# The Compatible Time-Sharing System

## A Programmer's Guide

The M.I.T. Computation Center

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

# The Compatible Time-Sharing System
## A Programmer's Guide

The M. I. T. Computation Center

F. J. Corbató
M. M. Daggett
R. C. Daley
R. J. Creasy
J. D. Hellwig
R. H. Orenstein
L. K. Korn

The M. I. T. Press
Massachusetts Institute of Technology
Cambridge, Massachusetts, 1963

# The Compatible Time-Sharing System

## A Programmer's Guide

Library of Congress Catalog Card Number: 63-19793
Printed in the United States of America

# PREFACE

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 elsewhere. 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 systems 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. Marjorie M. Daggett, Mr. Robert Daley, Mr. Robert Creasy, Mrs. Jessica Hellwig, Mr. Richard Orenstein, and Professor F. J. Corbató. Important contributions to some of the commands and the background system have been made by Mrs. Lynda Korn. Valuable criticism and advice 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

CONTENTS

# CHAPTER 1

# INTRODUCTION

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 monitor 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 hitherto available was 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 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 pre-time-sharing programming systems to continue to be operated. CTSS is used at a console which may be of several varieties, but which in essence is an electric typewriter. 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, is with no loss of generality since a command can also be used to start an arbitrary programming subsystem with its own control language level.

The consoles of CTSS form the foreground system, with computation being 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; all user programs are either kept in the 32,768 word B-bank of core memory, or swapped in and out of the disk (or drum) memory as needed.

Not only are the active user programs swapped in and out of the two secondary memory disk modules ($4.6 \times 10^6$ words each) and the drum ($.2 \times 10^6$ words) but it is expected that all console users will utilize the disk memory for semi-permanent storage of their active program and data files. Cards and magnetic tapes will still serve in secondary roles as long-time and back-up storage devices; they will be usable in CTSS only through the central Computation Center facilities and not directly through the remote user consoles.

Figure 1.1 is a system diagram of the 7090 computer at the Computation Center, which is to be converted to a 7094 Model 1 in August, 1963, with the addition of the 7320 drum at the same time. The motivation for the equipment configuration is described in the next chapter.

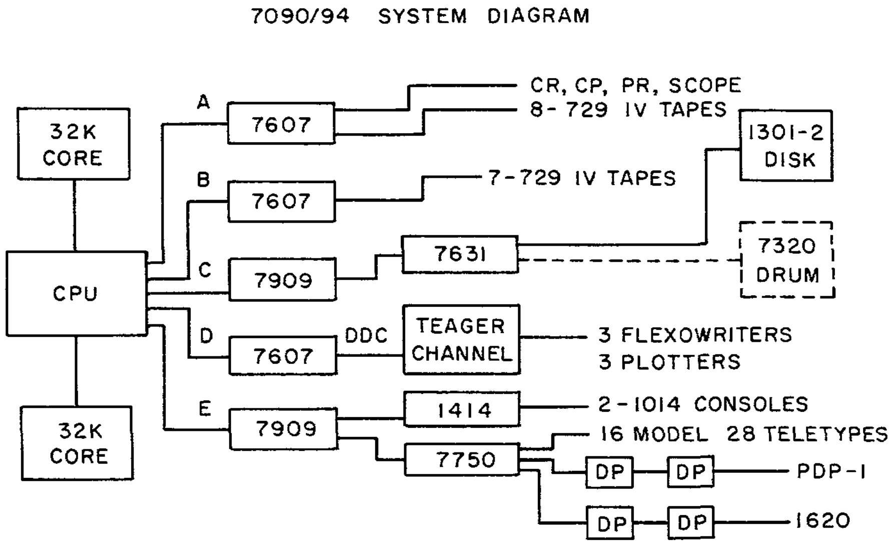

Figure 1.1 7090/94 System Diagram

# CHAPTER 2

## HISTORY, PLANS, AND CURRENT STATUS OF SYSTEM

Initially an experimental time-sharing system was developed using the special three-Flexowriter subchannel designed and built under H. Teager's direction. The first public demonstrations of this system were in November 1961, and the work was reported at the Spring Joint Computer Conference in May, 1962. During the latter half of 1962, in addition to refinement of the initial system, studies were made of how to remove several limitations in the system, namely: 1) the limited number of typewriter console stations, 2) difficulties of installing and maintaining remote connections, 3) lack of compatibility of operation with other 7090 programming systems, and 4) problems of information retrieval and information security associated with a large on-line secondary memory.

These studies resulted in plans for the conversion of the experimental time-sharing system to a pilot system which would be capable of operating simultaneously with most of the work load already handled by the 7090 at the Center. These plans for the 7090 included: 1) the implementation of the interrupt clock, memory protection, and relocation features; 2) the addition of an IBM 7750 communications channel which allows up to 112 Teletype consoles to be attached via phone lines at remote locations; 3) the addition of a second bank of 32,768 words of core memory; 4) the installation of the IBM 1301 disk file; and 5) the design and programming of a master disk control subroutine (memo CC-196) and an associated disk editor program (memo CC-208) for flexibility in using the crucial secondary memory. As separate experiments, two smaller computers are being specially attached to the 7750 channel by telephone line links: the Electrical Engineering PDP-1 is to be used as an experimental display console, and the Civil Engineering IBM 1620 will be used to control remote analog input-output equipment such as a pen plotter. All of the above equipment and attachments should be installed by spring, 1963, and the necessary programming adaptation required for the first pilot version of the time-sharing system is expected to be completed by summer, 1963. In fall, 1963, the addition of an IBM 7320 moderate-speed drum should further increase system performance by allowing somewhat faster secondary memory access and transmission speeds for swapping programs in and out of core memory, although the drum memory capacity is limited. At this time the general performance of the 7090 will be improved by upgrading it to a 7094 Model 1.

To implement effective operation of a pilot time-sharing system, the initial 16 Teletype consoles have been deployed as follows: two consoles for the Center system programming staff to assist in implementation and evaluation; one in the programmer consulting office to allow rapid assistance to users in the location of errors; one with the Center keypunch operators for service keypunching directly into the disk memory; one for the 7090 computer console operator; one for the 7090 tape-mount operator; one in Professor Minsky's research group for their applications and system evaluation; one in Professor Corbató's office for analysis, monitoring, demonstrations, and programming purposes; one more remotely located at Professor Miller's Civil Engineering Computation installation; and seven in an open pool for general time-sharing users at the Center.

One of the principal design features of the pilot system is that most older pre-time-sharing systems may be operated in the role of a background program simultaneously with the foreground time-sharing console system (memo CC-202-1). Because of the weakness of input-output memory protection in the 7090, it was necessary to develop an intricate input-output analyzer-monitor program for safe operation of older programs which had been developed without time-sharing in mind. Besides easing the programming transition problems of changing from batch processing to time-sharing techniques, this compatibility feature eliminates the necessity for immediately satisfying all of the Center's 400 research users (and a similar number of student users) with the initial version of the time-sharing system.

To achieve programming compatibility, the time-sharing system uses the same programming languages normally available in the major Center system, FMS: Fap, Mad, and Fortran in the form of the nearly equivalent Madtran (memo CC-188). Initial requirements dictated special core-memory-only versions of these compiler programs, but this restriction is being eliminated.

To contrast with batch-computing techniques, it is informative to summarize a typical time-sharing usage. A user has written a subprogram in a compiler language and wishes to incorporate it into his set of programs already developed and kept in the central disk file. After sitting down at a console, he first gives a login command to identify himself. His next command, input, turns his console into a pseudo-keypunch. Using simple conventions, he types in his subprogram for storage with other subprograms on the disk. If he should note any typing or logic errors in his new file, now or at any later time, he can correct the file using an edit command. (If he wanted to avoid the tedium of typing a long program, a suitably trained keypunch operator could just as easily have done the operation in advance.) Using an appropriate command the user can compile his subprogram, and if he has no diagnostics requiring correction, he can prepare to test it. Using the load command, he collects in core the newly compiled subprogram in binary form, any subprograms previously prepared, and the necessary library subprograms drawn from as many libraries as he wishes. If loading is successful, and does not require re-entry for the addition of missing subprograms, a start command initiates his program. He notes the results (if any) that he receives, and after stopping the program, if necessary, inspects the status of various locations and variables within the set of subprograms loaded. After a suitable amount of probing he detects his programming error, makes the necessary corrections, and continues in this manner until he wishes to terminate the session by giving a logout command.

To the above basic time-sharing system, which consists presently of a few dozen commands, many new features and additional commands must added before it will be generally acceptable. Among the more important ideas are the ability 1) to request from the consoles large-scale delayed printing or punching of programs and cards at the central Center facility

2) to create delayed graphical plots at the Center; 3) to allow foreground-initiated jobs to be run as background after a user leaves his console; 4) to assign by user request, for the duration of a console session, various input-output units, such as tape units, from the common central pool; 5) to allow inter-user console communication for activities such as management games; 6) to be more flexible and versatile in the allowable debugging techniques such as traces, interpreters, selective dumps, and so forth. Specifications for some of these features are described in later sections.

In addition to the obviously heavy programming development required for the above improvements, there are large operational questions which will need to be determined in the areas of reliability, error detection, programming system and hardware maintenance under conditions of near-continuous operation, automatic traffic and performance monitoring, and automatic accounting.

Finally, in the areas of hardware, there are evaluations to be made of the various consoles: the various model Teletypes, especially models 28 and 33, the Kleinschmidt 311 Teleprinter, the IBM 1050 Selectric typewriter, as well as possibly others. Vast questions lie in the area of self-maintaining display consoles, and finally questions of improved high-speed drums and future memory hierarchy schemes will need to be studied.

# CHAPTER 3

# GENERAL DESCRIPTION AND USAGE TECHNIQUES

The foreground system is organized around commands, which each user gives at his console, and the user's private program files which are kept on the disk. For convenience, the format of the disk files is such that they have titles with name and class designators. (Files can be entered from cards or 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 for the consoles; scheduling of console-initiated (foreground) and off-line-initiated (background) jobs; temporary storage and recovery of programs during the scheduled swapping; monitoring of all input and output from the disk, as well as input and output performed by the background system; and performing the general role of monitor for all foreground jobs. These tasks can be carried out by virtue of the supervisor's direct control of all trap interrupts, the most crucial of which is the one associated with the Interval Timer Clock.

The Interval Timer Clock is set for a small quantum of time, _q_. Every _q_ seconds the supervisor can interrupt the program currently running in B-core in order to accept input from the consoles or to issue output to the consoles. If the input from a console is other than a command line, it is left in the supervisor's core buffers until it is read by the user's program (whether a command or a user-written program) If the input line is a command, it is given immediate priority and the supervisor, after dumping as much as necessary of the current B-core program onto drum or disk, brings in the requested command program from the disk.

Except for this initial top priority, the time-sharing programs are each run for a burst of time which is some multiple of _q_ determined according to the scheduling algorithm. At the end of each program's appropriate time 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 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 to get a console line in or out; "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 will be another type of background, consisting of batch jobs initiated from consoles but left to run without console interaction; these jobs will be run with exactly the same type of scheduling as normal foreground programs.

## Command Format

The commands are typed by the user to the time-sharing supervisor (not to his own program) and thus can be initiated at any time regardless of the particular program in memory. (For similar reasons of coordination, the supervisor handles all input-output of the foreground system typewriters.) Commands are composed of segments separated by blanks; the first segment is the command name, and the remaining segments are parameters pertinent to the command. Each segment consists of the last 6 characters typed (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, "WAIT" is typed back. When the command is completed, "READY" is typed back. Where possible, the computer responses are in the opposite color from the user's typing. 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.

## Command Initiation

A user starts a command when he completes a command line at his console and is automatically placed in a waiting command status. The time-sharing supervisor uses the interrupt clock feature every quantum of time to interrupt the user program being run and assign the users in waiting command status to a working status with an initial priority based on the size of the requested command program.

## Program Termination

A foreground program leaves the working status by two means. It can re-enter the supervisor in a way which eliminates itself 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 the user may still restart or examine his program.

## Input and Output Wait States

User input-output is through each typewriter 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. Consequent there is an input-wait status and an output-wait status, both similar to dormant, 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 user program, as it enters the system to be run (or as a response to a user is completed), to an $\ell$th level priority queue. Programs are initially entered into a level $\ell_o$ corresponding to their size, such that

$$\ell_o = [\ \log_2 \left( [\ w_p / w_q\ ] + 1 \right)\ ] \tag{3.1}$$

where $w_p$ is the number of words in the program, $w_q$ is the number of words which can be transmitted in <u>and</u> out of the high-speed memory from the secondary memory in the time of one quantum, q, and the bracket indicates "the integral part of". Ordinarily the time of a quantum, the c time unit (currently about 200 ms.), is as small as possible without excessive overhead losses when the supervisor switches from one program in high-speed memory to another. The process starts with the time-sharing supervisor operating the program at the head of the lowest-level occupied queue, $\ell$, for up to $2^{\ell}$ quanta of time, and then if the program is not completed (i.e. has not made a response to the user) placing it at the end of the $\ell$+1 level queue. If there are no programs entering the system at levels lower than $\ell$, this process proceeds until the queue at level $\ell$ is exhausted; the process is then iteratively begun again at level $\ell$+1, where now each program is run for $2^{\ell+1}$ quanta of time. If during the execution of the $2^{\ell}$ quanta of a program at level $\ell$, a lower level, $\ell'$, becomes occupied, the current user is replaced at the end of the $\ell$th queue and the process is reinitiated at level $\ell'$.

<u>Change of Program Size</u>

Similarly, if a of size $w_p$ at level $\ell$, during operation requests from the time-sharing supervisor a change in memory size to $w_p''$, then the enlarged (or reduced) version of the program should be placed at the end of the $\ell''$ queue where

$$\ell'' = \ell + [\ \log_2 \left( w_p'' / w_p \right)\ ] \tag{3.2}$$

Again the process is re-initiated with the head-of-the-queue user at the lowest occupied level $\ell'$.

Message Interaction Considerations and Usage Times

Because a user cannot "see" his program running, it is important that the programmer resort to appropriate cues and confirmation messages in his programming. Thus when input is expected, a good technique is to have the program first type out a message such as "type next data set" or "A=". Similarly after input it is often reassuring, if a long computation is required, to see some program output acknowledging the receipt (and perhaps the accuracy) of the input message.

The reply response time is governed by the basic scheduling algorithm described earlier and depends on the number of other users. In the current implementation of the algorithm a user-computer interaction consists of the period from an input message to an output message. Although the response time is only a few milliseconds if the user's program is already in core, whenever there is more than one user the usual minimum time for a response, which is the minimum computer usage time per response, is (in the current drumless system) approximately S/8 seconds plus the response computation time, where S is the number of program words in thousands. For example, eight different user programs, of one thousand words each, could simultaneously respond within a second to input messages requiring only negligible computation.

For those interactions which require non-trivial computation, the computer usage time is approximately given by the sum of two series, the first of which is for the computation:

$$T = \sum_{i=0}^{n} \left( 2^i \frac{S}{8} \right) + \sum_{i=0}^{n} \left( \frac{S}{8} \right) \text{ sec.} \qquad (3.3)$$

where the nth term of the series is that just necessary to complete the computation. As can easily be seen, in the limit of large compu tations the ratio of usage time to computation time asymptotically approaches one.

## Storage Allocation

Presently only one user program is kept in core memory at a time. However, in order to improve response time, as many user programs as possible should be left in core memory. When required, as such as necessary of the lowest priority program in the scheduling algorithm may be moved to the disk (or drum) memory to make more core memory available. Because of the relative slowness of the 7320 drum and the 1301 disk transfers, moving program segments about in core is an effective way to consolidate space into which to read the user program next to be run. Thus the swap time required to give a response to a user will be only that dictated by the size of his own program. To carry out this procedure, several storage allocation algorithms are under consideration.

Although other possibilities exist, as in the Atlas Computer, initially no attempt will be made to operate any user program unless all of its program segments are contiguous and sequential in core memory. Also the possibility exists of performing a look-ahead operation of swapping a user program in or out while operating another user program (i.e., multiprogramming). Again, initially no attempt will be made to implement the look-ahead feature, since the effectiveness is seriously reduced if scheduling changes (i.e., program completions) are frequent, or if there is frequent competition with the operating program for use of the drum or disk channel.

## Memory Protection and Relocation

To avoid fatal conflicts between the supervisor and multiple users, the CTSS IBM 7090/94 includes a special modification which behaves as follows:

Core memory is considered to be 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 execution of 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 address, 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.

## 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 trap; control of each user's status, input 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, a convenient means of user-supervisor communication is for the user to cause a protection violation; if the violation conforms to an established convention it may be recognized as a call to a subroutine in the supervisor. Since the supervisor is in A-core at all times, and since the user is in B-core operating in the protection mode, the following convention is used:

```
             TIA         LOC
LOC          BCI         1,NAME
```

(or equivalently, TIA =HNAME) where NAME is the BCD name of the desired subroutine; the attempted transfer from B-core to A-core causes a protection violation.

In the future, should memory space in the supervisor region be available, some commonly-used library subroutines may be kept in this region and reached via dummy subprograms which issue the appropriate supervisor call.

## Time-Used Messages

Since usage time and computation time are different from real time, the user may periodically want to know how much computer time he has used. He may request from the supervisor a message of the following form:

MIN. USED = 12.070 + 2.005 IN 24.011 + 5.725

In this example, the user has logged 2.005 minutes of computer time since he last requested a time message, which was 5.725 minutes (real time) ago; at that time he had used 12.070 in 24.011 minutes; he has now used a total of 14.075 minutes since he logged in 29.736 minutes ago. Comparison of these sets of figures affords an estimate of his rate of service.

The time-request is made by typing in a special message at any level of console input; the message convention is initially set to "TIME" followed by carriage return. If the user wishes he may by use of the time command either change the time-request message to another set of characters or discontinue the facility altogether.

## Interval Timer Clock

To facilitate running programs for a limited amount of time, the CTSS IBM 7090/94 has an interval timer clock available. This clock is completely under control of the supervisor; its action is as follows: location 5, memory A, is incremented in the units portion every 1/60 sec; whenever it overflows on a count of $2^{35}$, an interrupt occurs which, if the clock is enabled, causes a trap to location 7 and the trap location to be stored in location 6.

(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 on-line printer clock. The supervisor can, however, simulate the interrupt clock behavior for each user. By supervisor calls similar to those of MITMR (memo CC-193), the user can program for nested interrupts and computation time readings.

## Break Characters

Whenever a user types into his console, regardless of whether his program is running or not, the input character is received at the supervisor level within 200 ms. The supervisor compares the character against the break character list of that user. (In routine circumstances, and after every command, the break character list includes only the carriage return.) 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 input 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.)

If a programmer desires to interact more frequently with input messages (including character-by-character reading), any arbitrary break character may be added to and deleted from his break character list by means of subroutine calls to the supervisor. The programmer, however, should anticipate response time delays and the extra computer time usage for each interaction.

## Quit Signals and Console Input Levels

When a user issues a command, two actions are initiated. First, his console input level is logically dropped from level 0 (command level) to level 1; second, a program is started (i.e., placed in working status), either the command program or his own, which executes until it terminates and enters dead or dormant status, or until the user manually terminates the program and places it in dormant status. Clearly this manual quit feature is desirable in that it allows the user to change his mind, correct mistakes, etc. The termination is performed by the user's issuing a single quit signal (varying with the console type) which may be issued even if the console is typing out. Upon issuance of the quit signal, the user's console input level is raised by one, normally back to level 0.

In addition to this basic two-level scheme, it is possible for the user to extend the number of input levels, thereby allowing for program subsystems, each with its own control language (e.g., for debugging). This is accomplished by the program giving subroutine calls to the supervisor, which on each entry drops the user console a level (to a limit of level 3). Whenever a quit signal from the console is received by the supervisor, the user's input level is raised by 1 (but no further than level 0); control is returned (by means of a push-down list) to the subsystem entry point previously assigned by the program to the current level, or finally, after the right number of quit signals, to dormant status.

## Character Mode Switch

For routine computer work, especially older applications, the normal 7090/94 BCD character set is sufficient for console messages. This set consists of 47 characters and blank, augmented by a few console control functions, namely: carriage return, tabulation, back space, color shift, delete-last-character, delete-last-message, and ignore; this normal BCD set is contained in a 6-bit code. When the character mode switch of a console is set to "normal", a console will transmit in the normal BCD mode.

The user's program, however, by issuing specific subroutine calls to supervisor entry points, may change the character switch to the "full" setting which, by means of a 12-bit character code, allows the user precise knowledge of console input as well as full flexibility upon output. Whenever a user program is completed and the supervisor is receiving input from a console at the command language level, the character mode switch is automatically restored to the "normal" setting.

## Consoles and Character Sets

Each type of console attached to CTSS has associated input-output mapping tables (Figure 3.1) for both the 6-bit and 12-bit character codes. (The high order 6 bits will be referred to as logical case bits.) In addition each console has a particular quit signal technique All characters are interpreted individually; hence any combination of legal characters may be used in a message. In particular, the physical case is automatically kept track of on both input and output, so that the user need not program physical case shifts.

In the case of 6-bit normal BCD mode input, characters which do not map into normal BCD characters or functions are enclosed in parentheses on the following chart and are ignored on input. The 6-bit normal BCD mapping is obtained by deleting the 6 high-order case bits. In a 12-bit mode input, all characters are coded and kept in a message.

In the case of console output, in the 6-bit normal BCD mode, logical case 0 will be used, while in the full 12-bit mode, all characters will be printed as specified with unused codes being ignored. After all output messages, the console physical case will be restored to its previous case before output, except for the IBM 1014 and 1050 Selectric typewriter consoles, where lower case will be restored after output.

When using Model 28 Teletype consoles, the quit signal is generated by depressing and relasing the <u>break</u> key. There are no backspace or color shift functions available.

When using the IBM 1014 Selectric typewriter consoles, the quit signal is generated by the sequence of: inquiry request, "#," inquiry release. To initiate each line of input, the inquiry request key must be depressed. If the check light comes on from typing too fast, inquiry cancel must be given and the line reinitiated. Inquiry release acts as a carriage return signal. The color shift function automatically reverses between input and output and is not codable. The only break character possible is carriage return. Type balls may be changed by a <u>chball</u> command. It is not possible to input a non-printing message, hence when it is desired to issue a carriage return with no message, the character "$\pi$," which is always ignored, is issued before the carriage return.

When using the Flexowriter consoles, the quit signal is generated, if there is no output typing, by the following sequence: depressing and releasing "code delete" (a lever above the keyboard), followed by a carriage return. If there is output typing, instead the red button on the control box in front of the keyboard must be depressed once. At present these consoles are only implemented for a single break character of carriage return.

## Code Table Conventions, Figure 3.1

### 12-bit mode

input to the user program

In the columns for the appropriate console, the keys with the characters or control functions indicated, when depressed, generate a logical case and 6-bit code as a single 12-bit code. A keystroke causes no CTSS control action, with the exception of a break character or quit signal. In the case of the quit signal, no code is transmitted to the program.

output from the user program

The character or control function corresponding to the logical case and 6-bit code in the columns under the appropriate console is typed out. If there is no character shown for the code, nothing types out.

### 6-bit mode

input to the user program

In either of the columns for the appropriate console, depression of a key with an indicated character or control function produces the corresponding 6-bit input code; if a character or control function is enclosed in parentheses the character is ignored and no input code is produced. In the case of the delete-character (d.c.), delete-message (d.m.) and quit signals, no code is transmitted to the user's program and instead the appropriate control action is taken. All other characters or control function codes are transmitted to the user's program. (The end-of-command character (e.o.c.) is generated by CTSS as a terminal flag to mark the end of the command

| 7090 Prog. Output | 7090 Prog. Input | CTSS Input Meaning | 7090 Octal Code | Mod. 28 Teletype | | Mod. 1014 Selectric | | Flexowriter | | PDP-1 | | 1620 | | Mod. 1050 Selectric | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Log. 0 | Case 1 | Log. 0 | Case 1 | Log. 0 | Case 1 | Log. 0 | Case 1 | Log. 0 | Case 1 | Log. 0 | Case 1 |
| 0 | 0 | 0 | 00 | 0 | (l.f.) | 0 | (]) | 0 | 0 | 0 | (Red) | 0 | | 0 | (°) |
| 1 | 1 | 1 | 01 | 1 | | 1 | (<) | 1 | | 1 | (Black) | 1 | | 1 | (±) |
| 2 | 2 | 2 | 02 | 2 | (bell) | 2 | (>) | 2 | 2 | 2 | (‾) | 2 | | 2 | (¢) |
| 3 | 3 | 3 | 03 | 3 | | 3 | (\|) | 3 | 3 | 3 | (_) | 3 | | 3 | (;) |
| 4 | 4 | 4 | 04 | 4 | | 4 | | 4 | 4 | 4 | (⊃) | 4 | | 4 | (#) |
| 5 | 5 | 5 | 05 | 5 | | 5 | (Δ) | 5 | 5 | 5 | (∨) | 5 | | 5 | (%) |
| 6 | 6 | 6 | 06 | 6 | | 6 | (#) | 6 | 6 | 6 | (∧) | 6 | | 6 | (@) |
| 7 | 7 | 7 | 07 | 7 | | 7 | | 7 | 7 | 7 | (<) | 7 | | 7 | |
| 8 | 8 | 8 | 10 | 8 | | 8 | (_) | 8 | 8 | 8 | (>) | 8 | | 8 | |
| 9 | 9 | 9 | 11 | 9 | | 9 | ([) | 9 | 9 | 9 | | 9 | | 9 | |
| | | | 12 | | | | | | | | | | | | |
| = | = | = | 13 | = | | = | → | = | | = | → | = | | = | |
| ' | ' | ' | 14 | ' | | ' | | 1 | | ' | | @ | | ' | |
| | | | 15 | | | | | | | | | | | | |
| | | | 16 | | | | | | | | | | | | |
| | | | 17 | | | | | | | | | | | | |
| + | + | + | 20 | + | | + | | + | | + | | + | | + | & |
| A | A | A | 21 | A | | A | a | A | a | A | a | A | | A | a |
| B | B | B | 22 | B | | B | b | B | b | B | b | B | | B | b |
| C | C | C | 23 | C | | C | c | C | c | C | c | C | | C | c |
| D | D | D | 24 | D | | D | d | D | d | D | d | D | | D | d |
| E | E | E | 25 | E | | E | e | E | e | E | e | E | | E | e |
| F | F | F | 26 | F | | F | f | F | f | F | f | F | | F | f |
| G | G | G | 27 | G | | G | g | G | g | G | g | G | | G | g |
| H | H | H | 30 | H | | H | h | H | h | H | h | H | | H | h |
| I | I | I | 31 | I | | I | i | I | i | I | i | I | | I | i |
| | | | 32 | | | | | | | | | | | | |
| . | . | . | 33 | . | | . | | . | | . | . | . | | . | (color up) |
| ) | ) | ) | 34 | ) | | ) | | ) | | ) | ] | ) | | ) | ( " down) |
| b.s. | b.s. | b.s. | 35 | : | : | : | : | b.s. | | b.s. | | | | b.s. | |
| c.s. | c.s. | c.s. | 36 | | | | | c.s. | | | | | | c.s. | |
| | | d.c. | 37 | | " | | " | | - | | " | | | | " |
| - | - | - | 40 | - | | - | | - | | - | | - | | - | (_) |
| J | J | J | 41 | J | | J | j | J | j | J | j | J | | J | j |
| K | K | K | 42 | K | | K | k | K | k | K | k | K | | K | k |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M | M | M | 44 | M | | M | m | M | m | M | m | M | M | m |
| N | N | N | 45 | N | | N | n | N | n | N | n | N | N | n |
| O | O | O | 46 | O | | O | o | O | o | O | o | O | O | o |
| P | P | P | 47 | P | | P | p | P | p | P | p | P | P | p |
| Q | Q | Q | 50 | Q | | Q | q | Q | q | Q | q | Q | Q | q |
| R | R | R | 51 | R | | R | r | R | r | R | r | R | R | r |
| | | | 52 | | | | | | | | | | | |
| $ | $ | $ | 53 | $ | | $ | | \| | | ↑ | \| | $ | $ | ! |
| * | * | * | 54 | * | | * | | : | | X | | * | * | |
| c.r. | c.r. | c.r. | 55 | c.r. | | π | | c.r. | | c.r. | | ‡ | c.r. | |
| | | d.m. | 56 | | ? | | ? | | _ | | ∿ | | | : |
| ignore | | quit | 57 | | | | √ | | | | | | | |
| blank | blank | blank | 60 | sp. | | sp. | | sp. | | sp. | | sp. | sp. | |
| / | / | / | 61 | / | | / | | / | | / | | / | / | (?) |
| S | S | S | 62 | S | | S | s | S | s | S | s | S | S | s |
| T | T | T | 63 | T | | T | t | T | t | T | t | T | T | t |
| U | U | U | 64 | U | | U | u | U | u | U | u | U | U | u |
| V | V | V | 65 | V | | V | v | V | v | V | v | V | V | v |
| W | W | W | 66 | W | | W | w | W | w | W | w | W | W | w |
| X | X | X | 67 | X | | X | x | X | x | X | x | X | X | x |
| Y | Y | Y | 70 | Y | | Y | y | Y | y | Y | y | Y | Y | y |
| Z | Z | Z | 71 | Z | | Z | z | Z | z | Z | z | Z | Z | z |
| tab | tab | tab | 72 | tab | | tab | | tab | | tab | | | tab | |
| , | , | , | 73 | , | | , | (%) | , | | , | | , | , | (single line) |
| ( | ( | ( | 74 | ( | | ( | | ( | | ( | [ | ( | ( | (double line) |
| | | | 75 | | | | | | | | | | | |
| | | | 76 | | | | | | | | | | | |
| | e.o.c. | e.o.c. | 77 | | | | | | | | | | | |

note 1) PDP-1, log. case 1, codes 00 and 01: red and black, output only.

note 2) 1014, log. case 0, code 55: pi as c.r. only if entire message.

Figure 3.1 Console Input-Output Mapping Table

parameter list.) It is not possible to input to a program the codes for which there are blanks in the column labelled "7090 program input."

output from the user program — The character or control function corresponding to logical case 0 and the 6-bit code in the column under the appropriate console is typed out. If for a code there is no character or control function indicated in the column labelled "7090 program output," the code will be ignored; for uniformity and future compatibility, it is recommended that the standard "ignore" code be used if this effect is desired.

## Special Input Modes

To anticipate noisy console lines, especially for future remote or dialed connections, a <u>confirmation</u> mode will be available. In this mode <u>every</u> input message (i.e., up to the break character) will immediately be copied back as an output message. If a user detects a garble he should issue a quit signal or take whatever action he has anticipated in his program.

When a person is working closely with his console at the command level, it will be possible to enter a <u>brief</u> mode. All programs which have been properly prepared will be able to interrogate the brief mode status indicator and to adjust the verbosity of their output accordingly Both the <u>brief</u> and <u>confirmation</u> modes are set by the use of a command.

## Interconsole Messages

Any user console can send a message to another user console by subroutine calls to the supervisor. These messages are placed in an input message pool for the receiving user along with the user number of the sender. The receiving user program can read its message pool at any time by a supervisor call similar to that for reading its own input console; an input-wait status can occur if no messages are present. If a receiver fails to read or aknowledge a message this is assumed to be intentional. The user number of another console must be determined by supervisor subroutine calls which give the desired user number on the basis of the console location, problem number and/or programmer number.

## Disk Memory Control

For a good understanding of the disk control subroutine, the following list of considerations concerning the use of the disk may be helpful.

1. The user is able to write and maintain permanent program and data files on the disk.

2. System programs (commands and standard library) are permanently recorded on the disk.

3. The user has only symbolic reference to his files.

4. The user is able to read and write many files simultaneously, and, for the sake of efficiency, may specify in which logical module a new file is to be written. In this way much unnecessary seek time may be avoided by using two separate modules for a simple read-write operatio

5. The user is not able to reference any files not authorized to him.

6. The user is able to initiate files in different modes, such as temporary files, permanent files, or permanent read-only files.

7. In order to utilize the maximum storage capacity of the disk file and to be compatible with any future I.B.M. programs using the the disk, Format 1 (i.e., a single record per track) is used.

A high level of user information protection can be achieved using the 1301 disk unit with the CTSS 7090. Complete protection is maintained (up to machine error and user identification errors) of all information on the disk unit. All users and systems use a single standard supervisor subroutine for reading and writing the disk. Because of the input-output trapping and memory protection features, mishaps will not occur if the user executes wild transfers, tries to store spurious information in the subroutine, or tries to execute explicit instructions for writing the disk.

It is desirable, from the point of view both of programming and of disk administration, that the user have no notion of the absolute location where his files of information are stored in the disk. Rather, the user will refer to his files only by symbolic names and logical mode number. Furthermore, if the user does not specify the logical module number, it will be assigned for him by the subroutine. Each of the user's files has two names; commonly the second name is descriptive of the type of file, as "FAP," "DATA," "BSS," etc. Each name is at most six BCD characters long; the characters may be any alphanumeric characters, but special characters should not be used; the special characters are used to distinguish the names of certain files created for the user by the supervisor.

Normally different programmers on the same problem are treated as different users. To allow convenient cooperation between programmers, such as students in classes or group projects, there is a feature which makes it possible to have files common to several different programmers with the same problem number. To gain access to these common files (referred to by programmer number zero) a special supervisor call must be given which places the user in a "common file mode;" another supervisor entry allows the user to return to the normal "personal file mode." More elaborate cross-referencing of files can be accomplished by the link or copy disk editor control cards described in Chapter 5.

## Organization of the Disk Memory

When the user initially requests use of CTSS in his Computation Center Problem Application Form he may also request the allocation of a number of tracks on the disk. If none are requested, an initial allotment will be made automatically which may later be extended by request. The track allocation information is kept on the disk in the master file directory of all users, which in addition contains the location of each user file directory. A track usage table is also maintained on the disk.

Each user has on the disk a user file directory (UFD) which contains a 4-word entry for each of his files. The first two words contain the file name and class name of the file. The third word contains the mode of the file in the prefix, a document number (a number assigned internally to each file, for tracing and retrieval purposes) in the decrement, and the address of the first track of the file in the address and tag. The fourth word contains the number of tracks used by the file in the address, and the date the file was last used (day, month, and last digit of year, coded as MMDDY, in octal) in the decrement.

If the user exceeds his track quota while writing a file, there will be an automatic temporary extension of his quota. This will allow him to complete any file he has begun; it will also allow for the automatic dumping of his machine status in case of system shut-down (see "System Service Changes and Supervisor Messages"). As soon as the track quota is exhausted the supervisor will notify the user of the fact. The extension will be maintained when the user issues logout. When he next logs in, he should relieve the excess in his track quota by adequate deletions. Should he fail to do this, a sufficient number of his oldest files will automatically be dumped at the next disk editing to restore his track quota to normal; the procedure for dumping these files will be similar to that for over-age files (see "Disk Editing Procedures").

## Keypunching and Card Input to the Disk

Normally, it is expected that casual program writing, editing and data generation will be done directly from the user's console. Even for the case of long programs which are too tedious for the programmer to type, a trained typist can still type the program directly into the disk memory. However, for various reasons such as distance, convenience, or exchange with other computer centers, it is still sometimes necessary to input punched cards. In this case, input cards must be brought, with an appropriate control card, to the Center Dispatching Area for entry into the disk during the disk editing period, which occurs at least once a day.

## Delayed Output and Disk Editor Control Cards from the Console

Normally, a programmer will find it more convenient to maintain brief but frequent interactions with his program through his console, but there will be occasions when large-scale output is desired. This output may still be obtained via the Center 1401 computers in the form of printing, punched cards, or pen plotting, by the user's issuing appropriate delayed output commands. The output files must be stored by the user on the disk in appropriate formats. The delayed output commands initially will cause appropriate disk editor control cards to be generated by the supervisor and automatically inserted in the disk editing procedure; thus the delayed output occurs at disk editing time. The user can also by appropriate commands or supervisor calls generate other disk editor control cards to be entered into the disk editing process.

## Disk Editing Control Cards

Periodically, requests will be processed by the Center Staff for input to the disk from cards and for output from the disk, either printed or punched. Control cards are submitted to the Center Dispatch Area which specify these requests. A variety of operations, specified by control cards, may be performed at disk-editing time, including, in addition to off-line input and output: editing of a binary program onto the disk; deletion of a file, with or without punched-card dumping; changing the mode of files (in particular of "read-only class 2" files) copying a user's file for another user; and linking several users (i.e., giving them access) to a particular user's file.

## Disk Reliability, Malfunctions and Recovery

The 1301 disk memory is expected to be quite reliable - more so, for example, than tape - but to date enough information has not been gathered to offer any definite appraisal of its reliability.

During times when CTSS is not in operation, the disk may on occasion be used by other systems. For this reason, as well as to protect information from loss due to malfunction of the disk, the contents of the disk are dumped onto magnetic tapes as needed and at least once a day. These tapes are used to reload the disk at the same time that any requests for new input to the disk are being processed. Disk dump tapes are written in duplicate to safeguard against tape errors. Dump tapes will be kept for backup purposes for several days, and a set of dump tapes will also be preserved from certain fixed points in the past (e.g.,week-old, month-old, three-month-old, etc.). Thus if any information on the disk should be lost or garbled, any or all user files could be restored to the status of an earlier day.

For example, were the entire contents of the disk to be erased, the last set of dump tapes would be used for reloading, and the loss confined to less than twenty-four hours worth of new information. This is not expected to happen since erasure does not occur easily and the disk normally retains all information in case of power failure.)

If both sets of duplicate dump tapes were to prove unreadable, the loss would be no greater than twenty-four hours, since the preceding day's tapes would be available. If the dump tapes were being used to restore information after disk loss, and both sets of tapes were unusable (an unlikely concatenation of misfortunes), the loss would still be confined to forty-eight hours since an earlier day's tapes would be available.

If an individual user's file is lost or garbled due to undetected failure of one or more disk tracks or to undetected tape error, the single file could be retrieved from earlier dump tapes either in earlier form or precisely, using the document number of the file for confirmation. If the user does not reference the file very frequently, however, it is possible that he might not ascertain the error until, say, a week after the loss occurred, and it might not exist on any older backup tape. Consequently, as with every type of information storage, it behooves the user to take his own precautions concerning valuable files.

## Disk Editing Procedures

Despite the rather large capacity of the IBM 1301 disk modules for secondary storage they are still capable of being filled by various means, from runaway programs to careless user file "housecleaning." To minimize the latter effect, each user is assigned an initial but usually adequate track quota, which can only be extended by a formal request to the Center. Furthermore a technique is used of automatically displacing, from the disk secondary storage to magnetic tape tertiary storage, those user files which have been unused and inactive for several weeks. These tertiary storage tapes will be created in duplicate each week and will be called history tapes. History tapes will automatically be erased after six months. No direct notice of file displacement will be made to each user; rather each user, upon examining his file directory, can by inspection see which files are still on the disk. Any attempt to use a displaced file will result in a console message to the user.

The above history tape procedure represents a compromise between automatic information disposal and the convenience of the user. The procedure which is illustrated in Figure 3.2 is done in conjunction with the normal disk editing procedures of dumping and loading the disk (at least daily).

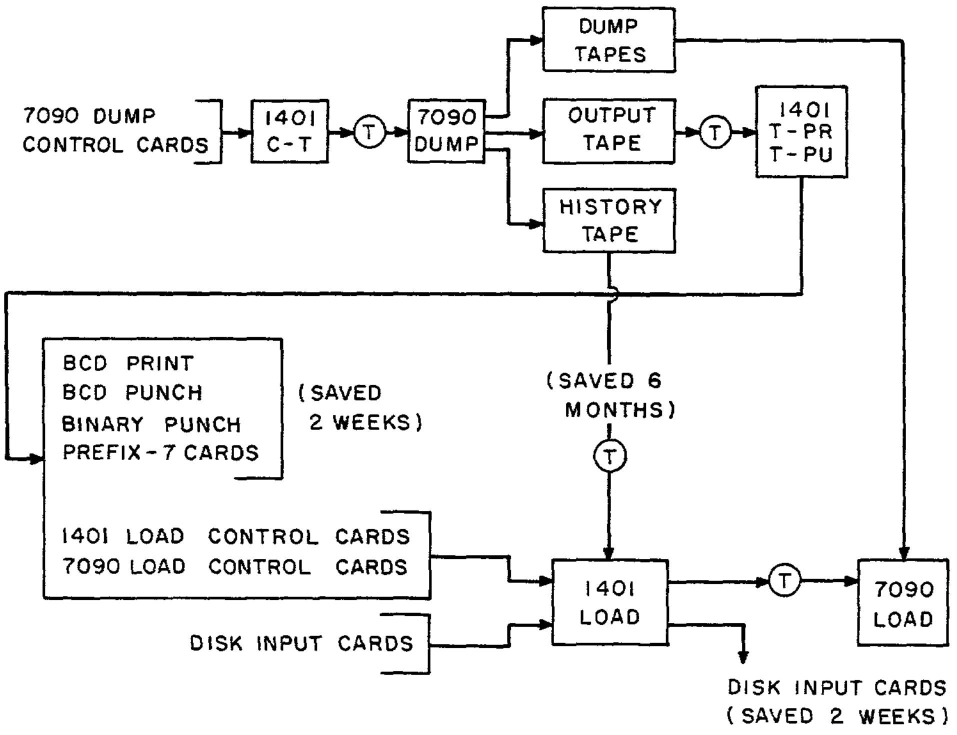

Figure 3.2 Diagram of Disk Editing Procedure

The first phase of the procedure starts with the 7090 dump editor and 7090 dump control cards being recorded on magnetic tape via the 1401 to run as a standard 7090 FMS job. The job first processes control card files left on the disk by the time-sharing supervisor, and then the input tape control cards, before producing the requested dump tapes. In addition, the current history tapes are extended by the over-age files and any files which the user has asked to be deleted; correspon-ding file directory entries are marked along with the date dumped and the history tape position. Finally, the dump program produces an out-put tape for the 1401 of output requested by the dump control cards, both those submitted directly and those generated from consoles. This output consists of: BCD printing, BCD and binary cards, prefix-7 binary cards (for convenient and safe reloading of the disk), and those remaining control cards not acted upon which were produced at the consoles and left by the supervisor. The latter control cards, which may include requests to restore to the disk some files which have been displaced to the history tapes, are sorted for systematic retrieval in the next editing phase. Requested disk output of all forms will be kept at the Center for a period of only 2 weeks before being disposed of. During this time, it is the user's responsibility to pick up his output at the Center.

The second phase of the procedure is that of loading the disk. The load-edit program is placed in the 1401 card reader, along with the control cards produced by the 7090 dump-edit program, as well as those load-control cards and input card decks submitted to the Center Dispatching Area. Input card decks, after they have been used by the edit program, will be placed in the Output Request Section of the Center and have a similar 2-week lifetime before disposal. By means of a special 1401 program, the above input is used to write a 7090 input tape which includes those files requested from history tapes. The latter is accomplished by the 1401 load-control cards produced by the 7090, which log on the 1401 printer giving the 1401 operator detailed instructions on the mounting and dismounting of history tapes. Finally the 7090 load-edit program, with input data, is run as a standard FMS job, where the input edit data and the previous disk dump tapes form the input.

The user will be able by a retrve command to retrieve his file from a history tape and replace it on the disk while at his console. This command will first give the user an estimate of the minutes of 7090 computer time required to position the appropriate history tape; if the user wishes to continue, the command will carry out the retrieval using the attach command; otherwise the user is free to create a 1401 load-edit control card such that the desired file will be entered into the disk at the next disk-edit period.

## System Service Changes and Supervisor Messages

Normally the CTSS system will operate for scheduled periods in blocks of time during each day. System operation will commence with messages typed on all consoles that are on (or can be remotely turned on and off) stating that the system is on and at what time the system will cease to operate again.

When the CTSS system is shut down, either on a scheduled basis or an emergency basis, the following procedure occurs. Each active user ceases to run, a cutoff message is sent to his console, all files are closed, all attached input-output units are disconnected, the user's program is created as a file of name cutoff with class saved, the user is given a logout command, and a message is placed in the user's console message file on the disk giving a summary of the cutoff process including the unread characters in the console input buffer.

Whenever there are individual or general disk failures as well as system changes of great importance, appropriate messages will be placed by the supervisor in each of the user console message files. A console message file (named "MESSAG E.FILE") is created for the user when necessary and is normally typed out by the login command, at which time it is deleted. During operation of his console, additional messages may be generated; a user can read his console message file at any time with the printf command, which will not delete the file. The console message files, which are created by the supervisor, use disk tracks of the user's quota.

For writing message files and for saving the user's program at the time of a system shut-down, the supervisor may have to take advantage of the feature which allows an extension of the user's track quota (see "Organization of the Disk Memory"). In this event the user should, the next time he is at a console, relieve this condition by dumping some of his files, or else older files will be dumped automatically for him at disk editor time. If at shut-down time the user himself has already exceeded his quota, no further extension will be allowed; his machine status will be lost and so may his message.

## Assignment of Input-Output Units

There are numerous devices such as tape units, scopes, plotters, satellite computers, etc., which are of such a nature that it is desirable to assign them to a single user for his session at his console. This assignment is done by means of an attach command, where the user may receive a busy signal whenever requested units are actively engaged. Addressing conventions of input-output units will always be by logical unit numbers except when location is relevant.

The supervisor will maintain for each user an input-output assignment table in which the logical-physical unit number interconnection will be kept. When a user gives a logout command at his console at the end of his session, all assigned units will be disconnected.

## Magnetic Tape Usage

Although for a large number of cases the disk memory, because of its quasi-random access, is more appropriate for program data and temporary storage requirements, there are still many problems such as payrolls, bubble chamber scanner output, etc., where magnetic tapes are more appropriate. This is true partly because of the vast quantities of information.

Within CTSS, the user may use private tape reels in the following way. The user types in an attach command specifying, for example, that file-protected private tape reel 103 should be mounted as logical tape 2 on logical channel 2. The computer replies "WAIT," as the command requests from the supervisor pool an available tape drive. Upon receipt of an appropriate tape drive assignment, the attach command sends an interconsole message to the tape operator's console (where his program is continuously processing messages), of the form:

"mount reel 103 as tape A3, file protected."

The tape operator, on completing the action, types "ok" which causes a confirming interconsole message to be sent back to the attach command The command can relay to the user the tape mounting information by repeating the input information and concluding with "READY."

## Dataphones

In addition to Teletypes, the IBM 7750 Communication Channel allows the attachment of higher speed devices (1200 bits/sec.) by means of Bell System Model 202B Dataphone sets, one at the 7750 and another at the device, with a telephone line between them. The devices (e.g., PDP-1 computer or a 1620 computer) will appear as input-output units, and a user, from his console, may request to have them logically attached to his program. For convenience each device can be considered by the programmer to consist of several logical subunits; in particular, for control purposes one of these subunits (e.g., the typewriter keyboard) can represent a user console, so that an auxiliary console is not needed Provision has been made for error detection and transmission correction in the internal character and message formats, which are further described in memo CC-210.

## Command Programming

Whenever a user issues a command at his console, the command name and parameter segments are placed in order in a command parameter list, and the corresponding command program is located and started wherever indicated by the initial machine conditions of the program. The command program reads the parameter segments by issuing supervisor calls.

The only exception to the above process occurs with the resume command where the format for resuming program $\alpha$, which has been "saved" by the save command, is

$$\text{resume } \alpha\ p_1\ p_2\ \ldots\ p_n .$$

In this case, just before starting the $\alpha$ program, the command list is reordered to contain the sequence $\alpha\ p_1\ p_2\ \ldots\ p_n$. The purpose of this exception is to allow the user a trivial way to develop and use private commands which can also be compatibly added to the general system.

A useful property of the command mechanism is the ability of one command to call another command. The setting of the next command name and parameter list is done by the operating command program issuing appropriate supervisor calls.

By letting a command program set up an internal "command location counter" (not seen by the user), it is possible to run a master command program which in effect operates a "program" of subcommands, each of which returns to the master command after increasing the command location counter; the master command effects the return to the supervisor. The master command and the subcommands may, by means of a supervisor subroutine call, modify the command parameter list.

When this procedure is used, a fresh copy of the master command program is brought in each time it is executed, and the "command location counter" is used to dispatch control to the next command sequence. Conditional branching can be realized by letting the subcommand increase the location counter by a variable amount. Communication between subcommands must be accomplished by leaving data files on the disk.

## Console-Initiated Background

Although the foreground console system is highly useful for preliminary programming, experimentation, and general man-machine interaction problems, there are occasions when long, uninterrupted computations must be performed by foreground programs. In these cases, the user ordinarily neither needs nor wants to occupy a console during the computation. To avoid this situation, it will be possible for the user, by placing his operating program in dormant status and issuing a bkgrnd command, to turn his job over to the supervisor for operation as console-initiated background. In the bkgrnd command the user will specify as parameters: the computation cut-off time; the name of the job file; a name for the file to contain the final status of the program; and the name of a file for any console messages which may develop (including information concerning the final job status).

After a console-initiated background program is turned over to the supervisor, the user, at a later time (even after he has issued logout and login commands), may interrogate the supervisor by means of a status command to obtain the status of his one or more jobs. From the status command, the user receives a report containing: whether the job has finished or not; the time the job was submitted; the computer time used to date; the cutoff time; the current program level number; and the level populations in the scheduling algorithm.

## Estimate of Computation Completion Time

When a user is present at his console operating a foreground program, he can easily determine the rate of computational progress by the use of the time command. However in the case of console-initiated background, estimation of completion time becomes a more difficult problem. For this purpose, an estimate command will be made available. Its properties are expected to be as follows:

The console user gives as command parameters the designation of the particular background job in question, and the total expected computation time required by the job. The estimate command can determine the position of the job in the scheduling algorithm, and it has available a cumulative history of the scheduling traffic on, for example, an hourly basis over a period of a week. For each hour, the traffic history consists of a histogram, for each level of the scheduling algorithm, of the number of job initiations versus the total computation time required (including time in subsequent levels). Knowing in addition the current job status and the scheduling algorithm itself, the estimate command is in a position to run a high-speed Monte Carlo simulation, into future time, of the expected system behavior. By running several such simulations, using different initial settings of the pseudo-random number generator, the estimate command can type out a message like: "Range of estimates over 3 trials = 131 min. to 237 min. Estimated Completion time between 3:16 pm July 1 and 5:02 pm July 1." (An alternative message might be "job already completed.")

If the estimated time is too long, the user will initially have no recourse except to cancel or stop the background job. Future systems may contain priority alternatives, with additional usage charges serving as restraints.

## Background System Restrictions

The normal user of CTSS will not need to concern himself with the following section which is primarily of interest to those programmers dealing with older pre-time-sharing programs.

Certain programming conventions or restrictions must be followed in programs running as background jobs in CTSS. These restrictions are principally dictated by the need periodically to interrupt all programs operating in CTSS. Normally, well-programmed systems will already satisfy the requirements. What is required, in the main, is a program which is timing-insensitive (i.e., one which would operate correctly were the computer to be put in manual operation at any moment).

1. The use of the following instructions is prohibited; if a system uses one of these instructions, a protection mode violation occurs and a diagnostic will be given.

| | |
|---|---|
| ECTM | LPI |
| ESNT | LRI |
| ESTM | SEA |
| ETM | SEB |
| LCH | TIB |

2. Data channel traps on channels A and B will operate normally except that when a trap occurs, instead of an effective XEC of location 13 or 15, control is transferred to location 13 or 15. Therefore, if data channel traps are expected, location 13 or 15 can only contain an unconditional transfer for proper program operation. (If this condition is not satisfied a diagnostic will occur.)

3. The background system, if authorized by the Center, may use the Disk Control Subroutines (Chapter 5) by calls identical with those used by foreground programs.

4. Since many foreground users may have programs operating, it is undesirable to have the operator intervene in the conventional manner to take terminal post-mortems. The "Load Cards" and "Load Tape" buttons are not to be used by the operator, since they reset the memory protection and relocation mode. The "Start" button cannot be used because the operator does not know when the background system is operating.

However, in order that the operator may effectively perform these functions, the address of the console keys is continually checked by the Time-Sharing Supervisor. (The background system cannot use the address section of the keys.) When the supervisor finds certain codes, it simulates the following functions:

a. Depressing the "Load Cards" button.

b. Depressing the "Load Tape" button.

c. Depressing the "Start" button.

d. Initiating the "standard error procedure."

The "standard error procedure" consists of storing the instruction counter in a specified location, and transferring control to some address which is the start of a post-mortem routine or a return to the background system monitor. The background system should therefore specify two locations for use by the Time-Sharing Supervisor. This is done by the following call to a supervisor subroutine:

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

where DEFERR contains: TIA =HDEFERR. DEFERR is the name of a supervisor subroutine which defines the error procedure. ERRILC is the address where the instruction counter will be stored, and control will be transferred to location ERRTRA.

5. GETTM is the subroutine used to read the date and time from the special Computation Center on-line printer clock. The original version of the program uses the "load channel" instruction (which is illegal), and is not interruptable. A new, interruptable version will be available.

The interval timer clock will be simulated for use by the background system. This clock will only operate when the background system is in control and should therefore be used to determine the amount of time run. MITMR, the Center's timer program (CC-193), may be used in the normal manner.

6. In general on the 7090, when an interrupt occurs, the interrupted program is resumed at the location specified in the appropriate lower core register. When the 7090 is stopped by an HPR instruction,

the instruction counter is set to the next location; it is this location which is stored if an interrupt occurs. Thus if there is an interrupt at this point, the program resumes at the next instruction after the HPR. Since interrupts are a normal occurrence in the Time-Sharing System, the 7090 cannot always be stopped by an HPR instruction. The HTR instruction, however, produces a genuine program stop and should be used in place of the HPR.

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

a. the card reader, card punch and printer;

b. tape units A1-A6, A10, B1-B6 and B10.

If other units are referenced, a diagnostic will occur.

8. The "Load Channel" instruction is prohibited. A diagnostic will occur if an LCHX instruction is used. A "Reset and Load Channel" instruction, if given, must immediately follow the select instruction; otherwise, the I/O check light will be turned on. An exception is made for three instructions; up to three SPR's, or SPU's, and/or NOP's may be inserted between the select and reset and load instructions.

9. All I/O commands (including TCH commands) must have a "1" in bit position 20. This will automatically be done on assembly if the BCORE pseudo-op is used in the Center version of the Fap assembler. I/O instructions, if generated, must also contain this bit. A diagnostic will be given if this condition is not met.

10. When the supervisor determines that the background system is to be saved while a foreground program receives attention, the background system's machine conditions must be saved. In order to save the status of an I/O channel, the supervisor must wait for current I/O activity to stop. If the background system reads or writes long records on tape, the time spent waiting in the supervisor can be detrimental to the foreground response time. It is normally required, therefore, that all physical records read or written on tape be less than 1000 words. If this condition cannot be met, the matter should be discussed with the Computation Center Staff.

11. To ensure insensitivity to timing, proper interlocking of I/O commands and the CPU program must be guaranteed, usually with TCOX instructions. When this is not done, it is sometimes possible to get erratic program operation due to the probabilistic nature of other program interrupts (and consequent program delays) occurring during the operation time of I/O commands which change the working data area of the interrupted program. This also applies to control-card coding.

# CHAPTER 4

## SUPERVISOR SUBROUTINE CALLS

The subroutines listed below are available to CTSS users through the supervisor program; a brief description of their function follows. The entries to the disk control subroutine are also via the supervisor; the function of each of these entries is described in Chapter 5.

| Supervisor Subroutines | Disk Control Subroutine Entries |
|---|---|
| RDFLXA | .DUMP |
| WRFLX | .LOAD |
| WRFLXA | .ASIGN |
| DEAD | .WRITE |
| DORMNT | .APEND |
| (EFTM) | .FILE |
| (LFTM) | .SEEK |
| GETMEM | .READK |
| SETMEM | .ENDRD |
| GETCOM | .RELRW |
| AKNOLG | .CLEAR |
| TSSFIL | .DLETE |
| USRFIL | .RENAM |
| SETBRK | .RESET |
| SAVBRK | .FILDR |
| GETBRK | .FSTAT |
| SETFUL | |
| SETBCD | |
| NEXCOM | |
| GETILC | |
| FNRTN | |
| INSTRT | |
| INPEND | |

Supervisor subroutines are called by means of the trapping convention described in Chapter 3, "User Communication with the Supervisor Where arguments are expected, they are located relative to index register 4 as in normal calling sequences; for example

```
                TSX     RDFLXA,4
                PZE     BUFF,,4
                        . . .
                        . . .
     RDFLXA     TIA     =HRDFLXA
```

is a call to RDFLXA with a single argument in (1,4) and a return to (2,4). The BCD name of the supervisor entry, as specified in the TIA instruction, must be left-justified with trailing blanks.

To read an input line from the console,

```
                TSX     RDFLXA,4
                PZE     BUFF
```

reads 14 words into memory starting at BUFF. On return the logical AC contains N where the break character is the Nth character; the word containing the break has blanks to the right of the break; subsequent words contain blanks.

To issue an output line to the console,

```
                TSX     WRFLX,4     or     TSX     WRFLXA,4
                PZE     BUFF,,N            PZE     BUFF,,N
```

where WRFLX writes the N words (N≤14) starting at BUFF, adding a carriage return at the end of the line, and a color shift at beginning and end where the color shift is available; blanks are deleted from the end of the line. WRFLXA differs in that it does not add carriage return or color shift, and does not delete terminal blanks.

All entries to the master disk control subroutine are interpreted like supervisor calls. See Chapter 5.

The call:

```
                TSX     DEAD,4
```

returns control to the supervisor and puts the user in dead status, i.e., machine conditions are not saved.

```
            TSX     DORMNT,4
```

returns control to the supervisor and puts the user in dormant status, i.e., machine conditions and the current status of the user's program are saved, unless a command is issued which causes a new program to be read into memory. If the start command is issued, control returns to (1,4)

The call:

```
            TSX     (EFTM),4
```

causes entry into the floating-point trapping mode, with trapping simulated in B-core.

```
            TSX     (LFTM),4
```

causes exit from this mode.

The call:

```
            TSX     GETMEM,4
```

returns in the address of the AC the size of the user's current memory allocation. The call:

```
            TSX     SETMEM,4
```

sets the size of the user's memory allocation to the value in the address portion of the AC. This subroutine must be used whenever the user wishes to increase the size of his program. If he fails to do so and refers to locations outside his original memory allocation he may not cause a protection violation, since the protection indicators are set by block count rather than word count; but dumping, when users are swapped in and out of core, is done by word count, and all information stored beyond the memory bound will be lost.

The call:

```
            TSX     GETCOM,4
            PZE     N
```

returns in the logical AC the Nth word of the user's latest command. Each command issued by the user is written in the supervisor region, one word per argument, in an 18-word buffer; after the last word (the last parameter in the command line) a marker is written consisting of a word of octal 7's. The command name itself is word zero.

The call:

```
            TSX     AKNOLG,4
            BCI     1,*
```

where * may be any console character (other than zero), puts the user in an input acknowledge mode; whenever a message is received from the console the specified character will immediately be typed out. To leave the acknowledge mode, the following call is given:

```
            TSX     AKNOLG,4
            PZE
```

The call:

```
            TSX     TSSFIL,4
```

allows the user to read CTSS system files from the disk (e.g., the CTSS library). When in this mode <u>only</u> CTSS files may be read; to reset the mode,

```
            TSX     USRFIL,4
```

allows the user to reference his own files again.

To drop the console input level (see Chapter 3, "Quit Signals and Console Input Levels") and set the corresponding quit entry point, the following sequence is used:

```
            TSX     SETBRK,4
            PZE     ILC
```

which drops the input level by one and sets the return corresponding to the new level to the value of ILC. The call:

```
            TSX     SAVBRK,4
```

raises the console input level by one and returns in the AC the quit entry point corresponding to the level just left. If the command level (level 0) is reached, the contents of the AC is zero. The call:

```
            TSX     GETBRK,4
```

returns in the AC the value of the instruction location counter at the time the user last gave the quit signal.

The call:

```
        TSX     SETFUL,4
```

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

```
        TSX     SETBCD,4
```

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

The next group of subroutines have been designed for fairly special purposes; they are most likely to be useful for the writing of commands.

The call:

```
        TSX     NEXCOM,4
```

where the AC contains the name of a command, right-justified with leading blanks, puts the user in waiting-command status with the specified command as his next command. (This is used, for example, by the madtrn command to chain into the mad command.)

The call:

```
        TSX     GETILC,4
```

returns in the AC the value of the instruction location counter at the time when the user last entered dormant status. The call:

```
        TSX     FNRTN,4
```

returns the user to dormant status; the user's instruction location counter is reset to the value it had when he last entered dormant status. (Both these entries are used by the pmr command.)

The call:

```
        TSX     INSTRT,4
```

puts the user into console input mode. The contents of the AC is taken as a decimal initial line number; after adding ten to this number the supervisor prints it out; when a line is typed in it may then be read by RDFLXA. The supervisor continues to generate line numbers, incrementing by ten. According to the conventions described under the input command, (Chapter 7), a manual mode may be entered where the supervisor suspends the generation of line numbers. To exit from the console input mode the following call is given:

```
                TSX     INPEND,4
```

(Both these entries are used by the _input_ command.)

Additional supervisor entries, to be added in the near future, will allow the following requests: setting a break character other than the normal one for the console; setting a character or set of characters which when typed by the user will denote a request for a time-used message; setting the input confirmation mode; requesting interconsole communication; calling adapters for the attachment of additional console units; requesting access to files assigned to other programmers with the same problem number; altering the increment used in automatic generation of line numbers (_input_ command); modifying by program the command parameter list.

# CHAPTER 5

# USE OF THE 1301 DISK MEMORY

## Master Disk Control Subroutine

The disk control subroutine may be considered an extension of the supervisor. It provides means whereby the CTSS user may store information on the disk for indefinite periods, such information to be completely protected and readily accessible for console and off-line reference.

Each user is assigned one or more tracks to serve as a directory of all his private files currently stored on the disk. The directory consists of a set of 4-word entries, each of which contains a file name (two BCD words) and two control words indicating the type of file, the date the file was last used, and the address of the first track assigned to this file. If the file is more than one track long, a second track is chained to the first such that the first data word of the first track assigned to the file will contain the address of the second track. Additional tracks as needed are chained in the same manner. The first data word of the last track of a file will contain a pointer to the last word in the file. Additional tracks for the user's private file directory may also be chained in a similar manner, if necessary. This chaining of tracks is done automatically by the disk control subroutine so the user will never need to concern himself with absolute addresses within the disk.

The user may specify any of four modes for a file. The mode is designated by a prefix code in certain of the disk-routine calling sequences. The following is a list of the permissible prefix codes with their meanings:

| | |
|---|---|
| PZE | Temporary file, will be deleted as soon as it is read. |
| PON | Permanent file, can be read or altered indefinitely. |
| PTW | Read-only (class 1), can be read but not altered until the mode has been changed. |

PTH Read-only (class 2), can be read but not altered except by edit cards submitted to the Center. Any attempt by a program to alter a file in this mode will cause a diagnostic comment to be printed.

Temporary files will be deleted when the user gives the <u>logout</u> command. They are not included in his track quota, and he may use as many as he wishes as long as there are enough free tracks on the disk.

All input and output for disk files is effected through the disk control subroutine. A set of entries is provided which are called like CTSS supervisor subroutines, i.e., by setting index register 4 and entering the subroutine by means of a TIA instruction (see Chapter 4).

To dump a continuous block of core onto the disk as a file the following sequence is used:

```
TSX     .DUMP,4
***     FILNAM,,MODNO
PZE     A,,n
```

where A is the location of the first word to be dumped, n is the number of words to be dumped, FILNAM is the location of the first of 2 consecutive BCD words which will become the name of this new file, and MODNO is the logical module number. Logical modules 1 and 2 will be available, although only 1 is used at present. If MODNO is zero, the disk control subroutine will assign the module number. *** is a prefix code which defines the mode of the file. Upon completion of a dump operation, the new file name is added to the user's private file directory.

To load a continuous block of core from a file previously recorded on the disk, the .LOAD entry is provided:

```
TSX     .LOAD,4
PZE     FILNAM
PZE     A,,n
```

where A,n, and FILNAM are of the same form as in the .DUMP calling sequence with the exception that n may be set to a larger value than necessary for loading files of indeterminate length. The prefix code and module number need not be specified.

For cases in which the .DUMP and .LOAD entries are insufficient (i.e., the blocks of core to be read or written are not contiguous or are to be read or written in small units) a more versatile set of entries is provided. The first is the .ASIGN entry that prepares the system for the writing of a new file. (More than one file may be written at one time.) The calling sequence is of the following form:

```
        TSX     .ASIGN,4
        ***     FILNAM,,MODNO
        PZE     WBUF1,,WBUF2
```

where WBUF1 and WBUF2 are the initial locations of two 470-word blocks of core to be used by the disk control subroutine for the buffering of subsequent calls for writing. If available storage space is not sufficient for double buffering, WBUF2 may be specified as zero and only the single buffer specified by WBUF1 will be used. FILNAM and MODNO are of the same form as described in the .DUMP calling sequence. The prefix code *** determines the mode of the new file and is as described under the .DUMP calling sequence.

The .ASIGN routine assigns a free track, in the module specified by MODNO, to be the first track of the new file and sets the disk control hardware in motion to find this track. While the apparatus is looking for this track, the .ASIGN routine returns to the calling program which may then make use of the seek time.

To write information into a file initiated by a call to the .ASIGN entry, any number of calls to the .WRITE entry may be used in the following form:

```
        TSX     .WRITE,4
        PZE     FILNAM
        PZE     A,,n
```

Starting from location A, n words will be written into the file specified by FILNAM.

To write additional information at the end of an existing file, the following entry may be used in place of the .ASIGN entry:

```
        TSX     .APEND,4
        PZE     FILNAM
        PZE     WBUF1,,WBUF2
```

which will locate the file specified by FILNAM and prepare to write starting at the end of the information already recorded. In other respects this entry is used just as .ASIGN.

To terminate the writing of a file and free the write buffers for other work, the following sequence is used:

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

The .FILE routine will write out the last buffer load of the file specified by FILNAM with a pointer to the last word of the file. The user's private file directory is updated with the new file name.

To initialize the disk control subroutine for reading a file from the disk, the following calling sequence is provided:

```
        TSX     .SEEK,4
        PZE     FILNAM
        PZE     RBUF1,,RBUF2
```

where RBUF1 and RBUF2 are the initial locations of two 470-word blocks of core to be used by the disk control subroutine for the buffering of subsequent calls for reading. If available storage space is not sufficient for double buffering, RUBF2 may be specified as zero and only the single buffer specified by RBUF1 will be used.

The .SEEK routine will locate the file specified by FILNAM in the user's private file directory, pick up the address of the first track of the file, and set the disk control hardware in motion to find this track. While the apparatus is looking for the track, the .SEEK routine returns to the calling program which may then make use of the seek time.

To read information from a file initialized by a call to the .SEEK routine, any number of calls to the .READK entry may be used in the following form:

```
        TSX     .READK,4
        PZE     FILNAM
        PZE     A,,n
        PZE     X
```

The first or next n consecutive words will be read from the file specified by FILNAM, and stored in consecutive locations starting with location A. If an attempt is made to read past the last word of the file, control is transferred to location X. When this occurs, the file being read is dropped out of the read status, and the buffers being used are available for other work.

To terminate the reading of a file without having to read past the last word in the file, the following sequence is used:

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

For some applications, it is logically much simpler to treat each file as an addressable secondary memory. To facilitate this procedure, the next group of calls are alternatives to the above tape-like calls.

The call:

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

will allow the specified file to be treated as an addressable secondary memory. Only one buffer should be specified, as double-buffering is not feasible in this case. This may be followed by calls to both .WRITE and .READK, using the calling sequences next described.

The call:

```
        TSX     .WRITE,4
        PZE     FILNAM,,RELADR
        PZE     A,,n
```

will write the n words starting at core memory location A into the n words of the file starting at relative location RELADR.

Similarly as in the .WRITE call, when the first parameter of a .READK call is

```
        PZE     FILNAM,,RELADR
```

and RELADR is non-zero, the reading process starts from that relative address of the file.

When reading and/or writing is terminated, a call to .FILE or to .ENDRD is given; when in the relative read-write mode, .FILE and .ENDRD are interchangeable.

The call:

```
          TSX      .CLEAR,4
          ***      FILNAM,,n
```

will create a file called FILNAM and of mode ***, and will write n zeros into the file.

To delete a file from a user's private file directory and thus free the tracks assigned to the file for other use by the same or by a different user the following sequence is provided:

```
          TSX      .DLETE,4
          PZE      FILNAM
```

The .DLETE routine does not require any buffer storage, but it still must read every track of the file in order to find the addresses of all the tracks used by this file and make the necessary changes in the track usage table. Any file which has been placed in the read-only status (either class 1 or 2) cannot be deleted in this way.

The name or mode of a file may be changed by the following sequence:

```
          TSX      .RENAM,4
          ***      NEWNAM,,FILNAM
```

The .RENAM routine will replace the file name specified by FILNAM with the name specified by NEWNAM (two BCD words at NEWNAM and NEWNAM+1) in the user's private file directory. The prefix code (***) determines the new mode for the file. A file which is in the class 1 read-only mode may be altered only by first changing its mode with a call to .RENAM. If it is desired to delete a file which is in the class 2 read-only mode, it is necessary to submit an edit card to the enter.

The call:

```
          TSX      .RESET,4
```

will drop from active read or write status any of the user's active files. Files which are reset from active write status will be lost. Temporary files reset from active read status will be deleted.

To obtain a copy of the user file directory, the following sequence is used:

```
                TSX     .FILDR,4
                PZE     0,0,BUFF
```

The first track of the UFD is read into the 470 locations starting at BUFF. The decrement of the first word is zero if there is more than one track to the UFD; in this case the address and tag of this word contain information to be transmitted to the supervisor to obtain the next track; this address and tag must be stored in the argument of the next call, as:

```
                TSX     .FILDR,4
                PZE     **,**,BUFF2
```

which will bring in the next track of the UFD.

The decrement of the first word of the last (or only) track contains the number of words in this track not counting the first. The address of the second word contains the user's total track count. The set of 4-word entries for each file follows, beginning with the third word.

Information concerning a file may be obtained by the following sequence:

```
                TSX     .FSTAT,4
                PZE     FILNAM
```

which will return the following information in the logical AC:

```
                ***     WDCNT,,MODNO
```

where *** specifies the mode, WDCNT is an estimated (maximum) word count based on the number of tracks, and MODNO is the logical module number.

Error Procedure. The disk control subroutine will handle error conditions, unless the user adds another argument to any of the above calling sequences, specifying an error return. If this additional argument has a prefix of PZE, a diagnostic will be printed and control will be transferred to the specified location with an error code in the AC. If the error-return argument has a prefix MZE, the diagnostic will be suppressed but otherwise action will be the same as with PZE. Note that an HTR instruction may not follow a disk-routine call as it will be interpreted as an error-return parameter.

The following is a list of possible error conditions with the corresponding error codes:

| Error Condition | Error Code |
|---|---|
| Illegal calling sequence | PZE 1 |
| Too many active files | PZE 2 |
| User not in Master File Directory | PZE 3 |
| Available space on module exhausted | PZE 4 |
| File not found | PZE 5 |
| Allotted track quota exhausted | PZE 6 |

## Disk Editor Control Cards

Requests for off-line disk input and output and for other operations carried out at the time the disk is edited (loaded or dumped) are submitted to the Center Dispatching Area in the form of punched control cards. In the case of input, the input deck is submitted with the control card.

The general format for control cards is as follows: starting in card column 1, a variable number of fields, each of variable length not exceeding 6 characters; fields are separated by commas, and reading of the card is terminated by a blank. In general the fields are:

1. Control word
2. Problem number
3. Programmer number
4. Primary file name
5. Secondary (class) file name

The following control words are used:

Input: This allows a file to be input from cards. The control card has the general format with a sixth field specifying the mode of the file, as a digit:

0 = temporary
1 = permanent
2 = read only, class 1
3 = read only, class 2

The deck immediately following the control card is put on the disk as a single file. The input deck must be followed by a standard end-of-file card or a decimal card, blank in cols. 1-7, with *EOF* in columns 8-12. Column binary and decimal cards may be mixed in the input deck. If the card is binary, 28 words are read; if decimal 14 words. When the end of the file is encountered, the file is added to the user's file directory.

Files which have been dumped from the disk onto cards may be reloaded using this control card. These dumped cards have a special prefix, 7, which is recognized by the editor program and causes the original file to be recreated. Such cards are sequenced, and they must be in order and form a complete set when reloaded.

Edit: The control card has the same format as Input. This provides a means of loading a binary deck to form a core image of a program, which is then written as one record on the disk. The control card is followed by a column-binary record card for the deck; the record card has a prefix of 1 in the first word; the third word contains in the decrement the last address used by the program. The cards following must be absolute column-binary cards with correct check sums. The deck must be followed by an end-of-file card as described under Input.

Print: The control card has the general format. The specified file is printed off-line. The file is assumed to be 14 words per line; a blank word is inserted before each line to ensure single spacing. If each line is preceded by a line mark, variable-length lines will be printed. The line mark has 7's in prefix and decrement, and the address contains the number of words in the line to follow. The first character of the first word following each line mark must be the print control character.

Dpunch: This is the same as Print except that the files are punched off-line. Line-marked files are also recognized.

Bpunch: The control card has the general format. The file is punched off-line, 28 words per card. No provision is made for 7-9 punches or check sums, which are assumed to be included in the card image where required.

Delete: The control card has the general format, with an optional sixth field. The specified file is deleted from the user's file directory. If the word "CARDS" appears in the sixth field, the file will be dumped in the form of special cards. These are column-binary cards with a prefix of 7 in the first word of each card; instead of a loading address they have a sequence number. These cards may be reloaded using Input.

7punch: The control card has the general format. The file is dumped in the form of prefix-7 cards, as in Delete with the "CARDS" option. The file is not deleted.

Chmode: This allows the mode of a file to be changed. The control card has a sixth field specifying the new mode desired for the file (according to conventions specified under Input). If the new mode is temporary, the user track count is adjusted.

Copy: This control card causes a specified file assigned to one user to be copied for another user so that each has an individual copy. The general format is used to specify where the file is going and what it is to be called. Four additional fields specify the file that is to be copied: problem number, programmer number, and two file names. The mode of the file is unchanged.

Link: The format is similar to Copy. The specified file is not copied, but is made accessible to the designated second user. More than one link may be made to a given file. The linked file may be read by all of the users who have access to it, but it may be altered only by the primary user.

# CHAPTER 6

# CTSS LIBRARY

The library routines briefly described below are presently available to console users. More detailed descriptions of usage, restrictions, calling sequences, etc., may be found in short subroutine writeups for each routine,which may be obtained from the Computation Center. For the manner in which the CTSS library is made accessible see the description of the load command, Chapter 7.

## Elementary Function Routines

| | |
|---|---|
| ASIN, ACØS | Arc sine, arc cosine functions; floating-point argument. |
| ATAN, ATN | Arc tangent function; floating-point argument. |
| EXP | Exponential function $e^x$; floating-point argument. |
| EXP(1 | Computes $I^J$; fixed-point arguments. |
| EXP(2 | Computes $X^K$; X floating-point, K fixed-point. |
| EXP(3 | Computes $Y^Z$; floating-point arguments. |
| INDV, DPNV | Numerical solution of a system of $N^{th}$ order non-linear simultaneous ordinary differential equations. |
| LØG | Natural logarithm, floating-point argument. |
| RANNØ, SETU | Generates a floating-point random number between 0.0 and 1.0 with normal, rectangular distribution. |
| SIN, CØS | Sine, cosine functions; floating-point radian argument. |

|  |  |
|---|---|
| SQRT, SQR | Square root; floating-point argument. |
| TANH | Hyperbolic tangent; floating-point radian argument. |
| TAN, CØT | Tangent, cotangent functions; floating-point radian argument. |
| XSIMEQ, XDETRM | Solves matrix AX=B; computes value of determinant A |
| .01300 | Computes $Y^Z$; floating-point arguments (used by Mad). |
| .01301 | Computes $X^K$; X floating-point, K fixed-point (used by Mad). |
| .01311 | Computes $I^J$; fixed-point arguments (used by Mad). |

<u>Library Versions of Fortran Built-in Functions (used by Madtran)</u>

|  |  |
|---|---|
| INT | Truncation; floating-point argument. |
| XINT, XFIX | Fixed-point truncation; floating-point argument. |
| MØD | Remaindering; floating-point. |
| XMØD | Remaindering; fixed-point. |
| MAX0 | Maximum; fixed-point argument, floating-point function. |
| MAX1 | Maximum; floating-point argument, floating-point function. |
| XMAX0 | Maximum; fixed-point argument, fixed-point function. |
| XMAX1 | Maximum; floating-point argument, fixed-point function. |
| MIN0 | Minimum; fixed-point argument, floating-point function. |
| MIN1 | Minimum; floating-point argument, floating point function. |

|  |  |
|---|---|
| XMINO | Minimum; fixed-point argument, fixed-point function. |
| XMIN1 | Minimum; floating-point argument, fixed-point function. |
| SIGN | Transfer of sign; floating-point arguments. |
| XSIGN | Transfer of sign; fixed-point arguments. |
| DIM | Positive difference; floating-point arguments. |
| XDIM | Positive difference; fixed-point arguments. |

## Input-Output Routines

Using the CTSS library the user may read and write files on the disk. By subroutine calls he may: 1) use the disk-control subroutine entries directly as described in Chapter 5 (see also "Supervisor Entry Subroutine" below); 2) use a set of comparable routines suitable for Fortran and Mad programs, which automatically take care of format, buffering, and error conditions ("Reading and Writing the Disk," below); or 3) via Fortran or Mad Input-Output statements use routines which simulate tape usage on the disk ("Simulated Tape Usage"). The last method allows the convenience of the compiler I/O statements but with some loss of flexibility. For reading and writing the typewriter consoles the READ and PRINT statements should be used by Fortran and Mad programmers.

A separate special library will be available for users who wish to read and write tape in accordance with the attach command. When this library is used, Fortran and Mad I/O statements will reference tape I/O subroutines; the usual set of I/O subroutines will be available.

Reading and Writing the Disk. To create an output file on the disk the user specifies a file name consisting of two 6-character BCD names. Since the master disk-control routine uses core buffering (see Chapter 5), the user may, if he chooses, specify buffer space in his own program. Buffering may be either single or double, and for large-volume input or output there is a considerable increase in speed where double-buffering is provided for. Each buffer used for input to or output from a single file must be a block of 470 words; for double buffering two 470-word blocks must be provided. The user may specify two, one, or no buffer areas; if he specifies none, a single buffer will automatically be assigned for a maximum of three simultaneously active files.

In calling the library subprogram ASSIGN, the user specifies the name to be given to the output file, and buffer space as desired. The file is placed in active write status. By calling DWRITE he may now write variable-length BCD records into the specified file. Several files may be active simultaneously (up to five, if adequate buffer space is provided), so each call to DWRITE must give the appropriate file name among its arguments. Conversion to BCD is implicitly through (IØH) and is controlled by a format specified as another argument in the subroutine call. The format conforms to Fortran specifications (see CC-186-6). After the file name and the format, subsequent arguments in the call to DWRITE constitute the output list.

The length of an output record is determined by the format and the output list. Each record is preceded by a one-word record mark containing 1's in bits 0-17 and the number of words to follow in bits 18-35

Successive calls to DWRITE will cause successive records to be written in the specified file. The file remains in active write status. The only limit on the size of a file is implicit in the user's track quota.

To conclude writing, to close out the file, drop it from active status, and add it to his file directory, the user calls the subroutine FILE specifying the appropriate file name as argument. When this is done the buffers assigned to that file are freed, whether provided by the user's program or automatically assigned for him. If the user fails to call FILE, but calls EXIT at the end of his program, any files still active will automatically be closed out for him. If he calls neither FILE nor EXIT, output files which are still active will be lost.

To write a binary file, a call is given to ASSIGN just as for BCD files; calls to BWRITE, specifying the file name and the output list, cause binary output to be written. The output exactly follows the argument list, and there are no record marks.

To read a file from the disk, the user first calls the subroutine SEEK, which is essentially similar to ASSIGN but puts the specified file in active read status. The same conventions apply to buffering. To read BCD records, DREAD is used, with argument requirements similar to those of DWRITE. Records are read according to the format specifications. Two types of record format are allowed; in addition to the variable-length record described above, BCD files may be composed of fixed-length 14-word records without record marks (suitable for off-line card input or console input).

To close out a file in active read status, a call to ENDRD, essentially similar to FILE,is given.

Binary files may be read by BREAD, which is the converse of BWRITE.

If it is desired to write additional records on an existing (inactive) file, this may be done by a call to the library subroutine APPEND. The arguments provided should be the same as those for ASSIGN; APPEND reopens the specified file and subsequent calls to DWRITE or BWRITE will cause additional records to be written following those already in the file. The conventions for writing and closing out the file are identical to those followed when writing a new file.

If an end-of-file condition is encountered while reading a file, the subroutine EOFXIT is called. This routine prints a diagnostic on the user's console and calls EXIT. If other action is desired, the user may substitute his own version of EOFXIT.

To dump a continuous block of core onto the disk as a file, the user may call the library subroutine DSKDMP, specifying the name to be given to the file, the location of the first word to be dumped, and the number of words to be dumped. To load a continuous block of core from a file previously recorded on the disk, the routine DSKLOD is called, with a set of arguments similar to those for DSKDMP.

To delete a file from his directory, the user may call the library subroutine DELETE, specifying the name of the file.

To change the name of a file the subroutine RENAME is called, specifying the old file name and the new one. To change the mode of a file the subroutine CHMODE is called, specifying the file name and the desired new mode. Files of mode "read-only, class 2" (see Chapter 5) may not be changed in this manner, nor may they be renamed.

By way of illustration, a Mad call to DWRITE, for example, might be:

```
     EXECUTE DWRITE. (NAME1, FMT1, A(1)...A(N) )
```

where NAME1 and FMT1 would be specified by

```
     VECTOR VALUES NAME1 = $ FIRST  DATA$
     VECTOR VALUES FMT1 = $(5F6.3)$
```

In Fortran, where the VECTOR VALUES facility is not available, the arguments might be specified as follows:

```
     CALL DWRITE (12H FIRST  DATA,7H(5F6.3), A(1), A(2), etc.
```

but for complicated formats, many calls, etc., this method is tedious. Two routines are provided to simplify calls in Fortran; they are SETNAM and SETFMT. An example of a set of statements utilizing these could be:

```
     DIMENSION NAME1 (2), FMT1 (10), BUFR1(470), BUFR2(470)
     CALL SETNAM (NAME1, 12H FIRST  DATA)
     CALL SETFMT (FMT1, 7H(5F6.3) )
     CALL ASSIGN (NAME1, BUFR1, BUFR2)
     CALL DWRITE (NAME1, FMTI,            list            )
                              etc.
```

In addition to these subroutines, the master disk control subroutine entries may be used directly for input-output. Entries to the disk control subroutine may be made through the library (see below).

<u>Summary of Disk Input-Output Subroutines</u>

| | |
|---|---|
| SEEK | DSKDMP |
| DREAD | DSKLOD |
| BREAD | DELETE |
| ENDRD | SETFMT |
| ASSIGN | SETNAM |
| DWRITE | APPEND |
| BWRITE | EOFXIT |
| FILE | |

(FTB) and (BTB), implicitly called by SEEK and ASSIGN, provide a directory of active files and associated buffers.

Simulated Tape Usage. In order to allow the use of Fortran and Mad standard input-output statements a set of library routines has been provided for simulation of tape usage. The major restrictions on the pseudo-tape usage are:

1. only BCD routines are available;
2. at most three "tape units" may be active (i.e., "positioned" for reading or writing) at one time.

Legal Fortran statements are:

| | |
|---|---|
| READ | WRITE OUTPUT TAPE i |
| READ INPUT TAPE i | END FILE i |
| PUNCH | REWIND i |
| PRINT | BACKSPACE i |

Legal Mad statements are:

| | |
|---|---|
| READ FORMAT | REWIND TAPE i |
| READ BCD TAPE i | END OF FILE TAPE i |
| PUNCH FORMAT | BACKSPACE RECORD OF TAPE i |
| PRINT FORMAT | BACKSPACE FILE OF TAPE i |
| WRITE BCD TAPE i | |

Fap programs may call:

(STH), (SCH), (SPH)
(TSH) (CSH)
(IOH), (FIL), (RTN)
(BST) (EFT), (RWT)

Usage. For simplicity we will refer to Fortran statements, but Mad and Fap usage is essentially similar. The format of all the statements and subroutines listed above is unchanged; they may all be used in the traditional manner. For description of a record, see above "Reading and Writing the Disk."

READ will read a record from the user's typewriter console.

PRINT will type a record on the user's console.

The first appearance of WRITE OUTPUT TAPE i will automatically cause the assigning of a disk file to the user, the file to be named ".TAPE. i" and of permanent mode. If a file of this name already exists in the user's directory, a new one will not be created, but output will be appended to whatever is already there. A variable-length record will be written in accordance with the format specified, and subsequent output statements designating this same logical unit will add records to the file. Until either a REWIND or END FILE statement is encountered or the program terminates in EXIT, the file ".TAPE.     i" is in active status.

PUNCH is identical with WRITE OUTPUT TAPE 3.

The first appearance of READ INPUT TAPE i will cause the disk control routine to seek a disk file, assigned to the user, named ".TAPE.     i" and of permanent mode. If no such file is found, an error condition will result. If it is found, a fixed- or variable-length record will then be read and subsequent input statements designating the same logical unit will read subsequent records from the file. Until either a REWIND statement is encountered or the program terminates in EXIT, the file ".TAPE.     i" is in active status. If an end-of-file condition is encountered, EOFXIT will be called.

END FILE i will cause the file ".TAPE.     i" to be dropped from active write status. If the designated file is inactive or if it is in active read status, no action will result.

REWIND i will cause the file ".TAPE.     i" to be dropped from active read or write status. If the designated file is inactive, no action will result.

BACKSPACE i will have no effect, but a diagnostic will be printed at the console.

When EXIT is called, all files still in active status will be closed out. If the program terminates without calling EXIT, any files still in active status will be lost. END FILE and REWIND are not normally necessary, provided the program terminates in EXIT and no more than three logical units are used. If more are required, or if a file is written and the same file read later in the program, it will be necessary to use REWIND or END FILE to drop a unit out of active status.

Auxiliary Subprograms for Simulated Tape Usage.

| | |
|---|---|
| (IOH), (FIL), (RTN) | Handles transmission and conversion of BCD data according to List and Format specifications. Implicitly called by Fortran READ, PRINT, READ INPUT TAPE, and WRITE OUTPUT TAPE Statements, by the analogous Mad statements, and by all calls to DREAD AND DWRITE. |
| .READ, .READL, .TAPRD, .PRINT, .COMNT, .TAPWR, .PUNCH, .PNCHL | Mad BCD input-output routines, implicitly called by Mad I/O statements. |
| (SPH), (STH), (SCH), (TSH), (CSH) | Fortran BCD input-output routines, implicitly called by Fortran I/O statements. |
| (SLI) (SLO) | Provide list indexing for non-subscripted arrays in Fortran I/O statements; called implicitly. |
| (EFT) (RWT) (BST) | Tape manipulation routines, implicitly called by Fortran statements END FILE, REWIND, BACKSPACE. |
| .BSF, .BSR, .EFT, .RWT | Tape manipulation routines implicitly called by Mad statements REWIND, END OF FILE, BACKSPACE. |

General Utility Programs

| | |
|---|---|
| .SETUP | Sets up floating-point trap to (FPT). Automatically compiled into Mad and Fortran programs. |
| MOVIE) | Contains the names, initial locations, entry points, and transfer vector length of all subprograms loaded into core by the load and use commands. |

| | |
|---|---|
| STOMAP | Prints a storage map of all user subprograms loaded into core by the <u>load</u> and <u>use</u> commands. |
| (EXE) | Decodes errors encountered in Fortran and Mad input-output library subroutines. Types out reason for error, saves machine conditions and transfers to RECOUP. |
| RECOUP | Provides a skeleton subprogram to which (EXE) transfers after an input-output error; calls EXIT, with no restart permitted. The user may write his own RECOUP to allow for recovery procedures. |
| (FPT) | Processes underflow and overflow in execution of floating-point operations. Underflows are set to zero, while overflows halt execution by a call to ERROR. |
| ERROR | Called by some math library subroutines in case of error. Prints diagnostic and trace of logical path of program flow where standard error procedure is provided. After printing, control returns to calling program. |
| SAVMC | Saves user's machine conditions in a buffer provided by the calling program. |
| RSTMC | Restores user's machine conditions from a specified buffer, previously filled by SAVMC. |
| LDUMP | Called by some math library subroutines in case of error. The library version of LDUMP calls EXIT. The user may write his own LDUMP to allow for recovery procedures. |

|  |  |
|---|---|
| EXIT, CLKOUT, ENDJOB, DUMP, PDUMP | Closes out any active disk files. Preserves machine conditions and goes to the supervisor in "dormant" status. If a start command follows, execution will be resumed at the location following the call to EXIT. |
| EXITM | Transfers control to the supervisor in "dead" status. Upon issuance of a new command, any active files not closed out prior to calling EXITM will be deleted. |
| XLOC | In compiled programs, finds the location where a variable is stored. |
| .03310, .03311 | In Mad, used to compute linear subscripts for two-dimensional arrays. |
| .MTX | In Mad, used to compute linear subscripts for arrays of more than two dimensions. |
| LISTM | For Mad list manipulation. |
| IOSET | Calls to IOSET are compiled into the object program during Madtran translations of Fortran input-output statements involving iterations. |

## Supervisor Entry Subroutine

The following subroutine from the library may be used in place of direct calls to the supervisor. (e.g., the Fap instruction TSX $DEAD,4 will effect a supervisor call to DEAD.) The entry names of this subprogram are:

|  |  |
|---|---|
| DEAD | .WRITE |
| DORMNT | .LOAD |
| .RESET | .DUMP |
| .RENAM | .APEND |

| | |
|---|---|
| .DLETE | SETFUL |
| .ENDRD | SETBCD |
| .FILE | WRFLX |
| .SEEK | WRFLXA |
| .ASIGN | RDFLX |
| .READK | RDFLXA |

There is no RDFLX in the supervisor. The routine RDFLXA reads a line from the console into the buffer specified along with the break character (usually carriage return); blanks are filled in to the right of the break character. The address of the AC contains N where the break character is the Nth character. The library routine RDFLX by using RDFLXA reads a line into the buffer specified with the break character stripped and all subsequent characters filled out with blanks. For both RDFLX and RDFLXA the maximum buffer length is 14 words.

# CHAPTER 7

# CONSOLE COMMANDS

The following index refers to brief summaries which are given of the presently available commands. The command programs normally enter from the disk but in certain instances (e.g. start) the command resides in the supervisor section of core memory. A block count is given to indicate the length of the commands. Each block is $256_{10}$ words long (128 blocks = 32,768 words), the increment for the protection and relocation registers. If a command is contained in the supervisor the block count is given as zero.

login $\alpha$ $\beta$

block count = 0

$\alpha$ = user problem number

$\beta$ = user programmer number

Should be given at beginning of each user's session at a console. Clears time accounting records and logs out any previous user of the console. Prints out the contents of the supervisor message file, if any, deleting the file after printing.

In the future an additional parameter may be required in order to afford greater security for the user. This will probably be in the form of a private code given separately; explicit instructions will be given by the login command if necessary.

logout

block count = 0

Should be given at end of each user's console session. Copies user file directory to disk; prepares usage accounting records for the system.

$ $\alpha$

block count = 0

$\alpha$ = arbitrary text treated as a comment.

listf $\alpha$ $\beta$ $\gamma$

block count = 4

a) If $\alpha$, $\beta$, $\gamma$ omitted, types out, in reverse chronological sequence, the list of all file names in the user's file with date-last-used, number of tracks, and file mode.

b) If $\alpha$ = "rev," and $\beta$, $\gamma$ omitted, same as a) but in chronological sequence.

c) If $\alpha$ = file name, $\beta$ = file class name, $\gamma$ omitted, types out a summary of information concerning the single file.

d) If $\alpha$ = numeric month, $\beta$ = numeric day, $\gamma$ = last 2 digits of numeric year, same as a) but only those files filed on or before given date.

## <u>input</u>

block count = 1

Initiates an automatic mode of input. The supervisor types out line numbers which will be attached to the lines input by the user. The user types a card image per line, according to a format appropriate to the programming language (see <u>file</u> command). Each line is processed by the input program. When in the automatic mode, a manual mode may be entered by giving an initial carriage return (for the 1014 Selectric console, the sequence: inquiry request, "$\pi$," inquiry release). In manual mode the supervisor types back the signal "MAN." instead of a line number. The following conventions may be followed when in manual mode:

a) A line number (all numeric) followed by space or tab, followed by the desired line: this allows insertion and correction of lines (cf. <u>file</u> command). If the line number is followed by tab, the first field is blank (cf. <u>file</u>).

b) DELETE, $n_1$, $n_2$ where $n_1$ and $n_2$ are previous line numbers: the lines from $n_1$ through $n_2$ will be deleted when the <u>file</u> command is issued; if $n_2$ is omitted, only $n_1$ will be deleted.

c) Initial carriage return or the corresponding 1014 sequence given above: the automatic mode will be resumed.

d) SEQUENCE, $n_1$, $n_2$ where $n_1$ is a line number and $n_2$ is a line-number increment: the automatic mode is resumed starting at $n_1$, with subsequent numbers incremented by $n_2$. If $n_2$ is omitted the normal increment of 10 is retained.

e) The command <u>file</u> $\alpha$ $\beta$: terminates the input mode and initiates the <u>file</u> command.

If it is desired to leave the input mode without filing the input lines, the normal quit signal is given; input lines will be lost.

<u>edit</u> $\alpha$ $\beta$

block count = 1

$\alpha$ = title of file
$\beta$ = class of file

The user is set in the automatic input mode with the designated file treated as initial input . The same conventions apply as to the input command.

<u>file</u> $\alpha$ $\beta$

block count = 7

$\alpha$ = title to be given to file
$\beta$ = class of language used during input

The created disk file will consist of the numbered input lines in <u>sequence</u>; in the case of duplicate line numbers, the last version will be used. The line numbers will be written as right-adjusted sequence numbers in the corresponding card images of the file.

For convenience the following editing conventions apply to input lines:

a) a delete-message signal signifies the deletion of the line;

b) a delete-character signal signifies the deletion of the previous character in the line.

The following formats apply:

a) fap: symbol, tab, operation, tab, variable field and comment.

b) mad, madtrn: statement label, tab, statement. To place a character in the continuation column: statement label, tab, logical backspace, character, statement.

c) data: 72 characters.

If a file $\alpha$, $\beta$ already exists in the user's file directory this older file will be deleted and replaced by the $\alpha$, $\beta$ just created; a message is given to this effect.

<u>printf</u> $\alpha$ $\beta$ $\gamma$

block count = 5

Types out file $\alpha$, $\beta$ starting at line number $\gamma$. If $\gamma$ is omitted, the initial line is assumed. If $\gamma$ does not match any line number in the file, printing commences at the first line number greater than $\gamma$. Even though the identification field of a card image contains alphabetic characters, $\gamma$ represents only the numeric portion.

If $\alpha$, $\beta$ is not in card-image form but written in the variable-length format, no line numbers will be printed. <u>Printf</u> will, if necessary, split a line which is too long for the console carriage.

<u>fap</u> $\alpha$

block count = 61

Causes the file designated as $\alpha$,fap to be translated by the Fap translator (assembler). Files $\alpha$,symtb and $\alpha$,bss are added to the user's private files giving the symbol table and the relocatable binary form of the file, respectively.

If the user's file directory already includes files $\alpha$,symtb and $\alpha$,bss these older files will be deleted and replaced by those created by this assembly.

Reference: IBM Fap Reference Manual, Form J28-6098-1 __mad__ $\alpha$

block count = 43

Causes file $\alpha$, mad to be translated by the Mad translator (compiler). Files $\alpha$,bss and $\alpha$,madtab are created giving the relocatable binary program file and the translation summary file, respectively.

If the user's file directory already includes files $\alpha$,bss and $\alpha$,madtab these older files will be deleted and replaced by those created by this compilation.

| | |
|---|---|
| References: | Mad Reference Manual |
| | Abbreviated Description, (forthcoming CC memo) |
| | I/O Format Specifications, CC-186-6 |

__madtrn__ $\alpha$

block count = 59

Causes file $\alpha$,madtrn (i.e., a pseudo-Fortran II language file) to be edited into an equivalent file $\alpha$,mad (added to the user's files); translation of this file then occurs automatically as if the command __mad__ $\alpha$ had been given.

If the user's file directory already includes file $\alpha$,mad this older file will be deleted and replaced by the file created by this translation.

| | |
|---|---|
| Reference on Madtran: | CC-188-1 |
| Reference on Fortran: | IBM Reference Manual, form C-28-6054-2 |
| | Abbreviated Description, CC-164-5 |
| | I/O Format Specifications, CC-186-6 |

__load__ $\alpha_1$ $\alpha_2$ ... $\alpha_n$

block count = 9

Causes the consecutive loading of files $\alpha_i$,bss. An exception occurs if $\alpha_i$ = (libe), in which case $\alpha_{i+1}$,bss is searched as a library file for all subprograms still missing. (There can be further library files.) If after all $\alpha_i$ have been processed there are still missing subprograms, the supervisor library file will be used in an attempt to complete loading. The only exception occurs if $\alpha_i$ = (nlib) in which case the supervisor library is not used at the end of loading; if a subsequent $\alpha_i$ = (lib), the normal case is restored. If, after the search of the supervisor library (where not suppressed), there are still missing subprograms, a message will be typed of the form:

need $p_1 \ldots p_k$

which may be followed by the use command.

use $\alpha_1\ \alpha_2 \ldots \alpha_n$

block count = same as load

This command is used whenever a load or previous use command notifies the user of an incomplete set of subprograms. Same $\alpha_i$ conventions as for load.

start

block count = 0

Starts the program set up by the load and use commands or restarts a dormant program at the location just after the point where the program entered the supervisor.

save $\alpha$

block count = 0

Creates file "$\alpha$,saved," consisting of the complete state of the user's last dormant program.

resume $\alpha$ $p_1 \cdots p_n$

block count = 0

File "$\alpha$,saved" is restored as the user's program and is restarted where it last left off. The parameters $p_i$ are entered into the user's command directory, which now contains: $\alpha$ $p_1 \cdots p_n$ (i.e., $\alpha$ replaces resume and all arguments are shifted correspondingly).

pm $\alpha$

block count = 0

Produces post-mortem of user's last dormant program (loaded by the load command) according to the request specified by $\alpha$. The following requests are permitted:

| | | |
|---|---|---|
| a) | pm ilc. | Gives the stop location or ILC (1 line). |
| b) | pm traps. | Gives contents of locations 0,2 and 8 (1 line). |
| c) | pm lights | Gives machine conditions and ILC (4 lines). |
| d) | pm stop | Gives ILC and contents of two locations on either side of the stop (5 lines). |
| e) | pm auto | Corresponds to "lights" plus "stop" (9 lines). |
| f) | pm stomap | Gives the BSS loading table, with origin and entry of all subprograms loaded. |
| g) | pm name | Gives contents of the four initial locations of subprogram "name" (5 lines). |

h) pm name $loc_1$ $loc_2$ mode direction

Gives contents of all locations from relative location $loc_1$ through $loc_2$ of subprogram "name," in the given mode and in the given direction. "Name" is (MAIN) for the main program. "$Loc_1$" is assumed to be decimal; if the number is preceded by a slash, "/," it is taken as octal. "Mode" specifies the form of printed output, and may be: fix, flo, dec, oct, bcd, or all. "Direction" specifies the order of printing over the range ($loc_1$, $loc_2$), and may be fwd or rev. If mode is omitted, all is assumed; if direction is omitted, fwd is assumed. $Loc_1$ and $loc_2$ may be replaced by the single argument entire to cause printing of the entire subprogram.

i) pm $loc_1$ $loc_2$ mode direction

Gives contents of locations from absolute location $loc_1$ through $loc_2$. This is normally used for post-mortem of the common region.

Reference: CC-167-6

patch $\alpha$

block count = 0

$\alpha$ = name of user subprogram loaded by load command

Sets up a mode for entering patches to relative locations within $\alpha$. If $\alpha$ is omitted, (MAIN) is assumed. In addition, three special patching modes may be used, i.e.:

a) $\alpha$ = (abs) allows patches to absolute locations;

b) $\alpha$ = (com) allows patches to relative locations in Common storage;

c) $\alpha$ = (pat) allows patches to be entered into locations above the user's current memory bound; this "patch space" is referenced by relative locations and is shared by all subprograms.

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

$\beta,\gamma,\delta,\epsilon$

where:

$\beta$ = octal address to be patched. This octal number may be immediately followed by a letter A, denoting an absolute location, C, denoting a location in Common, or P, denoting a location in the patch space.

$\gamma$ = type of field which follows, i.e.:

a) oct, octal word (used for instructions)

b) flo, fixed on floating-point number (E or F notation)

c) int, Fortran integer

d) dec, Mad integer

$\delta$ = number to be patched into $\beta$, according to format $\gamma$.

$\epsilon$ = relocation bits if $\gamma$ = oct; two alphabetic characters, the first for the decrement and the second for the address. The characters are:

A: absolute

R: relocatable

C: common

P: patch space

If $\epsilon$ is omitted, AR is assumed.

Successive $\delta$ fields may be specified in any line, with the following $\epsilon$ fields where necessary.

Exit from the patch command is by the quit signal.

tra $\alpha$ $\beta$

block count = 0

Where $\alpha$ is the name of a relocatable program loaded by the load command and $\beta$ is a relative octal location in $\alpha$, this command causes the setting up of a transfer to $\beta$. The transfer will be executed upon issuance of the start command. If $\alpha$ is omitted, the main program is assumed.

stopat $\alpha$ $\beta$

block count = 0

Parameters $\alpha$ $\beta$ as in tra. The instruction at $\beta$ is replaced by a transfer; when this transfer is executed the original contents of $\beta$ will be restored and the program will be placed in dormant status. The start command may then be used to cause execution of the original $\beta$ in $\alpha$. If $\alpha$ is omitted, the main program is assumed.

rename $\alpha$ $\beta$ $\gamma$ $\delta$

block count = 1

Changes the name of file $\alpha,\beta$ to $\gamma,\delta$. The mode of the file is unchanged. If $\delta$ is omitted, class $\beta$ is preserved.

chmode $\alpha$ $\beta$ m

block count = same as rename

Changes the mode of file $\alpha$, $\beta$ to that specified by m. The mode may be given as:

| | | | |
|---|---|---|---|
| T | or | 0 | : temporary |
| P | or | 1 | : permanent |
| R1 | or | 2 | : read-only class 1 |
| R2 | or | 3 | : read-only class 2 |

A file in class R2 may not be altered by this command, nor by the rename command.

delete $\alpha_1$ $\beta_1$ ... $\alpha_n$ $\beta_n$

block count = same as rename

Deletes files $\alpha_i$, $\beta_i$ from user's file directory. Files of class R1 or R2 may not be deleted by this command. A file of class R1 may be deleted only after its mode has been changed by chmode.

split $\alpha$ $\beta$ $b_1$ $s_1$ ... $s_{n-1}$ $b_n$

block count = 9

Splits the file $\alpha,\beta$ into n new files $b_1$, $\beta$ ... $b_n$,$\beta$. Splitting of $\alpha,\beta$ is done after the record sequence numbers $s_1 \cdots s_{n-1}$, which should be in ascending numeric order. The new files are appended to the user's file directory without resequencing. If any $b_i$ is "*," the corresponding file is not added to the user's files. This provides an easy way to extract subfiles from long master files. If $b_n$ is to be "*" it may be omitted.

If any $s_i$ is "*," or cannot be matched with a sequence number in the remaining portion of the original file, this is an error; the remainder of file $\alpha,\beta$ is included with the last $b_i$ processed, and an error comment is given. If two of the $s_i$ are the same, only the first file is created. Matching of sequence numbers is performed on the numeric portion only.

If file $\alpha,\beta$ cannot be found, the "need-use" convention is followed as in the <u>load</u> command.

<u>combin</u> s $\alpha$ $\beta$ $\gamma_1 \cdots \gamma_n$

block count = 9

Combines the files $\gamma_i,\beta$ into a single file $\alpha,\beta$. The new file is resequenced, starting with sequence number <u>s</u> x 10 and with numbers incremented by 10. The file is then added to the user's file directory. If <u>s</u> (a decimal number from 1-5 digits) is given as "*," no resequencing is performed. If any of the $\gamma_i$ cannot be found, a list of those missing is typed out and the "use" convention is followed as in the <u>load</u> and <u>use</u> commands. No new file is created unless the command is completed. If a "use" reentry to the <u>combin</u> command specifies a file name as "*," it is as if the original file name did not appear.

<u>rqu</u>est c $\alpha$ $\beta$ ...

Permits creation of control cards for requests to the disk editor. A file of special name and mode is created, consisting of control cards for those functions specified. At disk-edit time, this file will be read by the editor which will process the requests and then delete the file. If requests require special handling, the editor program will not service these but will create control cards which will be submitted to the Dispatcher. Requests will have the following parameters:

c = control word, e.g. Dpunch

$\alpha,\beta$ = file name

Subsequent parameters where needed, as:

rquest delete $\alpha$ $\beta$ cards

To request that a file $\alpha,\beta$ be restored from the history tapes next time the disk is loaded, the command has the form:

rquest reload $\alpha$ $\beta$

and the <u>rquest</u> command will provide the necessary information concerning the pertinent history tape.

The file created by this command (or added to by subsequent issuance of the command) has the name:

REQUES T.FILE

and is added to the user file directory. If the user wishes to change this file he must delete it and then reissue the desired commands.

<u>cpu</u> $\alpha_1 \ldots \alpha_n$

block count = 0

Gives user's current machine conaitions as requested by $\alpha_i$. As many of the following arguments may be given as desired, and in any order:

| | |
|---|---|
| AC | entire contents of AC |
| MQ | contents of MQ |
| SI | contents of sense indicators |
| IRS | contents of index registers |
| SLTS. | contents of sense lights |

The requests are serviced one per line in the order given in the command. This command may be used during execution of absolute programs, command programs, etc. In the case of programs loaded by the <u>load</u> command, it gives the same information as <u>pm</u>.

<u>octlk</u> m n

block count = 0

Prints out contents of <u>n</u> locations starting at absolute location <u>m</u>. If <u>n</u> exceeds the size, <u>s</u>, of the user's output buffer (internal to the supervisor) only the first <u>s</u> locations will be printed. The size of <u>s</u> is not fixed, but as a reasonable estimate <u>n</u> should not exceed 12.

<u>octpat</u> m $a_1$ $b_1$ ... $a_n$ $b_n$

block count = 0

Permits absolute patching into absolute locations beginning at <u>m</u>; $a_i$ and $b_i$ are octal left and right half-words, respectively.

<u>brief</u> $\alpha$

block count = 0

If $\alpha$ is omitted, the brief mode of console input is entered. This mode may be reset by repeating the command with $\alpha$ = OFF.

<u>confrm</u> $\alpha$

block count = 0

If $\alpha$ is omitted, the confirmation mode of console input is entered. This mode may be reset by repeating the command with $\alpha$ = OFF.

<u>time</u> $\alpha$

block count = 0

Causes the standard request for time-used messages to be changed from "TIME" to the set of characters specified by $\alpha$; $\alpha$ must be from 1 - 6 printing characters. Time messages may then be obtained by the user's typing an input message of $\alpha$ followed by a carriage return. If $\alpha$ = OFF, the time-used facility will be suspended until another time command is given or until the user gives a logout command.

chball $\alpha$

Allows for changing the type ball on the IBM 1014 or 1050 console. The designation of the desired ball is given by $\alpha$ (designations will be published later). The ball must be changed after "READY." is given.

memo $\alpha$
modify $\alpha$ $\beta$
ditto $\alpha$ p

block count = 37

Use of memo generates a file $\alpha$,memo; use of modify brings a user file $\alpha$,memo into core for modification and refiling as $\beta$,memo; if $\beta$ is omitted, the old $\alpha$,memo will be deleted and replaced by the modified version. Use of ditto generates a memorandum from file $\alpha$,memo, beginning with page p; if p is omitted, the entire memorandum is produced; use of the p parameter provides a restart procedure.

Memo, ditto, and modify are used with the typewriter to produce a memorandum. Textual information can be written, edited and manipulated by use of various control words.

The method of entering the textual matter is similar to the method used by the input command. The memo program types a line number on the typewriter. The programmer then types the lines of text which he wishes to enter and strikes a carriage return. Memo will then type the next line number. If the programmer strikes an initial carriage return before the line of text would have been given, he enters the manual mode. This allows him to type his own line number followed by the text to be entered in that line. An initial carriage return ( or "π" for the IBM 1014 or 1050) instead of a line number effects the re-entry to automatic typing of sequential line numbers. While operating in the manual mode, the programmer may cause a previous line to be replaced or a new line to be inserted between previous lines, depending upon the line number which was typed. In particular, control word lines may be inserted among text lines.

The modify command, which is essentially a restart of a previous memo command, brings the requested file into core memory. Line numbers are then typed sequentially as described in memo, except that the first number given is the line that closed the previous memo. Typed control words and line numbers are then accepted as in memo. If no β designation was given, the file α,memo will be replaced by the modified file. This new file will now be α,memo.

The ditto command loads the requested file into core memory, strips the line numbers, and types a copy of the memo divided into pages. This process is guided and controlled by the control words interspersed in the text. The control words which may be used to alter the format or facilitate correction are listed below; for ease in entering control words there is a set of abbreviated control signals which may be used as desired; these are listed also. A control word is recognized by a leading period.

| | |
|---|---|
| .EDIT FROM LINE XX | .ED XX |
| .END EDIT | .EE |
| .END MEMO | .EM |
| .DELETE LINE XX | .DE XX |
| .RESEQUENCE LINE NUMBERS | .RE |
| .COMMENT | .CO |
| .END COMMENT | .EC |
| .HEADER YYY... | .HE YYY... |
| .BEGIN PAGE | .BE |
| .FOOTNOTE IN LINE XX AFTER\|YYY...\| | .FO XX\|YYY...\| |
| .DOUBLE SPACE | .DO |
| .SINGLE SPACE | .SI |
| .SPACE XX | .SP XX |
| .CHANGE TYPE BALL TO XX | .CH XX |
| .END FOOTNOTE | .EF |
| .EDIT PREPRINT FROM LINE XX TO LINE YY | .ED PREPRI XX YY |

a) .HEADER YYY...

The 48 characters YYY... will be printed at the top of each page of the memo. The statement "Page xx of yy" will automatically be inserted to complete this header (or title) line appropriately for each page. Pages will not be numbered unless this control word is used. Normally this control word line should only occur once in a memorandum.

b) .EDIT FROM LINE XX

This control word line is not entered into the memorandum file. Memo and modify automatically switch to printing the line numbers which have previously been entered, instead of the consecutively incremented line numbers. This printing starts with line XX. The programmer can thus replace a series of lines without worrying about omitting any. The sequence will be interrupted by a new .EDIT FROM LINE XX control word or by the .END EDIT control word.

c) .END EDIT

This control word line, which is not entered into the memo file, causes the resumption of typing of new line numbers.

d) .END MEMO

The lines which have been entered will be filed in the user's file with the title given by the initial request for memo or modify.

e) .DELETE LINE XX

The line XX will be deleted, and no line will be entered into the memo file for this control word line.

f) .RESEQUENCE LINE NUMBERS

All the lines of the memorandum are assigned sequential line numbers starting with 000100 and incrementing by 100's. A copy of the text (with line numbers) is printed when this process is complete. This control word facilitates additional correction and editing of texts which may have had excessive corrections and insertions. The .END MEMO control word is then assumed and the memorandum is filed as explained under d).

g) .COMMENT

The lines of text which follow this control word line are considered to be notes or comments to the programmer. The .COMMENT control word line will not appear in the finished copy of the memorandum. The lines of notes following it will be printed on an initial page of the finished copy. These comments are extracted and printed at the beginning of the memorandum in spite of the fact that they had line sequence numbers inplying their position to be later. The signal to memo and modify that the comments are done is the .END COMMENT control word line. Other control word lines will not be recognized between .COMMENT and .END COMMENT.

h) .END COMMENT

This control word is the only means of indicating the end of a sequence of messages to the operator.

i) .BEGIN PAGE

The line following this control word will be positioned at the beginning of a page, (after a heading line, if present).

j) .FOOTNOTE IN LINE XX AFTER |YYY...|

The lines following this control word are considered the body of the footnote. The footnote is terminated by the .END FOOTNOTE control word or another .FOOTNOTE control word. The exact sequence of characters YYY... is found in line number XX and a footnote reference number is inserted in parentheses immediately following. Three or four blanks should be left in the correct position in line XX to permit this insertion without any overlapping. The body of the footnote will appear at the foot of the page containing line XX. Large footnotes may be extended to the next page.

k) .DOUBLE SPACE

A blank line will be inserted after each subsequent line of the memorandum.

l) .SINGLE SPACE

This control word causes a change in mode from double spacing to single spacing.

m) .SPACE XX

XX blank lines will be skipped before the next line of text is printed by the ditto program.

n) .CHANGE TYPE BALL TO XX

This control word, useful only for the IBM Selectric typewriter, implies the intention to print one or more characters of the preceding line using a different type ball. When the finished form of the memorandum is to be typed by <u>ditto</u>, a set of instructions to the operator will be printed on a preliminary page. A stop occurs after the affected line. The carriage should then be manually rotated back 2 lines. When the carriage return is struck, the printing of the line will continue assuming the type ball has been changed. It can be seen that blanks must be inserted carefully in the lines preceding the control word line and following the control word line to prevent over-printing.

When typing with a non-BCD character set extra care must be taken not to strike the period position on the keyboard at the beginning of a line unless a control word line is intended.

The memo may be continued using the new type ball until a change type ball control word is recognized.

o) .END FOOTNOTE

See the description of control word j).

p) .EDIT PREPRINT FROM LINE XX TO LINE YY

Lines XX to YY are printed after all pending editing, insertions. and deletions have been performed.

## <u>Additional commands</u>

The following commands have not yet been completely defined, although their function has been described elsewhere in the text. A more precise definition of their usage will be published later.

|  |  |  |
|---|---|---|
| a) | <u>attach</u> | see page 31; |
| b) | <u>releas</u> | a command ordering the release of a logical unit assigned by the attach command; |
| c) | <u>bkgrnd</u> | see page 33; |
| d) | <u>status</u> | see page 33; |
| e) | <u>retrve</u> | see page 29; |

f) a command requesting an estimate of how long a job will take (see page 34);

g) a command allowing the loading of absolute programs;

h) command versions of three programs which have been run as experimental user subroutines, described below.

## plot

The command plot is designed to give the user another means of communication with the computer via the plotters or scope. Once the user types plot, all further communication will be on a question-answer basis with the computer asking the questions.

The user types two functions X and Y of a dependent variable t which he wishes to see plotted over a closed interval for t, the maximum magnitude attained by X and Y for a scaling option, and the desired interval for t; otherwise, the ranges of X, Y, and t are assumed to be $-1 \le X, Y \le 1$ and $0 < t \le 1$. He also specifies a delta t, a constant, which controls the intervals at which the functions are evaluated (for a straight line approximation). Finally, he has the option of putting more than one plot on a single graph (perhaps for solving equations).

The notation for the functions is similar to a Fortran arithmetic statement, with the following exceptions:

1) mixed expressions are allowed;

2) a**b has been replaced by pow (a,b).

Symbols (first character always alphabetic) may be used to specify constants in these expressions; when their values are needed, they will be asked for.

The plot program was written by Jay Martinson as a special project in Course 6.68; a more complete description will be given in a forthcoming CC memo.

## fapdbg

This command will sent in operation a symbolic machine language debugging and control program, FAP DEBUG (similar to DDT, Flit, and others), which will have been loaded automatically by the load command. The program, in its currently available form, is capable of reading a SYMTB file and relocating the symbol values, in order to inform itself of the program symbols and thereby permit symbolic reference to memory locations. By typing requests to FAP DEBUG, the user can examine or change instructions or data, insert "break points" which will transfer back to FAP DEBUG when control reaches them, or begin execution of subprograms at any desired location. A full explanation of the present version of FAP DEBUG will be available in a forthcoming memo.

Before transferring control to a subprogram, FAP DEBUG will request the supervisor (by a call to SETBRK) to transfer control back to FAP DEBUG if a "quit" signal is sent. After a "quit" signal, FAP DEBUG will set up to continue, if requested where execution was interrupted, as it now does after a break point is encountered. Thus, even if the program gets into a loop, it will be possible to return to FAP DEBUG.

The FAP DEBUG program was written by Robert Campbell as a part of a bachelor's thesis.

sam

A symbolic algebraic manipulation program has been developed, which will be put into command form. This program consists of a set of operations that the user may initiate from his typewriter. These operations include: substitution of variables; numeric evaluation; algebraic simplification; algebraic solution of equations; and certain bookkeeping and error correcting aids for minimizing the typing of large expressions. Further information may be obtained from the bachelor's thesis entitled "An Algebraic Manipulation Program for Time-Shared Computer Consoles," by Stanley Dunten.

# APPENDIX A

## EXAMPLE OF A SESSION AT A CONSOLE

The following is a reproduction of a short time-sharing session at an IBM 1014 Selectric typewriter console. Ordinarily the user types in black and the computer responds in red (or vice versa); in this illustration the user types in lower case and the computer responds in upper case.

```
login m1416 1591
WAIT,
 M1416  1591 LOGGED IN   5/27 1112.9
READY.
listf 5 20 63
WAIT,
    10 FILES    20 TRACKS USED
DATE              NAME          MODE   NO. TRACKS
 5/20/63      MAIN      MAD     P          15
 5/17/63      DPFA   SYMTB      P           1
 5/17/63      DPFA      BSS     P           1
 5/17/63      DPFA      FAP     P           2
READY.
input
WAIT,
00010             entry    recoup
00020 recoup      tra      *+1
00030             cal      1,4
00040             sto      recoup
00050             trs      2,4
00060             end
00070 π
 MAN. 40          sta      recoup
 MAN. file subr fap
WAIT,
READY.
fap subr
WAIT,
 0    00005   .000 00 4 00002        TRS      2,4              00000050
00006  FIRST LOCATION NOT USED
FAILED
READY.
```

```
edit subr fap
WAIT,
00070 π
 MAN. 50          tra     2,4
 MAN. file subr fap
WAIT,
OLD FILE DELETED.
READY.
printf subr fap
WAIT,
00010        ENTRY   RECOUP
00020 RECOUP TRA     *+1
00030        CAL     1,4
00040        STA     RECOUP
00050        TRA     2,4
00060        END
READY.
fap subr
WAIT,
00006  FIRST LOCATION NOT USED
READY.
mad main
WAIT,
LENGTH 02076, T.V. SIZE 00020, ENTRY 00735
READY.
load main subr
WAIT,
 NEED      DPSUBR
READY.
use dpfa
WAIT,
READY.
start
WAIT,
 FILE   TEST  DATA NOT FOUND.
 NO ERROR RETURN SPECIFIED
READY.
pm lights
WAIT,
 PROG SEEK   STOP=   112 REL., 14273 ABS.  TSX  007400414161
 AC = 000014000000, S =0, Q =0 MQ = 000010000000 SI = 400004000000
 IX1 =      2 IX2 =    14 IX4 = 63505 SENSE LIGHTS ON       4
 FPT ON ,DCT OFF, ACOF OFF
READY.
save may27
WAIT,
READY.
```

```
listf
WAIT,
   16 FILES    66 TRACKS USED
DATE              NAME           MODE   NO. TRACKS
 5/27/63     MAY27   SAVED     P          31
 5/27/63      MAIN     BSS     P           5
 5/27/63      MAIN  MADTAB # QUIT,
READY.
logout
WAIT,
 M1416  1591 LOGGED OUT  5/27 1140.3
 TOTAL TIME USED=   01.6 MIN.
READY.
```

# APPENDIX B

## CURRENT RESTRICTIONS

The list given below specifies those features described in the text which at present writing are incompletely implemented. It was the intent in writing this book to present as complete as possible a picture of the scope of CTSS. During the programming of the system not all features, of course, can be considered of equal importance. The facilities listed here are being programmed as we go to press; users of CTSS can obtain from the Computation Center periodic notification of their availability.

7320 drum (to be added August 1963) - p. 3.
Time-used messages - p. 15 and p. 81.
Simulation of interval timer clock - p. 15.
Setting of arbitrary console break characters - p. 16.
Character mode (12-bit) switch - p. 17 and p. 43.
Brief and confirmation modes - p. 22 and p. 81.
Interconsole messages - p. 22.
Common files for programmers with same problem number - p. 24.
Automatic extension of track quota (available soon) - p. 25.
Generation from console of disk editor control cards (available soon) - p. 25-26 and p. 79.
History tape procedure - p.27-30 and p. 86.
Supervisor messages and automatic cutoff (available soon) - p. 30.
Attachment of input-output units - p. 31 and p. 86.
Dataphone attachment - p. 32.
Modification of command parameter list (available soon) - p. 33.
Console-inititated background - p. 33 and p. 86.
Estimation of completion time - p. 34
Link disk editor control function (available soon) - p. 54.
Library of subprograms for use of attached tapes - p. 57.
Library routines SAVMC and RSTMC (available soon) - p. 64.

Commands:

chball - p. 82.

cpu - p. 80.

memo, modify and ditto (available soon) - p. 82.

Macro facility in fap command - p. 72.

## REFERENCES

The references given here are not intended as a bibliography of literature in the field of time-sharing. Rather, they comprise a selection of documents which are direct predecessors of this book. The bulk of the reference material consists of internal technical memoranda which set down, sometimes in greater detail than is given in the text, the specifications for certain aspects of CTSS. These documents are for limited distribution only, and are listed here primarily for the benefit of those Center programmers directly involved with CTSS; it should be noted that in many cases the specifications, since they were preliminary ones, have become obsolete.

### Earlier Published Report on CTSS

"An Experimental Time-Sharing System," F. J. Corbató, M. Merwin Daggett, and R. C. Daley. Proceedings of the 1962 Spring Joint Computer Conference, AFIPS, 1962, 335-344.

### Reference Manuals for Source Languages

"FORTRAN Assembly Program (FAP) for the IBM 709/7090," bulletin, form J28-6098-1, July, 1961.

IBM 709/7090 Programming Systems: FORTRAN Assembly Program (FAP), form C28-6235, September 1962.

IBM Reference Manual 709/7090 FORTRAN, form C28-6054-2, January, 1961.

IBM 709/7090 Programming Systems: FORTRAN II Programming, form C28-6054-3, February, 1963.

The Michigan Algorithm Decoder, University of Michigan, September, 1961. (unpublished)

### Computation Center Memoranda (unpublished)

CC 164-5, "An Abbreviated Description of the Fortran Compiler Language," F. J. Corbató, September, 1960.

CC 167-6, "Description of a Post-Mortem Subprogram (F2PM) for Use with the Fortran-Fap Monitor System," L. Korn and F. J. Corbató, March, 1963.

CC 169, "Programming Details Concerning Use of the Direct Data Flexowriter," H. M. Teager, December, 1960.

CC 171-1, "Present Status of the MIT Time-Sharing System," M. D. Kudlick and H. M. Teager, December, 1961.

CC 172, "A Proposed System for Time-Sharing with two Flexowriters on the 709," M. L. Merwin and F. J. Corbató, January, 1961.

CC 177, "A Restricted, Potentially Time-Shareable Version of the FMS System," F. J. Corbató, M. L. Merwin, and R. C. Daley, June, 1961.

CC 179, "Description of the Modifications to the IBM 7090," F. J. Corbató, July, 1961.

CC 183, "General Description and Instructions for the Use of the Time-Sharing Console (TSC)," M. Kovarik and A. Rutchka, October, 1961.

CC 184, "Memory Protection and Relocation RPQ for the 7090," F. J. Corbató, November, 1961.

CC 185, "An Experimental Time-Sharing System for the 709 Computer," F. J. Corbató, M. L. Merwin, and R. C. Daley, November, 1961.

CC 186-6, "Fortran and Mad Format Specifications," J. Spall, May, 1963.

CC 188-1, "Madtran - A Fortran-to-Mad Language Translation," L. Korn, December, 1961.

CC 189, "Latest Writeup on the MIT RPQ for Relocation and Protection Modes," M. L. Merwin, December, 1961.

CC 192, "Time-Sharing System Notes," M. M. Daggett, February, 1962.

CC 193-1, "MITMR, an FMS Subprogram for Using the IBM Interval Timer Clock," M. M. Daggett, F. J. Corbató, and J. R. Steinberg, April, 1963.

CC 194, "Change of Input-Output Format; Correction of CC-171-1," A. Rutchka, April, 1962.

CC 196, "A Master Disk Control Subroutine," R. C. Daley and F. J. Corbató, July, 1962.

CC 199, "Abbreviated Instructions for the Use of the Mod 9 Time-Sharing System," F. J. Corbató, M. M. Daggett, and R. C. Daley, August, 1962.

CC 202-1, "Requirements for Background Systems Operating in the 7090/94 Compatible Time-Sharing System," R. J. Creasy, January, 1963.

CC 205, "Memo, Modify and Ditto Commands in the Compatible Time-Sharing System," M. J. L. Lowry, F. J. Corbató, and J. R. Steinberg, March, 1963.

CC 206, "Compatible Time-Sharing System Supervisor Console Input-Output: Interface Specifications," R. J. Creasy, R. C. Daley, F. J. Corbató, April, 1963.

CC 208, "LDEDT and DPEDT, the CTSS Disk Editors," R. H. Orenstein and R. C. Daley, May, 1963.

CC 209, "Use of the Memory Protection and Relocation Mode within the Mod. 10 Time-Sharing System," R. C. Daley and F. J. Corbató, May, 1963.

CC 210, "Conventions for Information Transfer over the High-Speed Channels on the IBM 7750 etc.," F. J. Corbató, R. J. Creasy, R. C. Daley, May, 1963.