D. J. Rees

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. Corbató
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. Corbató
Cambridge, Massachusetts
May 1963

TABLE OF CONTENTS (7/30/66)

(* denotes new file system section)

AG.9	Debugging		
.01	ERROR	Trace of subroutine calls	
AG.10	Conversion		
.01	BCDEC,BCOCT	BCD or Spread-Octal to Binary	
.02	DEFBC,DELBC,DERBC	Binary to BCD	
.03	OCABC,OCDBC,OCLBC, OCRBC	Binary to Spread Octal	
.04	BZEL,ZEL,LJUST, RJUST	Justification & padding	
.05	(IOH),(RTN),(FIL) IOHSIZ,STQUO	General purpose conversion	
.06	PAKR, PAKL, UNPAKR, UNPAKL	Pack and unpack, single characters to full words.	
.07	FINT, MINT	Fortran integers to/from full word integers.	
.08	COM, ORA, ANA	Complement, OR, and AND	9/24/65
.09	DECODE, ENCODE	Formatted Conversion	3/30/66
AG.11	Subroutine linkage processors		
.01	COLT	Variable-length calling sequence	
.02	GNAM	Type calling program, FILNAM	
.03	MOVE1,MOVE2,MOVE3	Move argument list	
.04	SETFMT,SETNAM	Format and File Name	
AG.12	Timer		
.01	GETIME, GETTM, GTDYTM	Current time and date	5/31/66
.02	TIMER,JOBTM,RSCLCK, STOPCL,KILLTR,TIMLFT, RSTRTN	Alarm clock, stop watch	
.03	CLOCON,CLOCOF	Simulation of Interval timer.	9/24/65
AG.13	Miscellaneous		
.00	List of miscellaneous library subroutines		2/14/66
.01	(FPT)	Floating-point overflow and underflow.	8/30/65

AH. Commands

AH.1 Logging

AH.2 Languages & Subsystems

AH.3 File creation & editing

(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 off 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.

Revised: 5/9/66

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 by 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; "FIB-wait", a very specialized status, used only by the FIB monitor where there is no FIB job waiting to be run; "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 time of the first clock trap following 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 is kept in a disk file (of the second name 'TSSDC') in the same format as a SAVED file, in a system file directory. 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 "TSSDC." 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 is 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.)

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 8 levels (0-7). 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. If a program changes its size, its new level is computed immediately. If the new level is different from the old, a new maximum time is also computed and becomes effective retroactively.

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 question mark (?) to cause the system to ignore ("kill") the entire line thus far, or by typing one or more quotation marks (") 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 Ext. 5948 (MAC machine) or Ext. 4121 (Computation Center machine). 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 quotation mark (") 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 question mark (?). 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.

The MAD Command for Compilation:

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.

The EDL Command for Editing:
(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 only 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 first 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". Blanks within the strings are significant: "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 consecutive 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" (from the Edit mode). 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.

The LOADGO Command for Execution:

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.

Program Logic "Debugging":

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
 PARTY LINE BUSY, STANDBY LINE HAS BEEN ASSIGNED
 M1416 3711 LOGGED IN   1/10/66   1315.6 from 20000K
 CTSS BEING USED IS     DMN2C5
```

```
LAST LOGOUT WAS 1/6/66 1756.0
R 6.783+750


edl simple mad
W 1316.4
 FILE SIMPLE   MAD NOT FOUND.
Input
normal mode is integer
floating point a
  Tab   print comment$numbers,pleeuhz""""""ase$
  Tab   read data
  Tab   a=sqrt(b+c)
  Tab   d=bc
end of?   Tab     end of program

Edit
t
1 mode
NORMAL MODE IS INTEGER
r  Tab  normal mode is integer
n
r  Tab  floating point a,d
1 a
        PRINT COMMENT$NUMBERS,PLEASE$
1 a=
        A=SQRT(B+C)

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


Edit
file
R 5.833+4.250


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.
 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
c qbqb*q                                        (10)
        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*

        A = 2.707999E 26,              D = 14.000000                (11)
  EXIT CALLED. PM MAY BE TAKEN.
R 6.166.1.050

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

```
c qqtag2q
TAG2   EXECUTE EXIT.
i
Input
tag  Tab print comment$negative argument$
  Tab  transfer to tag2

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.
NUMBERS,PLEASE
b=7.,c=2.*

     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.*
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
 LOGUT NOT FOUND.
READY.


logout m1416""""""""
W 1528.4
 M1416  3711 LOGGED OUT  1/10 /66   1536.3 FROM 20000K
 TOTAL TIME USED=    00.7 MIN.
-
```

14 15 16 17 18 19

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

(COMBI NFILE) AH.6.01:

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

(INPUT FILE) AH.9.01,
(INPT1 FILE) 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 LOGOUT SAVED file is resumed). When invoked, the editing commands will announce the presence of one of the intermediate files (if one <u>is</u> 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.

LOGOUT SAVED AH.1.02:

At any time an automatic LOGOUT may be initiated by the system. If the user is not in dead status, this SAVED file will be created. The file may be resumed at a later time.

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 MAILBOX.

(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.

Files With Fixed Second Names

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. ('BCD' is also the required secondary name for RUNCOM files; see AH.10.01.)

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 SAVED AH.3.03:

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

NAME1 SQZBSS AJ.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

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 TYPSET and RUNOFF where each section is a separate file of the name MSXYY (MEMO). Note the deletion of periods within the file name.

Users may request copies of complete manuals or any section thereof from the Project MAC or Computation Center's documentation rooms. Or, at the user's convenience copies may be RUNOFF on the user's 1050 Selectric console.

The table of contents will be maintained in two forms.

1) TABLE (MEMO) which may be RUNOFF 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 (MEMO) which may be RUNOFF 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 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., -PŻE 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.

Revised: 8/30/65

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, 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 SFTNAM 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 in MAD but currently is acceptable only to the new I/O system routines.

Any FAP subprograms written after March 1 may be coded to accept this new format, but their write-ups should say so specifically.

In FAP, a PZE prefix maybe 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.

Major Revision: 5/23/66

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)

Revised: 5/31/66

Identification
Timely, Temporary Tips
5/31/66

1. The Public "command" SEQ SAVED (AJ.3.03) has been withdrawn. The write-up should be removed. *

2. A section (AA.2.01) which tabulates system-fixed file names has been contributed by Miss Ann Berry of the Computation Center Staff, whom we thank. Despite being listed as a subsection of the "Primer", this section may be of interest to old users as well as new. *

3. Retrieval procedures under the DAEMON are discussed in (new) Section AE.2.01. *

4. The FIB, DELFIB, and PRFIB commands (AH.1.03) are available in systems numbered MAC5A1, or CTR5A1, and above. Such systems also will contain a change which will cause LOGOUT SAVED files to be created in Temporary mode during automatic logout. N.B. If you intend to RESUME such a file, it's probably a good idea to change the mode to Permanent first--in case you need to RESUME it more than once. Remember also that Temporary mode files are deleted by normal LOGOUT, so that if you intend to RESUME yesterday's LOGOUT SAVED tomorrow you should change it to Permanent mode today. *

5. The file directories containting the manual are currently T254 3212 at MAC and M1416 3212 at the Computation Center.

6. Future additions and corrections to the manual should be referred to M.A. Padlipsky, c/o 512 Tech Square. *

7. In order to facilitate identification and handling of revision sets, two new typographical conventions have been established: a) In place of the word "PAGE", the month and year of the revision set will appear in the heading line on each page. b) The symbol "(END)" will appear in the right-hand margin of the last page of each section. Also, a summary of the contents of each set of revisions will be given in a Filing Instructions Sheet, which will be included with the set, or may be RUNOFF as INSTR (MEMO) from the Manual's directories.

8. Note that there now exists a "Time-Sharing Primer", Section AA.2. Further tutorial sections are anticipated on such topics as linking, debugging, and miscellaneous tricks ("Pages from the Consultant's Notebook"). Your Kindly Editor earnestly solicits comments and contributions in this area.

(END)

Identification
Equipment Configuration

The primary terminals of CTSS are, at present, modified Model 35 Teletypes and IBM 1050 Selectric teletypewriters (adaptations of the "golfball" office typewriter). These are located mostly, but not exclusively, within the M.I.T. campus. Each of these terminals can dial, through the M.I.T. private branch exchange PBX, either the IBM 7094 installation of Project MAC or the similar installation at the M.I.T. Computation Center. The supervisory programs of the two computer installations may, independently, accept or reject a call, depending on the identity of the caller. Access to the MAC System can also be gained from any station of the Telex or TWX' telegraph networks. Some tests and demonstrations have been conducted from European locations, and experiments are in progress in collaboration with a number of universities to provide further experience with long-distance operation of the system.

Although Teletypes and other typewriter-like terminals are adequate for many purposes, some applications demand a much more flexible form of graphical communication. The MAC System includes for this purpose the initial model of 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, as well as some special-purpose digital equipment that performs the light-pen tracking, and simplifies the task of the computer in maintaining the display, and in performing common operations such as translating and rotating the display. The two oscilloscopes 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 such as knobs, push buttons, toggle switches, and a typewriter. The meaning of a signal from one of these input devices is entirely under program control. The whole display system communicates with the IBM 7094 of the MAC installation through the direct-data channel, and the display data are stored in the central memory of the 7094. Because of cable length requirements, the display must be located in a room adjacent to the computer installation. Remote operation would require the addition of a memory and some processing capacity for local maintenance of the display.

A separate, very flexible display terminal is provided by a DEC PDP-6 computer which can communicate from a remote location with the MAC computer installation through a 1200-bit-per-second telephone connection.

All of these terminals can operate simultaneously with the MAC computer installation by time sharing its central

processor. In order to insure prompt response, the number of terminals active at a given time is limited by the supervisor program to 30. This number has already grown to 30 from 10, and is expected to grow further in the next few months. However, maximization of this number is not a primary objective at this time.

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.

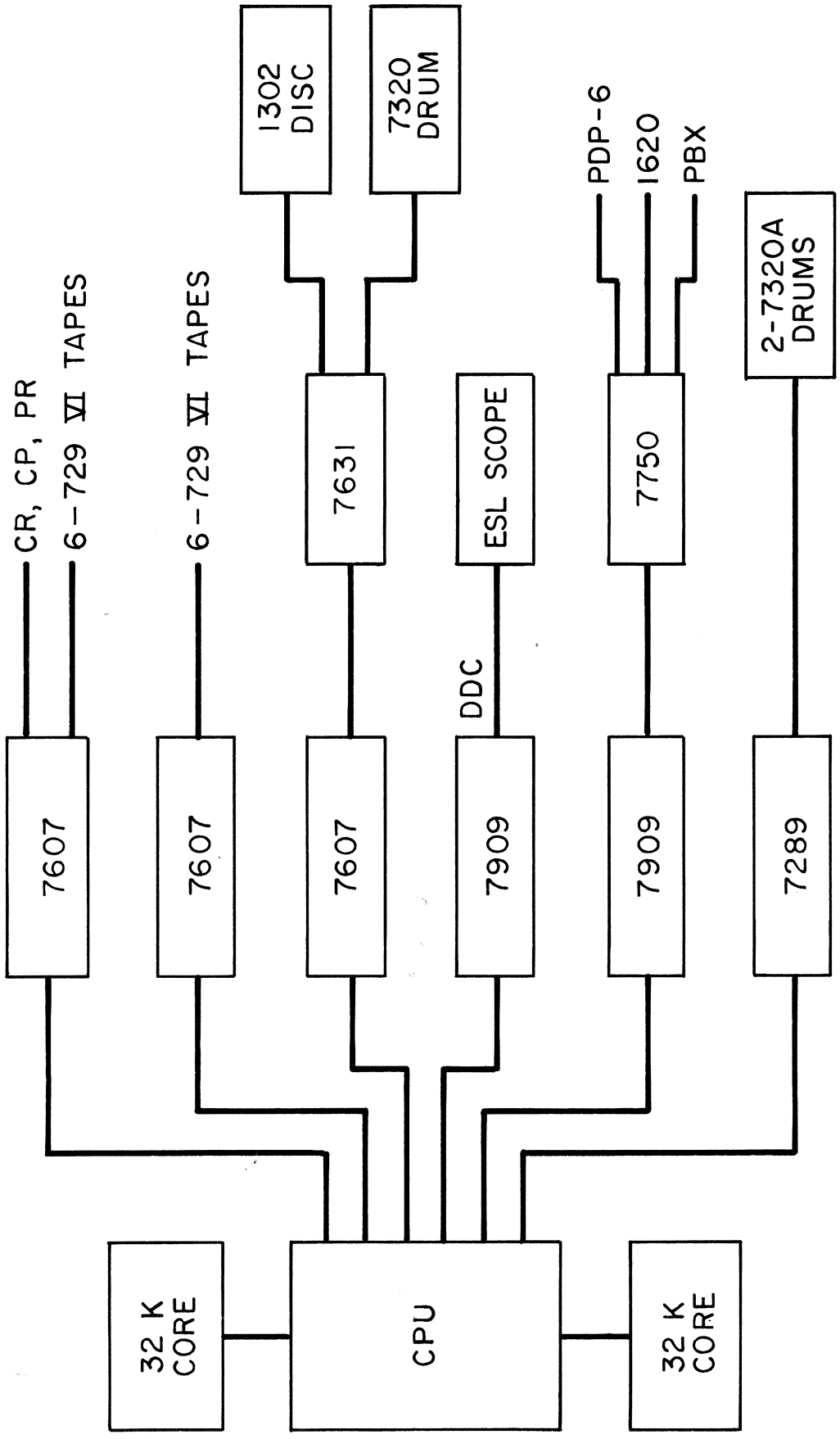
The central processor is equipped with six data channels, two of which are used as interfaces to conventional peripheral equipment such as magnetic tapes, printers, card readers, and card punches. A third data channel provides direct-data connection to terminals that require high-rate transfer of data,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

The fourth data channel provides communication with a disk (IBM 1302) and a low speed drum (IBM 7320). The theoretical storage capacity of the disk is 38 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.

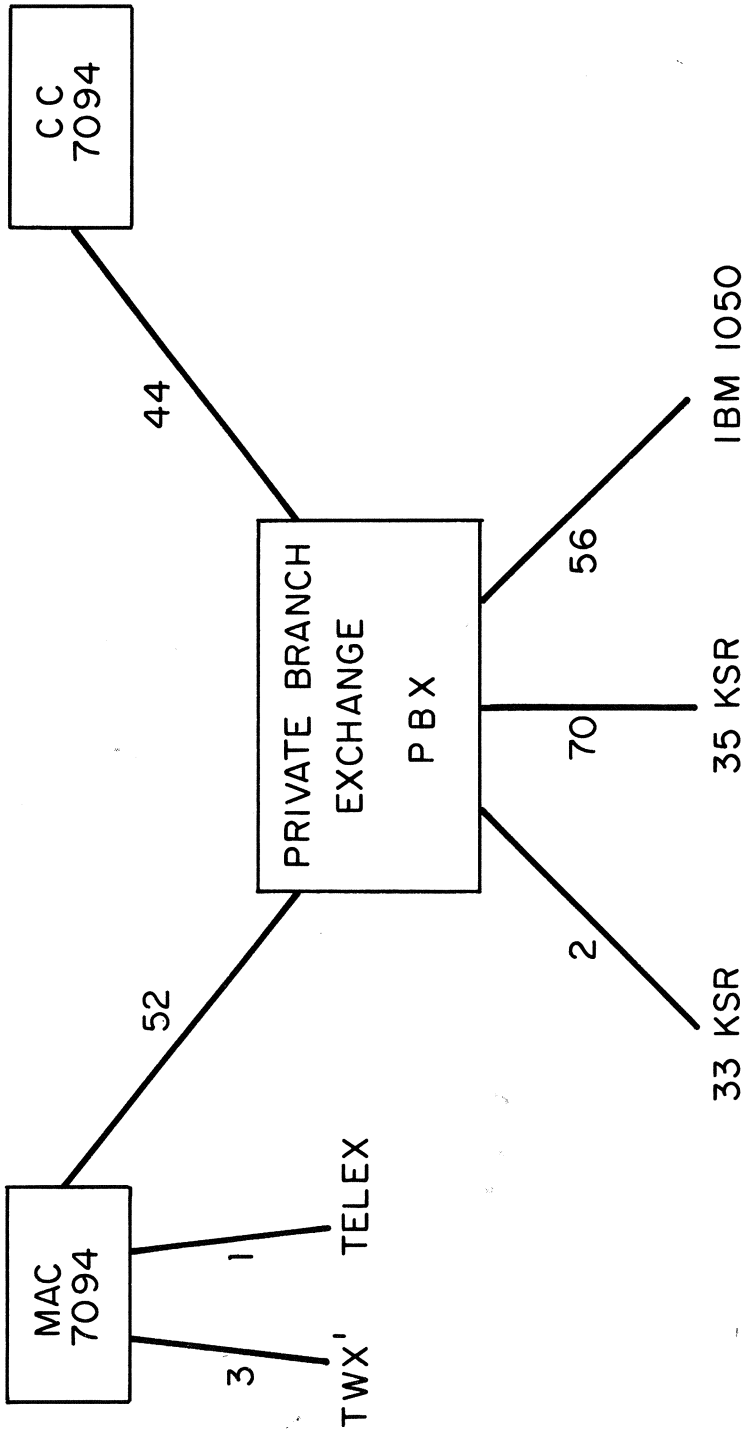
The fifth data channel provides communication with two high speed drums (IBM 7320A). The capacity of a 7320A is the same as that of the 7320 but the transmission time for 32K words is one-quarter second.

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, on the present MAC System, three 1200-bit-per-second terminals are installed, one of which provides the communication channel to the PDP-6 computer. 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. The design intent was and still is to provide sufficient input-output buffer space in the main memory to eliminate unnecessary core-to-core transfers; the present mode of operation is a makeshift made necessary by equipment limitations.



7094 SYSTEM DIAGRAM



SCHEMATIC OF COMMUNICATION NETWORK

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.

Revised: 3/30/66

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 in parentheses are discarded.
2. All other characters except double quote and question mark are truncated to six bits by discarding the left six bits.
3. Double quote is the "erase" character: the previous character is discarded.
4. Question mark is the "kill" character: the entire line is discarded.

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 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
C.R.	- Carriage return
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
CRLF	- 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	O	P
0050	Q	R	F.F.	$	*	C.R.		null
0060	blank	/	S	T	U	V	W	X
0070	Y	Z	tab	,	(			
0100	(|)	(↓)	(↑)	(;)	(#)	(¬)	(@)	(L.F.)
0110	(HLF)	(HLR)	(%)	(bell)	(!)	(H.U)	(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)	(D.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 code 0137 (double quote) is the erase character in 6-bit mode.
3. Character code 0156 (question mark) is the kill character 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 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	(¢)	(±)	(¤)			(¬)		
0110	ig	ig	ig	ig	ig	ig		
0120								
0130					ig		ig	
0140								
0150			(,)	(!)	(prefix)	(.)		
0160	ig	ig						
0170			ig	ig	ig		ig	ig

1. This table assumes a "standard correspondence" ball (938). On machines with an "old-style correspondence" ball (927), code 0100 will print a degree sign, and code 0102 will print a cent sign.
2. Code 0152 will type an upper case comma on output; an upper case comma on input will be given code 0073.
3. Code 0155 will type an upper case period on output; an upper case period on input will be given code 0033.
4. Code 0017 (Interrupt) can be generated by one push of the "reset line" button.
5. Code 0057 (Quit) can be generated by two pushes of the "reset line" button.

Old Style, MIT-modified Model 35 Teletype

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			(\)			(%)		ig
0110	ig	ig	ig					
0120		A	B	C	D	E	F	G
0130	H	I	ig	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			

1. Some machines may type different graphics for any of all of the following:

Character Code	Normal Graphic	Alternate Graphic
0102	\	
0104	#	≈
0105	%	⊃
0106	@	∨
0120	&	∧

3. Code 0017 (Interrupt) may be generated by one push of the "Break" button.
4. Code 0057 (Quit) may be generated by two pushes of the "Break" button.

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			

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. Code 0017 (Interrupt) may be generated by one push of the "Break" button.
5. Code 0057 (Quit) may be generated by two pushes of 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.

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" is used to stop the current program by placing it in dormant 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 "erase or kill line" 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 his program is running or not, 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 the command level (i.e., the user was in the dead or dormant status), he is placed in a waiting command status. If the user's program was in an input-wait status, it is returned to the working status so that it may resume by reading the input message. If the user's program was already in the 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 and returns to level 0. This manual QUIT signal allows the user to change his mind, correct mistakes, etc.

In addition to the basic level 0, the user may extend the number of interrupt levels, thus allowing externally controlled branching to subsystems. This is accomplished by the program's issuing calls to the supervisor, which on each entry drops the level by one (to a maximum of 3) and specifies a return entry. Whenever a console interrupt signal is received by the supervisor, the level is raised by one and control is returned (by means of a push down list) to the entry previously assigned.

The interrupt signal is sent from a 1050 selectric after a single push of the "RESET LINE" button. The QUIT signal is sent by a double push of the "RESET LINE" button. On the model 35 teletype, the corresponding button is labeled "BREAK".

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 character " (quote) is 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 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) is interpreted as a kill or delete message signal. The entire message back to the previous break character is erased.

Additionally, while in the 6-bit mode, the : (colon) is interpreted by most edit programs as a one space backspace. This is used in formatted input which uses tabs, such as MAD input. For example, A tab : R puts the R in position 11 as required for remarks cards in MAD.

Revised: 3/30/66

Identification

Data phone extensions & console ID.

Purpose

Consoles may be connected with the MAC or CC machines via telephone lines, through a special exchange or switchboard at M.I.T. Because of the difference in transmission rates, certain lines must be used by 1050 consoles and different lines must be used by teletypes. The exchange has the ability of searching several lines to find one which is not busy; therefore, dialing one digit will cause the exchange to find a line, if possible.

MAC computer:

35KSR: dial 7
1050: dial 8
33KSR and 35ASR: dial 403 or 404

Computation Center computer:

35KSR: dial 9
1050: dial 0

Consoles have specific, but not unique, identifications. These identifications are to be used in programming with the interconsole communication subroutines. The identification is a single BCD word, consisting, from left to right, of a type code (2 for 1050's, 3 for TELEX, 4 for TWX', 5 for inktronic, 33KSR and 35ASR, 6 for teletypes), 2 to 4 BCD zeroes, and 1 to 3 BCD characters of identification. *

Each dataphone used with an IBM 1050 console has a unique extension number which may be used for voice transmission between consoles rather than data transmission.

Locations and telephone extensions for teletypes:

(Asterisks will not be used in this list to indicate changes)

Locations	Ext. No.	I.D. Code	Machine No.
266 Hudson Rd., Sudbury	511	1	1
Terminal Rm A, Fifth Floor	512	2	2
1-109, 115 (jacked)	513	3	3
Harvard, 33 Oxford St., Cambridge	514	4	4
11 Holdenwood Rd., Concord	515	5	5
33-214	516	6	6
E19-319	517	7	7
Terminal Rm A, Fifth Floor	518	8	8
52-454C, 560, 561, 562, 076A (jacked)	519	9	9
Brandeis U, Sydeman Computer Center	520	10	10
5 Esquire Cr, Esq. Manor, Peabody	521	11	11
Terminal Rm A, Fifth Floor	522	12	12
54-610	523	13	13
7-344	524	14	14
Boston College, Gasson 21	525	15	15
Terminal Rm A, Fifth Floor	526	16	16
Lincoln Lab, B-232, A-166 (jacked)	417	17	17
Lincoln Lab, B-276, A-166 (jacked)	418	18	18
3-482	419	19	19
1-139	420	20	20
52-076A	421	21	21
26-352	422	22	22
20B-120	423	23	23

26-352	424	24	24
T.S. 922	425	25	25
32-108A	426	26	26
52-252B, 076A (jacked)	427	27	27
Terminal RmA, Fifth Floor	428	28	28
52-061, 062, 080, 083, 076A (jacked)	429	29	29
295 Harvard St., Apt 811, Cambridge	430	30	30
T.S. 809	431	31	31
T.S. Conference Room, 8th floor, 802, (jacked)	432	32	32
T.S. 901	433	33	33
Terminal Rm A, Fifth Floor	434	34	34
T.S. 526	435	35	35
T.S. 806	436	36	36
Terminal Fm A, Fifth Floor	437	37	37
1-139	438	38	38
33 Dawes Rd., Lexington	439	39	39
T.S. 908	440	40	40
35 Alcott St., Acton	405	41	41
T.S. 908	406	42	42
3-355	407	43	43
T.S. 922	408	44	44
7 Summer St., Watertown	409	45	45
Southeastern Mass. Tech Inst. Fall River	410	46	46
2-366	411	47	47
1-115 (jacked)	412	48	48

4 Hilda Rd., Bedford	413	49	49
Atlantic Union College, Math. Dept. South Lancaster	414	50	50
T.S. 922	415	51	51
T.S. 901	416	52	52
Terminal Rm A, Fifth Floor	221	53	53
Terminal Rm A, Fifth Floor	222	54	54
T.S. 913 (35ASR)	532	55	55
E10-009	216	56	56
52-521, 076A (jacked)	217	57	57
10-423	218	58	58
2-034	223	59	59
University of Rhode Island, Tyler Hall, Computer Lab	224	95	95
20C-112	225	96	96
20C-110	226	97	97
3-315	227	98	98
12-182	228	99	99
TS-901 (37KSR)	229	168	168
TS-515 (37KSR)	230	169	169
13-2143	527	140	140
Tufts, Comp Center, Medford	528	141	141
N51-310D	529	142	142
W91-223	530	143	143
Mass. Gen. Hosp. Lab. Computer Science, Thayer 2	531	144	144
54-1425	469	145	145
20A-121, 123 (jacked)	470	146	146

24-015	471	147	147
16-233	472	148	148
101 Wright St.,Arlington	473	149	149
56-711	474	150	150
Terminal Rm A, Fifth Floor	475	151	151
Mass.Gen.Hosp.,Warren Bldg Rm 603	476	152	152
32-102 (jacked)	477	153	153
T.S. 806	478	154	154
Harvard, School Public Health B-10, 665 Huntington Ave., Boston	479	155	155
5-332	480	156	156
131 Bedford St, Burlington	482	157	157
NW14-5124	481	158	158
34 Otis St., Newton	401	159	159

Locations and telephone extensions for 1050's:

1-139	541	U	63
26-142	542	C	60
26-265	587	8	61
26-139	544	J	62
5-225	545	1	64
24-521	546	2	65
26-269 (with attachments)	577	H	66
T.S. 404 (with attachments)	548	I	67
8-203	554	P	68
26-153	555	3	69

53-356A	556	4	70
26-265	557	5	71
26-265	558	6	72
13-5157	559	7	73
26-265	549	8	74
61 Shattuck Rd., Watertown	550	9	75
1-150	551	V	76
26-144	552	C	77
26-169	553	I	78
26-263	570	.(period)	79
26-261	573	.(period)	80
2-094	578	X	81
26-259	575	.(period)	82
Harvard Comp Lab, Room 207 33 Oxford St., Cambridge	571	Y	83
E19-325	547	Z	84
26-257	576	.(period)	85
26-150	572	.(period)	86
26-265	568	T	87
26-157	581	8	88
26-157	582	8	89
26-157	583	8	90
26-157	584	8	91
26-157	585	8	92
26-157	586	8	93
26-142	543	G	94
1-139	441	U	100

Lot 6, Deacon Haynes Rd., Concord	442	W	101
T.S. 803	443	A	102
T.S. 814	444	B	103
T.S. 817	445	D	104
T.S. 834	510	E	105
14S-322 (with attachments)	447	F	106
Terminal Rm A, Fifth Floor (with attachments)	448	F	107
T.S. 926	449	K	108
T.S. 823	450	L	109
3-482	451	M	110
T.S. 801A (jacked)	452	N	111
24-220T	453	O	112
14S-308	454	Q	113
13-5153	455	R	114
53-366A	456	&	115
Terminal Rm A, Fifth Floor	457	&	116
Terminal Rm A, Fifth Floor	458	&	117
Terminal Rm A, Fifth Floor	459	&	118
Terminal Rm A, Fifth Floor	460	&	119
24-109	461	$	120
E53-384	462	S	121
Lot 4, Deacon Haynes Rd., Concord	463	,(comma)	122
20B-226	464	,(comma)	123
32-108A	465	0(zero)	124
T.S. 519A	466	,(comma)	125
T.S. 922, 908 (jacked)	467	,(comma)	126

T.S. 922, 908 (jacked)	467	,(comma)	126
T.S. 922	468	,(comma)	127
T.S. 510	562	.(period)	128
T.S. 508	564	.(period)	129
T.S. 507	565	.(period)	130
192 Woodward St., Waban	560	.(period)	131
T.S. 835 (jacked)	574	E	132
T.S. 511, 513 (jacked)	561	.(period)	133
T.S. 509	563	.(period)	134
T.S. 506	566	.(period)	135
T.S. 505	567	.(period)	136
T.S. 518	579	.(period)	137
26-145	569	.(period)	138
20B-148	580	H	139
T.S. 928	588	.(period)	160
E19-789	589	P	161
T.S. 801, 836, 838 (jacked)	446	E	162
T.S. 516	540	.(period)	163
T.S. 831	489	4	164
T.S. 515	200	.(period)	165
T.S. 504	201	.(period)	166

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 April 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 *.

Revised: 5/31/66

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. mechanisms in preference to other files.

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.

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 ILOCK 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".

1301/1302 Disk and 7320 Drum Strategy

The file directory entry for a 1301, 1302 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:

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 "DISCUT (FILE)". The track usage file for the 7320 drum will be referred to as "DRUMUT (FILE)". Whenever possible, successive tracks of a file will be assigned to separate channels. This procedure will allow all available disk/drum channels to operate on a file in parallel.

1301/1302 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 P 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 )
```

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

Revised: 11/19/65

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.

Major Revision: 3/30/66

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 (currently Michael Padlipsky, Room 926, 545 Technology Square). 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 CMFL04) 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)

Major Revision: 7/30/66

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 four time-accounting files:

UACCNT	TIMACC
TIMUSD	TIMACC
PRTYGP	TIMACC
GRPUNI	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 for MAC only

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

UACCNT TIMACC

2 card images:

word1 word2 ...

1) NAME PROG PARTY STBY LINMUL UNIT RCODE GROUP PASS

2) DRUM DISK TAPE T1 T2 T3 T4 T5

GROUP 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
UNIT is unit group
RCODE has leading zeroes
GROUP is allocation group

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. LINMUL always "1"
e. items right-justified except:

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

f. GROUP always "0"
g. RCODE has leading zeroes
h. unused fields must be blank

A special entry type is distinguished, the kludge user entry. 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, 11, 3, 4, 6, 7, 8, 9, 10, 12, ...

intermediate: problem number

numeric order

minor: programmer number

numeric order, common files last.

Format of UACCNT TIMACC for Computation Center only

Although the CTSS programs which use UACCNT TIMACC treat this file as a format-free file, the

Center's version is fixed-field both for ease of scanning and to simplify the programming of other jobs which use this file. In addition, the Center has added four additional fields for its own administrative use. These fields are group name (by which the file is ordered) and two dates indicating when the entry was made, when it should be removed, and a request number.

There are two types of entries, problem number entries and programmer entries. Using MAD these entries can be read with the following statements.

The Problem Number Entry

```
DREAD. (FILNAM, FMT1, ASTER, PRBNO, DF1, CF1, TF1,
DF2, CF2, TF2, DF3, CF3, TF3, DF4, CF4, TF4, RQSTNO, GRPNM,
DATIN, DATOUT)

V's FMT1 = $(A6, 1X, A5, 1x, A4, 2X, 2(1X, A4), 7X,
3(1X,A4)/6(1X, A4), 11X, A4, 3(1X,A6))$
```

where the symbols have the following meanings:

ASTER	$*$, required first entry in Problem line
PRBNO	problem number
DFn	drum quota for COMFILn
CFn	disk quota for COMFILn
TFn	tape quota for COMFILn
RQSTNO	group request number
GRPNAM	group name
DATIN	date entry made
DATOUT	date entry to be removed

The Programmer Entry

```
DREAD. (FILNAM, FMT2, PRGNAM, PRGNO, PRTY, STNBY,
LMULT, UNIT, RESTR, GRPNO, PASS, RQDR, RQDS, RQTP,
TA1, TA2, TA3, TA4, TA5, RQSTNO, GRPNAM, DATIN,
DATOUT)

V's FMT2 = $(A6, 1X, A5, 1X, A4, 1X, A1, 2(1X, A4), 1X, A6,
4X, A1, 11X, A6/9(1X, A4), 3(1X, A6))$
```

where the symbols not previously defined are:

PRGNAM	programmer name
PRGNO	programmer number
PRTY	assigned party line
STNBY	"S" if assigned to standby
LMULT	line multiplier (not operative)
UNIT	restricted unit group if any
RESTR	restriction or privileges code

GRPNO	accounting group number (not used)
PASS	password
RQDR	drum record quota
RQDS	disk record quota
RQTP	tape record quota
TAn	time quota on shift n

More importantly, both types of entries can be read as shown below:

```
DREAD. (FILNAM, FMT, IN(1) ...IN(23))

V's FMT = $(A6, 1X, A5, 1X, A4, 1X, A1, 2(1X, A4), 1X, A6,
3(1X, A4), 1X, A6/9(1X, A4), 3(1X, A6))$
```

or

```
VREAD. (FILNAM, IN(1) ...IN(11))
VREAD. (FILNAM, IN(12) ...IN(22))
```

TIMUSD TIMACC

The file TIMUSD contains the following information for each user:

TUn	Time used for each shift.
DATE1, TIME1	Date and time of last logout.
DATE2, TIME2	Date and time from which used time has accumulated.
TL	Total time logged in since DATE2, TIME2.

LOGIN reads the TIMUSD file each time someone logs in. If the user is not found in this file, LOGIN adds him to the file with zero values for times used. LOGOUT updates the time used information and re-writes that portion of the file containing information on the user logging out.

Format of TIMUSD TIMACC

2 card images:

```
     word1   word2  ...

1)   PROB    PROG   NAME
2)   DAT1    TIM1   DAT2   TIM2  T1  T2  T3  T4  T5  CTU  KLT
```

DAT1	Last Logout	MMDDYY
TIM1		HHMM.T

DAT2 TIM2	First Login
T1-5	Time used, shifts 1-5, in seconds
CTU	Console time used, in minutes
KLT	Kludge time used, in seconds

PRTYGP TIMACC

The file PRTYGP contains the party group information and the maximum number of users. LOGIN reads this file to determine whether a user will be logged in as primary or as standby. (See also Section AH.1.01).

Format of PRTYGP TIMACC

free field card images:

	word1	word2
1)	MXUSRS	
2)...n)	GRP	MXGRP

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

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

(END)

Revised: 5/30/66

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 and CARRY 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 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 not be input. Decks will not be input "through links". Any errors discovered within the deck will cause the entire deck not to be input. Do not input a private or protected file to a common file directory since the author number will be set to zero.

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 10, 15, 30, (+15) ..., 120. Also, if the secondary name is ALGOL, LISP, or LSPOUT, a blank character will be inserted in front of each line to insure single spacing. However, an ALGOL file will be XPANDed to 132 characters by interpreting tabs for columns 11, 16, (+5)..., 66.

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

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

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

XX='CARRY' The file NAME1 NAME2 will be carried to the other computer and will be loaded onto the disk during the next load or update. It will be loaded in permanent mode, with the same name (NAME1 NAME2), within the same problem-programmer file directory. If a different problem-programmer specification is desired for the receiving file directory, OP may be PROB1 PROG1, (i.e., the desired problem/ programmer numbers). If a different file name is desired, OP may be PROB1a PROG1a NAME1a NAME2a, where PROB1a PROG1a must be the problem-programmer numbers for the receiving file directory and NAME1a NAME2a is the name to be given to the input file.

The file NAME1 NAME2 on this machine is in no way changed. If a PRIVATE or PROTECTED file * of the same name already exists on the receiving machine, the carried file will not be input. Input "through a link" will not

occur.

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', 'DPUDEL', 'BPUDEL', '7PUDEL':

The file(s) will be PRINTed, DPUNCHed, BPUNCHed, or 7PUNCHed, 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 - x6006

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.

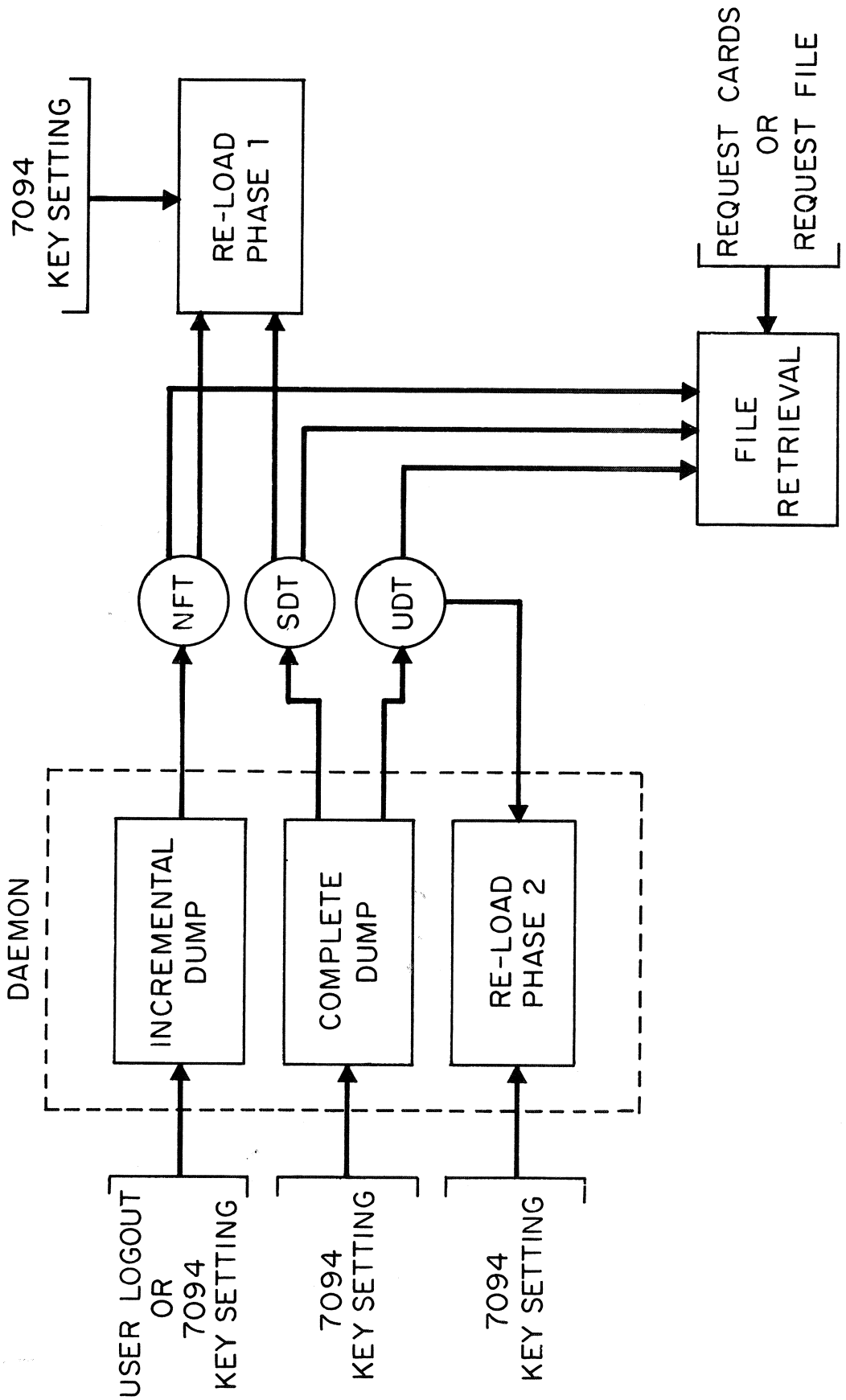


DIAGRAM OF DISK FILE BACK-UP AND RETRIEVAL

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 (SDR, UDT); and the New File Tape (NFT), or the incremental dump tape. CDTs are created periodically by the DAEMON at the behest 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. CDTs are saved for at least two complete dump dumping periods. 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 saved indefinitely.

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 at the beginning of the listing of files on the reels. Hence, users requesting retrievals from CDTs must first find their own directories and then "back-up" to the nearest reel-break message in the listings to determine which set of tapes is desired. 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. (If so many files are retrieved that a user's record allotment is exceeded, retrieval will continue, and temporary mode will be used for succeeding files.)

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 Areas. 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)

Major Revision: 1/3/66

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-A6, A10, B1-B6, B10
c. A7, the chronolog clock

Referencing of other units will cause a diagnostic.

(N.B. Tape units A6 and B6 will no longer be available to background systems when the DAEMON is implemented.)

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 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.

A program stop should only be used when the background system cannot possibly proceed, since the operator cannot restart by depressing the start button.

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
                                      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 as in the 7090 interval timer.

Revised: 5/9/66

Identification

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

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'

TSX  WRFLXA,4      optional(TIA  =HWRFLXA)
PZE  LOC,,'n'

TSX  WRFLX,4       optional(TIA  =HWRFLX)
PZE  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. *

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
                STO  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.

(END)

Identification

Set the console character mode switch.
SETFUL, SETBCD

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.

Upon return from either entry, the AC is zero if the previous setting was 6-bit mode, non-zero if the previous setting was 12-bit mode.

Both library entries may be called by MAD or Fortran programs.

Restrictions

Any completed lines waiting in the input buffer are reset (lost) by either of these calls.

Characters making up an incomplete line (that is, before the carriage return has been typed) are not reset by calls to SETBCD or SETFUL.

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.

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.

Identification

Slave remote consoles
ATTCON,RELEAS,SNDLIN,SNDLNA,REDLIN,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 10 slave. Similarly, an attached console which imitates the home console's output is an 00 slave. An attached console whose typed input appears as input at the home console is known as an 11 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.

Immediately after it has been dialed in, a console is in an unattachable state.

A quit signal issued from an unattachable console causes that console to become attachable.

A quit signal issued from an attachable console causes that console to become unattachable.

A quit signal issued from an attached console which is not an 11 slave releases that console and leaves it unattachable.

Usage

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.

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 (11,00,10) 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.

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.

Revised: 8/30/65

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.

Revised: 8/30/65

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 ONLINE FORMAT FMT, LIST              TSX  .COMNT,4
           FAP:  TSX  .PRINT,4     or      TSX  .COMNT,4
                 STR  FMT,,DIR     or      STR  SYMTB,DIR,FMT
                 OPS
                 STR  LIST,,ENDLST
                 OPS
                 STR  0

FORTRAN:  PRINT FMT, LIST
          FAP:  TSX  (SPH),4
                PZE  FMT,,SWITCH                              *
                OPS
                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.

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.

Identification

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

Purpose

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

Reference

MAD Manual, Chapter II, Section 2.16

Usage

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.

Revised: 8/30/65

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 11, 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.

Revised: 11/19/65

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.

Revised: 11/19/65

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 supervisor entries instead of the library subroutines means that buffers are not automatically assigned, only one buffer can be used, and errors cause execution of I/O system error procedures rather than the library error procedures.

For any given file, calls to the library routines may not be intermixed with calls to the core-B write arounds or I/O system entries.

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.

Revised: 11/19/65

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, PTW 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.

Revised 11/19/65

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.

To close an addressable disk file:
As supervisor or library entry:

```
    TSX  .ENDRD,4     optional(TIA  =H.ENDRD)
or  TSX  .FILE,4      optional(TIA  =H.FILE)
```

.ENDRD and .FILE are interchangable in the relative read-write mode. Write files not closed by one of these calls will be lost.

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.

Revised: 11/19/65

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".

Revised: 11/19/65

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: PŻE is temporary, PON is permanent, PTW and PTH are read-only and protected.

Revised: 8/30/65

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 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
06. 'LINKED' file not found

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.

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 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 read 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 written 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. Previous I/O out of bounds (membnd changed)
08. Input/Output error, see AG.4.06
09. Illegal use of write-only file (non-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).

Error codes:

03. File is not an active file
04. File is not in write status
05. No buffer assigned to this file
06. Previous 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 (non-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 completed 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 completed 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

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 COLT 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.

Revised: 1/3/66

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/O. 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 and 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).

Revised: 11/19/65

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.

Revised: 11/19/65

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.

Revised: 1/3/66

Identification

Switch current file directory
COMFIL, COMFL, TSSFIL, USRFIL

Purpose

To allow the user to switch between his file directory, common file directories associated with his problem number, or the system library file.

Usage

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 file directory.

BUSY is the location to which control will be transferred if the requested file directory were busy. The logical AC will contain the problem number, and the MQ the programmer number in BCD of the user currently using the requested file. N.B., 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:  A = COMFL.(N)      or EXECUTE COMFL.(N)
FORTRAN:  A = COMFL(N)       or CALL COMFL(N)
```

A is zero if switching has been performed or A will contain the programmer number of the user currently using the requested file directory.

To switch to the system library:
As supervisor or library entry:

TSX TSSFIL,4 optional (TIA =HTSSFIL)

TSSFIL switches the user to the system library file directory. This directory is composed of links to certain files in the system file directory which are in read-only, protected mode. The track 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 switches from the system library file back to the current file directory.

Note: the library entries, TSSFIL and USRFIL, may be called from MAD or Fortran programs.

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.

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.

Revised: 11/19/65

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.