double.

ERR is the location to which control will be transferred if an error is encountered either by the file system or by the buffering routines.

READ - WRITE:

```
MAD:  BFREAD.(NAME1,NAME2,A(N)...N,EOF,EOFCT,ERR)

      BFWRIT.(NAME1,NAME2,A(N)...N,ERR)

          FAP:     TSX   BFREAD,4      TSX   BFWRIT,4
                   TXH   NAME1         TXH   NAME1
                   TXH   NAME2         TXH   NAME2
                   TXH   A,,'n'        TXH   A,,'n'
                   TXH   EØF           TXH   ERR
                   TXH   EØFCT
          TXH    ERR
```

BFREAD(BFWRIT) transmits N words of data from (to) the current buffer assigned to file NAME1 NAME2 into (from) location A(N) through A(1).

N(or 'n') is the number of words to be transmitted.

EOF is the location to which control is transferred if the end of the data in the file is reached before N words can be transferred to location A(N) through A(1). For writing this does not apply since RELLOC = 0.

EOFCT is an integer variable into which is placed the number of words actually read when control was transferred to EOF.

CLOSE:

MAD: BFCLOS.(NAME1,NAME2, ERR)

BFCLOS is used to close an active file. If NAME1 NAME2 was a write file, the incomplete buffer will be added to the file before closing. If NAME1 is 'ALL' and NAME2 is -0, all active files will be closed.

ERRORS:

MAD: ERRCOD = BFCODE.(0)

```
          FAP:      TSX    BFCODE,4
                    STO    ERRCOD
```

BFCODE If called in the event of an error return, gives a non-zero code word (key below) if the error was detected by the buffering routines. If the error was detected by the file system, ERRCOD will be zero, in which case the user may call PRNTER or IODIAG to discover the nature of the error.

1. Too many active files - call to BFOPEN when ten files already were opened by BFOPEN.

2. Not enough buffers given - Call to BFOPEN to open a read (write) file and no (only one) buffers specified.

3. Attempt to (BF) read (write) a file not opened by BFOPEN.

4. Attempt to (BF) read (write) a file opened for writing (reading).

(END)

Identification

Old file system write-arounds to new file system

Purpose

In order to provide compatibility for programs (including many commands) written for the old file system, a set of write-arounds has been written which map the old disk calls into the new ones. These are available as library subroutines, and operate in core B. Unfortunately, this mapping is necessarily imperfect. Following is a list of the more painful and obvious discrepancies.

1. There is no .FILDR. The U.F.D. (FILE) can be opened and read with the same calls as any other files.

2. There is no double-buffering. Calls to .SEEK, .APPEND, and .ASIGN use only the first buffer specified in the call (the one specified in the address).

3. It is not possible to have more than one file with the same name. Therefore, a call to .ASIGN first deletes any file that already exists with the given name.

4. It is possible to create a file with a word count of 0. No telling how this incompatibility will show up.

5. Restrictions as to zero or non-zero relative addresses in calls to .READK/.WRITE following a .RELRW rather than .SEEK/.ASIGN have all been removed. Anything is legal.

6. All files which are specified to be written as R1 or R2 will be written as read-only, protected. Files which are created as read-only, protected will be treated as R1. There are no files with the former restrictions of R2.

A few conditions which formerly caused errors and no longer do were considered important enough to simulate. WARNING - since these error conditions are recognized by the write-arounds rather than by the file system, attempts to gain more information about the error (e.g. via PRNTER) will be misleading and meaningless.

1. "RELLOC too large" causes an EOF return from WRFILE, but an error return from .WRITE.

2. An FSTATE on a file in active write status gives valid information. For .FSTAT this results in an error return.

Error returns and error codes constitute the area of greatest inequality. The prefix of an error return is no longer significant (i.e. if an error return is provided, the comment is always suppressed). Also error codes meaning "file not found" (5), "too many active files" (2), and "track quota exhausted" (6) are translated, but all other errors are mapped into the catch-all code 1 (illegal calling sequence).

Approximate Mapping of Old Calls into New

.APEND	FSTATE to check for existence of file OPEN for Writing BUFFER
.ASIGN	DELFIL previous copy OPEN for writing BUFFER
.SEEK	OPEN for reading BUFFER
.RELRW	OPEN for reading and writing BUFFER
.LOAD	OPEN for reading RDFILE CLOSE
.DUMP	DELFIL OPEN for writing WRFILE CLØSE
.READK	RDWAIT
.WRITE	WRWAIT
.CLEAR	DELFIL OPEN for writing WRFILE n zeroes CLOSE
.FSTAT	FSTATE
.DLETE .ERASE	DELFIL

.FILE .ENDRD	CLOSE
.RENAM	CHFILE
.RESET	RESETF
.FILDR	"Subroutine not found"

Mapping of Modes

File Creation (.ASIGN)

Assigned mode	Resulting mode
Temporary	Temporary
Permanent	Permanent
Read-only, class 1 Read-only, class 2	Read only, Protected

File Testing (.FSTAT)

Actual mode	Mode returned by .FSTAT
Temporary	Temporary
Read-Only Protected	Read only, class 1
All others	Permanent

Note that a write-only file will appear to a program using .FSTAT to be permanent mode; the program may run into difficulty if it then attempts to read the file.

(END)

Identification

Change the mode or the name of a disk file.
CHMODE, RENAME, .RENAM

Purpose

To change the mode or the name of a disk file.

Usage

To change mode:
as library subroutine:

```
          FAP:  TSX  CHMODE,4
                PZE  FILNAM
                PZE  MODE


      FORTRAN:  A = CHMODE (FILNAM,MODE)
          MAD:  A = CHMODE. (FILNAM, MODE)
```

MODE is 0 for temporary, 1 for permanent, 2 for read only R1, 3 for read only R2. (R1 and R2 are READ-ONLY, PROTECTED).

A will be zero if successful or will contain the disk error code if the file cannot be found or changed.

To change name and/or mode:
as core-B write around

```
          FAP:  TSX  .RENAM,4
                OPN  FILNAM,,NEWNAM
```

To change name:
as library subroutine:

```
          MAD:  A = RENAME.(FILNAM, NEWNAM)
      FORTRAN:  A = RENAME (FILNAM, NEWNAM)
```

.RENAM replaces in the current file directory the file name specified by FILNAM by the new name located at NEWNAM by calling CHFILE. The standard supervisor error procedure may be followed.

OPN specifies the mode of NEWNAM. PZE is temporary, PON is permanent, PTW is R1, and PTH is R2. (R1 and R2 will be treated as READ-ONLY, PROTECTED files in the new system).

RENAME has two tries at changing the name of FILNAM to NEWNAM. If the first try fails because a file by the name of NEWNAM already exists, an attempt is made to delete this file with a call to the library subroutine DELETE. (if the first try fails for any other reason, AC will contain the error code from CHFILE). If the old version of NEWNAM cannot be deleted, AC will contain the error code from DELETE. When the old file NEWNAM has been deleted, the second try at renaming FILNAM is made. If this fails, AC will contain the error code from CHFILE.

If RENAME is successful the old file is given the new name and the mode is unchanged{ upon return from RENAME, AC will contain zero. If RENAME is unsuccessful, AC will contain the error code.

RENAME will not change the name of a linked file. If FILNAM is linked, an error code of octal 40(dec. 32) is returned in the signed AC.

(END)

Identification

Delete file from file directory
DELETE, ERASE, .DLETE, .ERASE

Purpose

To delete a file from a directory.

Usage

To delete a file:
core-B write around:

```
        FAP:  TSX  .DLETE,4
              PZE  FILNAM
```

as library subroutine:

```
        MAD:  EXECUTE DELETE.(FILNAM) or A = DELETE.(FILNAM)
    FORTRAN:  CALL DELETE (FILNAM)   or A = DELETE (FILNAM)
```

.DLETE calls the supervisor entry DELFIL. The FILNAM is removed from the current file directory and the tracks are made available for other use. Protected, read-only, write-only, or private files may not be deleted by this routine. Any error will invoke the supervisor error procedure.

DELETE calls the supervisor entry DELFIL. If the file is linked, a message will be typed asking if the file should really be deleted. If a 'linked' file is deleted, the link and file name still exist in the current file directory but the file to which they point is deleted. If the file (whether linked or not) is protected, read-only, write-only, or private, a message will be typed. Only the author may delete a protected file.

Upon return, if the file is not deleted the AC and A will contain an I/O error code, otherwise the AC and A will be zero.

To erase just the name:
as core-B write around:

```
          TSX   .ERASE,4
          PZE   FILNAM
```

as library subroutine:

```
      MAD:  EXECUTE ERASE.(FILNAM)  or A = ERASE.(FILNAM)
  FORTRAN:  CALL ERASE(FILNAM)      or A = ERASE(FILNAM)
```

ERASE is now the same as DELETE (.ERASE = .DLETE). In the earlier version of CTSS, as a result of a call to ERASE, the tracks were not made available for other use and the user's track count was not updated until the next time the disk was loaded.

(END)

Identification

Switch current file directory
COMFIL, COMFL, TSSFIL, USRFIL

Purpose

To allow the user to switch between his home file directory, common file directories associated with his problem number, or a public file directory.

Usage

To switch to a common file directory:
As supervisor or library entry:

```
            CAL   N
            TSX   COMFIL,4
            PZE   BUSY


            Optional:

     COMFIL   TIA   =HCOMFIL
```

N contains the integer of the common file directory desired. Zero is the user's home file directory.

BUSY It is no longer possible for a file directory to be "busy" but the calling sequence is preserved for compatibility. Control will always return to 2,4.

Unlike the old file system, active files are now not reset when a directory switch occurs.

As library subroutine:

```
         MAD:  COMFL.(N)      or EXECUTE COMFL.(N)
     FORTRAN:  CALL COMFL(N)
```

To switch to a public file directory:
As supervisor or library entry:

```
            TSX   TSSFIL,4     optional (TIA   =HTSSFIL)
          - PAR   PROB -
          - PAR   PROG -
          - PAR   LOC -
```

TSSFIL switches the user to the file directory named by PROB PROG. The user is permitted to switch into any of the following directories:

1) his home file directory
2) any public file directory
3) his current directory
4) any common file on his problem number, if he has common-file privilege

Any other values for PROB and PROG will result in an error. If the third argument is supplied, a transfer will be made to LOC; otherwise, the supervisor will print an error message and place the user in DORMNT status.

If the arguments PROB and PROG are not supplied, the user will be switched to the system public file directory, M1416 CMFL04. This directory is composed of links to certain files in the system file directory which are in read-only, protected mode. The record quota of the TSSFIL directory is 0, so that the user may not create files after a call to TSSFIL.

```
    TSX  USRFIL,4       optional (TIA   =HUSRFIL)
```

USRFIL restores the user to the directory he was in before the call to TSSFIL. If TSSFIL was not called, USRFIL does nothing.

Note: the library entries, TSSFIL and USRFIL, may be called from MAD or Fortran programs.

(END)

Identification

Query file status
FSTAT, .FSTAT

Purpose

To obtain the mode and word count of a specified file.

Usage

As supervisor or library entry:

```
          TSX  .FSTAT,4     optional (TIA   =H.FSTAT)
          PZE  FILNAM
```

As library subroutine:

```
     MAD:  A =  FSTAT.(FILNAM)
 FORTRAN:  A =  FSTAT(FILNAM)
```

.FSTAT If the file is not found, the supervisor disk error procedure is initiated.

Upon return from FSTAT, the AC or A will contain zero if the file was not found. Otherwise, it will contain a word of the form OPN WDCNT.

OPN is the mode of the file, PZE is temporary, PON is permanent, PTW is R1, PTH is R2.

WDCNT (the address and tag) is the word count of the file.

(END)

Identification

Get the name of next file
GTNAM

Purpose

If a program creates an unknown number of files, assigns them sequential primary names, and uses them in a push down list, it is necessary to be able to determine the next available primary name. GTNAM performs the search for the next available name.

Usage

As library subroutine:
FAP, MAD or FORTRAN

```
A   = GTNAM.($bCLASS$)
```

GTNAM searches for the first file which does not exist in the series of primary names ...001 thru ...999 with secondary name CLASS; then tries to delete the following file, if any; and returns in A the first BCD primary name available in the series.

(END)

Identification

Drop files from active status
.RESET, RESETF

Purpose

To remove all user's files in active status from the supervisor's list of active files.

Usage

Core-B write around:

```
                 TSX  .RESET,4
```

.RESET will remove all the user's active files from the active status. All files in active write status will be lost. All temporary files in active read status will be deleted. This call will not remove the user's active files from the library subroutines' list of active files.

As supervisor or library entry:

```
                 TSX  RESETF,4     optional (TIA  =HRESETF)
```

RESETF will remove all the user's active files from the active status. All files in active write status will be lost. All temporary files in active read status will be deleted. This call will not remove the user's active files from the library subroutines' list of active files.

(END)

Identification

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

Purpose

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

Usage

Change:

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

OLDNM1	OLDNM2 is the name of the file which is to be changed (right adjusted, blank padded). This file may not be in active status at the time of the change.
NEWMOD	is the desired mode of the file.
NEWNM1	NEWNM2 is the desired name of the file. NEWNM1 NEWNM2 may not be the same as OLDNM1 OLDNM2. To change just the mode, the new name must be specified as -0.

Error codes:

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

Delete:

```
DELFIL.($ NAME1$,$ NAME2$)
```

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

Error codes:

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

Status:

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

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

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

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

(N.B. "Open" means "opened by any user", <u>not</u> merely "opened by a caller".)

```
DEVICE is  1  Low speed drum
           2  Disk
           3  Tape
```

Error codes:

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

Size:

```
STORGE.(DEVICE,ALLOT,USED)
```

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

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

Error codes:

03. Illegal DEVICE specified
04. Machine or System error

Current UFD:

UPDATE.

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

03. Machine or System error

(END)

Identification

Historic File System Error Procedure

Purpose

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

Usage

Standard:

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

```
ILLEGAL CALL TO XXXXXX. NO ERROR RETURN SPECIFIED
```

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

User's option:

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

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

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

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

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

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

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

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

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

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

h. Tried to rename read-only class 2.

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

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

(END)

Identification

Library disk error procedure
SETERR, SNAP, RECOUP

Purpose

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

Method

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

Usage

SETERR:

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

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

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

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

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

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

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

Every call to SETERR supercedes the previous one.

RECOUP:

CALL RECOUP (ERCODE, IR4,-IND-)

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

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

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

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

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

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

SNAP:

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

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

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

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

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

(END)

Identification

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

Purpose

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

Method

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

Usage

EOFXIT:

The library routines call EOFXIT by:

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

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

SETEOF:

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

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

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

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

N must be set by an ASSIGN statement in Fortran

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

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

WRDCNT:

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

MAD or FORTRAN: CALL WRDCNT (LOC)

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

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

(END)

Identification

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

Purpose

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

Usage

EXIT, CLKOUT and ENDJOB are synonomous.

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

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

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

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

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

```
EXECUTE EXITM.
```

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

(END)

Identification

Error Exit from Math Library Routines
LDUMP

Purpose

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

Usage

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

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

ARG1	contains the first argument to the math library subprogram.
ARG2	contains the second argument, if any, to the math library routine.
NAME	contains the BCD name of the offending routine.
IN	is the return of 2,4 which the programmer should use if he is writing his own LDUMP and wishes to repeat the offended subprogram after he has corrected the error.
OUT	is the return of 3,4 which the programmer should use if he wishes to return from the offended routine without repeating its calculations.

(END)

Identification

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

Purpose

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

Usage

1. Standard:

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

2. Single return:

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

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

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

ERRLOC is the location to which control should be transferred if the I/O system detects an error. Upon entry to ERRLOC, index register 4 will contain the value set by the call to the I/O system that caused the error to be detected. To continue execution by ignoring the I/O call, transfer to 1,4. To continue execution by repeating the I/O call, transfer to 0, 4.

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

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

3. Individual returns:

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

4. Diagnostic information:

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

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

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

5. Printing of diagnostic:

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

B. Command: PRNTER -MASK-

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

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

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

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

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

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

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

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

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

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

C. Subroutine: PRDIAG.

PRDIAG will format and print the information supplied by IODIAG. No descriptive diagnostic is provided by PRDIAG; it is offered mainly for those situations where core space is at a premium.

Error codes

Standard error codes:

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

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

Input/output error codes:

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

1. Parity error reading or writing file
2. Fatal error reading or writing file, cannot continue
3. Available space exhausted on this device
4. Tape file not mounted or not available
5. Illegal operation on this device
6. Physical end of tape sensed while writing

   or

   Logical End of Tape of tape passed trying to open a file

   or

   End of tape file encountered unexpectedly.

(END)

Identification

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

Purpose

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

Usage

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

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

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

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

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

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

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

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

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

OPS may be indexing instructions.

SWITCH	is zero if the format is stored backwards and non-zero if the format is stored forward.
LIST,,ENDLST	are for standard list processing (see MOVE 1, 2, 3).
DIR	If zero, the format is stored forward. If one, the format is stored backward.
SWT	if zero with I format, the value is taken from the decrement of location LIST. If non zero with I format, the value is taken from the address of location LIST.
SYMTB	in a MAD call, refers to the start (bottom) of symbol table for this routine.
(FIL)	provides blank padding; with (SCH) to 80 characters and with (STH) to 132 characters.

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

(END)

Identification

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

Purpose

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

Usage

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

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

(TSH) and (TSHM) are synonymous.

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

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

OPS may be indexing instructions.

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

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

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

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

(END)

Identification

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

Purpose

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

Restrictions

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

Usage

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

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

N contains in the decrement the number of pseudo tape.

OPS may be indexing instructions.

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

(END)

Identification

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

Purpose

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

Restrictions

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

Usage

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

       MADTRN:  BACKSPACE N
                ENDFILE N
                REWIND N

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

      FORTRAN:  BACKSPACE N
                END FILE N
                REWIND N

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

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

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

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

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

(END)

Identification

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

Purpose

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

Restrictions

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

Usage

Mount:

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

MOUNT	must be used to direct the I/O system to mount a reel of tape on the unit to be subsequently referred to as UNIT.
CHAN	specifies which channel is desired. '1' specifies channel A; '2' specifies channel B; '0' or '-0' indicates "no preference".
UNIT	specifies a logical unit number (0 through 32767) by which the user will refer to this reel in other calls.
MESSAG	is the BCD message which will be printed for the operator in conjunction with the I/O system's mounting instructions. The message should contain information about "file protection" (write ring or no write ring) and reel identification. It should be stored "forwards" in memory; that is, the first word of the message should be in the highest-subscripted location of a MAD array. (This is not the order which MAD's VECTOR VALUE's statement normally furnishes, and it must be provided for.)

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

error codes:

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

Unmount:

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

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

UNIT is the logical unit number as defined by MOUNT.

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

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

error code:

03. Tape file in use.

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

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

UNIT has previously been defined by a call to MOUNT.

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

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

Error codes:

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

C5. Mount failed - illegal operation
(key code 11).
06. Mount failed - operations difficulties
(key code 12).

Label verification:

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

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

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

Error codes:

03. Tape file does not exist.
04. Machine error or bad status.
05. Mount failed - illegal operation
(key code 11).
C6. Mount failed - operations difficulties
(Key code 12).
C7. Labels do not match.

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

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

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

error codes:

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

Additional Information

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

Mount-tape requests are not queued. Thus before any MOUNT request is considered, the tape operator must have complied with any previous MOUNT request. Calls to MOUNT will result in "Tape-wait" status if another mount is already pending. Calls to LABEL or VERIFY when the tape in question is not yet mounted (mount pending) also result in tape-wait status.

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

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

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

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

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

Format of Tapes

BTL (header label)

End of File mark

Data File 1

End of File mark

EOFL (end of file label)

End of File mark

BTL (header)

End of File

Data File 2

.
.
.
.
.

Data File n

End of File mark

EOFL

End of File mark

EOLTL (end of logical tape label)

End of File mark

Format of BTL

Word(s)	Contents	Description
1	3Ebb60	Words 1-2 constitute
2	ObBTLb	Beginning of Tape label
3	MITMAC	
4	000000	
5	xxxxxx	File number on tape (in binary)
6	000000	
7	xxxxxx	Date file created (file system format)
8	xxxxxx	Number of days file is to be retained (usually . . . 999)
9-10	xx...x	File name
11-14	xx...x	User supplied label

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

Format of ELTL

1	bEOLTb
2-14	00...0

Format of EOFL

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

Format of Data_File_i

File i consists of 433-word records where

word	1	(Address of first word) = number of the record within this file. (Decrement of first word) = 0 unless this is the last record of the file. Then it equals the number of words in this last record excluding word 1.
words	2-433	User supplied data.

(END)

Identification

Program status
DEAD, DORMNT, GETILC, FNRTN

Purpose

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

Usage

DEAD: as supervisor or library entry:

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

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

DORMNT: as supervisor or library entry:

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

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

GETILC: as supervisor entry:

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

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

FNRTN: as supervisor entry:

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

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

Restrictions

DEAD, DORMNT and FNRTN result in an automatic logout if called from FIB.

DEAD, DORMNT and FNRTN may result in the execution of a command (subsystem), depending on the settings of the user's subsystem status words and options. Refer to sections AG.8.05 and AH.10.03 for details.

(END)

Identification

Interrupt execution for specified time
SLEEP, WAIT

Purpose

Allow a user program to place itself in dormant status, input-wait status, or timer-wait status, and be restarted automatically after a specified time.

Usage

Periodic dormancy:
As a supervisor or library entry

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

The program is placed in dormant status, and is restored to working status after 'n' seconds have passed.

General form:
As a supervisor or library entry

```
        WAIT.(MODE, N)

        TSX     WAIT,4        (TIA     =HWAIT)
        PAR     MODE
        PAR     N
        ...
   N    PZE     'n'
```

The program is placed in a waiting status as specified by MODE, and will be restarted after 'n' seconds have passed. (If 'n' is 0, it will not be restarted.) MODE is interpreted as follows:

0 - Timer-wait status: the program will be restarted after 'n' seconds. No commands are accepted. Input lines are saved; the program is not restarted when input lines arrive.

1 - Input-wait status: the program will be restarted after 'n' seconds have elapsed or when an input line is completed. If 'n' is zero, the program will be restarted only when an input line is completed.

2 - Dormant status: the program will be restarted after 'n' seconds. An input line while dormant is interpreted as a command. This mode is equivalent to SLEEP.

(END)

Identification

Interrupt levels
GETBRK, SETBRK, SAVBRK

Purpose

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

Restrictions

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

Method

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

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

Usage

SETBRK:
as supervisor or library entry:

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

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

SAVBRK:
as supervisor or library entry:

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

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

GETBRK:
as supervisor or library entry:

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

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

(END)

Identification

Storage Map
STOMAP

Purpose

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

Usage

As library subroutine:

```
TSX  STØMAP,4
```

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

(END)

Identification

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

Purpose

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

Method

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

Usage

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

```
TSX   .SETUP,4
```

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

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

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

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

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

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

(LFTM) leaves the floating-point trapping mode.

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

(END)

Identification

Memory allotment
GETMEM, SETMEM, GMEM, SMEM, EXMEM

Purpose

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

Method

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

Restrictions

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

Usage

As supervisor or library entries:

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

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

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

SETMEM sets the memory allotment to the value of n (low order 15 bits). If n is (77777)8, all of memory is allotted, including location (77777)8.

As library subroutines:
MAD or FORTRAN:

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

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

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

J contains an integer giving the memory bound desired.

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

SMEM sets the memory bound to the value desired.

To extend memory bound:

As library subroutine:

MAD, FORTRAN or FAP:

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

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

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

(END)

Identification

Free or erasable storage management
FREE, FRER, FRET

Purpose

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

Usage

As a library subroutine:

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

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

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

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

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

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

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

(END)

Identification

Reset file-wait return
TILOCK

Purpose

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

Usage

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

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

RETURN	is the location to which control will be transferred if an I/O call would normally result in file-wait. If RETURN is zero, the normal execution of file-wait will be reinstated.
OLDRTN	upon return, the AC will contain the address of the previous return setting, if any.

(END)

Identification

Get array from free storage
GETBUF

Purpose

To allow a MAD program to obtain buffer space by extending memory bound, and to address the storage area obtained as a subscripted array. This permits SAVED files of freshly loaded programs to be reduced in size, since the buffer area is not included in the SAVED file.

Usage

To obtain a buffer:

```
DIMENSION BUF(0)
A = GETBUF.(BUF, SIZE)
```

A block of core storage of length SIZE+1 is obtained by extending the memory bound. The value of BUF is set to the absolute address of BUF less the absolute address of the last addressable location of this block (i.e. old memory bound + SIZE), expressed in two's complement form, modulo 2.P.15. The old memory bound is returned in A.

Elements of the array obtained by GETBUF may be referenced by

```
BUF(BUF + I)
```

(for the Ith element), where I may have a value from 0 to SIZE. Multiple subscripts may also be used. Dimension declarations will be of the form:

```
DIMENSION BUF(0, BDIM)
VECTOR VALUES BDIM = (dimension vector, see MAD manual)
```

References are of the form:

```
   BUF(I, BUF+J)
or BUF(I, J, BUF+K)
   etc.
```

The last subscript (in the case of standard subscripts) is always the one to which the address contained in BUF is added.

To return buffer to free storage:

```
SMEM.(A)
```

where A is the old memory bound previously returned by GETBUF.

N.B. Beware of the following:

```
A = GETBUF.(B1, S1)
B = GETBUF.(B2, S2)
SMEM.(A)
```

This will release buffer B2 as well as B1, since the SMEM call resets the memory bound below both buffers.

Example:

Assign a buffer to a file

```
DIMENSION B(0)
OPEN.($R$, NAME1, NAME2)
GETBUF.(B, 432)
BUFFER.(NAME1, NAME2, B(B+432)...432)
```

(END)

Identification

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

Purpose

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

Restrictions

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

Usage

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

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

As library subroutine:

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

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

Get the contents of a block of A-core:
As supervisor or library entry:

```
          FAP:  TSX  GETLOC,4      optional (TIA   =HGETLOC)
                PZE  LOC,,'n'
                PZE  BUF
             or
                TSX  GETARY,4
                PZE  LOC
                PZE  BUF,,'n'

          MAD:  GETARY.(LOC, BUF(N)...N)
```

As a library subroutine:

```
          MAD:  GLOC.(LOC, BUF(N)...N)
```

Upon return, the 'n' word array beginning at BUF for a FAP call or BUF(N) for a MAD call will be set to the contents of the 'n' words of supervisor core beginning at LOC.

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

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

  As library subroutine:

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

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

Get a supervisor parameter:
  As library subroutine:

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

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

SYPAR returns a supervisor parameter in the AC.

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

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

(END)

Identification

Get common file number
GETCF, GETCFN

Purpose

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

Usage

As a supervisor entry:

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

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

As a library subroutine:

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

 FORTRAN:  CFS = GETCFN(CFN)

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

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

Restriction

If a user switches to a common file, and then uses ATTACH (command or file system call) to switch to another user's directory, GETCF will return the number of the common file to which he was switched, and give no indication of his current attached directory.

(END)

Identification

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

Purpose

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

Method

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

1	User may use common files
2	User may use privileged calls to the I/O system.
4	User may modify "PROTECTED" files of other users.
10	User may refer to "PRIVATE" files of other users.
20	User may modify the supervisor and I/O system.
40	User may use the ESL display routines.
100	User may use the 6.36 supervisor entries.
200	User may not use disk-loaded commands, except LOGIN and LOGOUT ("Restricted User", see Section AA.1).
400	User may not alter file directory (not yet implemented)
1000	User may modify standard options, subsystem status (see AG.8.05).
2000	User may remain logged in after system comedown initiated (system operators only).
1000000	User is background system.
2000000	User is foreground.
4000000	User is FIB.
10000000	User is incremental dumper.
20000000	User is privileged command.

A privileged command sets the 1, 2, 4, 10, 20 and 1000 bits on.

A command loaded while option bit 40 is on (see AG.8.05) sets the 1000 restriction code bit, making the command "subsystem privileged".

The bits which occupy the decrement may be moved left nine bit-positions to indicate the .not. condition, except in the case of the privileged command bit.

<u>Usage</u>

Update MFD:

```
                  UPDMFD.($ PROBN$,$  PROG$)
```

UPDMFD	places a new user (problem number programmer number) in the master file directory. With this call it is possible to update the MFD during time sharing rather than having to wait for a disk editor run.
PROBN	is the right adjusted problem number of the form ANNNN. A is an alpha character, and NNNN is a four digit number.
PROG	is a one to four digit programmer number. Note the right adjustment and blank padding.

Error codes:

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

Delete from MFD:

```
                  DELMFD.($ PROBN$,$  PROG$)
```

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

Error codes:

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

Attach to UFD:

ATTACH.($ PROBN$,$ PROG$)

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

Error codes:

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

Quota allotment:

ALLOT.(DEVICE,QUOTA,USED)

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

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

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

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

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

Error codes:

03. Illegal device specified

Move a file:

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

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

Error codes:

03. File not found in current U.F.D.
04. (Unused code)
05. File is 'PROTECTED'
06. File already exists in 'PROGN PROG'
07. Machine or System error
08. File already active.
09. Other U.F.D. not found
10. Illegal use of M.F.D.

Link to a file:

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

LINK	establishes a link in the current file directory to a file in some other file directory. Links may be established to the maximum depth of two, as specified by the supervisor.
NAME1	NAME2 is the name which will be used to refer to the file in the current file directory.
PROBN	PROG specifies the file directory to which the link is being made. This file directory may contain the actual file or it may contain a link to some other directory.
NAM3	NAM4 is the name by which the file is known in file directory PROBN PROG. If NAM3 NAM4 is not specified, it is assumed to be the same as NAME1 NAME2.
MODE	is an integer or integer variable which will be 'OR'ed with all the modes through all the links to the actual file. The resulting 'OR'ed mode will be used as the mode in the current file directory.

Error codes:

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

Remove a link:

```
UNLINK.($ NAME1$,$ NAME2$)
```

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

Error codes:

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

Date a file:

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

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

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

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

Error codes:

03.	Illegal device
04.	Machine or system error
05.	File is a link

Unlock a file:

```
RSFILE.($ NAME1$, $ NAME2$)
```

RSFILE is used to reset the ILOCK field in a file entry when, due to machine or system error, a file has become interlocked while no user is using it. This entry may only be used by system programmers privileged to patch the supervisor, and only while key 22 is down on the operator's console (to prevent accidental calls).

Error codes:

03. File not found
04. Linked file not found
05. Link depth exceeded
06. File is an active file
07. System or machine error

(END)

Identification

Get directory attached to
ATTNAM

Purpose

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

Usage

As a supervisor or library entry:

```
     MAD:   ATTNAM.(A(N)...N)       [N.LE.4]

     FAP:          TSX   ATTNAM,4
                   PAR   A,,'n'        or:    PTW   A,,N
                                                .
                                                .
                                                .
                                          N   PZE   'n'
```

Optional:

```
           ATTNAM TIA   =HATTNAM
```

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

MAD	Contents	FAP
A(N)	PROBNO	A
A(N-1)	PROGNO	A+1
A(N-2)	AUTHOR	A+2
A(N-3)	FPRIOR	A+3

where PROBNO-PROGNO is the user's currently attached file directory, FPRIOR is his file priority setting (set by SETPRI), and AUTHOR is his author number, in binary.

Only the standard error code 01 may be returned.

(END)

Identification

Obtain user status information from supervisor.
WHOAMI

Purpose

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

Usage

As supervisor or library entry:

```
    MAD:
            WHOAMI.(A(N)...N)      [N.LE.7]

    FAP:
            TSX    WHOAMI,4
            OPN    A.,'n'          (OPN=PZE or TXH; n .LE. 7)

    Optional:

    WHOAMI TIA    =HWHOAMI
```

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

MAD	Contents	FAP
A(N)	PROBNO	A
A(N-1)	PROGNO	A+1
A(N-2)	SYSNAM	A+2
A(N-3)	IDCODE	A+3
A(N-4)	LOGIN	A+4
A(N-5)	UFDNM	A+5
A(N-6)	UNAME	A+6

where PROBNO is problem number, PROGNO is programmer number, SYSNAM is the six-character system name of the currently operating version of CTSS, IDCODE is the console identification code, LOGIN is the name of the login command (changed during test sessions), UFDNM is the user's home file directory, and UNAME is the user's name (last six characters only).

(END)

Identification

Find named items in supervisor
COMLOC, SNATCH, GAC, ACORE

Purpose

A user program often wishes to know the location in core A of some supervisor data item. COMLOC returns the A-core location of any variable in CTSS common. SNATCH copies supervisor common into core B for later examination by GAC. ACORE returns the location in core A of any supervisor entry point and the load origin of the module containing the entry.

Usage

As library entries:

```
LOC = COMLOC.(SYMBOL, -ERR-)
```

COMLOC is called with the left-adjusted BCD name of a symbol in CTSS common. It returns the integer value which is the location of the symbol in core A.

ERR is the location to which a transfer is to be made if SYMBOL is not found. If ERR is not supplied and SYMBOL is not found, COMLOC will print an error comment and return zero.

The first time COMLOC is called, it switches to the system public file by a call to TSSFIL and reads in the current system common symbol table, extending the memory bound and packing the table into core. (The common symbol table is named "COMx00 SYMTB" where "x" is the fourth letter of the current system name returned by WHOAMI.) COMLOC then searches the table for a symbol matching its first argument. Subsequent calls to COMLOC do not require re-reading the symbol table.

```
SNATCH.
CONTS = GAC.(SYMBOL, -OFFSET-)
```

SNATCH on the first call, calls COMLOC to determine the size of CTSS common, extends the memory bound to make room for it in core B, and calls GETARY to move all of supervisor common to core B. Subsequent calls to SNATCH just call GETARY to refresh the saved copy of supervisor common.

GAC retrieves the contents of SYMBOL+OFFSET at the time of the last call to SNATCH by calling COMLOC.(SYMBOL), adding the integer OFFSET (if supplied), and looking in the saved copy of supervisor common. If SYMBOL is not found by COMLOC, an error message is printed and zero is returned. Since GAC does not call the supervisor or do I/O, it is very fast.

```
          WORD = ACORE.(NAME, -ERR-)
```

ACORE is called with the left-adjusted name of a CTSS module entry point. It returns a word which has the location in core A of the entry in the decrement, and the location of the origin of the module containing the entry in the address. If ERR is supplied and NAME is not found, a transfer will be made to the label ERR. If NAME is not found and no error return is specified, an error message is printed and zero is returned.

The first time ACORE is called, it switches to the system public file and reads the file "(LOAD FILE)" into core, packing it and extending memory bound as necessary. This file is a complete description of how the CTSS supervisor was loaded; it is written by the system loader every time the CTSS system is brought up. ACORE then returns to the previous directory by a call to USRFIL, and searches the core copy of the loading information for an entry name matching its first argument. Subsequent calls to ACORE do not require rereading of "(LOAD FILE)".

<u>Examples</u>

To find the number of users logged in:

```
          NU = GLOC.(COMLOC.($NUSERS$))
```

To print out the names of all logged-in users:

```
          SNATCH.
          T'H LL, FOR I = 1, 1, I .G. 40
          W'R GAC.($PROBN$, I) .E. 0, T'C LL
          PRMESS.(GAC.($UNAME$, I))
     LL   C'E
```

Note that the UNAME and PROBN arrays will be consistent.

(END)

Identification

User A-core variable
SETWRD, GETWRD

Purpose

Each logged-in user has one location in core A in the supervisor common vector "UARRAY". The GETWRD and SETWRD entries are provided so that the user may examine and set this location. The CTSS supervisor makes no use of this location; it is provided for such applications as multi-pass compilers, which may wish to pass options or success and failure indications from one pass to another.

Usage

As a supervisor or library entry:

```
          TSX  SETWRD,4      optional (TIA  =HSETWRD)
          PAR  WORD
```

This call will set the user's UARRAY location to the contents of WORD. The previous value will be returned in the logical AC.

```
          TSX  GETWRD,4      optional (TIA  =HGETWRD)
         -PAR  USERNO-
```

This call will return the contents of the UARRAY location beloging to USERNO in the logical AC. If USERNO is not specified, the current user's UARRAY contents will be returned.

Both SETWRD and GETWRD can be called by MAD or FORTRAN programs.

It is possible to use these entries for inter-user communication, since one user may look at another's UARRAY location. For example, to examine the UARRAY location belonging to PROB PROG, the following MAD code will work:

```
INDEX = ISIN.(PROB, PROG)
W'R INDEX .E. 0, T'O NOTIN
HISWRD = GLOC.(INDEX + COMLOC.($UARRAY$))
   or
HISWRD = GETWRD.(INDEX)
```

The other user's UARRAY value will be returned in the variable HISWRD. If the user PROB PROG is not logged in, the program will transfer to the label NOTIN.

(END)

Identification

Blip character
GETBLP, SETBLP

Purpose

The CTSS supervisor has a feature which allows the user to request that a sequence of characters be typed every few seconds of execution. The SETBLP and GETBLP entries are provided to set the character sequence and time interval, and to find out their current value.

Usage

As a supervisor or library entry:

```
          TSX  SETBLP,4     optional (TIA  =HSETBLP)
          PAR  CHARS
          PAR  N
```

This call will set the blip sequence to the three 12-bit characters contained in CHARS. The blip will be typed every N seconds. If N is zero, the blip feature is inhibited. (This is the state when the user first logs in.)

```
          TSX  GETBLP,4     optional (TIA  =HGETBLP)
          PAR  CHARS
          PAR  N
```

This call will return the current blip setting in CHARS and the current blip interval in N.

(END)

Identification

Get line number of logged-in user
ISIN

Purpose

All per-user arrays in CTSS supervisor common are indexed by a "line number" or "logical unit number" which is assigned to a user when he dials up. The maximum value for this index is "N", an assembly parameter for the supervisor. ISIN returns the logical unit number for a user, given his problem and programmer number.

Usage

As a supervisor or library entry:

```
        TSX  ISIN,4     optional (TIA  =HISIN)
        PAR  PROB
        PAR  PROG
        SLW  LUN
```

ISIN returns the logical unit number of PROB PROG in the AC. If PROB PROG is not logged in, zero is returned.

ISIN may be called by MAD or FORTRAN programs.

(END)

Identification

General discussion of MACRO command programs

Purpose

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

Discussion

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

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

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

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

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

(END)

Identification

Single command
XECOM, NEXCOM, NCOM

Purpose

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

Usage

NEXCOM:
as supervisor entry:

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

as library subroutine:

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

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

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

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

XECOM:
as library subroutine:

```
    MAD, FORTRAN, FAP:
        K =  XECOM.  (COMAND,LIST)
        EXECUTE XECOM.  (COMAND,LIST)
```

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

LIST is any legal list specifying locations which contain the BCD names of the arguments appropriate to the command. Right justification is not necessary but the number of items in the list must be .LE. 18.

K	will be zero if execution was successful; non zero if failure.
XECOM	builds a chain of SAVE, COMAND, RESUME and calls CHNCOM. Thus control will be returned to the calling program after execution of COMAND, if COMAND called CHNCOM.

(END)

Identification

MACRO command
SCHAIN

Purpose

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

Reference

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

Usage

MAD, FORTRAN or FAP:

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

FILNAM	specifies the BCD file containing the chain of commands to be executed. The secondary name need not be BCD as is required for RUNCOM.
ARG'S	are locations of BCD names of specific arguments to be substituted for the dummy arguments specified by the CHAIN pseudo-command. They may be single or list variables and the names need not be right justified.
A	Upon return may contain a word of the form...XXX, which is not an error, but the primary name of a SAVED file representing the last dormant status yielded by the last command in the chain.
SCHAIN	will intersperse SAVE's and RESTOR's or RESUME's so that the chain specified in FILNAM may be of any length. Control is returned to the calling program upon completion of the chain. The chain may include any number of RUNCOM specifications, since nesting and recursion are possible.

(END)

Identification

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

Purpose

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

Method

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

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

Restrictions

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

Usage

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

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

As library subroutine:
MAD or FORTRAN:

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

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

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

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

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

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

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

As library subroutine:
MAD or FORTRAN:

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

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

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

BUFF must be at least 20 words long for GETCLS.

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

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

As a library subroutine:

MAD or FORTRAN:

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

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

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

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

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

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

As library subroutine:
MAD or FORTRAN

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

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

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

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

To initiate or continue a chain:
    As supervisor entry:

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

    As library subroutine:
        MAD or FORTRAN:

```
                EXECUTE CHNCOM (J)
```

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

J or j	j=0 specifies to CHNCOM that no core image is available for the next command. j=1 means that a core image is available and may be used by the next command.
CHNCOM	determines whether or not another command exists in the chain. If one exists, it is initiated. If no chain exists; DORMNT is called if j=1, DEAD is called if j=0.

(END)

Identification

Fetch a current command argument
GETCOM, COMARG

Purpose

To extract the Nth argument from the current command buffer.

Usage

As supervisor or library entry:

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

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

As library subroutine:
MAD, FORTRAN or FAP:

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

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

(END)

Identification

Specify user options, subsystem status
SETOPT, RSOPT, LDOPT, GETOPT, SETSYS, GETSYS

Purpose

To allow a user or his subsystem to modify the settings of his standard options, subsystem name, and subsystem condition mask. Also to allow a user to examine his current options and subsystem status.

Discussion

Associated with each user, there are three status words maintained in the supervisor containing his standard options, his subsystem name, and his subsystem condition code mask and last condition code.

User standard options occupy a half-word (18 bits), and are interpreted as follows:

```
+-----------------+-----------------+
|                 |  user  options  |
+-----------------+-----------------+
```

1	Search user UFD first for command
2	Search user or system files (not both) for command
4	RESETF if command resets dormant prog.
10	User subsystem trap enabled
20	Inhibit quit signals for user
40	Current user program is subsystem
100	Automatic save before loading subsystem
200	User is 'dialable'

The two low order bits are taken together to specify four modes of command file searching:

0	Search system files then user files (normal mode)
1	Search user files then system files
2	Search system files only
3	Search user files only

The following disk-loaded commands are always taken from the system files (provided that the user is allowed to use them):

```
LOGIN
LOGOUT
OTOLOG (user may not issue)
DAEMON (incremental dumper only)
DSDUMP (incremental dumper only)
DSLOAD (incremental dumper only)
FIBMON (FIB user and FIBMON only)
OPTION (subsystem-privileged user only)
```

The RESETF bit specifies that if there is a dormant core image left from the last command, and the command currently being processed does not preserve this core image (i.e. not SAVE, MYSAVE, START, RSTART, SUBSYS, ENDLOG, RESETF, or any B-core transfer command: USE, DEBUG, PM, etc.), any active files will be reset by a call to RESETF instead of being closed normally. This provides compatibility with previous versions of CTSS.

The subsystem trap enable bit causes all program calls going to DEAD or DORMNT (including errors) to simulate a call to NEXCOM for the command SUBSYS, provided that the call does not come from the user's subsystem (option bit 40 off), and causes all new commands issued from the terminal to pass through the subsystem processor (with the exception of exempt commands).

The quit-inhibit bit causes all quit signals to be ignored for the user. Program status will be unaffected if the user attempts to quit and buffered output will not be reset. N.B. The only way to stop a non-quittable program that has gone into a loop is to force an automatic logout by hanging up the data-phone (or turning off power to the terminal). Use this feature at your own risk!

The subsystem execution bit, if on at command load time, causes a new core image being loaded to have subsystem privileges if the user does not have the subsystem privilege himself. Program calls going to dead or dormant status will execute normally if this bit is on, regardless of the setting of the subsystem trap bit.

The subsystem save bit if set causes the subsystem processor to simulate a 'MYSAVE progn T' before it loads the subsystem.

The dial-permit bit allows remote terminals to attach to the user via the DIAL command. See section AH.1.05 for details.

The user's subsystem name is interpreted as a six-character command name, which may be any system command or a user disk-loaded command (SAVED file).

```
+-----+-----+-----+-----+-----+-----+
|           subsystem  name         |
+-----+-----+-----+-----+-----+-----+
```

The subsystem condition code mask is a half-word quantity split into two 9-bit fields. The high order 9 bits are examined by the subsystem processor if the user has a core image left; the low order 9 bits are examined if there is currently no core image. Within each 9-bit field, the bits are interpreted as follows:

1	Trap new command
2	Trap direct program call ('DEAD', 'DORMNT')
4	Trap CHNCOM if end of chain or no chain set up
10	Trap error condition (file system, PMV, etc.)

The subsystem condition code occupies the high order 18 bits of the subsystem condition mask word. The low order 9 bits of these 18 indicate which of the possible subsystem trap conditions occurred to cause the subsystem processor to be entered (zero if the SUBSYS command was issued directly by the user or his program). The following 8 bits specify an error code if the subsystem condition code was 10 ('error'), in order to indicate the type of error that occurred. This is not yet implemented, and the error code will be returned as 0. The high order (sign) bit is on if there was a dormant core image left.

```
++--------+---------+---------+---------+
|| error  |  code   | condition  mask   |
++--------+---------+---------+---------+
```

<u>Usage</u>

To set (turn on) bits in the option status word:

As a library entry ...

```
FAP:      TSX  SETOPT,4   or      TSX  SETOPT,4
          VFD  O36/'bits'         PAR  BITS
                                  ...
                            BITS  VFD  O36/'bits'

MAD:      A = SETOPT.(BITS)
```

The bits specified as 'bits' will be ORed with the current contents of the user's option word and the result will replace bits 18-35 (right half) of the option word. The previous value will be returned in the accumulator.

To reset (turn off) bits in the option word:

As a library entry ...

```
FAP:      TSX  RSOPT,4     or      TSX  RSOPT,4
          VFD  O36/'bits'          PAR  BITS
                                   ...
                           BITS    VFD  O36/'bits'
```

```
MAD:      A = RSOPT.(BITS)
```

The specified bits will be masked out of the current contents of the option word, and the result will replace bits 18-35 of the option word. The previous contents will be returned in the accumulator.

To set the contents of the option word:

As a supervisor or library entry ...

```
FAP:      TSX  LDOPT,4     or      TSX  LDOPT,4
          VFD  O36/'bits'          PAR  BITS
                                   ...
                           BITS    VFD  O36/'bits'
```

```
MAD:      A = LDOPT.(BITS)
```

The specified bit configuration will replace the current contents of bits 18-35 of the option word. The old value will be returned in the accumulator.

To examine current option settings:

As a supervisor or library entry ...

```
FAP:      TSX  GETOPT,4      (optional TIA =HGETOPT)
        - SLW  A -
```

```
MAD:      A = GETOPT.(0)
```

Location A and the accumulator will contain the settings of all available options in bits 18-35. In addition, the left half will contain status flags pertaining to the user's current core image. In particular, bits 12-17 specify the current typewriter input mode as follows:

0	6-bit mode
1	12-bit mode
2	No-convert mode
4	No-break mode
10	Graphic input mode

Also, bit 11 will be on if the core-B simulated interval timer is running.

To specify subsystem name and condition mask:

As a supervisor or library entry ...

```
FAP:     TSX  SETSYS,4      (optional TIA =HSETSYS)
         PAR  COMMND
         PAR  MASK

MAD:     SETSYS.(COMMND, MASK)
```

The user's subsystem name will be replaced by COMMND; the subsystem condition word will be set to the contents of MASK. Option bit 10 (subsystem trap enable) is set by this call.

To examine subsystem status:

As a supervisor or library entry ...

```
FAP:     TSX  GETSYS,4      (optional TIA =HGETSYS)
         PAR  COMMND
         PAR  MASK

MAD:     GETSYS.(COMMND, MASK)
```

COMMND will contain the user's current subsystem name. MASK will be set to the contents of the subsystem condition word. Example: If the subsystem condition word contains 400004004016, this indicates that the subsystem is to be called in for any call to CHNCOM attempting to go dead or dormant because there is no chain set up (004004 mask), for any error not leaving a core image (000010 mask) and for a program call to DEAD (000002 mask); the subsystem will not be called in for any new command from the terminal (except SUBSYS of course), for an error leaving a core image, or for a program call to DORMNT. The condition code of 400004 indicates that the user's program called CHNCOM and fell out because no chain was set up, and that the call to CHNCOM specified a core image (400000 bit on).

Restriction

Only 'GETOPT' and 'GETSYS' may be called by a subsystem-restricted user from any program (or command) other than his subsystem.

(END)

Identification

Trace of Subroutine Calls.
ERROR

Purpose

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

Restrictions

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

Each subprogram executed must have at least one argument.

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

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

Usage

MAD, FORTRAN, or FAP:

```
ERROR.(MESS)
```

MESS	is a BCD fenced message of .LE. 132 characters which will be printed on the user's console when ERROR is entered.
ERROR	will trace back to the main program through the last subroutine calls and print comments of the following type and then return control to the calling program.

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

(END)

Identification

BCD or spread-octal to binary
BCDEC, BCOCT

Purpose

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

Usage

BCD to binary:
As library subroutine:

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

       MAD:  NUM = BCDEC. (X)

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

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

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

Spread-octal to binary:
As library subroutine:

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

       MAD:  NUM = BCOCT.(X)

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

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

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

(END)

Identification

Binary to BCD
DEFBC, DELBC, DERBC

Purpose

To convert a binary integer to BCD with leading zeros.

Usage

As library subroutine:

MAD or FORTRAN:

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

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

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

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

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

(END)

Identification

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

Purpose

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

Usage

As library subroutine:

MAC or FORTRAN:

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

X	contains the binary number to be converted
A	will contain the converted value in spread octal, i. e., six bits for each octal digit (0-7).
OCABC	converts the address field of X to 5 digits with leading blank.
OCDBC	converts the decrement field of X to 5 digits with leading blank.
OCLBC	converts the left half of X to 6 digits.
OCRBC	converts the right half of X to 6 digits.

(END)

Identification

Justification and padding
BZEL, ZEL, LJUST, RJUST

Purpose

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

Usage

Justification library subroutines:

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

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

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

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

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

Interchange leading zero and blanks, library subroutine:

```
        MAD, FORTRAN or FAP:

              A = BZEL (B)          A = ZEL (B)
```

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

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

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

(END)

Identification

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

Purpose

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

Reference

CC 186 FORTRAN and MAD Format Specifications Spall

Method

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

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

Usage

Output, binary to BCD:

```
     Fortran:                                  MAD:
              TSX   USRSTH,4                        TSX   USRSTH,4
              PZE   FORMAT,,SWITCH                  PZE   FORMAT,,SWT
       RTN    .                                RTN   .
              .                                     .
              LDQ   SYMBOL,t                        STR   FIRST,,LAST
              STR                                   .
              .                                     .
              .                                     .
              TSX   (FIL),4                         STR   0
              .                                     .
              .                                     .
    USRSTH    Set upper                     USRSTH  Set upper
                core locs                             core locs
              TRA*  (IOH)                           TRA*  (IOH)
       OUT    .                                OUT   .
              .                                     .
              TRA   2,4                             TRA   2,4
```

Input, BCD to binary:

Fortran:

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

MAD:

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

FORMAT	is the beginning location of the desired format.
SWITCH	is zero if the format is enclosed in parentheses and stored backwards in core. SWITCH is non zero if the format is enclosed in parentheses and stored forward in core (e.g. BCI).
SWT	is zero if format is forward. SWT is one, if the format is stored backward.
SYMBCL,t	locates the variable to be converted. A loop may be included here for arrays or a series of LDQ, STR. After each variable is converted by (IOH), return is made following the STR in order to find the next variable to be converted.
FIRST	is the starting location of the list.
LAST	is the final location of the list. LAST may be lower in core than FIRST. If the list is of length one, LAST is zero.
(FIL)	is called to indicate that all the output data has been converted and the current buffer should be truncated.

STR 0 terminates the list in a MAD call.

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

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

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

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

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

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

IOHSIZ:

MAD, FAP, or FORTRAN

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

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

STQUO:

MAD, FAP, or FORTRAN

TSX STQUO,4

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

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

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

(END)

Identification

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

Purpose

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

Usage

As a library subroutine:

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

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

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

I — is a full word (MAD) integer.

J — is a decrement (FORTRAN) integer.

A — is equivalent to J.

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

```
MAD INTEGER EXCEEDS 32767
```

MINT — converts from decrement integer to full word.

(END)

Identification

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

Purpose

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

Usage

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

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

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

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

(END)

Identification

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

Purpose

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

Usage

As library subroutine:

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

The FAP call may also simulate FORTRAN and MAD calls:

```
           FORTRAN                          MAD

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

where

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

1) In a FORTRAN (or FORTRAN simulated) call, it may be a setting from a call to SETFMT. If SETFMT is not used it should be the H-specification of the format statement, e.g.,
   A = DECODE(5H(5I3),TEXT,list)
The format is expected stored in reverse order with FMT pointing to the first location (normal FORTRAN compilation).

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

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

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

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

(END)

Identification

Binary/BCD Conversion
DTBC, OTBC, BTDC, BTOC

Purpose

Convert decimal or spread-octal BCD numbers to binary; convert binary to decimal or spread-octal BCD.

Restriction

These routines are usable from FAP programs only. They may not be called directly from MAD or FORTRAN programs, since the calling sequences are incompatible.

Usage

Decimal-to-binary conversion

```
        LDQ     DEC
        TSX     DTBC,4
        SLW     BIN
```

BIN will contain the binary integer represented by the bcd string contained in DEC.

Octal-to-binary conversion

```
        LDQ     OCT
        TSX     OTBC,4
        SLW     BIN
```

After the call, BIN will contain the binary integer represented by the spread-octal number contained in OCT.

Binary-to-decimal conversion

```
        LDQ     BIN
        TSX     BTDC,4
        SLW     DEC
```

After the call to BTDC, DEC will contain the bcd representation of the binary integer found in BIN.

Binary-to-octal conversion

```
        LDQ     BIN
        TSX     BTOC,4
        SLW     OCT
```

After the call to BTOC, the spread-octal representation of the <u>high order 18 bits</u> of BIN will be returned. The low order 18 bit of BIN will be returned left-adjusted in the MQ, to be used for another call to BTOC. I.e. the following code will store the spread-octal representation for all 36 bits of BIN in the locations OCT and OCT+1:

```
     LDQ     BIN
     TSX     BTOC,4
     SLW     OCT
     TSX     BTOC,4
     SLW     OCT+1
```

(END)

Identification

Padding
PAD, BZL, NZL, ZBL, NBL

Purpose

Allow the user to pad a bcd word with arbitrary leading characters.

Usage

Arbitrary padding:

```
          CAL     WORD
          TSX     PAD,4
          PAR     =HAAAAAA
          PAR     =HBBBBBB
          SLW     RESULT
```

All leading A's will be replaced by B's, and the result returned in the logical AC.

Example - to convert '   XYZ' to '***XYZ':

```
          TSX     PAD,4
          PAR     =H             (blanks)
          PAR     =H******
```

The following entries make internal calls to PAD:

BZL - Replace leading zeros by blanks

NZL - Replace leading zeros by nulls

ZBL - Replace leading blanks by zeros

NBL - Replace leading blanks by nulls

Calling sequences are all of the form

```
          CAL     WORD
          TSX     XXX,4
          SLW     RESULT
```

Restriction

These routines may be used by FAP calling programs only; they may not be called directly by MAD or FORTRAN programs.

(END)

Identification

Left and right justification
ADJ, LJ, RJ

Purpose

Left or right justify a character string within an arbitrary field.

Usage

General form:

```
        CAL     WORD
        TSX     ADJ,4
        PAR     =HAAAAAA
        PAR     SWITCH
        SLW     RESULT
```

If SWITCH is zero, leading A's will become trailing A's (left-justification); if SWITCH is non-zero, trailing A's will become leading A's (right-justification).

Example - to convert '   XYZ' to 'XYZ   ':

```
        TSX     ADJ,4
        PAR     =H              (blanks)
        PAR     =0              (left-justify)
        SLW     RESULT
```

The following entries make internal calls to ADJ:

RJ - Right-adjust, field of blanks

LJ - Left-adjust, field of blanks

Calling sequences for these entries are of the form

```
        CAL     WORD
        TSX     XX,4
        SLW     RESULT
```

Restriction

These routines may be called from FAP programs only; the calling sequences are incompatible with MAD and FORTRAN forms.

(END)

Identification

Strip leading blanks or zeroes
BZ57

Purpose

To convert leading zeroes or blanks to null characters (octal 57) for use in output formatting of BCD information.

Usage

```
     MAD:       A = BZ57.(B)

     FAP:         TSX   BZ57,4
                  PAR   B
                  STO   A
```

Location B contains the word to be converted. On return, both A and the AC contain the converted result.

(END)

Identification

Variable length calling sequence processor
COLT, SELAR, MDL

Purpose

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

Usage

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

COLT, as a library subroutine:

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

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

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

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

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

SELAR; what type of argument:

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

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

RETURN is the location to which SELAR is to return.

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

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

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

MDL, MAD list processor:

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

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

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

(END)

Identification

Determine type of calling program and FILNAM
GNAM

Purpose

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

Usage

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

As library subroutine:

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

OPN may be PZE, TXH, or TSX.

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

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

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

0	unknown
1	FAP
2	FORTRAN
3	MAD

(END)

Identification

List transmission
MOVE1, MOVE2, MOVE3

Purpose

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

Usage

As library subroutine:

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

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

BGDATA	is the beginning location of a block of core in the program in which the data will be stored.
ENDATA	(specified only when OP is TIX or STR) is the ending location of the data block.
BEGLST	is the begining location of the list which specifies the data to be moved.
ENDLST	(specified only when OP is TIX or STR) is the ending location of the argument list.

ALPHA	is the return from MOVE2 at which time the AC contains the first data item as specified by BEGLST.
STR	causes the storing of the AC in the data block specified by BGDATA. If this fills the data block, return is made to DATOUT and the AC is meaningless. The next data item from the list is then placed in the AC and return is made to BETA. If there is no next item in the list, return is made to LSTOUT.
	If BEGLST was specified as an array base, successive STR's will cause the transmission of successive elements of the array. The number of elements thus transmitted must be controlled by the user.
DATOUT	is the return location if the data block is full. The AC is meaningless. MOVE1 may now be called again to initialize another data block.
LSTOUT	is the return location if the list is exhausted MOVE2 may be called to specify another list or another STR may be executed if moving an array.
OPN	may be any programming to establish loops and use or modify the AC if desired.
MOVE1	initializes addresses and indexing for the data block and also initializes the STR trap cells to entries to this routine.
MOVE2	initializes addresses and indexing for the list, initializes the trap cells if not already done, and gets the first data item in the AC.
MOVE3	restores the trap cells.

(END)

Identification

Name a format or file name
SETFMT, SETNAM

Purpose

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

Usage

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

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

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

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

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

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

(END

Identification

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

Purpose

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

Method

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

Usage

1) GETIME

As supervisor or library entry:

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

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

2) GETTM

As library subroutine:

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

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

3) GTDYTM

As supervisor or library entry:

MAD

```
    TIME = GTDYTM.(0)
```

FAP

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

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

(END)

Identification

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

Purpose

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

Method

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

Restrictions

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

Usage

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

To initialize or reset the stop watch to zero:

```
                    EXECUTE RSCLCK.
```

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

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

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

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

```
EXECUTE  JOBTM. (J)
```

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

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

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

    MAD:  EXECUTE  TIMER. (J,S)

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

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

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

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

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

```
EXECUTE  RSTRTN.
```

To void the last setting of the alarm clock:

```
EXECUTE  KILLTR.
```

To provide foreground/background compatibility to job time remaining:

```
EXECUTE  TIMLFT. (J)
```

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

(END)

Identification

Simulated interval timer
CLOCON, CLOCOF, UPCLOC

Purpose

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

Restriction

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

Method

If the clock function is on, the B core interval timer cell (location 00005) will be updated by the supervisor at each time burst (200 milliseconds) or on a call to UPCLOC. It will be updated by the elapsed time (running time, not real time) in 60ths of a second. Any B-core interval timer overflow trap will be interpreted at the time of the update. The status of the simulated interval timer is not affected by commands which preserve the current core image: START, SAVE, PM, DEBUG, etc. In addition, it is restored from a saved file by RESUME or CONTIN. The clock function is normally off.

Usage

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

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

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

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

Update the clock and check for trap:
As supervisor or library entry:

```
            TSX  UPCLOC,4     optional (TIA   =HUPCLOC)
```

(END)

Identification

Print time used
RDYTIM

Purpose

To print a 'ready message' on the terminal indicating running time and swap time used since the last 'ready message'. The ready message is identical to that printed by the supervisor on calls to DEAD and DORMNT, and is of the form:

```
          R ttt.ttt+sss.sss
```

where 'sss.sss' is the swap time used in seconds and 'ttt.ttt' is the execution time used, also in seconds.

Method

The supervisor maintains incremental user charge time and running time to aid the user in judging efficiency of his programs. The RDYTIM entry is a user interface into the same program used by the supervisor in printing ready lines.

Usage

As a supervisor or library entry:

```
               TSX  RDYTIM,4      (optional TIA =HRDYTIM)
```

(END)

Identification

List of miscellaneous library subroutines:

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

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

(END)

Identification

Floating-Point Overflow and Underflow
(FPT)

Purpose

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

Method

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

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

1. It prints on one line the comment:

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

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

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

The spill codes are produced as follows:

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

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

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

(END)

Identification

Log in
LOGIN

Purpose

Log out any previous user of this console; identify the new user; initialize accounting information for the new user.

Usage

```
 --> login probn name
     W HHMM.M
     Password
 --> private password
      STANDBY LINE HAS BEEN ASSIGNED
      YOU HAVE  XXXXXX
      PRCBN PROG LOGGED IN  MM/DD/YY HHMM.M FROM UNITID
      LAST LOGOUT WAS MM/DD/YY HHMM.M FROM UNITID
      HOME FILE DIRECTORY IS  PROBN UFDNM
      -message of the day-
      CTSS BEING USED IS  SYSNAM
     R 4.183+.133
```

PROBN is the user's problem number, assigned to his project by the IPC administrative office.

NAME is the user's last name, of wich only the last six characters are used.

PASSWORD is the user's private password, which must match that found in the accounting files before the user can be logged in. After typing 'Password', the computer turns off the printing mechanism of the console, so that the password will not appear on the page.

PROG is the user's assigned programmer number which corresponds to the combination of PROBN, NAME, and PASSWORD.

XXXXXX may be any combination of the following files:

MAIL BOX	- Mail from other users
URGENT MAIL	- Mail from the system
PROGL SAVED	- Saved file from automatic logout. See section AH.1.02 concerning automatic logout.

UNITID is the console identification code

UFDNM is the user's home file directory, if not the same as his programmer number. This line omitted if UFDNM is same as PROG.

MESSAGE OF THE DAY is the contents, if any, of the file 'MESSAG TODAY' in the public file directory M1416 CMFL04, and contains information of interest to the user.

SYSNAM is the name of the current version of CTSS.

__Error messages__

ALREADY LOGGED IN
The user is already logged in from the same console. No further action is taken.

PROBN IS NOT A PROBLEM NUMBER
A problem number consists of a letter (usually 'C', 'M', 'N', or 'T'), followed by one to four digits. The problem number supplied on the command line does not satisfy this requirement.

PROBN NAME NOT FOUND IN DIRECTORY
The combination of problem number and name is not in the accounting files.

PASSWORD NOT FOUND IN DIRECTORY
The password supplied is incorrect for the user PROBN NAME.

NO TIME ALLOTTED FOR THIS SHIFT
The user has zero time allotment for the current shift. See TTPEEK, AH.1.04.

YOUR ACCOUNT IS OUT OF FUNDS
The user's account is overdrawn. He should make arrangements with the IPC administrative office, e.g. submit a new requisition.

YOUR ACCOUNT HAS REACHED ITS TERMINATION DATE
The user's account has expired. He should make arrangements with the administrative office.

USER MAY NOT USE THIS CONSOLE
The user's unit group restricts him to specific consoles, of which the current console is not one.

UNIT GROUP n NOT FOUND
System error, the user's unit group as specified in the primary accounting file does not appear in the unit group file. Notify the systems staff.

IF YOU LOG IN, YOUR FIB JOB WILL BE DELETED
DO YOU WISH TO LOG IN,
The user is currently logged in on FIB. If he replies 'yes', the fib job will be logged out, and he will be logged in; otherwise the fib job will continue to run.

PROBN PROG ALREADY LOGGED IN FROM UNITID
The user is already logged in from a different console. Notify the administrative staff in case of unauthorized use.

PARTY GRP NUMBER n IS WRONG
System error, illegal party group specified in the accounting files. Notify the systems staff.

CTSS IS BEING BROUGHT DOWN
The system is in the process of being shut down for scheduled routine maintenance or system difficulties.

ALLOTTED TIME EXCEEDED FOR THIS SHIFT
The user's time allotment for the current shift is exhausted. See TTPEEK, section AH.1.04.

YOUR DATA PHONE IS HUNG UP
Machine or system error; notify systems or operations staff.

DISK ERROR IN ACCOUNTING FILES
Machine or system error; notify systems staff.

TIME ACCOUNTING FILE IS LOCKED
One of the accounting files cannot be opened. Machine or system error; notify systems staff.

PROBN UFDNM NOT IN M.F.D.
The user's home file directory cannot be found. Notify systems or administrative staff.

SYSTEM FULL, TRY AGAIN LATER
The system is currently filled to capacity, and the user trying to log in does not have a priority line. Wait a few minutes and try again. The HELLO command may be used to determine if a login will be permitted: the number of users currently logged in must be less than the maximum allowed on. See AH.1.06.

LOGIN COMMAND INCORRECT
Error in command format or accounting files, or other error encountered during login process. The user is not logged in.

NO LOGIN
Login refused for whatever reason specified, e.g. account overdrawn, allotted time exceeded, etc.

Party groups and priority lines

The party group allotment specifies the number of users in each party group who may log in regardless of current system load, maximum number of users, etc. Such users are said to have priority lines. At present, only system administrators and programmers, ESL display scope users, and users involved in prescheduled demonstrations using CTSS are assigned priority lines; they are no longer assigned to user groups. All other users are assigned standby lines, and are subject to being automatically logged out by the system in the event of excessive overload due to too many users logged in, or to being refused access for this reason.

Party group 0 is always standby except that FIB and the DAEMON (incremental dumper) are always priority lines.

(END

Identification

Log out
LOGOUT, Automatic logout

Purpose

Allow the user to terminate his console session, update any accounting information, inform the user of his total time used, and in the event the system was full, allow another user to login. In addition, automatic logout allows the system to initiate the logout procedure; in this case, the user's current program if any is saved in a disk file from which execution may be continued at a later date.

Usage

User-initiated logout:

```
        LOGOUT
```

Logout will unmount any file system tapes currently mounted by the user, update the accounting files with the user's time used during this console session, delete all temporary files in the user's home file directory of which he is the author, release any attached remote consoles, hang up the user's home console, and exit to the CTSS supervisor to reset all switches and status words associated with the user.

System-initiated logout:

```
        (response)
        WAIT
        AUTOMATIC LOGOUT
```

Automatic logout is a chain of two commands, neither of which is directly issuable by the user:

```
        ENDLOG - Simulates 'MYSAVE progL T'
        OTOLOG - Special entry to logout, does not delete
                 temporary files
```

ENDLOG creates a SAVED file of the user's current program, if he is not in 'dead' status. The file name used is 'progL', where 'prog' is the user's programmer number. This file is created in temporary mode and may be restarted by

```
        RESUME progL
     or CONTIN progL
```

(END)

Identification

Foreground Initiated "Background"
FIB, DELFIB, PRFIB

Purpose

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

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

Restrictions

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

Usage

To initiate a FIB job:

```
        FIB  NAME1  -LIMIT-  -TIME-  -DAY-
```

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

LIMIT is the maximum execution time limit, in minutes, which the user wishes to place on the job. If LIMIT is not specified, a time limit will be set by FIB. No FIB job will be allowed to exceed a certain maximum time, which is currently set at 10 minutes (this is also the value used when no limit is specified). A FIB job which exceeds its time limit will be automatically logged out; it may be restarted by the user.

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

To delete a waiting FIB job:

```
DELFIB NAME1
```

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

```
PRFIB
```

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

Method

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

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

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

ALLOW, ATTCON, FORBID, GETBLP, RDFLXA, RDLINA, RDMESS, REDLIN, RELEAS, SET6, SET12, SETBLP, SLAVE, SLEEP, SNDLIN, SNDLNA, WAIT, WRHIGH, and WRMESS.

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

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

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

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

"Batching"

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

(END)

Identification

Examine time and storage quotas
TTPEEK

Purpose

Allow the user to list his administratively allotted CPU time and secondary storage quotas and his current time and storage used.

Usage

```
          TTPEEK
```

The user's total time used since login is printed, followed by his time alloted and used for each shift, and his drum, disk, and tape quota if any and current usage.

Shifts are as follows:

Shift 1	Mon.-Fri.	08:00 to 18:00
Shift 2	Mon.-Fri.	18:00 to 24:00
Shift 3	Daily	00:00 to 08:00
shift 4	Sat.-Sun.	08:00 to 24:00
Shift 5	FIB usage	

(END)

Identification

Attach remote terminals
DIAL

Purpose

To connect a dialed-in terminal to a user as an attached remote console.

Usage

From a dialed up but not logged-in console, issue the command:

```
DIAL prob prog
```

where 'prob prog' is the logged-in user expecting to attach the terminal. If 'prob prog' is logged in and has option bit 200 set (see OPTION command, section AH.10.04), the terminal is made an attached remote console of 'prob prog'.

A call to ATTCON by 'prob prog' is no longer necessary with this procedure (although not harmful).

Having once attached the console, 'prob prog' may call supervisor entries SLAVE, SNDLIN, REDLIN, etc., or may use the public command SLAVE SAVED, as desired, just as with previous CTSS systems.

To disconnect the terminal, a 'quit' signal may be issued from the terminal itself, or it may be released by the attacher ('prob prog') with a call to RELEAS. All attached remote consoles are automatically released by logout.

Restrictions

CTSS commands are not accepted from an attached remote console; any input typed is either saved for the attacher to read via REDLIN or is sent to the attacher's input buffer, in the case of an II slave.

The DIAL command may not be used when logged in.

(END

Identification

Dialup message
HELLO

Purpose

Print a message giving system name, number of users, maximum number of users, and the date and time when a user dials into CTSS. In addition, allow this information to be gotten (via the same mechanism) at any other time.

Method

With the CTSS system numbered MIT8A2, the supervisor core-resident module which printed the dialup message was removed, and replaced by a B-core command program to print the same message, together with a means whereby it is initiated for the user when he dials up to the computer. This was done primarily to free up supervisor core space, but in addition gives the added benefit of being expandable to provide more information, e.g. system response.

Usage

Dial into the 7094 installation at the Information Processing Center (see sect. AC.3), or issue the command

```
HELLO
```

Response is of the form

```
MIT8A2: 10 USERS AT 10/03/69 1242.7, MAX = 30
```

In addition, if the card image file 'DIALUP MESSG' exists in the public file directory M1416 CMFL04, it is printed.

This command may be used at any time, whether logged in or not.

(END)

Identification

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

Purpose

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

References

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

(END)

Identification

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

Purpose

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

References

IBM C28-6235 FORTRAN II Assembly Program(FAP)

MAC 179   BEFAP command within CTSS   R. U. Bayles

Usage

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

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

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

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

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

(END)

Identification

COGO-90 - Coordinate Geometry Language
D. Roos

Purpose

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

References

Research Reports:

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

Usage

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

Modifications

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

```
                    READ/DISK          NAME1  NAME2
```

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

```
                    DELAY/PRINT        N
```

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

(END

Identification

COMIT - Symbol manipulating and string processing
Bob Fabry, University of Chicago

Purpose

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

COMIT II Operation on CTSS

Summary

1. The command COMIT ALPHA accepts any input file named ALPHA COMIT as a COMIT II program.

2. Compiler and interpreter error comments appear on the typewriter.

3. Some interpreter errors put COMIT into BREAK status instead of terminating the run.

4. Pushing break button once puts COMIT into BREAK status.

5. In BREAK status, typing:

T or TERMINATE	terminates the run.
C or CONTINUE	continues the program from the next rule.
R or RESTART	restarts program at second rule, immune to further breaks until end of program is reached, then continues program from the point at which the break occurred.
D or DUMP	gives a COMDUMP.
number	executes that number of rules, then gives "OVERRULE" stop.

6. Channel L is for output on the typewriter, and channel R is for reading from the typewriter keyboard.

7. Other COMIT channels refer to files with names of the form ALPHA CHANEL, unless other names are provided by a COMSET.

8. Writing in COMIT on channel ALPHA will create or append to a disk file named ALPHA CHANEL.

9. Reading in COMIT from channel ALPHA will read from a disk file named ALPHA CHANEL, starting with the beginning of the file.

10. A channel being written or read is open. Only 3 files may be open at a time. *FW , *RW , *XF , reading an end-of-file, or program termination will close a file. Write after rewind deletes any older file.

11. The SAVE and LOAD comsets are used for creating and resuming CTSS SAVED files.

12. Do not use //*RAS, *RSS, *BS.

References

Introduction to COMIT programming, MIT Press, 1962.
COMIT Programmer's Reference Manual, MIT Press, 1962.
(New manuals for COMIT II to be published shortly. Those contemplating immediate use can obtain preliminary namuals from V. H. Yngve, Graduate Library School, University of Chicago, Chicago, Illinois 60637.)

Detailed Description of COMIT CTSS Operation

1. Command COMIT

The CTSS command COMIT ALPHA makes the COMIT programming language available to CTSS users. The COMIT system provided is compatible with the SHARE distributed COMIT II except for certain input-output functions. It is possible for a foreground user to write, compile, debug, and run a COMIT program directly from a console typewriter.

Input

The COMIT input source program may be composed at the typewriter using an edit command, or it may be loaded on the disk from cards, or it may be produced directly as the output of another COMIT program. In any case it must be named or renamed ALPHA COMIT.

Compilation

When the command has been issued, compilation begins. The title card may be omitted, but if present will be typed out as a check. There is a comment at the end of compilation. The compiled program then begins to run under the COMIT interpreter.

2. Compiler Error Comments

Any compiler error comments are printed on the typewriter during compilation. You may push the break button twice at any time to quit and make corrections before recompiling.

3. Interpreter Error Comments

Interpreter errors may occur at any time after compilation is finished. In this case, too, the error comments appear on the typewriter. Some interpreter errors cause the run to terminate, but others that are less serious halt the program temporarily after typing the word BREAK. The program is now in BREAK status, and various actions may be taken as indicated below under 5.

4. Manual Interrupt

Pressing the interrupt button once during compilation will have no effect except that the supervisor will destroy any information in the supervisor input-output buffer, then it will type INT. NO ACTION. But during interpretation, the supervisor will destroy the information in the buffers and then return control to the COMIT system, which will eventually type BREAK, enter BREAK status, in which interpretation is temporarily halted, and await further instructions from the keyboard. If an EXECUTIVE comset is used, a restart will be given instead of the break. (See below for description of restart.)

5. BREAK Status

When COMIT is in BREAK status, it expects one of the following instructions from the keyboard: TERMINATE, CONTINUE, RESTART, DUMP or a decimal number. TERMINATE, CONTINUE, RESTART, and DUMP may be abbreviated to their first letter. These instructions are explained below.

TERMINATE Typing TERMINATE when COMIT is in the BREAK status will cause the COMIT program to terminate normally. Any accumulated format A partial lines of output being held in the COMIT buffers will be written out.

CONTINUE Typing CONTINUE when COMIT is in the BREAK status will cause the COMIT program to continue in a normal fashion from where it was when BREAK status was entered.

RESTART Typing RESTART when COMIT is in BREAK status will cause the COMIT program to go back to the second rule and continue from there in immune status, in which it is immune to the manual BREAK signal until the COMIT END card is reached. At this point the COMIT program does not terminate, but instead it leaves immune status and continues normally from the point where it was when it entered BREAK status.

This facility allows one to stop a COMIT program at any time to provide it with input, change the flow of control, enter or leave a trace mode, take a dump, etc. Section 13 gives programming suggested for processing breaks.

DUMP Typing DUMP when COMIT is in the BREAK status will cause a special built-in format-S differential dump called a COMDUMP on the output unit, normally the console.

number   Typing a decimal number between 1 and (2**35 - 1) will cause an overrule stop to occur after that number of rules have been executed, allowing dumps to be taken, etc.

6. Console Input and Output

Output on the console typewriter can be produced by the COMIT routing instruction *WAL or *WSL.

Input from the typewriter keyboard can be obtained by *RCS, *RAR, *RTR, or *RSR. When the COMIT program reaches one of these instructions it will read from the supervisor typewriter input buffer. If the buffer is empty, the COMIT program will wait until the next carriage return, which enters material into the buffer. Because of this wait, it is a good plan to have the COMIT program type a comment just before entering the rule containing the read instruction so that the user will be alerted to the need for providing input.

7. Simulated Tapes

All other COMIT channels refer to the disk. It is here that most of the incompatibility with the standard SHARE-distributed COMIT is to be found. The disk routines provided by CTSS do not completely simulate tapes. File names are normally of form ALPHA CHANEL, but arbitrary names may be provided by COMSET cards on which the two words which normally have the form TAPE A6 or ONLINE PUNCH are treated as a file name. The special name /ON LINE/ is used for the remote console.

8. Writing on Disk

Material can be written on the disk by using *WAX, *WSX, *WBX, where X is any letter except L or R. The material written out appears in a file on the disk named X CHANEL. If there is no such file in the user's file directory, such a file is created and named automatically. If there already is a file by this name, the material written out is added to the end of that file.

9. Reading from the Disk

Material can be read from the disk by using *RCX, *RAX, *RSX, *RBX, or *RTX where X is any letter except L, R, or S. The material to be read should exist in a file named X CHANEL. The first read instruction starts reading from the beginning of the file. There is normal COMIT behavior on reading the end-of-file except that the next read instruction after an end-of-file has been read starts reading from the beginning of the file again. Note that channel K is no longer the input tape unless the input file has been split into a program file named ALPHA COMIT and a data file named CHANEL K.

10. Closing out Disk Files

A disk file is either open for reading, open for writing, or closed. Only three files may be open at any time. A file can be closed out at any time by the COMIT instruction *RW, by *FW after writing, or by *XF after reading. Reading an end-of-file closes the file. At the end of the COMIT program, all open files are closed out. The first read after a file has been closed is from the beginning. Writing is always at the end of a file, except that the first write after a rewind will first delete the old file. Since there may be only one file with a given name, end-of-file marks cannot be used for data separators.

If you want to stop a program which is writing a file and preserve the file, do not use QUIT, but press the break button once to get BREAK, then type TERMINATE. In this way the file will be closed.

11. Creating and Restoring SAVED Files

The SAVE and LOAD comsets may be used to create SAVED files containing COMIT programs and to restore any SAVED file.

12. Limitations

The COMIT instructions *BS, *RSS, *RAS may not be used.

13. Programming and Debugging Hints

To program an automatic BREAK, use a comma as a dump request. The dump is not automatically given, but rather the program enters BREAK status so the dump, or some other action can be requested.

Make liberal use of periods before rule names when left-halves are not expected to fail. If one does, a BREAK will occur, allowing for a dump.

To process the RESTART after BREAK, use the following organization:

```
COM
* START
* RESTART
START *

 .
 .
 .

RESTART
                END

 .
 .
 .

END *
END
```

(END)

Identification

DYNAMO - Model Simulation Language
A. L. Pugh III

Purpose

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

References

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

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

Usage

```
                DYNAMO   NAME1   P   R
```

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

P-Page Skip

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

R-Rerun

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

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

When DYNAMO is expecting this rerun information it will type out

PLEASE TYPE CHANGES IF RERUN DESIRED

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

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

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

QUIT

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

Differences In Input

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

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

(END)

Identification

ESL display system (not a command)
C. Garman

Purpose

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

References

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

(END)

Identification

FAP - IBM 7094 machine language
Programming Staff

Purpose

FAP is the IBM MACRO-FAP assembly program for the 7094 machine language code, slightly modified for increased utility. It accepts all 7094 operation codes and the standard data-defining pseudo-operations, as well as macro definitions. Input files may be line marked (tabs are assumed set at columns 8, 16, 30, and all columns thereafter; lines are truncated after column 72) or line-numbered.

References

IBM C28-6235   Fortran II Assembly Program (FAP)

Usage

```
FAP name1 -name2- -list-

    'name1 name2' is the name of the file to be assembled
    'name2' is assumed 'FAP' if not specified
    'list' consists of any of
      '(DATE) -NO-' print date and time of assembly in page
                       headings
      '(MACR) -NO-' macro nesting level will be listed and
                       multiply defined macros will be flagged
      '(LIST) -NO-' create 'name1 BCD', the listing file
      '(FLAG) -NO-'  list non fatal flags on the user's
                       console
      '(REFS) -NO-' force symbolic reference listing
      '(LONG)' #    set page length to #; default value is
                       56, minimum is 4. the maximum number
                       of references per line in the symbolic
                       reference listing is reduced by three.
                       this is intended for producing listings
                       that will be reduced to 8 1/2" by 11".
      'BCD -NO-'    same as '(LIST)'
        'NO' inverts the meaning of the argument it is
        applied to
        The default mode is (DATE) (MACR) (LIST) NO (FLAG)
```

General

CTSS FAP is completely compatible with FAP as described in IBM C28-6235 except that, since it is not useful in the CTSS environment, the UPDATE facility has been removed. These pseudo-operations will be listed with a non-fatal 'F' flag but will have no other effect. It accepts the standard 7094 BCD character set plus the characters 'colon' (octal 35) and 'tab' (octal 72). Colon is an alphabetic character; tabs are converted into strings of spaces by the preprocessor. Tab settings are at columns 8, 16, 30, and every column thereafter.

CTSS FAP uses temporary files with first names 'FAPTEM', 'FAPSYM', 'FAPBSS', and (if a listing is requested) 'FAPBCD'. The second name of any of these files is the user's programmer number. A symbol table file 'name1 SYMTB' is created which contains all defined symbols with their definitions and relocation information. The format is BCD card-image with four fields of three words each and two words of trailing blanks per card. Each field has the format VVVVVV bRbSSS SSSbbb. VVVVVV is the symbol value with one leading blank if < 32768, SSSSSS is the left-adjusted symbol name, and R is 0 if absolute, 1 if relocatable, 2 if common, and 4 (actually 5) if in the transfer vector. This file is used by debugging aids such as DEBUG (Sect. AH.8.08).

If no page title card appears in the first card group, a default header of

```
FAP ASSEMBLY LISTING  ...  ...  ...    FILE name1 name2
```

will be used. The date and time of assembly are normally listed to the left of the page number at the top of each page. Two new non-fatal error conditions have been defined to flag common errors: 'L' if the location field is numeric when a symbol is expected, and '7' if column 7 is non-blank.

A new form of literal has been defined: =V adds the versatility of the VFD pseudo-operation to literals. The string following =V is processed like the variable field of an OPVFD (only previously defined absolute symbols may be referenced, the result may not be more than 36 bits long). No tag or decrement is allowed in an instruction containing a VFD literal.

Machine Operations

Six new extended machine operations have been defined, three for use in calling sequences and three for use in I/O lists:

mnemonic	meaning	assembles as	requires
PAR	parameter	TXH	Address
EFA	effective address	NOP	Tag
BLK	block	TIX	Address, Decrement
IOP	I/O Proceed	TIX	Address, Decrement
ION	I/O Non-transmit	TXI	Decrement
IOD	I/O Disconnect	PZE	-

Some new machine operations relating to special time-sharing hardware have been defined:

mnemonic	function	op. code		
LRI	Load Relocation Indicators	0562		X I
SRI	Store Relocation Indicators	-0601		X I
LPI	Load Protect Indicators	-0564		X I
SPI	Store Protect Indicators	-0604		X I
TIA	set instruction references to A-core and transfer	0101		X I
TIB	set instruction references to B-core and transfer	-0101		X I
SEA	set data references to A-core	-0761	0041	X
SEB	set data references to B-core	-0761	0042	X
IFT	skip if instruction references are in A-core	-0761	0043	X
EFT	skip if data references are in A-core	-0761	0044	X
SSLx	Store Sense Lines - channel x	±0660 ±0662	±0661 ±0663	X I
PSLx	Present Sense Lines - channel x	±0664 ±0666	±0665 ±0667	X I
SCDx	Store Channel Diagnostic - channel x	±0644 ±0646	±0645 ±0647	X I

Symbol-defining Pseudo-operations

SETB: This pseudo-operation is provided to give a symbol a boolean definition and yet permit redefinition. It sets the symbol in the location field to the value of the boolean expression in the variable field. If the symbol had been previously defined by a SET or SETB it will be redefined; if it was previously defined in any other manner it will be redefined but a warning flag will be generated. All symbols used in the variable field must have been previously defined.

Storage-allocating Pseudo-operations

COMBSS and COMBES have been provided to allow more intelligible use of common.

COMBSS: The common counter is decremented by the value of the expression in the location field; then the symbol, if any, in the location field is defined as the value of the common counter plus one. This is analogous to the BSS pseudo-operation.

COMBES: The symbol, if any, in the location field is defined as the value of the common counter plus one; then the common counter is decremented by the value of the expression in the variable field. This is analogous to the BES pseudo-operation.

Data-generating Pseudo-operations

BCI and BCD have been modified and 12BIT (to generate full-mode text information) has been added. As used herein, "quoted text string" means "/text/" where "/" is any character except blank. Any character except "/" may appear within the string.

BCI: The variable field may contain a quoted text string beginning in columns 12 through 16 and followed immediately by a blank or column 73. The characters in the string are converted to BCD and stored six to a word in consecutive memory locations. If the number of characters is not a multiple of six, the last word is padded to six characters with nulls (octal 57). The algorithm for deciding whether the variable field is a qouted text string or a word count plus data is:

1) the first two characters are 1-9 followed by a comma -- count and data
2) there is a comma in column 12 -- count and data
3) the first and last non-blank characters are the same -- quoted text string
4) the first character is 1-9 -- count and data (gives non-fatal 'F' flag)
5) otherwise -- gives fatal 'E' flag (one word of blanks is assembled)

BCD: is like BCI except that the quoted text string must start in column 12.

12BIT: This pseudo-operation is used to generate full-mode text information in a program. A symbol in the location field will be defined as the next location to be assigned when the 12BIT pseudo-operation is encountered. The variable field contains a quoted text string beginning in columns 12 through 16 and immediately followed by a blank or column 73. The characters in the string are converted to their octal values in the CTSS character set (Sect. AC.2.01), packed three to a word, and stored in consecutive memory locations. If the length of the string is not a multiple of three, the last word will be padded to three characters with nulls (octal 0057). Letters are normally lower case. The character "*" is special: its code is not assembled but the 100(base 8) bit of the code for the next character in the string is complemented; this creates upper case letters from lower case ones and some special characters from numbers and BCD special characters. All 6-bit characters except '*' and most 12-bit characters can be produced.

Program-linking Pseudo-operations

LINK and NOLNK are provided to control the existence of the linkage director.

LINK: The variable field must contain either 'ON' or 'OFF'. 'ON' will cause the linkage director to be included in this subprogram, 'OFF' will delete it.

NOLNK: This is an obsolete form retained for compatibility. It is equivalent to 'LINK OFF'.

List Control Pseudo-operations

These pseudo-operations affect the listing (if any) and the terminal output but have no effect on the binary output.

FLAG: The variable field must contain 'ON' or 'OFF'. 'ON' causes non-fatal error flags to be listed on the terminal, 'OFF' causes these flags to appear only in the listing. 'ON' is the default mode. If '(FLAG)' appears in the command line, this operation does nothing.

LSTNG: Alternate occurrences of this pseudo-operation cause initiation and termination of printing the listing on the user's terminal.

MLEVEL: The variable field must contain 'ON' or 'OFF'. This pseudo-operation controls the appearance of the macro nesting depth to the right of a macro expansion. If '(MACR)' appears in the command line, this operation has no effect.

MMACRO: The variable field must contain 'ON' or 'OFF'. This pseudo-operation controls the flagging of macro re-definitions (in the listing and at the terminal). The mode is initially on. If '(MACR)' appears in the command line, this operation has no effect.

REF: The pseudo-op has been modified to accept a variable field. The variable field may be blank or may be any FAP expression (blank = 0). The variable field is evaluated, and its value is interpreted as follows:

- 0 All symbolic reference listings are deleted (except multiply defined symbols)
- 1 Listing of unreferenced symbols is suspended
- 2 Listing of unreferenced nonrelocatable symbols is suspended
- 3 or more (e.g. -1) Normal mode of symbolic reference listings

If '(REFS)' is specified on the command line, this pseudoop has no effect.

SYMREF: The variable field may be 'ON', 'OFF', or blank. This pseudo-op controls the accumulation of symbolic references. Normal mode is ON; blank variable field inverts the mode.

SEQ: The variable field must contain 'ON' or 'OFF'. This controls sequence checking of card-image input. Checking is initially on.

NOSEQ: This is an obsolete form retained for compatibility. It is equivalent to 'SEQ OFF'.

Macro Processor

MACRON: This pseudo-operation is identical to 'MACRO' except that unspecified trailing arguments will never be replaced by created symbols.

SAVCRS: The current created symbol count and character are saved.
RESCRS: The created symbol count and character are restored to what they were when the most recent SAVCRS was encountered. N.B. SAVCRS/RESCRS pairs can not be nested.

Miscellaneous

BCORE: This pseudo-operation indicates that this program is to be run as background to CTSS. It will be checked for illegal instructions and the B-core flag will be set in all I/O commands.
ACORE: Counteracts a previous 'BCORE'.

INSERT: The variable field contains one or two file names separated by a comma. The entire contents of the named file replace this pseudo-operation. If name2 of the file is not specified, it will be taken to be the same as the second name from the command line. INSERTs may not be nested. The INSERTed file need not be the same type (line-marked or card-image) as the current source file.

ABBREV: The variable field may contain 'OPCODE' or 'SYMBOL'. 'OPCODE' will cause over 400 operations not generally useful in foreground programs to be deleted from the combined operations table. This more than doubles the number of available slots. 'SYMBOL' is intended to remove all created symbols from the symbol table. It will delete any five character symbol beginning with a '.'. This operation is effective only during pass 1.

SAVE: The variable field contains one or two file names separated by a comma. This operation is intended to create private versions of FAP that have been initialized with macro and symbol definitions, etc. When encountered during pass 1, this causes loss of all generated code then saves itself as the first file name specified. When resumed, this saved file will act like FAP except that it will retain all symbol and macro definitions, mode settings, and unexpanded remote sequences. Preset modes may be overridden by arguments when resumed or by operations in the file to be assembled. The assumed second name of input files will be set to the second name specified by the SAVE, or, if none, to the first.

(END)

Identification

GPSS - General Purpose System Simulator
M. M. Jones

Purpose

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

References

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

(END)

Identification

LISP - List Processing Language
J. Moses

Purpose

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

References

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

Usage

LISP -NAME1-

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

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

(END)

Identification

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

Purpose

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

References

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

RESTRICTIONS

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

Usage

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

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

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

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

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

CHANGES:

1. A new statement

   INSERT FILE ALPHA

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

2. An addition has been made to the '...' block notation in MAD. Formerly only the form

   A...B or A,..., B

   was allowed, where A and B are variables. Now the second expression may be a constant, e.g.,

   A ... 7.

   See MAD Manual, November, 1963, page 16.

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

   ..RTN. This symbol, which is obviously invalid in a statement, stands for the operation of placing the appropriate value(s) in the arithmetic register(s) and then returning from a function to its calling program. It is analogous to the right hand side of a substitution statement (the B operand) and then a transfer to a given address (there is no designation for this address _within the triple_). As such, there is no result. As an example, if the result of a function were a double precision number, say mode 5, the following would be a reasonable definition.

```
MODE STRUCTURE 4..RTN.5
JMP  *+3,BT,*+1
CLA  B
LDQ  B+1
TRA  FF
OUT  ACQ
END
```

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

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

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

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

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

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

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

(END)

Identification

MADTRN - Fortran II to MAD translator
Programming staff.

Purpose

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

References

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

Usage

```
MADTRN  NAME1  OP
```

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

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

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

(END)

Identification

SNOBOL - A String Manipulation Language
D. Shea

Purpose

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

Usage

```
          SNOBOL NAME1 -'ONLINE'-
```

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

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

References

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

(END)

Identification

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

Purpose

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

References

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

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

Usage

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

```
OPS
```

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

```
GUIDE INFORM
```

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

(END)

Identification

Information Retrieval and Text Management
TIP
W. D. Mathews, Project TIP, Rm. 14S-310, X5687

Purpose

TIP is a flexible and powerful information handling system developed by the Technical Information Program to provide capabilities for information retrieval and text manipulation in a wide variety of on-line applications. The command and the many associated subsystems are described in the documents listed below. These publications and additional information are available from:

Project TIP Document Room
Room 14S-310
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

References

TIP-TM-104	The TIP Reference Manual	W. D. Mathews
TIP-TM-105	The Edit Reference Manual	L. H. Morton
TIP-TM-106	The Reduce, Expand, Sort and Merge Reference Manual	E. M. Mattison
TIP-TM-107	A TIP Sampler	J. R. Ebright
TIP-TM-111	The Run Reference Manual	W. I. Nissen, Jr.

(END)

Identification

FORTRAN IV Translator FOR4
T. Burhoe, 868-9840

Purpose

To translate source programs written in the FORTRAN IV language to MAD and compile them; to provide source-level compatibility between CTSS and background FORTRAN IV; and to provide useful diagnostics on the source program.

FOR4 is an author-maintained CTSS command, offered on an experimental basis. As such, all inquiries, suggestions, or complaints should be addressed to Mr. Tom Burhoe, IBM Scientific Center, 545 Technology Square (4th floor), Cambridge, Mass. Telephone (617)-868-9840.

Usage

```
FOR4   NAME   -'NOCOMP'-   -'MADFIL'-
```

NAME is the primary name of file NAME MADTRN, the user's FORTRAN IV source program. It must be a line-numbered disk file. All source programs should be written in accordance with the FORTRAN IV language specifications as described in IBM Form C28-6390-2, available at the Coop. These are the specifications for FORTRAN IV as implemented under IBSYS, Version 13.

NOCOMP is an optional parameter indicating that the user does not wish the translated MAD program to be compiled into object code. (For instance, he may wish only to find the syntactic errors in his FORTRAN programs prior to submitting them for background runs.)

MADFIL is an optional parameter indicating that the generated MAD program should be left in permanent (0) mode in the user's directory, rather than in temporary mode as in the normal case. Also, detection of source errors will normally suspend the generation of further MAD code; the MADFIL option will create a MAD program regardless of errors in the source program.

In normal operation, FOR4 will generate an equivalent MAD program in temporary mode, compile it via MAD, and exit to CHNCOM. If source errors are detected, they are printed out along with the offending statements, and the final exit is to DEAD. If translation is interrupted via the BREAK key, an immediate exit to CHNCOM will occur.

There are nearly one hundred explicit one-line diagnostics in FOR4, thus permitting much flexibility and specificity in helping the user debug his programs. The diagnostics closely resemble those of IBSYS FORTRAN IV. One important objective of FOR4 is to permit an exact correspondence between programs which will compile and run under CTSS, and those which are run under batch processing. Thus, all IBSYS FORTRAN IV errors are diagnosed in FOR4, even though some may be "translatable" - e.g., mixed modes in arithmetic expressions.

The so-called "built-in functions" of FORTRAN IV may be used in FOR4 in external function form by including file 'F4LIBE BSS' at load time. This program contains entries for all the built-in functions which do not process "complex" or "double-precision" data exclusively. To link to this (1-track) library, do: LINK F4LIBE BSS M1416 CMFL04 .

Restrictions

Because of dependence on the IBJOB system, or lack of a MAD counterpart, the following FORTRAN IV facilities may not be used in FOR4:

'BLOCK DATA' subprograms
'DOUBLE PRECISION' and 'DOUBLE PRECISION FUNCTION'
'COMPLEX' and 'COMPLEX FUNCTION'
'NAMELIST'
Named-common conventions in 'COMMON' declarations
'ENTRY'
'RETURN i' (non-standard return)

Attempts to use these facilities will cause a special diagnostic to the user reminding him that they are legal in IBSYS FORTRAN IV, but cannot be processed by FOR4. In addition, the user should note the inconsistency in array storage between FORTRAN and MAD, the former storing by columns and the latter by rows. Thus, an attempt to equivalence a vector to a column of a matrix would be acceptable in FORTRAN, but the resulting MAD program would have the vector equivalenced to all or part of one or more rows of the matrix. Care is urged in this usage of FOR4.

All facilities of FORTRAN IV excepting those noted above are available to users in their full generality.

(END

Identification

An Algebraic Desk Calculator version of the Formula Manipulation Compiler developed by the IBM Boston Programming Center.
FORMAC
R. Kenney, 491-0321

Purpose

To allow the user to manipulate a class of formal expressions and compute the values of arithmetic expressions. Included among the capabilities of the program are formal differentiation, substitution for one or more variables in an expression and expansion of expressions. After the expression has been manipulated in a way requested by the user, the results are simplified; like terms (i.e., terms which differ only by a constant factor) are combined, zero terms and unit factors are eliminated, etc. The result of the computation or manipulation is then available for further manipulation or for printing on the user's console.

FORMAC is an author-maintained CTSS command offered on an experimental basis. Inquiries, suggestions or comments should be addressed to Mr. Robert Kenney, IBM Boston Programming Center, 545 Technology Square (3rd Floor), Cambridge, Massachusetts, telephone (617) 491-0321.

Reference

CC-257. Description of Time-Shared FORMAC.

Method

The Desk Calculator statements are executed immediately, as they are typed in through the user's console. No provision is made for a stored program. That is, although results are saved and are usable from one computation to the next, the statements which caused the computations to take place are not saved. Thus, it is not possible to establish loop controls and various paths of program flow in the classical stored program sense.

The Desk Calculator accepts lines of up to 84 characters and prints 84 characters per line. The '$' is used as the end of statement marker. A statement may extend over any number of lines. FORMAC will respond 'READY' initially and between statements; do not type a new statement until the response has appeared. (Editor's note: If the response seems unduly delayed, check to see if you've forgotten to terminate the statement with a '$'.) Standard CTSS erase and kill characters apply during the typing of a line. However, there is no "context editing"; if an already typed line is found to be incorrect, the entire line must be retyped.

Usage

FORMAC

Summary (see CC-257 for details):

Variables:

A variable name represents a fixed point variable if its initial character is one of the letters I-N. A variable is made either atomic or assigned from context , "upon its first appearance" in a FORMAC statement. It is considered "assigned" if its first appearance is on the left-hand side of an = sign. It is "atomic" (i.e., it stands for itself) if its first appearance is on the right hand side of an = sign. An atomic variable may not have an expression assigned to it.

Operators and Functions:

Expressions may be composed of the following operators and functions:

*,+,-,**,/, DERIV,SIN,COS,ATN,HTN,FAC,EXPON,LOG,DFC,COMB.

DERIV (A,B,N)	is the nth derivative of A with respect to B.
SIN (A)	is the SINE of expression A.
COS (A)	is the COSINE of expression A.
ATN (A)	is the ARC TANGENT of expression A.
HTN (A)	is the HYPERBOLIC TANGENT of expression A.
LOG (A)	is the NATURAL LOG of expression A.
EXPON (A)	is E**A.
FAC (A)	is the FACTORIAL of A.
DFC (A)	is the DOUBLE FACTORIAL of A (n(n-2) (n-4)...).
COMB (A,B)	is the COMBINATORIAL of A things taken B at a time.

Executable Commands:

The assignment statement assigns the variable name on the left of the equal sign as the name of the expression on the right. The expression may be composed of any of the operators and functions above.

```
Example:  A = B+C+DERIV (X**2,X,1)$
Result:   the variable A names the expression,
          B+C+ X*2.
```

SUBST Substitution for variables in the expression. The list of parameters may be written explicitly or a label of a 'PARAM' statement may be used. Substitution proceeds from left to right.

```
Example 1:  D = SUBST A, (X,0)$
Example 2:  ABC = PARAM (X,0)$
             D = SUBST A,ABC$
```

Result: Both examples result in D naming the expression B+C. (Assume A was the expression B+C+X*2 from the example above.)

EXPAND Performs multinomial expansion and the distributive law on the expression. The 'CODEM' option causes the result to be placed over a common denominator.

```
Example 1:  F = EXPAND D**2$
Result:     F names the expression B**2+B*C2+C**\
            (Assume D was the expression B+C from
            above.)

Example 2:  F = EXPAND H/B+C, CODEM$
Result:     F names the expression:
            (B*C+H)*B**(-1)
```

PRINT Prints on the user's console the variables and their expressions which are specified.

```
Example:   PRINT A,D$
Result:    Results in the following print-out on
           the console,
           A = B+C$
           D = B**2+B*C*2+C**2$
```

DUMP Prints on the user's console all assigned variables and their expressions.

```
Example:   DUMP$
Result:    All assigned variables printed in the
           same format as the 'PRINT' command
           above.
```

ERASE Erase the expressions specified and return the storage they require to the storage pool.

```
Example:   ERASE A,D$
Result:    The expressions which A and D names
           are erased.
```

CLEAR Clears the symbol table and reinitializes the storage pool. It has the same effect as reloading the Desk Calculator.

Example: CLEAR$
Result: All symbols removed from the symbol table and the storage pool reinitialized.

STOP   Places the Desk Calculator in dead status and returns to the CTSS supervisor.

Example: STOP$
Result: R XXX.X+XXX.X

Declarative Statements:

DEPEND   Declares dependence relationships between atomic variables for use with the differentiation operator.

Example: DEPEND (M,B/X,Y)$
Result: M depends on X and Y, B also depends on X and Y.
(NOTE: M does not depend on B ncr X or Y or Y on M or b.)

Example: DEPEND (X/Y)
A =DERIV (X**2,Y, 1) $
Result: A names the expression:
2*X*DERIV (X,(Y, 1))

PARAM   Sets up parameter pairs for use with the 'SUBST' command.

Example: ABC = PARAM (B,2) , (C,X+Y) $
Result: When the label ABC is referenced in the 'SUBST' command, the number 2 will be substituted for every occurrence of B likewise X+Y will be substituted for C.

Interrupt Level:

The Desk Calculator has one interrupt level which returns to the input routine. When the Desk Calculator types 'READY', the next statement may be typed in. If the Desk Calculator is interrupted during execution of any command except 'PRINT' or 'DUMP' there can be no guarantee that further execution will give correct results. "USER BEWARE".

Errors

Error messages are tabulated in CC-257.

(END)

Identification

Interface between user and CTSS.

N. I. Morris

Purpose

The "." command (read "dot" - or "point") 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

Since "." allows many user modified options, it must maintain a file containing these option settings in the user's own tracks. This file is named "USER PROFIL" and is one record long. The first time "." is used, a standard copy of "USER PROFIL" will be copied from public file.

1. Initiation:

```
--> .
```

Response is the character "R" followed by two carriage returns.

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 restarting itself at the end of the chain. For example:

```
FAP ALPHA (LIST) , LOAD ALPHA GAMMA , SAVE DELTA , START
```

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 '(EOC)' followed by the continuation on the next console line. The only restriction on command line length (not to be confused with command 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 restarting of "." at the end of a command chain, terminate the line with the parameter "slash" ("/"). Note that this tactic will allow the chaining of six commands. Commands which leave a desired core-image (e.g. RESTOR, LOAD, etc.) and are the last link of a command chain should be followed by the slash to prevent destruction of the core-image by the ".".

d. Commands and parameters may be iterated by the use of parentheses. For example, the command line

```
( BEFAP NCLOAD SAVE ) PROG
```

will generate

```
BEFAP  PROG
NCLOAD PROG
SAVE   PROG
```

and the command line

```
FAP ( ALPHA BETA GAMMA ) (LIST)
```

will generate

```
FAP ALPHA (LIST)
FAP BETA (LIST)
FAP GAMMA (LIST)
```

2. Abbreviations:

a. Often-used commands and parameters may be abbreviated in ".". The abbreviation definitions are stored in the user's "USER PROFIL". They may be determined through the use of the internal commands

```
ABBREV COM
ABBREV PAR
```

Note that "USER PROFIL" cannot be PRINTed successfully. The "ABBREV" command must be used to examine its contents.

b. To define abbreviations, 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
```

For example: DC LO LOGOUT

c. To remove abbreviations, the internal commands are "RC" and "RP". General form is as follows:

```
RC ab1 ab2 ... abn
RP ab1 ab2 ... abn
```

Example: RC LO

d. The abbreviation tables can each contain a maximum of 75 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 "CMFL0x".
".x" is the abbreviation for "(CFLx)".

3. Private commands:
"SAVED" files may be resumed without explicit use of the "RESUME" command. "." maintains a list of the primary names of "SAVED" files in "USER PROFIL". This list is consulted whenever a command is issued to determine if a "SAVED" version exists and a "RESUME" command should be generated. For example, if the user issues the command "ZILCH" and "ZILCH SAVED" exists in his files, then "." will automatically generate a "RESUME ZILCH" command. This private command list may be updated through the use of the internal command "UPDPRI" which searches the user's file directory for "SAVED" files. Note that if "ZILCH SAVED" is created and "UPDPRI" not issued, "." will not "know" that "ZILCH" refers to a private "SAVED" file.

The private command list checking feature may be turned off by issuing the internal command

"SVLIST OFF". In this mode, "." will perform an FSTATE for each command it generates in order to check for a private "SAVED" version. This mode is <u>considerably</u> <u>slower</u> than the "SVLIST ON" mode. When "." is used for the first time, it will be in the "SVLIST OFF" mode. Issuing the "UPDPRI" command will update the user's "SAVED" files into "USER PROFIL" and turn "SVLIST ON" automatically.

Note that it is also possible to run with option bit 2 set, and let the supervisor check the existence of the SAVED file if necessary (see AH.10.04, OPTION command).

4. File system calls:
In a similar manner to the treatment of RESUMEs ("private commands"), "." will also automatically 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	FSTATE	OPEN
ATTNAM	IODIAG	SETPRI
CHFILE	MOVFIL	TRFILE
DELFIL		

For example, "STORGE 2" will generate the command "CALL STORGE 2".

5. Interconsole communications:
When a user is in "." 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 will prevent the user from receiving interconsole messages in his input buffer while editing, typsetting, etc.)

Interconsole messages are sent using the revised ASCII character set. This permits the user to communicate using upper and lower case and all special characters. As in "TYPSET", "#" is the erase character, and "@" is the kill character.

a. To send an interconsole message, use the internal command "WRITE" followed by the problem and programmer number of the user to whom the message is being sent. For example, "WRITE T234 3187" would initiate the transmission of a message to T234 3187. Subsequent input lines would be received by T234 3187 if he desired to communicate. T234 3187 could reply by typing his answer.

b. To prohibit the reception of all interconsole communications, use the internal command "FORBID".

c. To permit a particular user to communicate, use the internal command "ALLOW PROB PROG". If PROG is omitted, all users with problem number "PROB" will be allowed to communicate. If "PROB" is "*", all users with programmer number "PROG" will be allowed. If both "PROB" and "PROG" are omitted, everybody will be allowed.

d. The interconsole section of "." may be left by giving a single "break" signal. This will cause the interconsole communications routines to exit to a fresh version of ".".

6. Sleeping:

The internal command "RS" will initiate a sleeping program which will wake up every ten minutes and print a comment. Monopolization of lines to the computer in this fashion is rather anti-social, and should not be done unless absolutely necessary. Interconsole messages may be received and replied to while sleeping. After communications are completed, a single "break" will return the user to sleep.

7. Response supression:

Successive usages of the internal command "V." (verify) will alternately suppress and permit printing of the acknowledgement characters "R" and "W" and of the sleeping comment.

8. 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 DEAD. It is wise never to quit out of ".", but to always use "Q". Otherwise, the user may receive interconsole messages unexpectedly.

(END

Identification

Context editor for card image files
ED

Introduction

ED is a command for editing 14-word BCD card image files within CTSS. The command is based on TYPSET (CC-244, MAC-M-193 by J. H. Saltzer) and many of the conventions of TYPSET are used by ED. Tabs are automatically interpreted for FAP, MAD, MADTRN, GPSS, COMIT, and ALGOL(i.e., AED) programs. Tabs may also be set by the user for other purposes. Although line numbers may be generated by the ED command, editing is done entirely by context.

Usage

The ED command is initiated with the following CTSS command.

```
                    ED  -NAME1-  NAME2  -NAME3-
```

NAME2 is the secondary name of the file to be edited or created and must be provided. NAME1 is the primary name of the file to be edited. If NAME1 NAME2 is not specified, ED will assume that a new file is to be created and will start in the high-speed INPUT mode. If NAME1 is provided, the command will look for the file NAME1 NAME2. If the file is not found, the high-speed INPUT mode will be entered. If the file is found, the EDIT mode will be entered.

If NAME3 is specified and the file NAME1 NAME2 is found, the subsequent FILE will create a file NAME3 NAME2 and NAME1 NAME2 will remain unaltered. Any arguments to the FILE request, however, will take precedence.

New files will be created only on the disk (device 2) and old copies resident on the drum will disappear if modified. The original file to be edited may be a linked file, however, any attempt to replace a linked file by the edited version will be rebuffed. The modified version must be filed under a different name. Linked files of the name '(INPUT prog' and '(INPT1 prog' where 'prog' is the user's programmer number may not exist in the file directory or ED will not function.

HIGH-SPEED INPUT MODE:

When the user enters this mode, the ED command will type "INPUT:" on the user's console. While the user is operating in this mode, the ED command will accept input lines from the user's console. Tabs will be interpreted automatically for each input line. Backspace characters may also be used to move back one character position in the input line. No response is typed for input lines and as a result, the user may type successive lines as fast as he wishes. When the user types a line consisting only of a single carriage return, the ED command will place the user's console in the EDIT mode.

EDIT MODE:

When the user enters this mode the response "EDIT:" will be typed on the user's console. At this time the user may type requests to the ED command. All changes made to a file become effective immediately and as a result, the user is able to make recursive modifications to his file. We may think of a pointer which is positioned at a line in the edited file. When the user enters the EDIT mode from the INPUT mode, this pointer will be positioned at the last input line typed by the user. When the user starts the ED command in the EDIT mode, the pointer is positioned before the first line in the old file. If the end of file is reached by an EDIT request, the comment "END OF FILE REACHED BY:" is typed on the user's console followed by the request which caused the end of file to be reached. At this time the pointer will be positioned after the last line in the file. When in the EDIT mode, any line which is not a legitimate EDIT request will cause the comment "NOT A REQUEST:" to be typed on the user's console followed by the line which caused the error. In many cases it is possible for the user to stack EDIT requests. If one of the requests causes an error message to be typed, any stacked requests will be ignored. This is done in case one of the stacked requests depended on the successful completion of the request in error.

Any number of initial tabs or spaces (including 0) may occur in a request line. Arguments and the request must be separated by at least one space or any number of tabs or spaces. Wherever the argument is line image, however, tabs and spaces retain their normal significance.

Error messages

Some errors from the file system will result in a PRNTER type error message followed by a question to the user of whether or not he wishes to continue. An answer of 'NO' will result in a call to DORMNT so that the user may SAVE the command, fix the problem, and RESUME the command.

EDIT <u>REQUESTS</u>:

REQUEST:	FIND line
ABBREVIATION:	F
RESPONSE:	none
ERRORS:	END OF FILE

The FIND request is used to move the pointer forward from its present position to the line specified by "line". "Line" is a normal input line and may contain tabs and backspaces. This line is used as a mask for selecting the desired line in the edited file. Matching is done only on the non-blank characters specified in LINE. For example, the request,

```
          F (tab)-(tab)-ALPHA,1
```

might be used to find the line,

```
          LOOP    TIX     ALPHA,1,4
```

REQUEST:	LOCATE string
ABBREVIATION:	L
RESPONSE:	none
ERRORS:	END OF FILE

The LOCATE request is used to move the pointer forward from its present position to the first line which contains the entire character string specified by "string". The full line of 84 characters is scanned, so that "string" may specify line numbers. It is recommended that "string" include the leading zeros of the line numbers to avoid any undesired match with program constants.

REQUEST:	NEXT I
ABBREVIATION:	N
RESPONSE:	none
ERRORS:	END OF FILE

This request is used to move the pointer forward from its present position in the file. "I" specifies the number of lines to be skipped over. If I is "0" or not specified, it is assumed to be "1" and the pointer will be moved to the next line in the file. If the NEXT request is given after the end of file has been reached, the pointer will be reset to the beginning of the file and moved "I" lines from there.

REQUEST:	DELETE I
ABBREVIATION:	D
RESPONSE:	none
ERRORS:	END OF FILE

The DELETE request will delete "I" lines from the file starting with the line at which the pointer is currently positioned. If I is "0" or left unspecified, only the current line will be deleted. _The pointer is left at the position vacated by the last line deleted by this request_.

REQUEST:	PRINT I -L-
ABBREVIATION:	P
RESPONSE:	printed lines
ERRORS:	END OF FILE

The PRINT request will print "I" lines from the file starting with the line at which the pointer is currently positioned. Upon completion of this request, the pointer will be left pointing to the last line printed. If I is "0" or left unspecified, one line will be printed. Normally lines are printed without line numbers. If the character "L" is present in the PRINT request, line numbers will be printed to the right of the printed lines.

REQUEST:	RETYPE LINE
ABBREVIATION:	R
RESPONSE:	none
ERRORS:	none

This request will cause the line at which the pointer is currently positioned to be replaced by LINE. LINE is a normal input line and may contain tabs and backspaces. _The pointer is not moved by this request_.

REQUEST:	TOP
ABBREVIATION:	T

This request will cause the pointer to be reset and positioned before the first line in the file. In addition, an automatic TOP is performed by the FIND, LOCATE and NEXT if the pointer was positioned at the end of the file.

REQUEST:	BOTTOM
ABBREVIATION:	B
RESPONSE:	INPUT:
ERRORS:	none

This request will cause the pointer to be positioned after the last line in the file. Upon completion of this request the user's console will be placed in the high-speed INPUT mode. All subsequent lines will be treated as input and added to the end of the file.

REQUEST:	INSERT or (C.R.)
ABBREVIATION:	I
RESPONSE:	INPUT:
ERRORS:	none

This request will cause the user's console to be placed in the high-speed INPUT mode. All subsequent lines will be treated as input and inserted after the line at which the pointer is currently positioned. If the INSERT request is given immediatly following a TOP request, the inserted lines will be placed at the beginning of the file.

REQUEST:	INSERT line
ABBREVIATION:	I
Response	none
Errors:	none

The INSERT request may be used to insert a single line without changing to the high-speed input mode. Line is a normal input line. It is inserted following the line at the present pointer position.

REQUEST:	CHANGE Qstring1Qstring2Q I G
ABBREVIATION:	C
RESPONSE:	none
ERRORS:	END OF FILE

This request will examine "I" lines starting at the line at which the pointer is currently positioned. Upon completion, the pointer will be left positioned at the last line examined by this request. If I is "0" or left unspecified, it is assumed to be "1" and only the current line will be examined. The character "Q" is taken to be the delineator or "Quote character" and may be any character in the 6-bit BCD set. "string1" and "string2" are arbitrary BCD character strings and may be of different lengths. If the character "G" (GLOBAL) is present, every occurrence of string1 will be replaced by string2. If "G" is not present, only the first occurence of string1 will be replaced by string2 in each examined line. EXAMPLES:

```
line:              ALPHA= ALPHA+ALPHA
request:   C *ALPHA*BETA*
new line:          BETA= ALPHA+ALPHA
request:   C *ALPHA*DELTA* 1 G
new line:          BETA= DELTA+DELTA
```

```
request:   C *DELTA+**
new line:        BETA= DELTA
```

REQUEST:	BLANK line
ABBREVIATION	BL
RESPONSE:	none
ERRORS:	none

The BLANK request will put blanks in the current line wherever non-blank characters appear in "line". For example 'BL ******' will clear the label field of a line in a FAP file.

REQUEST:	OVRLAY line
ABBREVIATION:	O
RESPONSE:	none
ERRORS:	none

The OVRLAY request will place the non-blank characters of "line" into the corresponding position of the current line. Notice that only non-blank characters of "line" replace what was in the current line. For example in a FAP file, if the current line is

```
                         TXI        *+1
then
              O  EOF(tab)   bbH  (tab)(tab)    comment
will produce
                 EOF        TXH        *+1       comment
```

REQUEST:	VERIFY
ABBREVIATION:	VE
RESPONSE:	none
ERRORS:	none

The VERIFY request sets the verify mode. In the verify mode, completion of any of the requests FIND, NEXT, LOCATE, OVRLAY, BLANK and CHANGE will cause the printing of the current-pointer line. In addition, CHANGE will cause the printing of all changed lines. Requests may not be stacked while in the verify mode.

REQUEST:	BRIEF
ABBREVIATION:	BR
RESPONSE:	none
ERRORS:	none

The BRIEF request sets the brief or normal mode. Within the brief mode, the FIND, NEXT, LOCATE, OVRLAY, BLANK, CHANGE requests will not give the responses expected in the verify mode.

REQUEST:	CLIP 'ON' or 'OFF'
ABBREVIATION:	CL
RESPONSE:	none
ERRORS:	ILLEGAL ARGUMENT:

The request CLIP ON sets a mode such that any input line which exceeds column 72 will cause the message "TRUNCATED:" followed by the faulty line image. Any waiting input lines will have been deleted. Requests on which this may occur are FIND, INSERT, RETYPE, OVRLAY, BLANK and high-speed INPUT. The request CLIP OFF resets the mode . The normal mode is CLIP ON for all files except FAP files which are normally CLIP OFF.

REQUEST:	SERIAL N
ABBREVIATION:	S
RESPONSE:	none
ERRORS:	none

This request is used to change the increment between line numbers of successive lines to the increment specified by the decimal integer "N". Initially, this increment is set to 10 by the ED command. If N is "0" or not specified, it is assumed to be "10". Lines inserted after a line with the line number "L" will be sequenced L+N, L+2N, L+3N, etc. If the lines following the inserted lines have line numbers which are less than or equal to the line number of the last inserted line, as many lines as necessary will be resequenced to insure that all line numbers are unique and in ascending order. For example, assume that "N" is 2 and the user wishes to insert 9 lines after line 25 in a file that was previously sequenced by fives. The inserted lines would be numbered, 27, 29, 31 ... 43. The lines previously numbered, 30, 35, 40, 45 and 50 would be renumbered to, 45, 47, 49, 51 and 53 respectively. The remaining lines in the file would be unchanged.

REQUEST:	COLON a
ABBREVIATION:	CO
RESPONSE:	none
ERRORS:	ILLEGAL ARGUMENT

A colon (or backspace on 1050) is a logical backspace anywhere, eg., 'ABC ::: Db(C.R.) is interpreted as 'DbC'. The colon moves the character pointer back one but does not erase the characters over which it has moved. One should be careful in using this convention that the total number of characters does not exceed 84, as any extras will be added to the next line during INPUT, or result in a request during EDIT.

The COLON request allows the colon character to be inserted as text. (They may also be 'CHANGE'd in as desired.) If 'a' is T or TEXT, all ':' will be treated as text except for the ':' as the first character after a tab. If 'a' is B or BACKUP the normal mode will be reinstated and all ':' will be backspaces.

REQUEST:	TABSET T1 T2 ... TN
ABBREVIATION:	TA
RESPONSE:	none
ERRORS:	ILLEGAL TAB SETTING

Ti specify the columns at which tabs are to be set. Tabs must be set in ascending order and may not exceed column 72.

REQUEST:	FILE -NAME4-
ABBREVIATION:	FL
RESPONSE:	*
ERRORS:	NO FILE NAME GIVEN
or	FILE WORD COUNT ZERO
	NOTHING IN FILE
	INPUT:

This request is used to terminate the editing process and write the new edited file on the disk. NAME4 specifies that the new file will be created as NAME4 NAME2. If NAME4 NAME2 is not specified, the old file will be replaced by the edited file or a new file NAME3 NAME2 will be created. If no name was given by the initial ED command or by the FILE request, an error message will be printed and the FILE request will be ignored.

If a file to be deleted is either READ-ONLY or PROTECTED, confirmation of deletion will be requested. If confirmation is denied or if file is LINKed, the EDIT mode will be reentered with the pointer at the top of the file. Any modes associated with the previous copy of the file will be transferred to the new copy.

(END)

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 programmer 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

```
     FILE NAME1 SAVED IS LOCKED, SAVE NOT EXECUTED.
```

Another user was referencing file NAME1 SAVED; the SAVE or MYSAVE must be repeated.

```
     ERROR n FOUND AT loc IN CALL TO entry FOR NAME1 SAVED
     SAVE NOT EXECUTED.
```

A file system error involving the file NAME1 SAVED has occurred. The SAVE must be repeated, probably with a new NAME1.

Restriction

If the user's memory bound is zero (i.e. no core image left), the SAVED file will contain only command buffers and directory switching information. Such a file cannot be restarted by RESUME or CONTIN. It can, however, be restored by RECALL, in order to load command buffers and restore directory attachment.

(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 progL (where 'prog' is the user's programmer number, used as the saved file name) 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.

(END)

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	I	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. If either name contains imbedded *'s, the "LISTF * convention" will be applied. That is, the * will match any character including blank.

This command in no way affects permission.

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

<u>Method</u>

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. 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)

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)

Identification

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

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)

Identification

Binary file editing program
EDB
J. H. Saltzer

Purpose

EDB is a reincarnation of the TYPSET and EDL editing commands for use with arbitrary binary files.

Usage

```
     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

QED text editor
Ken Thompson

Purpose

QED is a command for editing symbolic text. Its input and output are either console, 6-bit, 12-bit, or ASCII files, or a combination of these. QED keeps all text internally in the ASCII character set. extensive facilities for inserting, deleting and changing lines of text, a search feature, a macro feature and a large number of possible text buffers.

Discussion

QED, like most editors, performs operations on text in a workspace. In QED the workspace is called a 'buffer'. A buffer consists of from zero to (ideally) any number of lines of normal text. Each line _must_ be terminated in an end-of-line (carriage return) character. Not counting the end-of-line character, a line consists of from zero to (ideally) any number of characters.

QED, unlike most editors, has another level of hierarchy. The text in QED's workspace is broken up into from one to (ideally) any number of buffers. Each buffer is identified by a name of from one to five characters. There is _one_ current buffer and all of the other buffers are auxiliary buffers. The auxiliary buffers allow temporary workspace to store text. Any of the auxiliary buffers can become the current buffer; at which time the old current buffer becomes an auxiliary buffer.

QED accepts commands and text from a stream of characters. This stream normally comes from the console. Special characters in the stream can divert the stream to a text buffer. In this way, predefined commands can be placed in a buffer and then executed by diverting the command stream to this buffer. This buffer in turn may divert the stream to another buffer or (recursively) to the same buffer. At any time, the stream can be diverted to the console for one line of text.

QED has a very uniform command format. Each command acts on text in the current buffer and possibly on an entire auxiliary buffer. The text in the current buffer is specified by a series of from zero to (ideally) any number of line addresses. Two adjacent line addresses are separated by either a comma or a semicolon. Only the last two addresses are 'remembered' although one address may affect the evaluation of subsequent addresses. The command is represented by a single character. This character is usually mnemonic of the action of the command. Depending upon the command, qualifying data may be needed after the command character.

Actual details on commands and addresses follow.

REGULAR EXPRESSIONS

Regular expressions can best be described by example. In the following examples, the characters '/', '|', '*', '(' and ')' are operators in the expressions.

/a/ will match the letter 'a' anywhere on a line.
/abcd/ will match the word 'abcd' anywhere on a line.
/ab*c/ will match the words 'ac', 'abc', 'abbc', 'abbbc' ...
/abc|def/ will match the words 'abc' or 'def'
/(i|o)nto/ will match the words 'into' or 'onto'

The operators '{' and '}' have the same meaning as the operators '(' and ')'. When braces rather than parentheses are used to bracket sub-regular expressions, the regular expression in braces is named by the character immediately following the right brace. (See SUBSTITUTE and VERIFY commands.)

In addition, the characters '^', '.' and '$' are special. They are not operators, but just special characters. The character '^' will match the 0th character on a line. The character '$' will match the character after the last character on a line. The character '.' will match any character on a line.

/.*/ will match an entire line regardless of length.
/^begin|end$/ will match a line beginning with 'begin' or ending with 'end'.
/in.*to/ will match a line containing 'in' and 'to' in that order.
/^beg.*end$/ will match a line starting with 'beg' and ending with 'end'.
/^$/ will match a blank line.
/$^/ will also match a blank line.
/$.^/ will match nothing.

A null regular expression is identical to the last regular expression. (Upon initial entry, after a syntax error in a regular expression, and after a Read, Write, or List command, a null regular expression is an error.)

BUFFER NAMES

Buffers are named with a one to five character name. The name is enclosed in parentheses. If the name is one character long (not 'cr' or '(') the parentheses may be omitted. The buffer name can be any length, but only the last 5 characters are significant. The buffer names 'X' and '(X)' are identical.

TEXT ADDRESSING

Lines in the current buffer may be addressed in the following ways:

1) By relative line numbers.
   A decimal number is interpreted as a relative line number. The first line is numbered 1, the second 2, etc. The relative number of a line is its current position in the text buffer. This number may change during editing.

2) By absolute line numbers.
   The character ' immediatly followed by a decimal number is interpreted as an absolute line number. After a successful read command, every line in the current buffer is assigned an absolute line number that is the same as the relative line number at that time. The absolute line number does not change during editing except after a read. New lines created during editing have undefined absolute line numbers.

3) By '.'.
   The value of '.' is the current line. This value is changed by most editor commands.

4) By '$'.
   The value of '$' is the last line in the text buffer. This number may change during editing.

5) By context.
   The structure '/regular expression/' causes a search for a text pattern that matches the regular expression. The search begins at the line after the current line and cycles to the current line. If the search is successful, the value of '/regular expression'/ is the first line found containing the text pattern.

6) By additive combinations of 1-5.
   An address followed by '+' or '-' followed by another address is also an address. The value is obvious. Evaluation is done left to right. At no time during evaluation may an address exceed the bounds of the number of lines in the text buffer. In all unambiguous cases, the '+' may be omitted. ('.+4' is the same as '.4', but '5+2' is not the same as '52'.)

In subsequent discussion, '<u>A</u>' will indicate any legal address.

EDITOR INPUT

The input to QED is a stream of characters. Depending upon the context of the stream, some of the characters are interpreted as commands to the editor, and some of the characters are interpreted as literal text. In either case, the following characters are recognized by the editor as directives to the character stream and not as any editing function:

\Bx

These character are removed from the character stream and are replaced by all of the characters (in sequence) in buffer x (where x is the name of a text buffer.) Recursion is allowed to a depth of 500. The special case where the characters '\Bx' are the last two characters in a text buffer is treated specially and does not cause an increment of the recursion count.

\R

This character is removed from the character stream and is replaced by the next complete line from the console. Any partial line remaining from the console is skipped. In the line that replaces the \R character, the characters '\R' and '\B' cause no special action.

During console input, corrections may be made with the following control characters:

\E

Delete the preceding character. (The default alternate to \E is '#'. See 'e' option in the OPTION command.)

\W

Delete the preceding characters up to, but not including, the first blank followed by a non-blank character. In other words, delete the preceding word.

\K

Delete the entire line. (The default alternate to \K is '@'. See the 'k' option in the OPTION command.)

TEXT INPUT

There are three QED commands that expect to be followed by literal text input. This text must be preceeded by a space or a carriage return. The text itself consists of an arbitrary string of characters that terminates in the sequence '(cr)\F'. The '\F' character is not part of the literal text, but only serves to show the end of the text.

In subsequent discussion, '-text-' will indicate literal text input.

EDITOR COMMANDS

1) APPEND command.

   a) A A-text-

      The editor accepts text which is inserted after the line addressed. The value of '.' is set to the last line inputted.

   b) A-text-

      is identical to '$A-text-'.

2) BUFFER command.

   a) Bx

      The current buffer will become an auxiliary buffer and buffer x will become the current buffer. Initially the editor has buffer 0 as the current buffer.

3) CHANGE command.

   a) A1,A2 C-text-

      Lines in the current buffer A1 through A2 are deleted. The editor accepts text which is substituted in place of the deleted lines. The value of '.' is set to the last line inputted. The line number of A1 must be less than or equal to the line number of A2.

   b) A1 C-text-

      is identical to 'A1,A1C-text-'.

   c) C-text-

      is identical to '.C-text-'.

4) DELETE command.

   a) A1,A2 D

      Lines A1 through A2 are deleted. The value of '.' is set to the line after the last line deleted. The line number of A1 must be less than or equal to the line number of A2.

   b) A1 D

      is identical to 'A1,A1D'.

   c) D

      is identical to '.D'.

5) EXECUTE command.

   a) Ex

      The editor will execute CTSS commands out of buffer x. The execution is done four at a time. The current state of the editor is saved during the execution of a command in a temporary file PROGN SAVED, where PROGN is the users programmer number. The CTSS commands are taken one per line. Blank lines are ignored. No abbreviations are allowed. The core image left by every fourth command executed is destroyed by the restoration of PROGN SAVED. The value of '.' in the current buffer and in buffer x is not changed.

6) FACTS command.

a) Fx

The contents (if any) of buffer x will be replaced by the following six lines: date, time, problem number, programmer number, system name, and console id. The value of '.' in the current buffer is not changed. The value of '.' in buffer x is set to 0.

7) GLOBAL command.

a) A1,A2 Gc/regular expression/

Lines A1 through A2 are searched for text matching the regular expression. For every line found containing the regular expression, the QED command c will be executed. (c may only be 'p', 'd', 'z0', 'z1', 'z2', '=', or ':'.) The character '/' need not be used to delimit the regular expression. The first character after the command character will be used as the delimiting character. The value of '.' will be set to the last line searched.

b) A1 Gc/regular expression/

is identical to 'A1,A1Gc/re/'

c) Gc/regular expression/

is identical to '1,$Gc/re/'.

8) INSERT command.

a) A I-text-

The editor accepts text which is inserted before the line addressed. The value of '.' is set to A.

b) I-text-

is identical to '.I-text-'.

9) SORT command.

a) A1,A2 K

Lines A1 through A2 in the current buffer are sorted according to their assending ASCII colating order. The sorting time is 20*(A2-A1)**2 micro seconds. The value of '.' is unchanged. The line number of A1 must be less than or equal to the line number of A2.

b) A1 K

is identical to 'A1,A1K'.

c) K

is identical to '1,$K'.

10) LIST command.

a) Lx n1 n2(cr)

The editor will read file type x (x="a" for ASCII, "s" for line marked six-bit, "t" for line-marked 12-bit files) with CTSS name 'n1 n2' and print it. If n1 or n2 are missing, ASCII files have the default name 'ASCII', 6-bit files have the default name 'FAP', and 12-bit files have the default name '(MEMO)'. The value of '.' is unchanged.

11) MOVE command.

a) A1,A2 Mx

Lines A1 through A2 will be moved to buffer x. The old contents of buffer x will be deleted. The lines moved will no longer be in the current buffer. The value of '.' in the current buffer will be set to the line after the last line moved. The value of '.' in buffer x will be set to 0. Absolute line numbers of lines moved will also be moved. If x is the current buffer, this command is treated the same as '1,A1-1D A2+1,$D'. The line number of A1 must be less than or equal to the line number of A2.

b) A1 Mx

is identical to 'A1,A1Mx'.

c) Mx

is identical to '.Mx'.

12) OPTION command.

a) O-list-(cr)

The Option command is used to set internal options or modes of the editor. The list consists of any number of the following:

i) 'S' sets the editor input mode to convert all lower case letters in the command stream to upper case. Other characters are not affected. This option is automatically set when a type 's' file is mentioned in 'L', 'R', or 'W' commands. (Note that all characters in the command stream are affected, not just the characters typed at the console.)

ii) 'T' will set the editor input mode back to normal. This option is automatically set when a type 'a' or 't' file is mentioned in 'L', 'R', or 'W' commands.

iii) 'O' will remove the special meaning of the characters '(', ')', '*', '|', '{', '}', '.', '$', and '^' in regular expressions. The special meaning is restored locally by preceding the characters by '\C'.

iv) 'I' will restore the special meaning of the nine control characters in regular expressions. The special meaning of the characters is locally removed by preceding the characters by '\C'.

v) 'Bx' will give the character x the same meaning as '\B'. If x is a blank or a carriage return, any previous use of 'Bx' is removed.

vi) 'Cx' will give the character x the same meaning as '\C'.

vii) 'Fx' .. '\F'.

viii) 'Ex' .. '\E'. This option is preset to the number sign. (#)

ix) 'Wx' .. '\W'.

x) 'Kx' .. '\K'. This option is preset to the commercial at sign. (@)

xi) 'Rx' .. '\R'.

xii) 'pl' will set a printing option to preceed all lines printed with their absolute line number.

xiii) 'pa' will set a printing option to preceed all lines printed that have undefined line numbers with an asterisk. When two lines are printed with defined, non-sequential absolute line numbers, an asterisk is inserted between the lines.

xiv) 'pn' will set the printing option back to normal.

xv) 'v' (verbose) will cause QED to print any unexecuted commands after detection of an error while expanding a buffer. This is a useful mode to use while debugging QED programs.

xvi) 'q' (quick) will set the 'v' option back to normal.

xvii) 'd' (forbi_d_) will cause QED to reject any incomming inter-console messages. The PREN/PROBN of anyone sending an interconsole message is printed, but the command 'WRITE' is not called.

xviii) 'a' (_a_llow) will cause normal acceptance of inter-console messages.

xix) 'md*' (_m_ode) will cause QED to create any files in mode d*. (where d* is any number of octal digits.) 'Om104' will set the file mode to PROTECTED/READ-ONLY and 'Om1' will set the file mode to TEMPORARY.

13) PRINT command.

a) _A_1,_A_2 P

Lines _A_1 through _A_2 will be printed. All characters that have no graphic representation on the printing console are printed according to

MULTICS escape conventions. Two or more spaces that land on a tab stop will be printed as a tab. This convention is used to speed printing. (Tabs are assumed set at columns 11, 21, 31, ...) The value of '.' is set to the last line printed.

b) A1 P
   is identical to 'A1,A1P'.

c) P
   is identical to '.P'.

d) A1 (cr)
   is identical tc 'A1P'.

e) (cr)
   is identical tc '.+1P'.

14) QUIT command.

a) Q
   The editor will return to CTSS command level through the use of the CTSS routine 'CHNCOM'.

15) READ command.

a) A Rx n1 n2(cr)
   The editor will read file type x (x=a, s, t) with CTSS name 'n1 n2' and insert it after the line addressed. The value of '.' is set to the last line read.

b) Rx n1 n2(cr)
   is identical tc '$Rx n1 n2(cr)'.

16) SUBSTITUTE command.

a) A1,A2 S/regular expression/string/
   Lines A1 through A2 are searched for all occurrences of the regular expression. In general, the string is substituted for each occurrence. The value of '.' is set to the last line searched. The line number of A1 must be less than or equal to the line number of A2.

   The specific action of the substitute command is best described as follows:

   i) The next line is searched and all sequences of characters that match the regular expression are noted. If the search is done, stop. There are N1 such sequences where N1 is greater than or equal to zero.

   ii) If N1 is zero then go to i).

   iii) Of the N1 sequences, pick the sequences that end farthest to the right on the line. There are N2 such sequences where N2 is greater than or equal to one and less than or equal to N1.

iv) Of the N2 sequences, pick the sequence that starts farthest to the left. This sequence is unique. This unique sequence is replaced by the string to be substituted as described below.

v) Of the original N1 sequences, remove all sequences that end farther to the right in the line than the unique sequence begins. This will give a new N1 that is at least N2 sequences less in number.

vi) Go to ii).

During each substitution (step iv) the following characters in the string to be substituted are recognized and treated specially.

i) '&' is replaced by the unique sequence to be substituted.

ii) '!' is replaced by a character of the unique sequence as follows: If the unique sequence is N characters long, the string to be substituted for the unique sequence is scanned N times. On each scan, each character '!' is replaced by each character in the unique sequence in turn.

iii) A character that has been used to name a sub-regular expression in the regular expression (with braces) is treated as follows: The character is replaced by the sub-sequence of the unique sequence that matched the regular expression. If the sub-regular expression was not matched, the character will be replaced by a null sequence.

examples:

command	text	result
s/a/b/	abcdabcd	bbcdbbcd
s/cat/&&/	cat	catcat
s/^.*$/!/	abcde	abcde
s/^.*$/!!/	abcde	aabbccddee
s/cat/!&/	cat	ccatacattcat
s/cat/&/	cat	cat
s/c{a}xt/x/	cat	a
s/{{c}w{a}x{t}y}z/zyxw/	cat	cattac

b) A1 S/regular expression/string/
is identical to 'A1,A1S/regular expression/string/'.

c) S/regular expression/string/
is identical to '.S/regular expression/string/'.

17) TRAP command.

a) Tnx

The editor sets up buffer x to be expanded when an error of type n is detected. (Error numbers are at the end of this paper.) A subsequent 'T' command for a particular error number will override all previous settings for that error number. After the expansion of a buffer for an error, any errors of the same type will not cause buffer expansion. The expanded buffer can, of course, contain another 'T' command to allow buffer expanding on another error. A trap can be 'unset' with program control by setting the trap to expand buffer octal zero. (Example to 'unset' trap four: 'T4\000'.) Buffers 'carriage return' (oct 012) and 'octal zero' (oct 000) cannot be used as trap buffers. This command is meant mainly for QED programs and not for normal editing.

18) AUDIT command.

a) Ux

will replace the contents of buffer x with a QED edit that will reproduce all editing done on the current buffer since the last read on the current buffer. The value of '.' in the current buffer is not changed. The value of '.' in buffer x is set to 0. Buffer x becomes the current buffer.

19) VERIFY/SUBSTITUTE command.

a) $\underline{A}$1,$\underline{A}$2 V/regular expression/string/

is the same as the substitute command except that before each substitution, the line is printed with the found regular expression in red case. If substitution is to take place, an 's' must be typed at the console. Anything else will be taken as an indication that substitution is not to take place. No erase, kill, word erase, or escape processing is done on the response line to this command.

b) $\underline{A}$1 V/regular expression/string/

is identical to '$\underline{A}$1,$\underline{A}$1V/regular expression/string/'.

c) V/regular expression/string/

is identical to '.V/regular expression/string/'.

20) WRITE command.

a) $\underline{A}$1,$\underline{A}$2 Wx n1 n2(cr)

Lines $\underline{A}$1 through $\underline{A}$2 will be written on file type x (x=a, $\underline{s}$, t) with CTSS name 'n1 n2'. If no file exists with that name, one is created in permanent mode. (Alternate modes are possible with the 'm' option of the OPTION command.) If a file exists in a writable mode, it is first truncated to zero length and then rewritten. The contents of the buffer are not changed. The value of '.' is not changed. The line number of A1 must be less than or equal to the line number of A2.

b) A1 Wx n1 n2(cr)
   is identical to 'A1,A1Wx n1 n2(cr)'.

c) Wx n1 n2(cr)
   is identical to '1,$Wx n1 n2(cr)'.

21) BUFFER LIST command.

a) X
   The editor will list the name and length (in lines) of every buffer previously mentioned by the user in B, E, F, M, or U commands or in buffer expansion with the '\B' character. The first buffer listed is the current buffer. The value of '.' is not changed.

22) CANONICALIZE command.

a) A1,A2 Z0
   All overstruck characters on lines A1 through A2 are reordered according to their ASCII colating sequence. Trailing blanks and tabs are removed. This operation is automatically done to every line read from the console. The value of '.' is set to A2. The line number of A1 must be less than or equal to the line number of A2.

b) A1 Z0
   is identical to 'A1,A1 Z0'.

c) Z0
   is identical to '. Z0'.

d) A1,A2 Z1
   Lines A1 through A2 will undergo the Z0 transform and then two or more spaces that land on a tab mark are converted into a tab. (Tab stops are assumed set at columns 11, 21, 31, ...) The line number of A1 must be less than or equal to the line number of A2.

e) A1 Z1
   is identical to 'A1,A1Z1'.

f) Z1
   is identical to '.Z1'.

g) A1,A2 Z2
   Lines A1 through A2 will undergo the Z0 transform and then all tabs will be converted to the 'right' number of spaces. (Tab stops are assumed set at columns 11, 21, 31, ...) The line number of A1 must be less than or equal to the line number of A2.

h) A1 Z2
   is identical to 'A1,A1Z2'.

i) Z2
   is identical to '.Z2'.

23) RELATIVE LINE NUMBER command.

a) A =

The editor will print out the relative line number of the addressed line. The value of '.' will be set to A.

b) =

is identical to '$='.

24) ABSOLUTE LINE NUMBER command.

a) A1 :

The editor will print out the absolute line number of the addressed line. The value of '.' will be set to A.

b) :

is identical to '$:'.

25) COMMENT command.

a) "

The editor will skip all characters up to and including the next carriage return.

NOTES

The characters '{', '}', '\' and '^' are idealized versions of the ASCII left brace (oct 173), right brace (oct 175), back slant (oct 134) and circumflex (oct 136) respectively. To input these characters from different devices, normal MULTICS escape conventions apply. (See Multics Systems Programmers' Manual Section BC.2.04.)

If a semicolon is used to separate addresses instead of a comma, the value of '.' is set to the address immediately preceding the semicolon. This makes '/RE/,/RE/+10' identical to '/RE/;.+10'. The second example is more efficient.

If more addresses are preceding a command than are required, only the last addresses are used.

All letters recognized by the editor are recognized in both upper and lower case.

While the editor is accepting commands, blanks that are not in regular expressions or strings are ignored. Note that the buffer name immediately following 'T', 'B', 'E', 'F', 'M', 'U', or '\B' is a string.

In most cases, preceding a special character by '\C' will remove the special meaning from the character. The one exception to this rule is that one cannot search for a carriage return in a regular expression.

Hitting the interrupt button once will immediately drop recursion to 0. The editor will then be ready to accept commands from the console.

An address search for a regular expression that fails will drop buffer recursion by one. If recursion is at level 0 (commands being taken from the console), an error is noted.

An absolute line address for which there is no absolute line is treated as an error.

There is no safeguard to keep the editor from editing a buffer that it is using for edit commands. If this is ever done, havoc can be expected.

QED can be called with any number of parameters. If it is called with no parameters, the editor will take commands from the console. If it is called with parameters (say P1 P2 ... Pn) then the following edit will appear in buffer '.':

```
     b0
     ra P1 QED
     \b0
     P2
     .
     .
     Pn
```

The editor then simulates '\B.' typed at the console. This allows a bootstrap edit to be executed out of buffer 0.

QED will set inter-console communications permission from the file 'USER PROFIL'. If an inter-console message arrives, QED will save its current status in a temporary file PROGN SAVED and call the CTSS command WRITE. When WRITE returns, PROGN SAVED will automatically be restored with no changes in QED.

All text is kept in core. Core storage limits the maximum size of text that can be around. This maximum is about 20 disk tracks of text. In lines, it is between 2000 and 3000 depending upon density.

COMMENTS

The ideas for QED have come from a variety of sources. The most notable are TYPSET at MAC, and QED at U. of Calif.

SUMMARY

QED COMMANDS

	Command	Function	Value of '.'
($)	A text	append	last line input
	Bx	buffer	previous '.' for this buffer
(.,.)	C text	change	last line input
(.,.)	D	delete	line after last line deleted
	Ex	execute	unchanged
	Fx	facts	'.' of buffer x set to 0
(1,$)	Gc/re/	global	set by last c executed
(.)	I text	insert	unchanged
(1,$)	K	sort	remains on same line.
	Lx n1 n2	list	unchanged
(.,.)	Mx	move	after last line moved, in x to 0
	O-list-	option	unchanged
(.,.)	P	print	last line printed
	Q	quit	unchanged
($)	Rx n1 n2	read	last line read
(.,.)	S/re/st/	substitute	last line searched
	Tnx	trap	unchanged
	Ux	audit	unchanged, in x to 0
(.,.)	V/re/st/	verify	last line searched
(1,$)	Wx n1 n2	write	unchanged
	X	status	unchanged
(.,.)	Z0	canonicalize	last line canonicalized
(.,.)	Z1	canonicalize	last line canonicalized
(.,.)	Z2	canonicalize	last line canonicalized
($)	:	absolute line	addressed line
($)	=	relative line	addressed line
	"	comment	unchanged

SPECIAL CHARACTERS

\Bx	expand buffer x (=o030)
\C	escape next character (=o031)
\F	end of text (=o034)
\E	character erase (=o032)
\W	word erase (=o037)
\K	line erase (=o035)
\R	expand console line (=o036)

ERROR MESSAGES

?0	QED internal table overflow
?1	address search typed at console that fails
?2	illegal command or address
?3	illegal syntax in regular expression
?4	interrupt

?5	address reference out of buffer
?6	M, D, S, V, or C command with addresses that cross line 0. (ie 5,2D)
?7	maximum recursion level reached
?8	file cannot be opened. This includes reading a non-existant file, writing after track quota is reached, writing a protected file, etc
?9	illegal format for a command. This includes an illegal file type (not a, s, t), unrecognizable option in option command, etc.

(END)

Identification

Edit ASCII files
EDA

Purpose

EDA is a context-editing program for ASCII character stream files. Almost all of the EDA command's operation is identical to that of the TYPSET command (see AH.9.01).

Usage

```
          EDA name1 name2
```

EDA will edit the ASCII file "name1 name2". Editing conventions are identical to those of the TYPSET command, except that input and output is in the ASCII character set with the normal Multics conventions, and the "break" line used to transfer between input and edit modes is a line consisting of a single period instead of an empty line. The 'break' request is not implemented.

(END)

Identification

Move a file; Append one file to another
MOVE; APND
N. I. Morris

Purpose

MOVE may be used to copy a file from disk onto file system tape or to read a file from file system tape onto the disk. In addition, MOVE can be used in the same manner as TRNSMT SAVED to move a file through a link. APND is used to append one file to another.

Usage

```
MOVE  NAME1  NAME2  NAME3  -NAME4-
APND  NAME1  NAME2  NAME3  -NAME4-
```

NAME1 NAME2	is the name of the file to be MOVEd or APNDed.
NAME3 NAME4	is the name of the file to be written. If NAME4 is omitted, NAME2 is assumed. If NAME3 is '*', NAME1 is assumed.

Execution

NAME1 NAME2 is opened for reading and NAME3 NAME4 is opened for writing. (If MOVE is being used, NAME3 NAME4 is truncated to zero-length.) Then, the contents of NAME1 NAME2 are copied into (or appended to) NAME3 NAME4 by a high-speed file copying routine. This copying routine is triple-buffered so that tape to disk and disk to tape copying can actually run two data channels simultaneously. Note that the file NAME1 NAME2 is unaffected by MOVEing or APNDing.

Writing Tape Files

A file may be MOVEd from disk to tape by the following sequence of commands. (Assume that the tape has already been MOUNTed and LABELed.)

```
TAPFIL  NAME3  NAME4  UNIT  0
MOVE  NAME1  NAME2  NAME3  NAME4
```

Reading Tape Files

The first time a freshly TAPFILed tape file is read, several words of garbage may be appended to the last record of the file. This is caused by a paradox in the file system. (The first time a tape file is read, the file system doesn't know the correct length of the file. After the *entire* file is read once, the correct length will be updated into the file system.) MOVE will compensate for this file-system problem by determining the correct length of the file and ignoring the garbage words. Thus, MOVE should always be used to read a tape file for the first time. To MOVE a file from tape to disk, use the following sequence:

```
TAPFIL  NAME1  NAME2  UNIT  FILE
MOVE  NAME1  NAME2  NAME3  NAME4
```

<u>References</u>

<u>Section</u>	<u>NAME</u>
AG.5.05	Use of tapes in foreground
AH.3.06	Tape-handling commands

(END)

Identification

Context editor for card-image files
EDC

Purpose

To allow editing of fixed-length-record (14 word card image) files with EDL/TYPSET editing conventions, and provide slightly greater flexibility than is offered by ED.

Usage

Editing conventions are identical to those of EDL, with the following exceptions and additions:

The 'break' request is not implemented.

When editing a file with name2 .e. MAD (or MADTRN), a colon (":") in column 12 (or 7 with MADTRN) immediately following a tab is treated as a logical backspace, i.e. the next character appears in column 11 (6) on the card. All other colons are treated as ordinary text.

NCOLS n (abbrev. NC)

> Sets the number of card columns available to text. All columns after 'n' are blanked, except that card serialization may occupy columns 76 to 80. 'n' is initially set to 72, and may be changed to any value between 1 and 84. Setting NC to any number but 72 will reset serialization to OFF; to restore serialization, use 'SERIAL ON' (see below).

TABSET n1 n2 n3 ... (abbrev. TB)

> Informs EDC that tabs are to be interpreted as skips to the columns specified. For MAD files, tabs are initially set at 12, 17, 22, 27, etc.; for FAP files, tabs are initially set at 8, 16, 30, and every 4 columns thereafter; for MADTRN files tabs are initially set at 7 and every 5 columns thereafter. For all other files, tabs are set at every column (i.e. a tab becomes one space).
>
> Tabs inserted into the current line will print as tabs; tabs in any other lines will print as the appropriate number of spaces.

```
SERIAL m n          (abbrev. SR)
SERIAL OFF
SERIAL ON
```

The file will be resequenced in card col. 76-80, starting at 'm', incrementing by 'n', beginning at the current line. Resequencing is repeated on every pass through the file (i.e. after a 't' request). If 'OFF' is specified, sequencing is discontinued; the sequencing field will be blanked (or will contain data if NC .G. 76), beginning with the current line; sequence numbers will be removed on successive passes through the file. If 'ON' is specified, sequencing will be resumed from where it was discontinued; the entire file will again be resequenced on successive passes after 't' requests. This option is initially set to 'SERIAL 0 10'.

Restriction

Do NOT use EDC if it is desired to preserve existing sequence numbers in a file, except with 'NC 84', being careful that substitutions do not push col. 73-80 off the end of the line.

EDC is somewhat inefficient, and is not recommended for editing very large files (i.e. several hundred records or more).

(END)

Identification

Combine seldom-used files
ARCHIV

Purpose

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

Restrictions

An archive file may contain files of any name, but unless the secondary name of the archive file is "SOURCE" or "ARCHIV", files in the archive whose secondary name is different from the archive file's secondary name cannot be referenced. For example, if the archive "ALL MAD" contained "XXX FAP", XXX could not be extracted without renaming the archive.

If the second name of an archive file is "SOURCE" or "ARCHIV", the arguments FIL1 ... are taken in pairs to represent primary and secondary names of files to be operated on. Otherwise, the arguments are taken singly, and represent primary names of files which have secondary name the same as the archive.

Files to be deleted are deleted by the library subroutine DELETE. Its conventions are 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=TOFF:	Like "T", but writes the table of contents into the file "ARCHIV OUTPUT", a line-marked six-bit file.

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, a message is printed and the command is executed as ARCHIV C NAME1 NAME2 NAME1 FILi.

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., as if the call had included every entry in the archive.

<u>Method</u>

Each entry in the archive file consists of a header, in which file name, date and time last update, 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 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'.

(END)

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.

(END)

Identification

File compression and expansion
SQUASH, XPAND

Purpose

SQUASH converts a card-image file to a 6-bit linemarked file. XPAND converts a linemarked file to a card-image file.

Usage

```
SQUASH NAME1 NAME2 NAME3 -NAME4-

XPAND  NAME1 NAME2 NAME3 -NAME4-
```

At least three arguments must be given to SQUASH or XPAND.

NAME1 NAME2	is the name of the file to be converted.
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.
SQUASH	converts 6-bit card-image files to linemarked format. 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 preceded 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. Experience with a variety of FAP and MAD programs indicates that a saving of from 60% to 75% of storage space is typical.
XPAND	converts linemarked files to card image format. 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.

(END)

Identification

Compress and expand BSS files
PADBSS and SQZBSS

Purpose

To compact BSS files by a factor of 2 in order to save disk space.

Usage

```
SQZBSS ALPHA -BETA-
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)

Identification

Archive ASCII files
AARCHV

Purpose

AARCHV is a version of the ARCHIV command (see section AH.4.01) for files in the ASCII character set. The reason for having a separate command is so that archives of ASCII files can be printed off-line. The ARCHIV command scatters header information in BCD through the archive file, which prints as garbage in ASCII. In addition, since ASCII files usually end with the character ETX (octal 003) to indicate the end of the file, only the first file in a regular archive of ASCII files will print off-line.

Usage

```
AARCHV KEY NAME1 NAME2 FIL1 ... FILn
```

AARCHV	is an adaptation of the regular ARCHIV command. See section AH.4.01 for details on its use. Only the differences between AARCHV and ARCHIV are described below.
KEY	The keys "P" and "IOFF" are not implemented.
FILi	The secondary names "SOURCE" and "ARCHIV" are not treated specially by AARCHV. All names FILi refer to a file "FILi NAME2".

The internal format of an ASCII archive file is somewhat different from a regular archive. All name and date information is kept in ASCII, and the length of the file is given in characters. Since the AARCHV header contains a NP (octal 014) character, each file in an ASCII archive file will begin printing on a new page if the file is printed offline.

(END)

Identification

Create or append to ARCHIV format files
APENDA

Purpose

To allow more efficient addition of new subfiles to existing archives, and to allow somewhat more efficient creation of archives. To allow creation of achives on CTSS foreground tapes without creating an intermediate disk file.

Usage

```
APENDA N1 N2 f1 f2 f3 f4 ...
```

Files 'f1 f2', 'f3 f4', etc. are appended to file 'N1 N2', preceded by archiv headers.

```
APENDA N1 N2 '*' f1 f2 f3 ...
```

Files 'f1 N2', 'f2 N2', 'f3 N2', etc. are appended to file 'N1 N2', preceded by archiv headers.

Method

File 'N1 N2' is initially opened for writing. Each file to be appended is then in turn opened, copied onto the end of 'N1 N2', and closed. In the event of any file errors, all files are closed and a diagnostic is printed. Note that if a file-not-found error is received while in the middle of a long series of append operations, all previously appended files will have been processed properly, and the operation can be continued later.

Restriction

If APENDA is used to append a file which is itself an archive to another archive ('super-archive'), the entire file appended is treated as a single subfile. Performing the same operation with the 'ARCHIV C' option results in extracting all the subfiles from the file to be added, and including these in the archive as distinct subfiles. Furthermore, using the 'ARCHIV U' option to update such a 'super-archive' will cause all embedded archive subfiles to appear as distinct subfiles of the main archive.

(END)

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.

SPECIFICATION TABLES

SEARCH SPECIFICATIONS

META-ARG	MODIFIERS	ACTION	DEFAULT	COMMENTS
(FILE)	None	ignores links		
(LINK)	None	links only		
			lists all files	
(UFD)	name of file linked to other U.F.D. (FILE)	searches the linked directory	U.F.D. (FILE)	
(SYS)	None	searches the public files		
(CFLn)	None	searches the user's common file n		
(AUTH)	author nos.	files created by specified author only	any author	
(MODE)	1 or more arguments, each having 1-4 of: 0,T, S,R,W,L,P,*, enclosed in parentheses	files with specified modes only	any mode	(RP) = 104, (RP*) = at least 104, (R) (P) = 100 or 004
(USED)	up to two dates MMDDYY or '(OLD)' or '(NEW)'	files with date used between the dates specified	(NEW) (OLD)	(NEW) is the present date (OLD) is the oldest date
(MADE)	see (USED)	files with date modified between dates given	(NEW) (OLD)	see (USED)

SORTING SPECIFICATIONS

META-ARG	MODIFIERS	ACTION	DEFAULT	COMMENTS
(SDIR)	None	file directory order		
(SNA1)	None	sort on NAME1		
(SNA2)	None	sort on NAME2		
(SMOD)	None	sort on octal file mode, in descending order		
(SREC)	None	sort on file size, largest first		no links*
(SUSE)	None	sort by date used		no links*
(SMAD)	None	sort by date modified		no links*
			(SUSE) for files, (SNA1) for links	
(REV)	None	reverses sorting order		

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

OUTPUT SPECIFICATIONS

META-ARG	MODIFIERS	ACTION	DEFAULT	COMMENTS
(LSUM)	None	summary lines only		
(LNAM)	None	NAME1-NAME2 only listed		
(LONG)	None	normal form plus date/time modified, auth. device, lock for files; mode, date/time used and modified, auth., norecs, device for links		if (UFD) was requested, links listed in normal form
(ON)	None	output printed on terminal (normal mode)		
(OFF)	None	output written into file LISTF OUTPUT for offline printing via 'RQUEST'		(HDR) is assumed
(HDR)	None	listing prefixed by date/time, file directory name		
(NHDR)	None	suppresses the header when (OFF) requested (normal mode)		

Example_1

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

 (8)    (9)    (10)    (11)    (12)     (13)    (14)    (15)
(MODE)  (RP)   (W*)   (AUTH)  99999   (CFL2)  (USED)  090165

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

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

1. non-linked$^{(4),(11),(14),(17)}$

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

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

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

5. in common file 2$^{(13)}$

6. last used on or before 9/1/65$^{(15)}$

7. last modified on 8/1/65$^{(18),(19)}$

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)
```

(END)

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.

(END)

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.

(END)

Identification

Print contents of a file in octal
PRBIN

Purpose

Print on the user's console (or in a file for later disk editor printing) 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 -start- -"THRU"/"TO"/"..." end- -nwords-
   -delta- -blksize- -"OFF"/"OFFON"/"ONOFF"- -name3- -name4-
```

start (optional) indicates the location of the first word in the file to be printed. If it is not specified or is "*", it is assumed to be "1". A starting location of "0" will be turned into a "1".

end (optional but only when accompanied by "THRU", "TO", or "...") specifies the last location to be printed. If omitted or "*", the remainder of the file will be printed.

nwords (optional) indicates the number of words to be printed in a block. These blocks are printed in groups of 'n' words per line where 'n' is either 'nwords' or the number of words which can be typed on the console ("8" on a 2741 or 1050, "5" on an ARDS, "9" in the offline file, or "6" otherwise) whichever is the smaller. If omitted or "*", 'nwords' is assumed to be "5", "6", "8" or "9" depending upon which console the user is at and whether or not he is creating an offline file. If 'nwords' is used, the 'start' location must also be present.

delta (optional) allows the "skipping" through the file printing every 'delta' words. If this argument is used both the 'start' location and 'nwords' must be specified. If it is not present or is "*", it is assumed to be "1".

blksize (optional) is the number of words to be printed starting at every delta'th location in the file. It must not be greater than delta. If not specified or is "*", it is assumed to be "1".

"OFF" (optional) will _append_ to a file 'name3 name4' the octal print of the file 'name1 name2'. If "OFFON" or "ONOFF" is used, the printing will also appear online. If 'name3' is omitted, it is assumed to be the same as 'name1'. If name4 is missing, it is assumed to be "   BIN".

"O" (optional) may be used to indicate any of the above counts are specified in octal. It appears as a separate argument preceeding the count (either 'start', 'end', 'nwords' or 'delta'). If missing the counts are assumed to be decimal. If the start location is specified in octal using the "O" argument, the location printed at the beginning of each block will also be expressed in octal.

(END)

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.

(END)

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 mnemonics.

'-----' 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:

```
NAME1 SAVED NOT FOUND.
SAVED FILE HAS IMPROPER FORMAT.
LOCATION NOT IN SAVED FILE.
File system diagnostics from 'PFNTER'.
```

(END)

Identification

Print an ASCII file
PRINTA

Purpose

PRINTA will type the contents of an ASCII file.

Usage

```
          PRINTA name1 name2
```

PRINTA will type the file name in red, followed on the same line by the current date and time, then a blank line, followed by the file contents.

Restrictions

Lines longer than 480 characters will be truncated. There is no provision for special action such as printing parts of lines or beginning in the middle of the file. The only arguments to the command are the file's names.

(END)

Identification

Print a file as rapidly as possible
P

Purpose

To print the contents of a file, which may be 6-bit linemarked or card-image or 12-bit linemarked or ASCII. Sequences of spacing characters are chosen to take the minimum possible time for the particular console being used.

Usage

P NAME1 NAME2 -options-

If no options are specified, P will look at the first part of the file to determine whether it is 6- or 12-bit or ASCII. If the file is 6-bit and linemarked and every line of the sample begins with a valid carriage-control character, carriage control will be interpreted. Tabs in the input file and at the console are assumed to be set at every 10 columns (11,21,...). An interrupt will cause P to go to CHNCOM.

Options are:

6:	forces 6-bit mode
9:	forces ASCII mode
12:	forces 12-bit mode
CC:	forces interpreting carriage control
NCC:	forces not interpreting carriage control
WR:	if a line in the file is too long for one output line, it will be continued on the next line. Normally the excess will be ignored.
PS:	two lines will be skipped at the beginning and end of each page
HDR:	a header giving file name, date and time, and page number will be printed at the top of each page; two lines will be skipped at the end of each page
DATE:	a header giving file name, date, and time will be printed at the top of the first page; two lines will be skipped at the beginning and end of each page
HELP:	a list of valid arguments to P will be printed
ITAB #:	tabs in the file are interpreted as set at every # columns
OTAB #:	tabs at the console are assumed set at every # columns
LL #:	the maximum number of characters per output line is set to #

```
       PGL #:  the number of lines per page is set to #
  FILE name4:  an ASCII file called name1 name4 is written
               instead of console output being produced
  Any of the above arguments may be enclosed in parentheses
    BLKSIZ #:  linemarks, if any, are ignored and a carriage
               return is inserted after every # words of
               input
ITABS or OTABS list:  input or output tabs are assumed to be
               at the locations specified by 'list'.
               'list' may be any combination of:
                  #:  a tab is set at column #
               CTSS:  tabs are set at 15,30,45,...
                FAP:  tabs are set at 8,16,35,40,45,...
                MAC:  tabs are set at 12,17,22,27,...
                      Also, a colon in column 12 is taken to
                      be a backspace
```

(END)

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".

(END)

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 (RCNC).

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.

(END)

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 file system entry DELFIL to delete file NAME1i NAME2i from the current file directory. If for any reason a file cannot be deleted, a message is printed:

```
NAME1i  NAME2i  NOT DELETED
```

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 "* * NOT FOUND" will be printed. If the name contains imbedded *'s, the "LISTF * convention" will be used. That is, the * will match any character including blank.

NAME2i is the secondary name of a file to be deleted. If NAME2i is *, all files of primary name NAME1i will be deleted. If the name contains imbedded *'s, the "LISTF * convention" will be used.

Change mode:

```
CHMODE NAME1 NAME2 MODE1 ... NAME1n NAME2n MODEn
```

Modes may be expressed in combinations of octal or alphabetic mode designations (see below) and the special characters * (taken to be the present mode of the file) and "/" (to mean "remove the following mode bits" from the mode being created). An initial "/" implies a preceding *.

letter	octal	meaning
0	0	permanent
T	1	temporary
S	2	removed ("secondary")
R	4	read-only
W	10	write-only
V	20	private
L	40	reserved for system use
P	100	protected
M	200	"being restored from tape"

Up to six of the letters or octal numbers can be concatenated to form combination modes; for example:

PR = 104	protected/read-only
*V	add private to previous mode
4P20 = 124	protected/private/read-only
/T = */1	remove temporary mode bit

NAME1i NAME2i - The same * conventions are 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. The deleting of NAME3 NAME4 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
```

NAME1i NAME2i - The same * conventions are used as in the DELETE command.

NAME3i NAME4i - If either NAME3i or NAME4i contain imbedded *'s, the *'s will be replaced by the appropiate character from NAME1i and NAME2i. If NAME4i is missing, it is assumed to be NAME2i.

(END)

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.

(END)

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.

(END)

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 11, 21, 31, (+10) ..., 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='SSPRNT': The BCD file NAME1 NAME2 will be printed with a leading blank on each line to insure single space printing. Line numbered files are always printed single spaced.

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='PLOT': The file NAME1 NAME2 will be placed on the plot output tape for plotting on the CalComp plotter. (see APM-1)

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

The file(s) will be PRINTed, SSPRNTed, DPUNCHed, BPUNCHed, 7PUNCHed, or PLOTted 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.

<u>Method</u>

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. If either of the names contains a "*", the RQUEST command will search the file directory for all file names corresponding to the requested name according to the LISTF "*" conventions. 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)

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. 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 *. Trailing optional arguments may simply be omitted.

Responses

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

Summary of Possible Uses of Call

1. CALL UPDMFD PROB PROG
2. CALL DELMFD PROB PROG
3. CALL ATTACH PROB PROG
4. CALL MOVFIL N1 N2 PROB PROG
5. CALL SETFIL N1 N2 MMDDYY HHMM MMDDYY AUTHNO -MODE- -DEV-
6. CALL LINK N1 N2 PROB PROG -P1- -P2- -MODE-
7. CALL ALLOT DEVICE -ALLOTTED- -USED-
8. CALL UPDATE

9. CALL OPEN STATUS N1 N2 -MODE- -DEVICE-

10. CALL BUFFER N1 N2 -LENGTH-

    a. Only one active file can be buffered at any one time.
    b. If 'LENGTH' is not given, it is set to zero.

11. CALL RDFILE N1 N2 -RELLOC- -COUNT-

    a. If 'COUNT' is not given, it is set to 1.
    b. 'COUNT' .LE.20; if 'COUNT' .G.20, then it is taken as 20.
    c. If 'RELLOC' is not given, it is taken as 0.
    d. 'COUNT' words are printed out. If the EOF is reached or passed, the word 'EOF' precedes the output.

12. CALL RDWAIT N1 N2 -RELLOC- -COUNT-

    a. See remarks a-d Item 11.

13. CALL WRFILE N1 N2 -RELLOC- -OCTLH- -OCTRH-

    a. 'OCTLH' and 'OCTRH' are converted to octal and treated as one word. Default value for 'OCTLH' and 'OCTRH' is 0.
    b. Default value for 'RELLOC' is 0.
    c. If 'RELLOC' lies beyond the end of file, the word 'EOF' is printed and no writing is done.

14. CALL WRWAIT N1 N2 -RELLOC- -OCTLH- -OCTRH-

    a. See remarks a-c Item 13.

15. CALL TRFILE N1 N2 -RELLOC-

    a. Default value for 'RELLOC' is 0.
    b. If this is the first usage of the 'CALL' command, the following lines are generated.

```
OPEN.($W$,N1,N2)

BUFFER.(N1,N2,BUFF(432)...432)

TRFILE.(N1,N2,RELLOC)

CLOSE.(N1,N2)
```

    Otherwise, the standard call to 'TRFILE' is generated.

16. CALL FCHECK N1 N2

a. If the I/O is completed for 'N1 N2', the word 'FINISH' is printed, otherwise nothing is printed.

17. CALL CLOSE N1 N2 (or CALL CLOSE ALL)

18. CALL FWAIT N1 N2

19. CALL SETPRI -PRIOR-

20. CALL RESETF

21. CALL CHFILE N1 N2 -MODE- -NEWN1- -NEWN2-

22. CALL DELFIL N1 N2

23. CALL FSTATE N1 N2

    a. Response is: LENGTH MODE STATUS DEVICE
       NEXT-READ NEXT-WRITE
       DLM TLM DLU AUTHNO
    b. If file is a link, 'MODE' will appear as 'L' MMM.

24. CALL UNLINK N1 N2

25. CALL STORGE DEVICE

    a. Response is: ALLOTTED USED

26. CALL ATTNAM

    a. Response is: ATTACHED-PROB ATTACHED-PROG
       AUTHNO PRIORITY

27. CALL IODIAG

28. CALL EXIT

    a. Returns via a call to CHNCOM. If any file was opened during this usage of 'CALL', EXIT will leave core image, otherwise, no core image is left.

<u>Special Arguments to 'CALL'</u>

1. (GO) - Inclusion of this argument anywhere in a call command line will cause the command to type the word 'CALL' and then wait for further file call instead of exiting after processing the indicated call. Sample uses of this argument might be:

```
CALL OPEN R   A   B   (GO)
CALL RDWAIT   A   B   1  5
CALL CLOSE ALL
CALL EXIT
```

2. (FNS) - Inclusion of this argument indicates to 'CALL' that a fence (Octal 777777777777K) is to be placed in this position and that scanning of the command line is to continue.

3. * - This argument specifies that an optional parameter has not been supplied. 'CALL' will pass a null parameter to the file system.

4. (STAR) - Inclusion of this argument indicates that an asterisk is to be placed in this position and scanning of the command line is to continue Note that this is not equivalent to the special argument,'*'.

(END)

Identification

Attach to another user's file directory
ATTACH

Purpose

To allow a user to attach to another user's file directory for the purpose of examining and/or modifying his files.

Usage

```
        ATTACH Prob Prog
```

If the command user is explicitly permitted to LINK to the file "U.F.D. (FILE)" in the directory "prob prog" in mode 0, ATTACH will change the command user's working directory to be "prob prog".

If "prob prog" are omitted, ATTACH will reattach the user to his home file directory. A user may also return to his home directory by using the COMFIL command, or by executing a program which calls COMFIL.

If a user's program calls TSSFIL (see AG.3.03), the supervisor will save the name of his current attached directory, and will restore it when USRFIL is called.

To give another user permission to attach to your directory, type "PERMIT U.F.D. (FILE) 0 probn progn", where "probn" and "progn" identify the user or set of users to whom you wish to give permission. Note that you must name "U.F.D. (FILE)" explicitly: "* *" will not do.

(END)

Identification

Append files
APEND

Purpose

To allow files to be combined together; to allow new files to be appended to existing files; to allow files to be combined into a CTSS tape file without creating an intermediate disk file.

Usage

```
APEND N1 N2 f1 f2 f3 f4 ...
```

Files 'f1 f2', 'f3 f4', etc. are appended to file 'N1 N2'.

```
APEND N1 N2 '*' f1 f2 f3 ...
```

Files 'f1 N2', 'f2 N2', 'f3 N2', etc. are appended to file 'N1 N2'.

Method

File 'N1 N2' is initially opened for writing. Each file to be appended is then in turn opened, copied onto the end of 'N1 N2', and closed. In the event of any file errors, all files are closed and a diagnostic is printed. Note that if a file-not-found error is received while in the middle of a long series of append operations, all previously appended files will have been processed properly, and the operation can be continued later.

(END)

Identification

Off-line ASCII printing
RQASCI

Purpose

Allow the user to request printing of ASCII character stream files with the ASCII chain on the Center's specially modified 1401.

Usage

```
          RQASCI NAME1 NAME2 NAME3 NAME4 ...
```

The files 'NAME1 NAME2', 'NAME3 NAME4', etc. will be printed during the next run of the ASCII editor. Currently, the ASCII editor is run twice daily, at 0400 and 2200.

If NAME2 of a file to be printed is RUNOFF (See AH.9.01, AH.9.06: 'print' option on RUNOFF and ROFF commands), the file is printed exactly at it appears. If NAME2 is <u>not</u> RUNOFF, form-feed characters (ASCII 014) are inserted at the bottom of each page, to skip over the perforations on the paper.

Method

RQASCI writes (through a link) into the file ASCII RQUEST maintained in the directory M1416 2962. This file is read by the ASCII editor. The file is in private mode, and is accessible only via the RQASCI command. The link made to ASCII RQUEST is left in the current file directory.

Restrictions

A request made with RQASCI <u>cannot</u> be deleted, as is possible with the standard disk editor and RQUEST commands.

(END)

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)

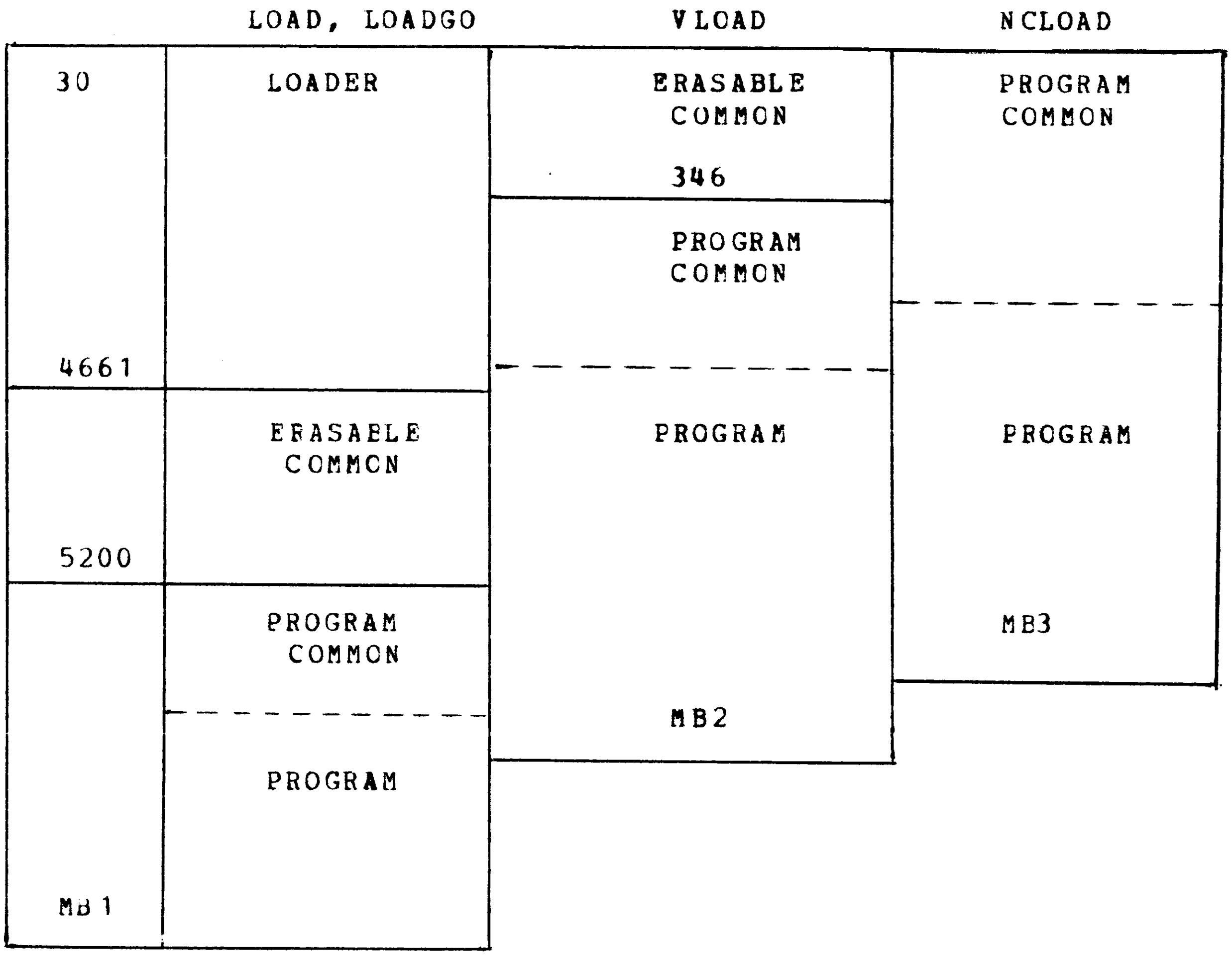
.. Pairs of words 5 and 6 for each entry to the subprogram.

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)

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)

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.

<u>Summary</u>

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

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 satisty 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.

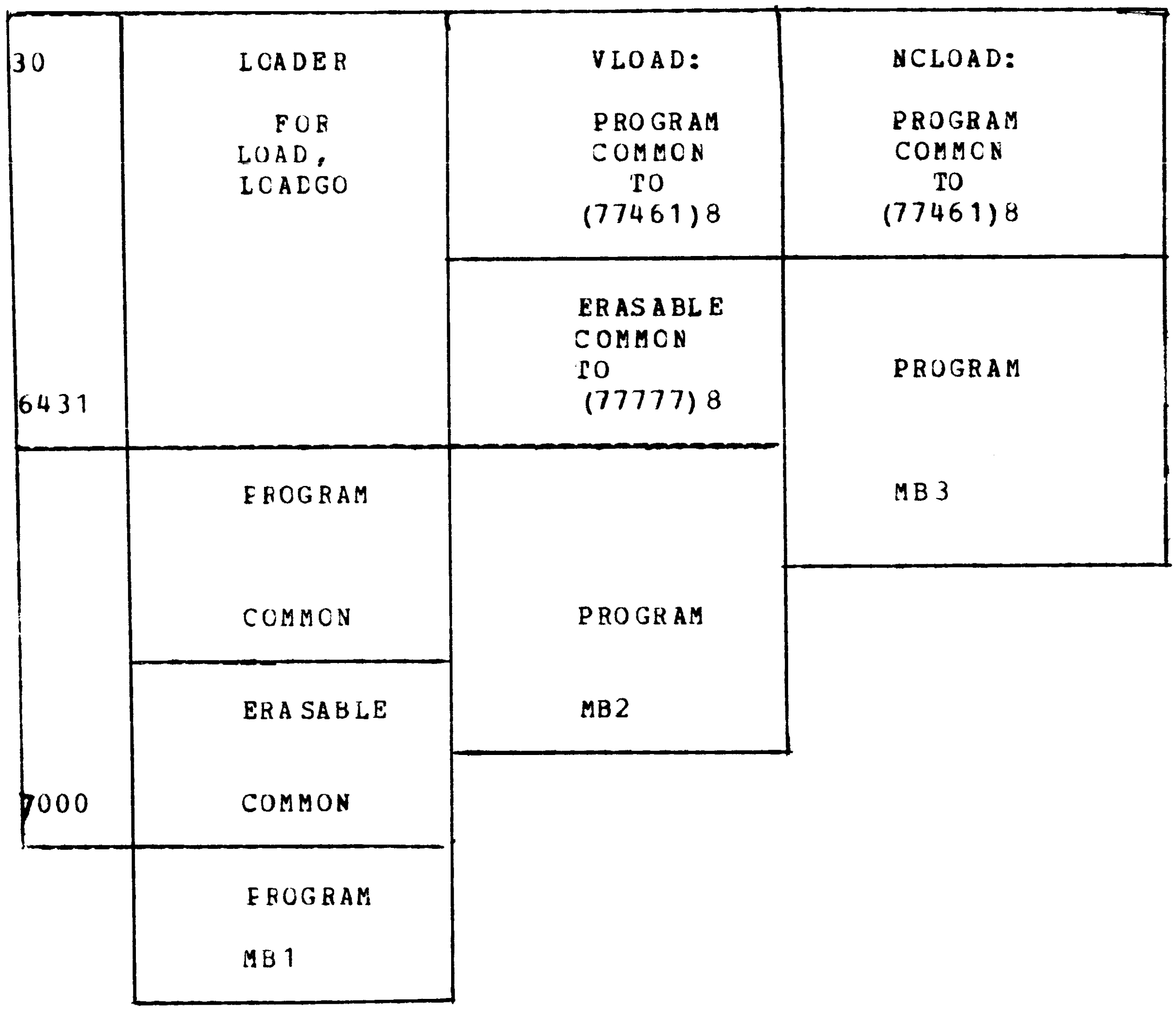
*** controls the printing of missing subroutines. If *** is PZE these will be listed by LAED. The message will be supressed if MZE is used.

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

Execute saved programs from common files
DO

Purpose

DO enables the user to execute saved programs from his common files or from the public file directory without using links.

Usage

```
DO n NAME arg1 arg2 ...
```

DO will switch to the directory specified, load the saved file "NAME SAVED" into core, and start it at location 24(8). No machine conditions are restored. "NAME" will have its arguments available to it just as if it had been RESUMEd.

The parameter "n" specifies the location of the saved file. If "n" is "*", the saved file is loaded from the current directory. If "n" is "P", the saved file is loaded from the system public file (M1416 CMFL04). If "n" has any other value, the saved file is loaded from the user's common file "n".

(END)

Identification

Simulation of the loading commands
PLOAD

Purpose

PLOAD simulates the loading of a system through the use of a load file. It will produce a list of missing files and/or subroutines, if any, as well as a cross referenced storage map.

Usage

```
PLOAD NAME -type- -'COMB'- -'NOLIST'-
```

NAME	The primary name of the LOAD file from which the list of files is to be taken. The secondary name is "LOAD" and the file must be line numbered.
type	may be used to specify the type of loading desired. If a type of "NCLOAD" is used loading will be simulated starting at 30(8); if the type is "VLOAD" the loading will be simulated starting from location 346(8); otherwise 7000(8) is used when preparing the cross reference storage map.
COMB	is used when one or more of the files is a library (i.e. contains more than one program).
NOLIST	is used to suppress the creation of the cross reference storage map (NAME STCMAP).

Restrictions

Under the present implementation of PLOAD it is not possible to use the special options recognized by the LAED loaders (e.g. "(LIBE)", "(SRCH)", "(CPLn)" etc.).

Note also that PLOAD does not relocate program common, as is done by the relocatable loaders. Therefore, if any common is used, the loading addresses shown in the storage map will be incorrect by the amount of common in use.

(END)

Identification

Set execution timing response
BLIP

Purpose

To allow a user to set his 'blip' switch from command level.

Usage

To set the 'blip':

```
                    BLIP -n-
```

If 'n' is not specified, it is assumed 2. If n .NE. 0, the command responds with 'Type: '. The following characters, up to but not including the carriage return, and to a maximum of three (12-bit characters) will thereafter be typed on the user's terminal every 'n' seconds of execution time.

To reset (turn off) 'blip':

```
                    BLIP 0
```

The switch indicating that the user has his 'blip' option on is reset, and the command exits.

Example

The character sequence 'space backspace' is useful, since it does not cause any printing, but gives evidence of execution timing by the carriage motion.

Note: Swap time and disk-command load time are not included in 'execution time' for this application.

(END)

Identification

Dispatching to and Accessing TIP Utility Programs and other Data-manipulation Programs
RUN

Purpose

The RUN system provides a uniform method of obtaining access to programs associated with the use of the TIP data-manipulation system.

TIP users have occasion to employ many different programs in the process of establishing data bases, manipulating data, and formating useful output. Some programs are of command status (e.g., QED), some are public commands (e.g., TAPLF), some are maintained in TIP common files (e.g., SORT), some are accessible exclusively through the RUN system (e.g., PUTOUT), and some are EDIT, FAP, and MAD programs written and maintained by individual users for their own purposes. The RUN system makes accessible in a uniform manner programs of all five types. This simplifies the maze of links, saved files, calling sequences, etc., which confronts the CTSS user. The system insulates the user from changes in system organization, etc. and relieves him from maintaining many links to many saved files.

The RUN system permits a number of programs to be loaded in a single saved file. If these programs call many of the same subroutines, then by eliminating the duplication of these subroutines over many saved files disk storage is conserved, at the cost of a very slight increase in execution time.

Usage

The calling sequence for RUN is:

```
RUN COMAND -ARG1- ... -ARG17-
```

COMAND is any program listed in the file, RUN SYSTEM, which is described below; or any TIP command, or any saved file in the user's current file directory, or any public command or any CTSS command.

-ARG1- ... -ARGn- are the arguments to COMAND, if any.

Method

RUN reads an internal table to determine if COMAND is a RUN command, and if so, which saved file contains it. RUN then attaches to TIPFIL, M4959 CMFL02, where the RUN saved files and TIP commands are found and attempts to resume the proper saved file. If the desired saved file cannot be found there, RUN will look in the user's own directory. If this is unsuccessful, RUN attaches to the public file, M1416 CMFL04, and makes another attempt. If the saved file is still not found, COMAND is assumed to be a CTSS command and NEXCOM is called. If COMAND is found in a saved file, RUN resumes the proper saved file, transferring control to the called program, except when the program is contained in a RUN saved file, in which case an intermediate program returns to free storage all subroutines not used by the called COMAND before transfering control to the entry point of COMAND.

The 6-bit line-marked file, RUN SYSTEM, in M4959 CMFL02 contains a list of all available RUN programs and the saved files in which they are found. It may be LINKed to by any user and PRINTed:

```
          run link run system m4959 cmfl02
          run print run system
```

Examples

A user wishes to convert some personal information to TIP format. He might use the following sequence of commands:

```
          run eda my field
           FILE     MY FIELD NOT FOUND.
          Input
          *!name!address!phone
          .
          Edit
          file
          *

          run reduce my field my fld 052 041
          1 items in, 1 out.
```

```
run eda my input
 FILE     MY   INPUT NOT FOUND.
Input
*
!name John D. Smith
!address 2 High Road
!phone 864-6900
.
Edit
file
*

run reduce my input my data 052 041 (tabl) my fld
 1 items in, 1 out.
```

The file MY DATA now contains TIP-searchable information detailing the personal situation of John D. Smith.

Suppose that a user had a large file named CIG DATA containing data about newspaper articles on cigarettes, including the authors' last names in field 3 of each item. Then,

```
run sort cig data cig sorted 3
```

will create the file CIG SORTED with items sorted alphabetically by author's last name.

If a user wants to put all files with first name LASER into a tape (or disk) file LASER FILE, and to delete the separate files, he might proceed as follows:

```
run listof (off) laser direct laser * (lnam)
run putout laser direct laser file
run deldir laser direct
```

If he later wanted to know just which files were in the file LASER FILE, he could:

```
run p laser direct
```

and to get the separate files back, he could:

```
run pullin laser direct laser file
```

These examples illustrate the uniformity which is available to the occasional user of CTSS, when he is using programs related to data-manipulation. PUTOUT, DELDIR, REDUCE and PULLIN can be reached only through the RUN system; SORT and LISTOF are independent saved files kept in TIP common files; and EDA and P are CTSS commands. However, the user need not know or remember these facts. He can reference the programs he needs quite efficiently using the single vehicle of RUN.

Current Status

The RUN commands presently available are these:

ADDON, APPND, ASCCHK, ASEMBL, BLIP, BSSEDT, CARDMK, CHKPIK, CUT, DELDIR, DOUBLE, EXPAND, FIBCHK, FPRINT, FREQUE, INDEX, INTERM, IPATCH, MAKCOD, MAKLOD, MAKTIA, NOTE, PULLIN, PUTOUT, QEDIT, REDO, REDUCE, REJOIN, REMAKE, RENUMB, REVNAM, SAMPLM, SEQUEN, SPREAD, SQUASH, TALLYB, TALLYT, TRIPLE, TSTOCK, UPSORT.

The TIP commands presently available include:

CALC, COMPIL, CVFILE, DCALC, DISPIC, DISPLY, EDIT, ENTCNT, FORMAT, FREI, LABEL, LISTOF, MERGE, RNAME, SETFIB, SHARE, SIZE, SORT, TAP, TIP, VERIFY.

Restriction

Whenever a name conflict exists between, e.g., a public command and a RUN command, then only one program can be referenced through the RUN system. For this purpose, RUN commands take precedence over TIP commands which take precedence over private commands which take precedence over public commands which take precedence over CTSS commands. An up-to-date listing of the RUN commands is kept in the file RUN SYSTEM.

(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 "disk-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 FIEXPM 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 disk-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 disk 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.

(END)

Identification

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

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

(END)

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.

(END

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.)

(END)

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'.

(END)

Identification

Absolute program patching
SPATCH

Purpose

Programs loaded with LDABS, NCLOAD, or VICAD 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.

(END)

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. AI BI 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.

(END)

Identification

STRACE - A trace debugging routine
B. L. Wolman

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 _before_ every Nth call. If N is zero, the STOP condition will be removed.

HALT N - Go to dormant _after_ every Nth call. Execution will be interrupted _after_ the specified subroutine has been executed and _before_ it has returned to the program making the call. This request should _not_ 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.

(END)

Identification

DEBUG - Symbolic debugging aid for CTSS.
Lewis Morton, M4959 4710. Room 14S-330, X5692.

Purpose

DEBUG is an extension of FAPDBG, described in the CTSS Programmer's Guide, section AH.8.01. DEBUG may be used with any compiler or assembler generated code which is loadable by the standard CTSS loaders (see section AH.7). DEBUG acts as an execution monitor by allowing register examination and modification, and conditional execution of program sections. Core locations may be refered to by their symbolic names, if a FAP style symbol table is available on the disk. Interaction with DEBUG may be from the console or a disk file.

Usage

For a general discussion of core-B transfer commands and debugging tools, see section AH.8.00 of the CTSS Programmer's Guide. At any time the user is at command level with the loader in core, or after DEBUG has been explicitly loaded, the user enters the monitor by giving the command-

```
DEBUG -FILE-
```

if a line-numbered file of the name "FILE" DEBUG exists, it will be used as the source of requests. If not, or when this file is exhausted, requests will be read from the console. If not already loaded, the loader will read DEBUG into core from a system library. DEBUG is exactly (15000)8 words long.

All requests are single letters followed by arguments, separated by blanks. Requests may be concatenated on a line by using the equal sign or apostrophe between them. If a request fails for any reason, other requests on the same line, with the exception of ".", "G" and "P", will still be executed.

Manipulation of the Symbol Table

DEBUG maintains an internal table for user defined symbols. Currently there is space for 800 symbols. Each symbol must be six or fewer character, at least one character of which must not be an octal number. The symbol table is created from a FAP style SYMTB file by the "T" and "L" requests, described below. There is available a conversion program to create FAP style files from those produced by MAD. Notice that as of July, 1968, FAP puts into the SYMTB file indication of whether the symbol is absolute, relocatable or common. This information is maintained in DEBUG. However, SYMTB files produced before this date will appear to DEBUG to contain only relocatable symbols.

The "L" request locates a subroutine origin and entry point. Its usage is

```
L ENTRY
```

where "ENTRY" is a subroutine name, as found in the movie table. The octal origin and entry point of this routine will be typed on the console.

The "T" request reads in a symbol table and relocates all relocatable symbols using the last origin found by an "L" request. Its usage is-

```
T -NAME1-
```

"NAME1" SYMTB is read from the disk. Symbols are added to those already in the table, and any duplicates will be redefined. If "NAME1" is omitted, the symbol table is reset to contain only DEBUG's predefined symbols.

The "W" request is a concatenation of "L" and "T". Its format is-

```
W ENTRY -NAME1-
```

and is equivalent to "L ENTRY'T NAME1". If "NAME1" is omitted, it is assumed to be the same as "ENTRY".

The "E" request defines a single symbol. Its usage is-

```
E EXPRESSION SYMBOL
```

where "EXPRESSION" is a sequence of constants or defined locations separated by plus or minus signs. A defined location is an octal number or symbol, possibly followed by a comma and a second number or symbol. If the comma is present, the symbol following is interpreted to be a number between zero and seven, and the saved contents of the appropriate index register subtracted from the value of the first symbol. The resulting value will be assigned as the value of "SYMBOL".

The predefined symbols in DEBUG are mostly locations where current active registers are stored. These symbols are listed in appendix 1, and may not be redefined. In addition, for relocatable mode (see below), the origins of all subprograms may be defined in the symbol table as the value of the subprogram's name.

There are three other special symbols: "*", "." and "**". "*" is the last location referenced by the user in any DEBUG request. "." is equivalent to the last symbol typed to DEBUG by the user. "**" is equal to zero.

Register Examination and Modification

Core locations and active registers may be examined in several modes. Every register is printed out with the location of the word being dumped in the left margin followed by the contents of the word. The address may be printed in any of four modes, while the contents may be printed in any of twelve modes.

The four basic modes are "R", "N", "S" and "U". In all of these modes, FAP operation mnemonics are used, followed by the address, tag and decrement fields of the word. Note, however, that all values are octal, even for FAP prefix mnemonics. Instructions with eighteen bit address fields will be printed with a fifteen bit address and a tag.

In "R" mode, address and decrement fields are printed in the format "ORG+reloc". "ORG" is the name of the first entry point of the subroutine in which the address is located, "reloc" is the octal distance from the load point. Fields above the initial memory bound are not relocated, they are rather printed in absolute octal notation. Entering "R" mode will add the names of all subroutines to the symbol table. However, programs with the same name as a symbol already in the table will not permanently redefine the symbol. The old definitions will be restored on entering "N", "S", or "U" mode.

In "N" mode address and decrement fields are printed in octal.

"S" mode attempts to simulate FAP assembly listings. Fields are printed in one of the following forms- "SYMBOL+offset", "absolute octal" or "*+n". Here "*" has the standard FAP meaning. Notice that this mode is derived from FAPDBG's symbolic mode, but has several additional features that make it more readable.

"U" mode is the original symbolic mode of FAPDBG. All fields are printed in the format "SYMBOL+offset". This mode, therefore, allows the user to find the nearest defined symbol to a given location.

In addition to these modes, the contents may be printed in any of eight other modes. The location field of this word will be printed in whichever of the four modes described above was last entered.

"O" mode causes printing as a signed octal number. For convenience, twelve digits will be typed in the format "N_NNNNN_N_NNNNN" with numbers greater than 377777777777 printing as negative quantities.

"H" prints the word as six bcd characters.

"I" prints as a full word integer, "J" as a fortran decrement integer.

"F" prints a floating point number.

"X", "Y" and "Z" modes interpret the word as a TIP style pointer. The pointer is printed in octal, followed by the text it points to. "X" is for six-bit pointers, "Y" for nine-bit and "Z" for twelve bit. If the pointer is longer than 84 characters, only the first 84 will be printed. If any part of the pointer is above memory bound, "***" will be printed instead of the contents of the pointer. In order to use these modes, the TIP subroutine TRITE must be loaded with DEBUG. For more information, contact the TIP programming staff.

These modes are all entered in the same manner. The format is-

```
MODE -LOC1- -LOC2- -SKIP-
```

If "MODE" is given alone, the output mode is set to this style for future print-outs. "LOC1" will be dumped, if specified, and the block between "LOC1" and "LOC2" if two arguments are given. If "LOC2" is a smaller number than "LOC1", the array will be printed backwards. If the "SKIP" argument is given, every "SKIP"th location between "LOC1" and "LOC2" will be printed. If a sequence of locations contain the same value, the word REPEAT will be typed, instead of the value. "LOC1" and "LOC2" may be any expression as defined in the "E" request.

"R" mode is automatically entered on initializing DEBUG, if possible. If the movie table is missing or damaged, "S" mode is entered. "S" mode is also entered after a "W" or "T" request.

The contents of a single location or block may be printed in the current output mode by typing just the location expressions and skip expression. Of course, the first location must not be a single letter recognized by DEBUG as a request. The contents of "*+1" may be printed by an empty request (carriage return or concatenation character).

Indirection

In some cases it is desirable to trace a chain through core. The asterisk request allows this by causing chains to be printed to any desired depth when the initial entry is printed out by one of the register examination requests. The request format is-

```
                    * DEPTH
```

where "DEPTH" is a decimal integer giving the required depth. If the number is positive, the indirection will be taken from the address and tag. If it is negative, the next location to be dumped will be taken from the decrement. If any word in the chain points to itself, the word REPEAT is typed and the chain terminated. Any word which is above memory bound will break the chain and cause "***" to be printed in the location field. "*" by itself is equivalent to "* 0", meaning that only the word requested, and no words pointed to by it, will be typed.

In "F", "H", "I", "J", "X", "Y" and "Z" modes, only the last level printed will be given in the specified format. All other levels will be printed in octal.

Register Modification

The "D" request may be used to alter a core location or active register. Its format is-

```
     D LOC OP1,ADD1,TAG1,DEC1 -OP2,ADD2,TAG2,DEC2- ...
```

"OP1...", "OP2...", etc, will be deposited in sequence starting at "LOC". "LOC" may be any expression as defined above, and the "OP"s may be any valid FAP instruction or DEBUG psuedo-op. However, all except the last deposited instruction must have at least an address field. The FAP op-code may be separated from the address field by a blank or a comma. Again, all numbers must be octal in a symbolic FAP instruction, and only type D and A instruction formats are recognized.

The DEBUG pseudo-ops are OCT, BCD, FLO, INT, DEC, INT, and JNT.

The OCT pseudo instruction accepts a signed or unsigned octal integer of absolute magnitude 777777777777 or less.

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

The DEC, INT, and JNT pseudo instructions accept signed or unsigned decimal integers. DEC and INT are equivalent, and cause assembly of full word integers with maximum value of 34359738367. JNT creates a fortran decrement integer of maximum value 131071.

The BCD instruction is followed by a single blank or comma. The next six characters (including blanks) are converted to Hollerith, and deposited. If the request terminator (carriage return, apostrophe or equal sign) appears before the sixth character, the word will be right justified and blank padded.

Depositing into DEBUG's special locations wil alter machine conditions. Changing $MEM will cause the memory bound to change at the next "P" or "G" request.

Manipulation of the Accumulator

The request formats are-

A -FAPWORD-   K -FAPWORD-

"FAPWORD" is a sequence of op-code, address field, tag field and decrement field as described in the "D" request. The "A" request places "FAPWORD" in the signed accumulator and clears the P and Q bits. It is equivalent to "D $A FAPWORD".

The "K" request places "FAPWORD" in the logical accumulator and clears the sign and Q bits. It is equivalent to "D $K FAPWORD".

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

Search Core for a Given Word

Core locations satisfying a given requirement may be found and dumped. Format is-

```
/ -LOC1- -LOC2- -.REL.- -VALUE- -MASK-
```

All locations between "LOC1" and "LOC2" will be examined to see if they contain "VALUE". The locations may be any expression, and "VALUE" may be any operation or pseudo-op recognized by DEBUG. Note that this instruction must contain at least an address field, if "MASK" is omitted. ".REL." is a MAD type comparative (.E., .L., .G., .NE., .LE., .GE.) indicating the relation the core location must bear to "VALUE". If specified, "MASK" will be anded with the core location and value before the comparison is made. "MASK" is a octal number of twelve or fewer digits.

If the "LOC2", ".REL." or "VALUE" arguments are missing, the last value used will be assumed. If "MASK" alone is missing, 777777777777 is assumed. If "/" alone is typed, the search will be started at the location following the last one found by the "/" request, using "LOC2", ".REL.", "VALUE" and "MASK" as previously set.

If a word is found meeting the requirement, it will be printed in the current mode. If one is not found, an appropriate message will be printed.

Compare Core with a Disk BSS File

The format is-

```
C ENTRY -NAME1-
```

"ENTRY" is the name of an entry point of a subprogram already loaded into 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".

The "C" request will relocate each word in the bss file and compare it to the corresponding word in core. If a discrepancy is found, DEBUG will type in the current mode the location, the word from the file, and the memory location. The phrase COMPARISON DONE will signal the end of the comparison.

Execute a Single FAP Instruction

If the user wishes to execute a single machine instruction, he may type

```
: FAPWORD
```

"FAPWORD" will be interpreted and executed. Machine conditions may be altered by this request. The instruction must cause no skips. If it is a transfer, it must return to the next instruction.

Storage Map

Typing "M" with no arguments causes a storage map to be printed. Subprograms, including origins and all entry points, are listed in the order of their loading.

Control of Program Execution

The program being debugged may be run in segments, or stopped when a given criterion is met. Five conditional breakpoints may be set. If the required condition is not met at the time execution reaches a breakpoint, DEBUG automatically restarts the program. An option is provided to cause printing of a location at that time, whether or not the condition is met (see "V" request). Format of the break request is-

```
B -N- -LOC1- -LOC2- -.REL.- -VALUE- -MASK-
```

"N" is a number between one and five, indicating the number of the breakpoint to be reset. "LOC1" is the location at which the break will be placed. When control passes to this location, a break will occur if and only if "LOC2" bears the relation ".REL." to "VALUE". The last four arguments are interpreted as they are in the "/" request, and omitting them creates an unconditional breakpoint. "LOC1" and "LOC2" may be any valid expression as defined in the "E" request, except that if the expression for "LOC2" contains an index register modification, the expression must be a single term.

Active registers may be used as the break condition. In addition to the standard registers, the special DEBUG symbol $COUNT is a location which contains the number of times the breakpoint has been passed without breaking. More than one breakpoint may be using $COUNT as the test register without conflict.

When a break is finally reached, DEBUG informs the user which break caused return to the monitor. It will also print out the number of times each breakpoint was passed without causing a break. At any break, all counts are reset to zero.

A breakpoint may be removed by typing "B N". All breakpoints will be removed by typing "B" alone. Two breakpoints may not be set at the same location.

<u>Return to the User's Program</u>

The "G" and "P" requests will trasfer control to the user's program. "P" has no arguments, and returns to the last breakpoint, interrupt or entry into DEBUG. The "G" request format is-

```
G LOC
```

and causes execution to begin at "LOC". Note that if "LOC" is at a breakpoint location, and the break condition is met, the break will be taken immediately.

If a request fails for any reason, "G" and "P" requests on the same line will be ignored.

A CTSS break level is set before control is given to the user's subroutine. Therefore, during execution, an interrupt will return control to DEBUG. The location at which the interrupt occurred will be printed in the current mode.

<u>Verify Registers at a Breakpoint</u>

Any block of locations or active registers may be dumped in the current mode whenever a breakpoint is reached, whether or not the condition for the breakpoint is met. The request format is-

```
V -N- -LOC1- -LOC2-
```

All locations between "LOC1" and "LOC2" will be dumped in the current mode whenever break "N" is reached. This is independent of the "B" request, that is, changing a breakpoint will not change the verify locations associated with that breakpoint. If "N" is an asterisk, all verify locations for all breakpoints will be set to the same.

"LOC1" and "LOC2" are expressions as defined in the "E" request, except that if they contain index register modifications, they must be a single term.

To shut off this feature for a specific breakpoint, type "V N". "V" alone, or "V *" will turn off all verification at all breakpoints.

<u>Interaction with CTSS</u>

Any CTSS command may be executed from within DEBUG. Format of the request is-

```
. COMAND -ARG1- -ARG2- .... -ARGN-
```

The current core image is saved in the file (BUG) SAVED, and the command "COMAND ARG1 ARG2 .... ARGN" executed. Command buffers are saved during the execution of the command. Return to DEBUG is indicated by the message DEBUG RESUMED.

The "Q" request may be used to return to command level. A core image is retained which may be patched, saved or restarted.

<u>DEBUG as a Subroutine</u>

An additional entry point to DEBUG is provided, called FBUG. It is provided to allow a subroutine to call DEBUG directly, but is otherwise equivalent to the command. Requests are read from the console, as usual. The "$" request, with no arguments, may be used to return control to "1,4" in the calling subroutine.

Appendix 1

DEBUG Special Symbols

NAME	CONTENTS
$MEM	The Current Memory Bound
$ILC	The ILC at the Last Entry into DEBUG
$K	The Logical Accumulator
$A	The Signed Accumulator
$MQ	The MQ
$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
$SI	The Sense Indicators
$LS	Lights and Switches (see below)
$COUNT	Count of Times this Breakpoint was Passed

These locations are stored in this order, and may therefore be dumped in block notation by the output requests.

The meaning of each octal digit in the $LS register is-

DIGIT	MEANING
1-3	Unassigned
4	Console in Twelve-bit Mode if on
5	Floating-point Trap Mode Indicator
6	Divide Check Light
7	AC Overflow Light
8	Multiple Tag Mode Indicator
9	Sense Light 1
10	Sense Light 2
11	Sense Light 3
12	Sense Light 4

If the digit is "1", the indicator or light is on.

Appendix 2

Summary of Requests

REQUEST	MEANING
A	Manipulate signed AC
B	Breakpoint request
C	Compare memory with disk BSS file
D	Deposit into core
E	Define symbol for DEBUG symbol table
F	Set output mode for floating point
G	Go to core location and start execution
H	Hollerith output mode
I	Integer output mode
J	FORTRAN decrement integer mode
K	Manipulate logical AC
L	Locate subroutine in core
M	Print storage map of core
N	"NICE" output mode (octal fields)
O	Octal output mode
P	Proceed from breakpoint or interrupt
Q	Quit and return to CTSS command level
R	Relocatable output mode
S	Symbolic output mode
T	Set up symbol table
U	"UGLY" output mode
V	Set verification criteria
W	Concatenation of "L" and "T" requests
X	TIP 6-bit pointer output mode
Y	TIP 9-bit pointer output mode
Z	TIP 12-bit pointer output mode
.	Execute a CTSS request
:	Execute a single machine instruction
$	Return to calling subroutine
*	Set indirection level
/	Search for a given word in core

Appendix 3

Sample DEBUG Session

Lines typed by the user are numbered in the left column, and commented on after the session.

```
(1)   debug
      W 1341.4
      DEBUG ENTERED.  MEMORY BOUND IS 24220.
(2)   l zot
      'ZOT' IS LOADED AT 7000, ENTRY POINT IS 7004.
(3)   t zot
      SYMBOLS LOADED.
(4)   7000 7032
      WRFLX/     TTR 22033
      CHNCOM/    TTR 24044
      CHNCOM+1/  HTR 0
      ZOT-1/     TXL 6060,6,14663
      ZOT/       LMTM 0
      ZOT+1/     LDI INDEX+2
      LOOP/      PXD 0
      CYCLE/     ADD INDEX+1
      CYCLE+1/   TIF SHIFT
      INCREM-1/  AXT 5,1
      INCREM/    ACD INDEX+1
      INCREM+1/  SXA INDEX,1
      INCREM+2/  TIF SHIFT
      SHIFT-2/   TIX INCREM,1,1
      SHIFT-1/   TRA CYCLE
      SHIFT/     PIA 0
      SHIFT+1/   ARS 1
      SHIFT+2/   AXT 0
      SHIFT+3/   TRA LOOP
      MESS-4/    TSX WRFLX,4
      MESS-3/    PZE MESS,0,1
      MESS-2/    TSX CHNCOM,4
      MESS-1/    HTR 0
      MESS/      TIX 53360,2,44645
      INDEX/     HTR 0
      INDEX+1/   HTR 1
      INDEX+2/   HTR 77
(5)   b mess
      MESS/      DONE.
(6)   r 7000==
      ZOT/       TTR WRFLX
      ZOT+1/     TTR CHNCOM+3
      ZOT+2/     HTR 0
```

```
(7)   *+4 *+10 2
      ZOT+6/     PXD 0
      ZOT+10/    TIF ZOT+17
      ZOT+12/    ACD ZOT+31
(8)   o zot+6==
      ZOT+6/     -0 75400 0 00000
      ZOT+7/      0 40000 0 07031
      ZOT+10/     0 04600 0 07017
(9)   n *
      7010/      TIF 7017
(10)  * 1'r zot+22
      ZOT+22/    TRA ZOT+6
       *ZOT+6/   PXD 0
(11)  *
(12)  p
      PROGRAM RESTARTED.
      EXECUTION.
(13)  INT. 0
      ZOT+14/    TIF *+3
(14)  d shift+2 pai,0 tnz,loop
(15)  b 1 mess-4'g zot+4
      PROGRAM STARTED.
      BREAK 1 AT ZOT+23.
(16)  o $a
      $A/           0 00000 0 00000
(17)  b 1 loop $count .e. oct,7'b 2 mess-2
(18)  v 1 $si'g zot+4
      PROGRAM STARTED.
      MONITOR 1 AT ZOT+6.
      $SI/        0 00000 0 00077
      $SI/        0 00000 0 00037
      $SI/        0 00000 0 00017
      $SI/        0 00000 0 00007
      $SI/        0 00000 0 00003
      $SI/        0 00000 0 00001
      DONE.
      BREAK 2 AT ZOT+25.
       LOC 1 AT ZOT+6 PASSED 6 TIMES.
(19)  / zot zot+40 .e. tsx,0 777700000000
      ZOT+23/     0 07400 4 07000
(20)  s'/
      MESS-2/    TSX CHNCOM,4
(21)  /
      NO WORD SATISFIES REQUIREMENT.
(22)  o $mq'$a': xca'$mq'$a
      $MQ/       -3 77777 7 77777
      $A/         0 00000 0 00000
      $MQ/        0 00000 0 00000
      $A/        -3 77777 7 77777
```

```
(23) . ttpeek

     08/10 1529.7 TIME USED =    2.0

     SHIFT          MINUTES
             ALLOTTED    USED
       1        85       24.6
       2        20        8.0
       3        10        3.2
       4        30        3.2
       5        30         .0

             STORAGE
     DEVICE    QUOTA    USED
     DISK        200      79


     DEBUG RESUMED.
(24) b 1 wrflx $count .e. oct,-1
(25) v 1 1,4
(26) * 1
(27) h
(28) g zot+4
     PROGRAM STARTED.
     MONITOR 1 AT WRFLX.
     ZOT+24/     0 00001 0 07027
      *ZOT+27/ DONE.
     DONE.
     BREAK 2 AT ZOT+25.
       LOC 1 AT WRFLX PASSED 1 TIME.
(29) q
     GOOD BYE.
     R 4.012+.416
```

Explanation of the DEBUG session.

(1) At command level, the monitor is entered with the DEBUG command.
(2) Locate the subroutine we will be using,
(3) and create the symbol table from ZOT SYMTB.
(4) Look at locations 7000 through 7032 in the current mode.
(5) Look at location MESS in Hollerith.
(6) Enter relocatable output mode and look at location 7000. The empty requests (concatenation characters) cause the next locations to be printed also.
(7) Look at every other word between *+4 and *+10. Note that * is the last location printed by the previous request, and that all numbers used in the request are octal.
(8) Set octal mode and look at ZOT+6.
(9) Go to "N" mode and look at the current location.
(10) Set indirection level to 1, and look at ZOT+22 in relocatable mode.
(11) Restore indirection level to zero.
(12) Proceed from last entry into DEBUG. In this case, since there had been no execution before DEBUG was entered, start at the beginning.
(13) We decide there is an infinite loop, and give an interrupt. DEBUG responds with the location of the interrupt.
(14) Deposit the correct instructions. Note the address field on the PAI instruction.
(15) Set a breakpoint at MESS-4 and start execution at ZOT+4. Note that we are in relocatable mode, therefore the symbol ZOT refers to the program origin, rather than the value of the location ZOT in the subroutine.
(16) Look at the accumulator.
(17) Set a conditional breakpoint at LOOP. The break will be taken at the seventh time execution reaches this point. Set a second unconditional breakpoint at MESS-2.
(18) Verify the sense indicators at breakpoint 1, and start from ZOT+4.
(19) Look for a TSX instruction between ZOT and ZOT+40. The octal mask insures that any TSX will be found.
(20) Switch to symbolic mode, and try to find another TSX.
(21) Try for one more TSX.
(22) Look at the MQ and AC. Then execute an XCA instruction, and look at them again.
(23) Execute the CTSS command TTPEEK from within DEBUG.

(24) Put a breakpoint at WRFLX. Note that $CCUNT will never reach -1, hence the break will never be taken.
(25) However, we may verify a register whenever control reaches that point. Here we wish to look at 1,4, which will be the argument to the subroutine WRFLX.
(26) Set indirection level, so that we can see what the argument points to,
(27) and go to Hollerith mode, since the argument is BCD text.
(28) Begin execution at ZOT+4.
(29) We are all finished. Exit back to CTSS command level.

(END)

Identification

Print storage map
STOMAP

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

```
     STOMAP
```

Prints the file '(MOVIE TABLE)' on the user's console.

```
     STOMAP ALPHA
```

creates a file 'ALPHA MAP' containing a numeric and an alphabetic storage map of the file '(MOVIE TABLE)'.

```
     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.

(END)

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. Corbato	(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.) The preset erase character is # and the preset kill character is @ .

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 character "c" becomes the erase character. normally, the character "#" is the erase character. (The erase character is used to delete the previously typed character or characters.)

KILL c

The character "c" becomes the kill character. Normally, the character "@" is the kill character. (The kill character is used to delete the entire line currently being typed.)

APPEND character string

The string of characters "character string" is appended to the line being pointed at.

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 <u>starts</u> <u>with</u> 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.

BREAK c

The character "c" becomes the break character, i.e. to switch from input to edit mode or from edit to input mode, type "c" followed by a carriage return. If "c" is not specified on the BREAK request, the normal mode (carriage return only) is restored.

QUIT

This request is used to terminate the editing process without making any changes to the original file, and without creating a new file. All intermediate files are deleted, and the user returns to command level.

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 prog
(INPT1 prog

where 'prog' is the user's programmer number. Following a QUIT sequence (or a CTSS system breakdown) 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 prog 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 prog contains only that part of the file <u>above</u> the pointer, and therefore contains only a partial record of the original file. The original file is never deleted until the new, edited file has been successfully written and closed.

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
A	APPEND string	none
F	FIND string	line found *
		end-of-file
Q	QUIT	Ready message

* 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.)

-ne 4

.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 xxxxxxxxxxxxxxxx

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 NOWAIT, the program stops before the first page for loading of paper. The STOP parameter will cause a stop between all pages. 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 <u>continue</u> 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 tabs on a 1050 are not set at the CTSS standard settings of 11, 21, 31, etc., the supervisor may mistime characters or insert extra carriage returns. For this reason, use of tab characters is not recommended.

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", followed by 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. Since the supervisor assumes that tab stops are at 11, 21, 31, etc., a line with too many tab characters may appear to overflow the carriage size, and the supervisor may insert extra returns.

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

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.

(END)

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 perioically - 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.

<u>Restriction</u>

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

(END)

Identification

Run off ASCII memorandum files
ROFF

Purpose

ROFF is a program used to type out memorandum files of English text, in manuscript format. Control words within the ASCII source file may be used to provide detailed control of the format of the document produced. ROFF is an adaptation of the RUNOFF command (see AH.9.01) for ASCII file input. Since ASCII files contain no line-marks and have four characters per word instead of three, a user can obtain a significant saving in file space by using EDA and ROFF instead of TYPSET and RUNOFF.

Usage

```
ROFF name1 -p1- -p2- ...
```

ROFF	will read the file "name1 ASCII" and produce output in manuscript form. The optional parameters p1, p2, etc. may appear in any order. They may be any of the following:
STOP	will cause a pause between pages so that the paper may be changed. Typing will resume when carriage return is struck.
NOWAIT	will cause the initial pause to load and position paper to be suppressed. Typing will begin immediately.
PAGE n	will begin output with the page numbered "n". If pages are not numbered, no output will result.
BALL n	will cause ROFF to assume that the console used for typed output has a typing element of type "n". ROFF assumes that all characters in the file can be typed with the typing element. CTSS characters in the file "name1 ASCII" which do not appear on the ball will print as blanks, so that they can be drawn in by hand.
PRINT	will cause ROFF to produce an ASCII character stream file "name1 RUNOFF" as output instead of typing on the console.

Control Words

The control words for ROFF are the same as those for RUNOFF, with the following exceptions:

NEED control word: ".need n"

Start a new page of output if there are not "n" lines remaining to be printed on the present page. This control word is useful for keeping tables and figures from being split across a page. (This control word must follow a control word which causes an automatic break.)

INSERT control word: ".insert file name3"

The input stream will be diverted to file NAME3 ASCII. When all input from NAME3 is exhausted, ROFF will revert to taking input from NAME1 ASCII. ".insert file" control words may not be nested. This control word causes an automatic break.

The abbreviation for ".insert file" is ".if" and the abbreviation for ".need" is ".ne".

(END)

Identification

Print files in public file directory
PINFO

Purpose

To allow users to print files (INFO files in particular) accessible through the public file directory M1416 CMFL04.

Usage

```
          PINFO NAME1 -NAME2-
```

PINFO will switch to the public files via a call to TSSFIL, open NAME1 NAME2 for reading, return to the user's file directory via a call to USRFIL, and print the file on the terminal. The file may be either line-marked or card-image. If 'NAME2' is omitted, 'INFO' is assumed.

Examples

There are short descriptions of most system commands in M1416 CMFL04 as files of second name 'INFO'. To find out how to use a command, for example 'APEND', type:

```
          PINFO APEND
```

If the appropriate INFO file exists, it will be printed. To obtain a summary of changes to INFO files, print the file 'NEWS INFO':

```
          PINFO NEWS
```

To obtain a list of all available INFO files, type:

```
          LISTF (SYS) * INFO
```

To print the message of the day (printed also by LOGIN), type:

```
          PINFO MESSAG TODAY
```

(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 RUNCOM (or 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 RUNCOM, 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  ATTACH
```

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.

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.

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.

Some examples of macro-commands:

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

```
    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 1C30, 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 RUNCOM MACRO N
        * N IS THE NUMBER OF THE LINE CONTAINING 'EXECUTION'
```

```
EDIT MACRO RUNCOM
FILE MACRO RUNCOM
$ EXECUTION
```

(END)

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
```

(END)

Identification

User subsystem control
SUBSYS

Purpose

To allow a user program ('subsystem') to have reliable control over the manner in which a console user may interact with CTSS.

Discussion

During recent years there has evolved on CTSS a class of specialized interactive programs best thought of as subsystems under CTSS which, with few exceptions, are intended to be usable by persons having little or no experience with CTSS as a general-purpose computing facility. Examples of such programs are the various information retrieval systems, teaching aids, and the command interface program '.' .

In many cases it is desirable (or necessary) that such a subsystem be the only access a user has to CTSS, i.e. that he can't QUIT and then go do something else. This means that the time-sharing supervisor must provide a means whereby the subsystem may regain control in situations which would ordinarily allow the user to issue commands directly to the system (e.g. program termination, pushing the QUIT button, error conditions, etc.).

The following considerations led to the current implementation of the subsystem facility:

1. Provide for subsystem-restricted users, i.e. users whose subsystems are initialized at LOGIN and who may not access CTSS except as allowed by the subsystem. (This was the principal reason for the implementation of the subsystem feature, and is intended to provide better control over a user's activities than the old 'disk-restricted user' facility.)

2. Allow a subsystem to load and execute programs or execute CTSS commands (e.g. EDL) by using command chaining, and recover control when execution terminates. In the case of restricted users, such programs must <u>not</u> be able to modify the supervisor subsystem status words; however, the subsystem need not be so restricted.

3. Allow a subsystem to intercept a new command typed while at command level. Since the QUIT button is the only real safety valve available when a program has entered

an endless loop, and even well-coded subsystems are not immune to program bugs, it was decided that the best way to give control to the subsystem after a quit is to wait for a new command to be issued from the terminal, and then load the subsystem instead of executing the command typed. By making the command typed at the terminal available in the current command buffer, it is possible for the subsystem to execute the command via CHNCOM or NEXCOM.

4. Allow the subsystem to specify conditions under which it should be loaded (program call to DEAD or DORMNT, call to CHNCOM with no command chaining, intercepting new command, error condition), and allow the subsystem to determine which of these conditions caused it to be loaded.

5. Allow the subsystem to specify that a SAVED file of a possible dormant core image is to be automatically produced before loading the subsystem.

Method

Six special TIA's have been provided which allow a program to specify and examine the conditions under which a subsystem is to be loaded. These may only be used by a subsystem or by a subsystem-privileged (i.e. not restricted) user. Refer to section AG.8.05 for details.

Associated with each user, there are three status words maintained in the supervisor containing his standard options, his subsystem name, and his subsystem condition code mask and last condition code.

User standard options occupy a half-word (18 bits), and are interpreted as follows:

```
+-----------------+-----------------+
|                 | user  options   |
+-----------------+-----------------+
```

1	Search user UFD first for command
2	Search user or system files (not both) for command
4	RESETF if command resets dormant prog.
10	User subsystem trap enabled
20	Inhibit quit signals for user
40	Current user program is subsystem
100	Automatic save before loading subsystem
200	User is 'dialable'

The two low order bits are taken together to specify four modes of command file searching:

0 Search system files then user files (normal mode)
1 Search user files then system files
2 Search system files only
3 Search user files only

The following disk-loaded commands are always taken from the system files (provided that the user is allowed to use them):

```
LOGIN
LOGOUT
OTCLOG (user may not issue)
DAEMON (incremental dumper only)
DSDUMP (incremental dumper only)
DSLOAD (incremental dumper only)
FIBMON (FIB user and FIBMON only)
OPTION (subsystem-privileged user only)
```

The RESETF bit specifies that if there is a dormant core image left from the last command, and the command currently being processed does not preserve this core image (i.e. not SAVE, MYSAVE, START, RSTART, SUBSYS, ENDLOG, RESETF, or any B-core transfer command: USE, DEBUG, PM, etc.), any active files will be reset by a call to RESETF instead of being closed normally. This provides compatibility with previous versions of CTSS.

The subsystem trap enable bit causes all program calls going to DEAD or DORMNT (including errors) to simulate a call to NEXCOM for the command SUBSYS, provided that the call does not come from the user's subsystem (option bit 40 off), and causes all new commands issued from the terminal to pass through the subsystem processor (with the exception of exempt commands).

The quit-inhibit bit causes all quit signals to be ignored for the user. Program status will be unaffected if the user attempts to quit and buffered output will not be reset. N.B. The only way to stop a non-quittable program that has gone into a loop is to force an automatic logout by hanging up the data-phone (or turning off power to the terminal). Use this feature at your own risk!

The subsystem execution bit, if on at command load time, causes a new core image being loaded to have subsystem privileges if the user does not have the subsystem privilege himself. Program calls going to dead or dormant status will execute normally if this bit is on, regardless of the setting of the subsystem trap bit.

The subsystem save bit if set causes the subsystem processor to simulate a 'MYSAVE progn T' before it loads the subsystem.

The dial-permit bit allows remote terminals to attach to the user via the DIAL command. See section AH.1.05 for details.

The user's subsystem name is interpreted as a six-character command name, which may be any system command or a user disk-loaded command (SAVED file).

```
+-----+-----+-----+-----+-----+-----+
|          subsystem  name          |
+-----+-----+-----+-----+-----+-----+
```

The subsystem condition code mask is a half-word quantity split into two 9-bit fields. The high order 9 bits are examined by the subsystem processor if the user has a core image left; the low order 9 bits are examined if there is currently no core image. Within each 9-bit field, the bits are interpreted as follows:

```
 1  Trap new command
 2  Trap direct program call ('DEAD', 'DORMNT')
 4  Trap CHNCOM if end of chain or no chain set up
10  Trap error condition (file system, PMV, etc.)
```

The subsystem condition code occupies the high order 18 bits of the subsystem condition mask word. The low order 9 bits of these 18 indicate which of the possible subsystem trap conditions occurred to cause the subsystem processor to be entered (zero if the SUBSYS command was issued directly by the user or his program). The following 8 bits specify an error code if the subsystem condition code was 10 ('error'), in order to indicate the type of error that occurred. This is not yet implemented, and the error code will be returned as 0. The high order (sign) bit is on if there was a dormant core image left.

```
++-------+--------+--------+--------+
|| error |  code  | condition  mask |
++-------+--------+--------+--------+
```

When the CTSS supervisor determines that a user's subsystem is to be called in (option bit 10 is on and user is about to go dead or dormant or is at command level and types a command), it initiates the special command 'SUBSYS' for the user, in the same way that ENDLOG is set up for an automatic logout, placing the user in the queues in waiting command status. The SUBSYS command may also be issued by the user directly, from the terminal or via CHNCOM; this is considered to satisfy any condition mask.

When SUBSYS is entered, the following occurs:

1. If the user's current core image is not that of his subsystem (option bit 40 off) and the automatic save

option is specified (option bit 100 on), SUBSYS simulates a 'MYSAVE progn T'.

2. If the SUBSYS command was initiated by the user, either by typing SUBSYS at the terminal, or within a command chain, the subsystem is unconditionally loaded, whether or not the current core image belongs to the subsystem. The condition code is set to 0. (This is the only way to reenter the subsystem if a protection mode violation or file system error with no error return specified occurs during execution of the subsystem.)

3. If the user's current core image is that of his subsystem, and the SUBSYS command was initiated by the supervisor, the user's program (subsystem) is restarted by simulating the 'START' command. If a command line was entered and trapped, it will be available in the current command buffer. This is the case when a user, while executing in the subsystem, quits and tries to issue a command, or when the subsystem itself has called DORMNT and the user issues a new command. (This occurs only if bit 1 of the subsystem mask is on.)

4. If the user's current core image is not that of his subsystem and the SUBSYS command was initiated by the supervisor, SUBSYS compares the current subsystem condition code with the condition code mask. If any condition is satisfied, the user's subsystem is loaded, option bit 40 is set (this bit on while a command is being loaded instructs the command processor to set the restriction code bit in the user's current restriction code that allows his subsystem to call the TIA's which modify options and subsystem status), and the program is started. If none of the conditions are satisfied or there is no subsystem set up, SUBSYS exits via DORMNT unless a command was trapped, in which case SUBSYS will return to the command processor to execute the command.

Exceptions

The following commands when issued from the terminal are not subject to being trapped by the subsystem facility, but will execute normally:

```
SAVE
MYSAVE
START
RSTART
OPTION (restricted user may not use)
SUBSYS
```

Restriction

Beware of attempting to use 'SUBSYS' as a subsystem. Results will be peculiar.

(END)

Identification

Set user options
OPTION

Purpose

Allow a user to set his standard option and subsystem status words maintained in the supervisor to modify system characteristics to suit his own needs.

Discussion

With systems numbered 8A0 and higher, the command processor has been completely redesigned to provide a more general user interface.

Associated with each user, there are three status words maintained in the supervisor containing his standard options, his subsystem name, and his subsystem condition code mask and last condition code.

User standard options occupy a half-word (18 bits), and are interpreted as follows:

```
+-----------------+-----------------+
|                 |  user  options  |
+-----------------+-----------------+
```

1	Search user UFD first for command
2	Search user or system files (not both) for command
4	RESETF if command resets dormant prog.
10	User subsystem trap enabled
20	Inhibit quit signals for user
40	Current user program is subsystem
100	Automatic save before loading subsystem
200	User is 'dialable'

The two low order bits are taken together to specify four modes of command file searching:

0	Search system files then user files (normal mode)
1	Search user files then system files
2	Search system files only
3	Search user files only

The following disk-loaded commands are always taken from the system files (provided that the user is allowed to use them):

LOGIN
LOGOUT
OTOLOG (user may not issue)
DAEMON (incremental dumper only)
DSDUMP (incremental dumper only)
DSLOAD (incremental dumper only)
FIBMON (FIB user and FIBMON only)
OPTION (subsystem-privileged user only)

The RESETF bit specifies that if there is a dormant core image left from the last command, and the command currently being processed does not preserve this core image (i.e. not SAVE, MYSAVE, START, RSTART, SUBSYS, ENDLOG, RESETF, or any B-core transfer command: USE, DEBUG, FM, etc.), any active files will be reset by a call to RESETF instead of being closed normally. This provides compatibility with previous versions of CTSS.

The subsystem trap enable bit causes all program calls going to DEAD or DORMNT (including errors) to simulate a call to NEXCOM for the command SUBSYS, provided that the call does not come from the user's subsystem (option bit 40 off), and causes all new commands issued from the terminal to pass through the subsystem processor (with the exception of exempt commands).

The quit-inhibit bit causes all quit signals to be ignored for the user. Program status will be unaffected if the user attempts to quit and buffered output will not be reset. N.B. The only way to stop a non-quittable program that has gone into a loop is to force an automatic logout by hanging up the data-phone (or turning off power to the terminal). Use this feature at your own risk!

The subsystem execution bit, if on at command load time, causes a new core image being loaded to have subsystem privileges if the user does not have the subsystem privilege himself. Program calls going to dead or dormant status will execute normally if this bit is on, regardless of the setting of the subsystem trap bit.

The subsystem save bit if set causes the subsystem processor to simulate a 'MYSAVE progn T' before it loads the subsystem.

The dial-permit bit allows remote terminals to attach to the user via the DIAL command. See section AH.1.05 for details.

The user's subsystem name is interpreted as a six-character command name, which may be any system command or a user disk-loaded command (SAVED file).

```
+-----+-----+-----+-----+-----+-----+
|          subsystem  name          |
+-----+-----+-----+-----+-----+-----+
```

The subsystem condition code mask is a half-word quantity split into two 9-bit fields. The high order 9 bits are examined by the subsystem processor if the user has a core image left; the low order 9 bits are examined if there is currently no core image. Within each 9-bit field, the bits are interpreted as follows:

```
 1  Trap new command
 2  Trap direct program call ('DEAD', 'DORMNT')
 4  Trap CHNCOM if end of chain or no chain set up
10  Trap error condition (file system, PMV, etc.)
```

The subsystem condition code occupies the high order 18 bits of the subsystem condition mask word. The low order 9 bits of these 18 indicate which of the possible subsystem trap conditions occurred to cause the subsystem processor to be entered (zero if the SUBSYS command was issued directly by the user or his program). The following 8 bits specify an error code if the subsystem condition code was 10 ('error'), in order to indicate the type of error that occurred. This is not yet implemented, and the error code will be returned as 0. The high order (sign) bit is on if there was a dormant core image left.

```
++-------+--------+--------+--------+
|| error |  code  | condition  mask |
++-------+--------+--------+--------+
```

Usage

To turn on option bits:

```
OPTION SET nnnnnn
```

To turn off option bits:

```
OPTION RESET nnnnnn
```

To specify all options:

```
OPTION LOAD nnnnnn
```

(where 'nnnnnn' is the octal representation of the option half-word)

To specify subsystem status:

```
OPTION SETSYS command nnnnnn
```

(where 'command' is the six-character or fewer name of the command which is the desired subsystem and 'nnnnnn' is the subsystem condition mask. Option bit 10 is turned on by this operation.)

To obtain a summary of options and subsystem mask bits:

```
     OPTION HELP
```

To print out current options and subsystem information:

```
     OPTION PRINT
```

To find out how to use the command:

```
     OPTION
```

Restriction

The OPTION command will be 'NOT FOUND.' in the system files for a subsystem-restricted user. For a non-restricted user, OPTION is always loaded from the system files, regardless of the settings of the command loading options.

The OPTION command will not be trapped by the subsystem mechanism for a normal user (i.e. one who is not subsystem-restricted).

(END)

Identification

Print I/O error diagnostics PRNTER

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 = <u>binary</u> 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

CTSS usage
WHO

Purpose

To determine who is using CTSS at any given time.

Usage

```
     WHO
 or  WHO N
 or  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.

(END)

Identification

Listing control
SPACE, EJECT

Purpose

Insert blank lines between commands in a chain, or eject a page.

Usage

```
          EJECT -A1 A2 ... An-
          SPACE n -A1 A2 ... An-
```

'SPACE n' causes 'n' carriage returns to be typed. If 'n' is omitted, '1' is assumed. EJECT is equivalent to 'SPACE 66'. If A1 ... An appear, they will be passed back to the supervisor as a command via SETCLS and NEXCOM.

(END)

Identification

Print command line
ECHO

Purpose

To print command line before executing the command.

Usage

```
          ECHO -A1 A2 ... An-
```

The command name 'ECHO' followed by any arguments will be typed on the terminal. If A1 ... An appear, they will be passed back to the supervisor as a command via SETCLS and NEXCOM.

(END)

Identification

Octal/decimal conversion
OCT, DEC

Purpose

Allow octal-to-decimal or decimal-to-octal conversion from command level.

Usage

```
          OCT/DEC nnnnnn -A1 A2 ... An-
```

OCT will convert the decimal number nnnnnn to octal and print the result; DEC will convert the octal number nnnnnn to decimal and print the result. If A1 ... An appear, they will be passed back to the supervisor as a command via SETCLS and NEXCOM.

(END)

Identification

Turn printer on or off
PON, POFF

Purpose

To engage or disengage the printing element on the terminal.

Usage

```
          PON/POFF -A1 A2 ... An-
```

The printer is turned on or off as specified. Note that output is always printed in the case of a model 35 teletype; only input printing is affected by POFF. If A1 ... An appear, they will be passed back to the supervisor as a command via SETCLS and NEXCOM.

(END)

Identification

Change ribbon shift
RED, BLACK

Purpose

Change ribbon color setting on 1050 or 2741 terminal.

Usage

```
          RED/BLACK -A1 A2 ... An-
```

The ribbon shift if any on the terminal is set to the desired setting. Note that input typed on a 2741 after setting ribbon shift to red will print black, whereas on a 1050 the ribbon setting is permanent. If A1 ... An appear, they will be passed back to the supervisor as a command via SETCLS and NEXCOM.

(END)

Identification

Command chain checkpoint
YES

Purpose

Print a response on the terminal, generally between commands in a chain or at the end of a chain.

Usage

```
          YES -A1 A2 ... An-
```

An asterisk ('*') is typed on the terminal. If A1 ... An appear, they will be passed back to the supervisor as a command via SETCLS and NEXCOM.

(END)

Identification

Pause between commands
WAIT

Purpose

Pause in execution for specified time

Usage

```
          WAIT n -A1 A2 ... An-
```

The command will sleep for 'n' seconds, then exit. If A1 ... An appear, they will be passed back to the supervisor as a command via SETCLS and NEXCOM.

(END)

Identification

Print date and time
TIME

Purpose

Print date and time on terminal

Usage

```
          TIME -A1 A2 ... An-
```

The current date and time will be printed in the form

```
          MM/DD/YY HH:MM:SS
```

If A1 ... An appear, they will be passed back to the supervisor as a command via SETCLS and NEXCOM.

(END)

Identification

Explain file error code
PERROR

Purpose

To allow the user to obtain a diagnostic message explaining a file system error at a time other than immediately after the error occurs.

Usage

```
          PERROR ENTRY ERCODE -IOCODE-
```

A diagnostic message of the same form as generated by the PRNTER command (see AH.11.01) will result, for error 'ERCODE' in call to file system entry 'ENTRY'. If 'ERCODE' is the error code for I/O error, the diagnostic will be for I/O error 'IOCODE' in call to 'ENTRY'.

Example

```
          PERROR WRFILE 8 5
```

This command will result in the message:

```
          ERROR 5: ILLEGAL I/O REQUEST FOR DEVICE --WRFILE
```

(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)

<u>Identification</u>

MADIO
Simplified i/o package for MAD programs
Reference: MAC-M-270
Peter J. Denning

<u>Purpose</u>

MADIO is designed for use in the CTSS environment as a compact input-output 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 M1416 CMFL04. 'READ' or 'PRINT' may be extracted from 'MADIO BSS' by means of the 'EXTBSS' command.

<u>FORMAT FREE READING</u>

<u>Program Name</u>: READ.

<u>Transfer Vector</u>: RDFLX, WRFLX, WRFLXA.

<u>Use</u>: 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 per memory location, the contents of the final location 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 <u>memory locations</u> to be filled (called for by the list A,B,...,L). Hence the n arguments of the list could be entered on as many as n console lines if desired.

<u>Restrictions</u>:

(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 arguments typed is less 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. Suppose the list A,B,...,L calls for M memory locations to be filled, the READ program is operating in OFF-mode, and the items typed would fill N locations. The remaining (M-N) locations will be filled with zeros. In the OFF-mode, only one line of type is accepted.

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 MAD-type 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.

<u>Special Features</u>

(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 <u>insert</u> ')' into the output string, the '=' sign being ignored. To insert '=)' into the string, use '==)'. <u>Example</u>. 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. <u>Example</u>. 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 is normal in CTSS.

Restrictions.

(1) No E-formats are allowed.

(2) Integers must be less than 2.P.36 in magnitude.

(3) If X.Y is a floating point number then the integer XY must be less than 2.P.36.

(4) All C-formats are interpreted as C6.

(5) 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)).

(6) 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. Suppose the specified field width is w, and the number to be printed contains x digits, where x is larger than w. The rightmost (w-1) digits of the number are printed, and an asterisk (*) is inserted at the left of the field. For example, suppose the format F5.2 is given and the number 1234.5 is to be printed. Since the number is too wide for the field, the following is actually printed:

```
*34.5
```

(END)

Identification

Public file commands

This section of the CTSS Programmer's Guide documents user-supplied programs which are analagous to system commands. They are maintained as SAVED files, accessible through the public file directory M1416 CMFL04.

The public commands may be used by commands of the form:

```
          DO P NAME ARG1 ARG2 ...
```

Alternatively, one may link to the SAVED file,

```
          LINK NAME SAVED M1416 CMFL04
```

and thereafter initiate the program with 'RESUME' or 'DO 0', or, with option bit '2' set, use the link as a 'user command' (see AH.10.04 about the OPTION command):

```
          NAME ARG1 ARG2 ...
```

(END)

Identification

GPM - A General Purpose Macrogenerator
Christopher Strachey

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

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

Compress or expand BCD files
SQZBCD SAVED, PADBCD SAVED
B. Wolman

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

Text display on ESL console
DISPLY SAVED
H. Murray

Purpose

To display text on the ESL console (second floor, Building 39).

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)

Identification

List links in a file directory
LSTLNK SAVED
C. Garman

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) T0999 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

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

Print list of files on file system tape.
TAPLF SAVED
N. I. Morris

Purpose

TAPLF will print a listing of all tape files in a user's directory. For each file, it will print the file name, the number of records, the logical unit of the tape file, and the physical file number. This program is quite useful to those who regularly use file system tapes.

Usage

```
R  TAPLF  -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 public file (M1416 CMFL04).
USE	consists of three parameters which specify a file to be treated as 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 is the name of the file to be used. This tactic is used to list files in another user's directory by reading through the link FNAM1 FNAM2 to that user's U.F.D. (FILE).
NAME1 NAME2	specify the file name(s) to be listed. If both are omitted, all tape files are listed. If either parameter is '*', all files of given primary or secondary name are listed. If NAME2 is omitted, '*' is assumed.

References

Section	Name
AG.5.05	Use of tapes in foreground
AH.3.06	Tape-handling commands

(END)

Identification

Convert 6-bit to 12-bit files
6TO12 SAVED
J. H. Saltzer

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.

(END)

Identification

Combine line-marked files
APPEND SAVED
C. Garman

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 (mode 'P'){ otherwise the file will be added to by using '.APEND'.

The files 'NAME3i NAME4i'|The files NAME3i NAME4i 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 NAME4e are '*' (single wasterisk), 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.

(END)

Identification

Enciphering, deciphering of files
GARBLE: ENCIPH SAVED, DECIPH SAVED
R. Fenichel, D. Edwards

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 intc 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.

(END)

Identification

Compare two files
CMPARE

Purpose

To perform a word-by-word comparison of two files.

Usage

```
CMPARE NAME1 NAME2 NAME3 -NAME4-
```

If NAME4 is omitted, it is assumed the same as NAME2. If the two files are identical, the message 'FILES ARE IDENTICAL.' is printed. If the two files are not identical, a line is printed for each word that is not the same, giving RELLOC within the files, the contents of the word in NAME1 NAME2, and the contents in NAME3 NAME4. If the end of file is reached in one of the files, 'EOF' is printed in the column for that file, and the remainder of the other file is listed.

(END)

Identification

Convert 12-bit file to 6-bit
12TO6

Purpose

To convert a 12-bit file (e.g. $$$FIB OUTPUT) to 6-bit form for offline processing via the disk editor.

Usage

```
            12TO6 NAME1 -NAME2- -NAME3- -NAME4-
```

File NAME1 NAME2 will undergo the 12-bit to 6-bit mapping described for typewriter input in section AC.2.01; the resulting file is NAME3 NAME4. A single space is prefixed to each line of the output file to serve as carriage control for the disk editor.

If NAME2 is omitted, it is assumed '(MEMO)'; if NAME3 is omitted, it is assumed the same as NAME1; if NAME4 is omitted, it is assumed 'BCD'.

(END)

Identification

Search a saved file
SRCH SAVED
N. I. Morris

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.

(END)

Identification

Generate dump of SAVED file for off-line printing
DUMPER
D. Widrig

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)'- -'(TEMP)'- -'(PRNT)'-
```

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. If the optional argument '(TEMP)' is used, NAME1 (DUMP) will be created in temporary mode. If the optional argument '(PRNT)' is used, a summary of the saved file's machine conditions will be printed on the user's console, in addition to the other activities.

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 .5 - .8 seconds per record of SAVED file.

(END)

Identification

Check success of RUNCOM
QUES SAVED
C. Garman

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.

(END)

Identification

Parameter identification within RUNCOM
RUNPRT SAVED
C. Garman

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 <u>not</u> 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
```

(END)

Identification

Slave consoles
SLAVE SAVED
N. I. Morris

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.

(END)

Identification

Sample programs

Purpose

This section of the CTSS Programmer's Guide will be devoted to sample programs illustrating techniques for using some of the more obscure facilities provided by the system.

Source files for these programs will be found in the CTSS manual directory (M1416 3212), and will be named by the section number in which they appear, e.g. the sample subsystem described in section AK.8.01 will be available as AK801 MAD, just as the manual section itself is AK801 ASCII.

Disclaimer

Although the programs described herein have in general been tested and found to work, no guarantees are made concerning their correctness, and time-back credit requests involving attempts to use them cannot be honored.

(END)

Identification

Usage of subsystem facility: sample program

Purpose

Allow a user to edit, compile, print and load programs, and to logout; prohibit any other non-exempt commands. Give the user the option to restart his program if a saved file was created.

Program

```
          R SAMPLE PROGRAM TO USE SUBSYSTEM FACILITY
          R
           DIMENSION COMMND(20), WHO(1)
           NORMAL MODE IS INTEGER
          R
          R
START      GETSYS. (NAME, MASK)
           CODE = (MASK .RS. 18) .A. 777K
          R
           WHENEVER CODE .E. 001K
             GCLS. (COMMND, 0)
             THROUGH LOOKUP, FOR I = 0, 1, COMTBL(I) .E. FENCE
               WHENEVER COMMND(0) .E. COMTBL(I)
                 RSOPT. (40K)
                 NCOM. (COMMND(0), COMMND(1))
               END OF CONDITIONAL
LOOKUP       CONTINUE
             PRMESS. (BZ57. (COMMND(0)), $ IS NOT A LEGAL COMMAND.$)
             TRANSFER TO CKSAVE
           OR WHENEVER CODE .E. 002K
             TRANSFER TO CKSAVE
           OR WHENEVER CODE .E. 004K
             TRANSFER TO WAIT
           OR WHENEVER CODE .E. 010K
             PRMESA. ($ERROR $, 406057575757K)
             TRANSFER TO CKSAVE
           OR WHENEVER CODE .NE. 0
             PRMESS. ($UNKNOWN SUBSYSTEM TRAP CODE.$)
           END OF CONDITIONAL
          R
WAIT       SETSYS. (NAME, MASK .A. 017017K)
           DORMNT.
           TRANSFER TO START
          R
CKSAVE     WHENEVER MASK .L. 0
             WHOAMI. (WHO(1)...2)
             FSTATE. (WHO, $ SAVED$, SIZE...1, WAIT)
          WHENEVER SIZE .E. 0, TRANSFER TO WAIT
          PRMESA. ($DO YOU WANT TO RESTART YOUR PROGRAM... $)
          RDFLXA. (REPLY...1)
          WHENEVER REPLY .RS. 30 .E. $00000Y$
            RSOPT. (40K)
            NCOM. ($CONTIN$, WHO)
          END OF CONDITIONAL
        END OF CONDITIONAL
        TRANSFER TO WAIT
       R
       R
        VECTOR VALUES COMTBL = $    EDL$, $ PRINT$, $   MAD$,
       1  $LOADGO$, $LOGOUT$, 777777777777K
        VECTOR VALUES FENCE = 777777777777K
       R
        END OF PROGRAM
```

(END)