The Compatible Time-Sharing System
A Programmer's Guide
The Compatible Time-Sharing System
A Programmer's Guide
SECOND EDITION
The M.I.T. Computation Center
P. A. Crisman, Editor

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

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

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

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

To the President of the United States:

This Commission has held its final session, and has submitted below its Final Report containing the conclusions of the Commission and its recommendations to the Congress. The Commission has fulfilled its duties to the nation and to the Congress.

The Commission was created by the President of the United States with the advice and consent of the Senate of the United States, under the authority of the Act of Congress approved March 3, 1977 (81 Stat. 72). The Commission's mission was to study and report on the role of the United States in the world, with particular emphasis on the environment, human rights, and international law.

The Commission's work has been marked by a commitment to the principle of international law and the rule of law, and by a respect for the diversity of cultures and societies around the world. The Commission has sought to promote a better understanding of the world, and to encourage dialogue and cooperation among nations.

The Commission's recommendations are based on a thorough analysis of the issues it has examined, and are intended to provide guidance for the United States government and for the international community. The Commission believes that the United States has a vital role to play in the world, and that it should continue to be a leader in promoting peace, prosperity, and human rights.

The Commission thanks all those who have contributed to its work, and wishes to express its appreciation for the support it has received from the Congress, the President, and the American people. The Commission looks forward to continuing its efforts to build a better world.
article in the January 1965 issue of the IEEE Spectrum.

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

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

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

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

Acknowledgements

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

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

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

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

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

F.J. Corbató
May 1965
Cambridge, Massachusetts
This handbook is an attempt to document the techniques of using a current version (model 13) of the compatible time-sharing-system (CTSS) which has been developed at the MIT Computation Center. It is primarily a manual of how to use the system, in contrast to many of the research memos, which have been more detailed in their documentation of the techniques of implementation. Because CTSS is basically a system which will allow an evolutionary development of time-sharing while continuing to allow more conventional background systems to operate, it is expected that the present manual will of necessity be revised many times before it reaches a final form. A good deal of the difficulty arises from, on the one hand, the rather drastic change in user operating techniques which time-sharing permits, and on the other hand the immense amount of programming required to fully implement the system.

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

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

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

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

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

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

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

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

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

F.J. Corbató
Cambridge, Massachusetts
May 1963
Reference to the report indicates: 1) supplementation of the previous reports, and 2) showing further steps to earn trust and access to economical activities. Regulations and the national policy have been developed to improve and strengthen the development of new industrial systems. We have been working on the establishment of new communication centers as a follow-up to the previous activities. The importance of developing new and modern systems such as telecommunication, computerized management, and comprehensive engineering tools has been acknowledged.
## TABLE OF CONTENTS

### AA. Introduction to Time Sharing
- 0 Introduction
- 1 General Description and usage techniques

### AB. Documentation
- 1 Conventions for this manual
- 2 Glossary and conventions
- 3 System documentation
- 4 Timely - Temporary Tips

### AC. Hardware
- 0 Equipment Configuration
- 1 Clocks
- 2 Consoles and character sets
  - 01 Character sets
  - 02 Meaning of some special characters
- 3 Dataphones

### AD. Files
- 1 Historic file system
- 2 A new look in the file system
- 3 Library files
- 4 Common files
- 5 Time accounting files

### AE. File Editing (off-line)
- 1 Card input and delayed output
- 2 Retrieval

### AF. Background
- 1 Restrictions for background programs

### AG. Subroutines
- 1 Console I/O
  - 01 RDFLX, WRLFLX
  - 02 SETFUL, SETBCD
  - 03 PRNTP
  - 04 WRMESS, RDMESS
    - ALLOW, FORBID
  - 05 ATTCON, RELEAS,
    - SLAVE, SET, SNDLIN,
    - REDLIN
  - 06 (CSH), .READ,
    - .READL,.LOOK,.SCRDS
  - 07 (SPH), (SPHM),
    - .PRINT,.COMNT,
    - .SPRNT
  - 08 .PCOMT
  - 09 .PRSLT,.PRBCD
    - .PROC

General I/O
- Character mode
- Fenced output
- Inter-program communication
- Slave remote consoles
- MAD, Fortran, on line input with format
- MAD, Fortran, on line output with format
- Print comment
- Print list without format
<table>
<thead>
<tr>
<th>Section</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>.2 Disk File I/O</td>
<td></td>
</tr>
<tr>
<td>.01 LOAD, DUMP, DSKLOD, DSKDMP</td>
<td>Unbuffered string I/O</td>
</tr>
<tr>
<td>.02 SEEK, READK, ENORD, B-D-VREAD</td>
<td>Input, buffered</td>
</tr>
<tr>
<td>.03 ASSIGN, WRITE, FILE, B-D-V-FWRITE, APPEND</td>
<td>Output, buffered</td>
</tr>
<tr>
<td>.04 RELRW</td>
<td>Relative read/write</td>
</tr>
<tr>
<td>.05 SETVBF</td>
<td>Set fixed record length</td>
</tr>
<tr>
<td>.06 SRCH, BLK, FLK, ENDF COUT</td>
<td>Library service</td>
</tr>
<tr>
<td>.07 CLEAR</td>
<td>Zero file</td>
</tr>
<tr>
<td>*.08 OPEN, BUFFER, RDFILE WRFILE, TRFILE, CLOSE, SETPRI</td>
<td>Input and output with new I/O system</td>
</tr>
<tr>
<td>.09 LDFIL</td>
<td>Load a file into free storage</td>
</tr>
<tr>
<td>.3 File Status</td>
<td></td>
</tr>
<tr>
<td>.01 CHMODE, RENAME</td>
<td>Change mode or name</td>
</tr>
<tr>
<td>.02 DELETE, ERASE</td>
<td>Delete or erase files</td>
</tr>
<tr>
<td>.03 CONFIL, TSSFIL, USRFIL</td>
<td>Switch file directories</td>
</tr>
<tr>
<td>.04 FSTAT, .FSSTAT</td>
<td>File status</td>
</tr>
<tr>
<td>.05 GTNAM</td>
<td>Get name of next file</td>
</tr>
<tr>
<td>.06 .RESET</td>
<td>Drop files from active</td>
</tr>
<tr>
<td>*.07 CHFILE, DELFIL, FSTATE, STORGE</td>
<td>Change mode or name, delete, find status</td>
</tr>
<tr>
<td>.4 Errors and Exits</td>
<td></td>
</tr>
<tr>
<td>.01 Historic File System Error Procedure</td>
<td></td>
</tr>
<tr>
<td>.02 SETERR, SNAP, RECOUP</td>
<td>Library Disk Error Procedure</td>
</tr>
<tr>
<td>.03 EOFXIT, SETEOF, WRDCNT</td>
<td>Library end-of-file procedure</td>
</tr>
<tr>
<td>.04 EXIT, CLKOUT ENDJOB, DUMP, PDUMP</td>
<td>Terminal procedure</td>
</tr>
<tr>
<td>.05 LDUMP</td>
<td>Error exit from Math routines</td>
</tr>
<tr>
<td>*.06 I/O System Error Procedures</td>
<td></td>
</tr>
<tr>
<td>.5 Tapes and Pseudo tapes</td>
<td></td>
</tr>
<tr>
<td>.01 .PUNCH, .PNCHL, .TAPWR, (SCH), (STH), (STHM)</td>
<td>Write BCD with format</td>
</tr>
<tr>
<td>.02 .TAPRD, (TSH), (TSHM)</td>
<td>Read BCD with format</td>
</tr>
<tr>
<td>.03 (STB), (TSB), (WLR), (RLR)</td>
<td>Read/write binary</td>
</tr>
<tr>
<td>.04 .BSF, .BSR, .EFT, .RWT (BST), (EFT), (RWT)</td>
<td>Backspace, rewind, write end of file</td>
</tr>
</tbody>
</table>
6 Program Status
   .01 DEAD, DORMNT
   GETILC, FNRTN
   .02 SLEEP
   .03 GETBRK, SETBRK, SAVBRK
   .04 STOMAP
   .05 .SETUP,(FPT), (EFTM), (LFTM)
   .06 GETMEM, SETMEM
   GMEM, SMEM, EXMEM
   .07 FREE, FRER, FRET
   *.08 TILOCK

   Dead and dormant status
   Alarm clock
   Interrupt level
   Storage map
   Floating point trap
   Memory allotment
   Free storage management
   File-wait return

7 Supervisor
   .01 GETLOC, SETLOC, SYPAR
   GLOC, SLOC
   .02 GETCF, GETCFN
   *.03 UPDMFD, DELMFD, ALLOT
   ATTNCH, MOVFIL, SETFIL, LINK

   Supervisor parameters
   Current commfil number
   Privileged I/O calls

8 Commands (Macro-Chain)
   .00 General discussion of chains
   .01 XECOM, NEXCOM, NCOM
   .02 SCHAIN
   .03 CHNCOM, (GET, G, SET,
   S) CLS and CLC
   .04 GETCOM, COMARG

   Single commands
   General MACRO command
   Supervisor chain control
   Fetch command argument

9 Debugging
   .01 ERROR

   Trace of subroutine calls

10 Conversion
   .01 BCDEC, BCOCFT
   .02 DEFB, DELB, DERBC
   .03 OCABC, OCDBC, OCLBC,
   OCRBC
   .04 B2EL, £EL, LJUST,
   RJUST
   .05 (10H), (RTN), (FIL)
   IOHSTZ, ISTQUO
   .06 PAKR, PAKL,
   UNPAKR, UNPAKL
   .07 FINT, MINT

   BCD or Spread-Octal to Binary
   Binary to BCD
   Binary to Spread Octal
   Justification and padding
   General purpose conversion
   Pack and unpack, single characters to full words.
   Fortran integers to/from full word integers.

11 Subroutine linkage processors
   .01 COLT, SELAR, MDL

   Variable-length calling sequence
   Type of calling program
   Move argument list
   .02 GNAM
   .03 MOVE1, MOVE2, MOVE3
<table>
<thead>
<tr>
<th>Section</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>.04</td>
<td>SETFMT, SETNAM</td>
</tr>
<tr>
<td>12 Timer</td>
<td>Current time and date</td>
</tr>
<tr>
<td>.01</td>
<td>GETTIME, GETTM</td>
</tr>
<tr>
<td>.02</td>
<td>TIMER, JOBTM, RSCLOCK, STOPCL, KILLTR, TIMLFT, RSTRTN</td>
</tr>
<tr>
<td>.03</td>
<td>CLOCON, CLOCOF</td>
</tr>
<tr>
<td>13 Miscellaneous</td>
<td>Simulation of</td>
</tr>
<tr>
<td>.00</td>
<td>List of miscellaneous library subroutines</td>
</tr>
</tbody>
</table>
AH. Commands

.1 Logging

.01 LOGIN

.02 LOGOUT, AUTOMATIC LOGOUT

.03 FIB

Log in
Log out
Foreground-Initiated-Background

.2 Languages & Subsystems

.01 AED

.02 BEFAP

.03 COGO-90

.04 COMIT

.05 DYNAMO

.06 ESL

.07 FAP

.08 GPSS

.09 LISP

.10 MAD

.11 MADTRN

.12 SNOBOL

.13 STRESS

.14 BLODI

ALGOL Extended for Design
Bell Laboratories 7094 Machine Code Assembler
Coordinate Geometry Language
String Processing Language
Model Simulation Language
ESL Display System
7094 Machine Code Assembler
General Purpose System Simulator
List Processing Language
Michigan Algorithm Decoder
Fortran II to MAD Translator
String Manipulation Language
Structural Engineering Language
Block Diagram Compiler

.3 File creation & editing

.01 INPUT, EDIT, FILE, TFILE

.02 ED

.03 SAVE, MYSAVE

.04 SAVFIL, RERUN

*.05 LINK, UNLINK, PERMIT, REVOKE

Line numbered files
Context edit line numbered files
Save dormant program
Save and restore files created by RUNCOM
Link to files in other U.F.D.'s

.4 File Compression

.01 ARCHIV

.02 CRUNCH

Archive seldom-used files
Compress BCD files
.5 File Printing
   .01 LISTF
   .02 PRINTF
   .03 PRINT
   .04 PRBIN
   .05 PRBSS
   .06 SDUMP

.6 File Housekeeping
   .01 COMBIN
   .02 SPLIT
   .03 CHMODE, RENAME,
     DELETE
   .04 COMFIL, COPY,
     UPDATE
   .05 EXTBSS, UPDBSS
   .06 RQUEST

.7 Program Execution
   .01 LOAD, LOADGO, VLOAD,
     NCLOAD, USE, MOVIE
   .02 LDABS
   .03 START, RESUME,
     RESTOR, RSTART, CONTIN,
     RECALL

.8 Debug
   .00 General discussion of debugging tools
   .01 FAPDBG
   .02 MADBUG
   .03 PM
   .04 PATCH, STOPAT, TRA
   .05 SPATCH
   .06 SD, SP
   .07 STRACE, TRACE

.9 Document
   .01 TYPSET, RUNOFF
   .02 LOG
   .03 REMARK

Section TABLE
Table of contents of file directory
Print BCD card image files
Print BCD file
Print any file in octal
Table of contents of BSS file
Print SAVED file
Combine files
Split files
Change mode or name, or delete files
Common file writing and copying
Library file housekeeping
Off-line processing
Relocatable Program
Absolute Program loading
Start or Continue
Symbolic FAP Language Debugger
MAD Language Debugger
Post Mortem dumping
Relocatable program patching
Absolute program debugging
Supervisor debugging tools
Subroutine trace
Memo editor and printer
Document modifications
Send message to system programmers
.04 INFO

On-Line system documentation program

.10 Commands
  .01 RUNCOM
  .02 GENCOM

MACRO commands
Command arguments in octal
Identification

Introduction to Time-Sharing

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

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

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

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

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

Introduction to Implementation

The implementation of a new computer system involves the integration of various components and processes. The purpose of this report is to outline the steps involved in the implementation process.

1. Preparation of the Environment
   - Installation of software and hardware
   - Configuration of the system

2. Data Migration
   - Transfer of existing data to the new system

3. User Training
   - Provision of training to end-users

4. Testing
   - System testing to ensure functionality

5. Deployment
   - Integration of the system into the organization

6. Documentation
   - Maintenance of system documentation

Implementation of these steps is crucial for the successful implementation of the new computer system.
decision-tree problems; real-time management
problems (airline reservations, hospital
administration, etc.); gaming problems;
sociological experiments; teaching machines;
language learning problems; library retrieval;
text editing; algebra manipulators; and many more.

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

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

Not only are the drum and disks used for swapping of active
user programs, but all console users utilize the disk memory
for semi-permanent storage of their active program and data
files. Cards and magnetic tapes still serve in secondary
roles as long-time and back-up storage devices.
The Community Development Program (CDP) is a comprehensive community empowerment program designed to address various social and economic issues within communities. The program focuses on several key areas including the provision of affordable housing, educational assistance, health services, vocational training, and community development projects. The goal is to enhance the quality of life for residents by improving infrastructure, providing job opportunities, and fostering social cohesion.
Identification

General Description and Usage Techniques

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

The Supervisor

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

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

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

In addition to maintaining input and output buffers for each user console, the supervisor keeps a record of the status of each user. The status of a user may be: "working", where a program is ready to continue running whenever it is next
The command plans for the first three months of the current year will be based on the plans of the Army Commanders and the command structure. The command plans will be developed in coordination with the Army Commanders and the command structure. The command plans will be developed in coordination with the Army Commanders and the command structure.

The command plans will be developed in coordination with the Army Commanders and the command structure.
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 a free output buffer; "file-wait," where the program is temporarily delayed until another user has finished using the requested program or data file; "FIB-wait," a very specialized status, used only by the FIB monitor where there is no FIB job waiting to be run; "dormant," where the program has stopped running and returned control to the supervisor, but machine conditions and the status of memory are preserved for inspection, modification, or re-entry; and "dead," where the program has terminated, control has been returned to the supervisor, and machine conditions and the status of memory have been scrapped.

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

Command Format

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

Command Initiation

At the time of the first clock trap following completion of a command line at a user's console, that user is placed in
process is referred to as "geography." Geography is the study of the relationships among the various features of the Earth's surface. It is concerned with the distribution of natural phenomena and human activities on the Earth. Geography can be divided into several sub-disciplines, including physical geography, human geography, and cultural geography. Physical geography deals with the study of the physical features of the Earth, such as climate, landforms, and vegetation. Human geography focuses on the study of human activities and their relationship to the physical environment. Cultural geography examines the ways in which human societies interact with the environment. Geography is a useful tool for understanding and predicting human behavior, natural resources, and environmental change.
waiting-command status. He is then set at the end of a scheduling queue which is chosen according to a rule assigning higher priority to shorter programs. When this user reaches the head of the highest-priority active queue, he will be placed into working status.

Program Termination

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

Input and Output Wait States

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

Scheduling

In order to optimize the response time to a user's command or program, the supervisor uses a multi-level scheduling algorithm. The basis of the algorithm is the assignment of each program as it enters working or waiting command status to an nth level priority queue. Programs are initially entered at a level which is a function of the program size (i.e., at present, programs of less than 4k words enter at level 2 and longer ones enter at level 3). There are currently 8 levels (0-7). The process starts with the supervisor operating the program which is first in the queue at the lowest occupied level, L. The program executes for a time limit = 2,P,L quanta; a quantum of time is one half second. If the program has not finished (left working status) by the end of the time limit, it is placed at the end of the next higher level queue. The program at the head of the lowest occupied level is then brought in. If a program P enters the system at a lower level than the program currently running, and if the current program P1 has run at least as long as P is allotted, then P1 will be returned to the head of its queue and P will be run. If a program changes its size, its new level is computed immediately. If the new level is different from the old, a
new maximum time is also computed and becomes effective retroactively.

There are several different time limits whose current values may be of interest to the users. If a data phone is dialed into the computer and the user does not log in within 2 minutes, there is an automatic hangup. If a program in working status waits one minute without being run, it is bumped to the next higher priority queue. If a user stays in any non-working status for one hour, he is automatically logged out. The clock burst which enables the supervisor to housekeep the console input and output and to change program status is currently set to 200 ms. The quantum of time used in the scheduling algorithm is one-half second.

Memory Protection and Relocation

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

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

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

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

User Communication with the Supervisor

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

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

```
TSX NAME1,4
...
...
NAME1 T1A =HNAME
```

where NAME is the BCD name of a legitimate supervisor entry point. The details of each supervisor entry are described in section AG. The T1A instruction is described in IBM manual L22-6636; it may usefully (but inexactly) be read as Trap Into A core.
Identification

Conventions of this manual

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

M is the manual designation. Since the CTSS Programmer's Guide for the IBM 7094 is the first manual in a series, its designation will be "A".

S is an alphabetic major section designation, e.g., this is section "B".

X is the one or two digit subsection designation. This first publication will have subsections numbered from 1 to 13. Note that they will not be designated as 01 to 13.

YY is the minor subsection designation. This is a two digit numeric designation (00, 01, 02,...)

The manual was prepared by the CTSS commands TYPSET and RUNOFF where each section is a separate file of the name MSXYY (MEMO). Note the deletion of periods within the file name.

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

The table of contents will be maintained in two forms.

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

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

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

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

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

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

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

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

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

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

Glossary and Conventions

Documentation Conventions

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

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

Glossary

* in front of an entry in the table of contents, indicates the new I/O system. An * in the right-hand margin, indicates a modification to the write-up.

AC 36-bit signed accumulator.

b denotes a required blank in a character string.

C.R. carriage return.

Console In general, the word console means a typewriter console (e.g., 1050, teletype) rather than a special display console (e.g., ESL scope).

Current File Directory is the file directory to which the user is currently switched. It is usually the user's file directory but may be switched to a common file directory by COMFILL or to another user's file directory by ATTACH.

External Routines are subprograms (with entry points) which are called by other subprograms. The library entries and library subroutines are external routines. The FAP calling sequences
CRICK MECHANISMS
NOTICE
Section 6.5

Disassembly Manual

Disassembly Procedure

1. With the engine warmed up, remove the engine from the frame. Disconnect power steering lines and electrical connections.
2. Remove the engine from the frame and place it on a clean, level surface.
3. Remove the engine mounts and the rear suspension.
4. Disconnect the battery and the alternator.
5. Remove the radiator and the exhaust system.
6. Remove the engine from the frame and place it on a clean, level surface.
7. Remove the engine mounts and the rear suspension.
8. Disconnect the battery and the alternator.
9. Remove the radiator and the exhaust system.
10. Remove the engine from the frame and place it on a clean, level surface.

Note: For further assistance, please refer to the service manual.
give the entry point name. The FAP convention for calling external routines is: 1) EXTERN pseudo-op specification, or 2) preceding the name by $, or 3) CALL pseudo-op. All the FAP calling sequences in this documentation assume EXTERN specification so that the CALL and $ are not shown.

Fence is a magic number used to designate the end of a variable-length string of parameters. The fence referred to in this documentation is a word of all octal sevens.

FILNAM is used in calling sequences to indicate the initial location of 2 BCD words containing the name of a disk file (right justified and blank padded). In Fortran programs, FILNAM may be set by the subroutine SETNAM or it may be the file name in H specification form. In MAD programs FILNAM may be set in a Vector Values statement.

FMT or FORMAT is used in calling sequences to indicate the beginning location of a format or a location containing a pointer to the beginning of the format, if SETFMT is used.

Library Entry - The majority of the required TIA's for the supervisor entries have been placed in the library as library entries.

Line-Marked Files are files composed of variable length records. Each logical record is preceded by a word containing binary ones in bit positions 0-17 and the number of words to follow in bits 18-35.

Line-Numbered Files are files composed of 14 word logical records. Characters 73-80 are a sequence field (the leftmost 3-6 may be alphabetic and the rightmost 2-5 must be numeric).

LIST is used in calling sequences to provide a list of parameters to the subroutine being called. It usually specifies parameters for input or output. A list may consist of a combination of single variables, dimensioned or subscripted variables, or block notation as described in the MAD manuals. In Fortran, the implied DO may be used only in I/O statements, not in calls to subroutines.

In MAD, a LIST might be: A, B(1) .. B(10), C(N) ... C(1), G(J). The notation D(N) ... N,
E(1) ... 10, is also available in MAD but currently is acceptable only to the new I/O system routines.

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

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

The FAP equivalent of the above MAD LIST is:

```
TXH A
TIX B-1,,B-10
TIX C-'n',,C-1
TXH G-'j'
TIX D-'n',,N
TIX E-1,,L(10)  i.e., location of a 10
```

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

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

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

With the current file system there are eight modes and the mode of a single file can be any combination of the eight.

```
000. PERMANENT
001. TEMPORARY
002. SECONDARY
004. READ-ONLY
010. WRITE-ONLY
020. PRIVATE
040. LINKABLE
100. PROTECTED
```
NAME1 NAME2 are used in calling sequences to indicate the actual name of a disk file. NAME2 is the secondary (class) name. The actual names are right adjusted, blank padded, BCD words.

String Files - files having no logical record breaks. Processed as strings of words by externally specified word counts.

Supervisor Entry - supervisor routines which reside in a core can be entered only by a special calling sequence convention.

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

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

Purpose
In order that the users may study the system and offer additions, modifications, or improvements, the listings, source decks, and table of contents of the system must be available. There are two different user requirements which dictate two different kinds of documentation: 1) for the user who wants a complete listing of everything or at least a large volume of what is in the system, 2) for the user who wishes to study or modify a relatively small section. To satisfy the first user, there are document tapes. To satisfy the second there are source language files available on-line which may be assembled and listed by the user at his console.

User Procedure
All source files will be CRUNCHed without sequence numbers and placed in COMFIL 1 of M1416. The system module BSS files will also be in COMFIL 1. The library BSS and command TSSDC. files will be in COMFIL 2 of M1416, which is accessible to the user as COMFIL S. The listings and source files of modules, commands, and library routines will be maintained on document tapes which will be updated at least once a month. Between updating times, the listings of all modified files will be kept in COMFIL 3 of M1416. Following the document tape update runs, these BCD listing files will be deleted from COMFIL 3. All of these files will be LINKABLE, READ-ONLY, AUTHOR protected files.

When a user modifies any component, the unmodified version (source and BSS) should be kept in his own file directory for a few days just as insurance since only the current version is maintained in the common files. All modifications should be documented to the extent of adding remarks to the front of the source file giving date, author, and purpose of modification.

PUBLIC COMMANDS:

When a new command is created, the SAVED file should be named 'COMMAND TSSDC.' The (MOVIE TABLE) file which is created by the loader should be named COMMAND MOVIE) of permanent mode. The movie table will provide a table of contents and storage map of the command saved file. All of the source files used to create the BSS files which were loaded (except the library) should be CRUNCHed without sequence numbers and ARCHIVEd into a file COMMAND CRUNCH. All of the BCD listing files which
should have been created during the translation process should be ARCHIVEd into a file COMAND BCD. The author may then submit a write-up to the manual editor and ask that the command be placed in the system as a public command. The responsible system programmer will then copy the above mentioned files into the common files and place the command in the system. If any of the required files is missing, the author will be notified and the command will not be entered in the system.

AUTHOR MAINTAINED COMMANDS:

The author may submit a write-up to the manual editor and ask that the command be placed in the system as a semi-public command. The documentation files are not required but it is strongly recommended that they be supplied as a courtesy. The documentation files may be author protected by the author rather than the system programmer so that the author may perform maintenance directly. The document files may either be copied into common files or linked to in the author's file directory from the common files (to be decided by the system programmer as a function of size and track quotas). The manual write-up will direct all criticism to the author and the users are protected only by the integrity of the author. Users' remarks may be placed in the author's MAIL BOX and any documentation may be placed in the editor's MAIL BOX.

LIBRARY:

Any user may modify or submit a new library routine, but the mechanics of updating the library may be performed by only one or two people. Short library routines will be grouped into families in order to optimize track usage. A table of contents listing family names with the components' file names and entry points will be maintained in COMFIL 1 of M1416 as Lib INFO which may be PRLNTFed. Each family will have a CRUNCHed ARCHIVEd source file in COMFIL 1 called FAMILY CRUNCH. Any single large routines will have a CRUNCHed source file called ROUTIN CRUNCH. The USS files will be CRUNCHed into the library file TSLIBN USS. The listings will be maintained on document tapes and recently modified listings will be available in COMFIL 3. A user who wishes to modify a library routine must supply all of these requested files to the system programmer who is responsible for the library.

Additional information about the library routines, (entry points, transfer vector, length and use of common storage) is printed by the command PRSUS TSLIBN.
INFORMATION RETRIEVAL:

1) To retrieve a large volume of data, such as source decks or listings, requests may be submitted to the dispatcher for off-line processing of the document tapes.

2) To obtain source files or listings of one or a few of the commands ? system modules, or the library, a user may LINK to the necessary files and:

'PRINTF LIB INFO' If the required routine is a library routine and the family name is unknown.

'ARCHIV X' FAMILY 'CRUNCH' FILE If there is an ARCHIV file called FAMILY containing the required FILE. This extracts and creates FILE CRUNCH.

'CRUNCH UN' FILE KIND 'NUMBER' will uncunch and create FILE KIND with line numbers.

'PRINTF' FILE KIND Will print the source file and thereby disclose what KIND should be, if unknown.

The source file FILE KIND may now be modified and translated with listing, if desired.

If a command is being inspected, the ARCHIV command may be used to print the table of contents of the COMAND CRUNCH file.

Maintenance Procedure

The few people who take the responsibility of actually updating the system must follow a set procedure. The following is a sketch of what these people must do. Either obtain from the modifying-user or do the work necessary to have a CRUNCHED (without sequence numbers) source file(s). For commands and library families, ARCHIV these crunched files and place them in COMFIL 1. Place the COMAND TSSDC, files in COMFIL 2. UPDBSS the library BSS files into TSLIBn BSS in COMFIL 2. Place the BCD listing files in COMFIL 3. Place the system module BSS files in COMFIL 1. One thing the responsible system programmer must do all the work for
Improvement Remarks:

To improve a fiscal volume ofawaii known as product of operation at increased year on year
convert to operational for off-line production of the

To apply more effort or late to one of the year

PRINTABLE BE Forex

LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFORE LIFE OFFICE
LONDON A FAMILY CONTRIBUTIONS EF
BEFOR...
is maintaining the document tapes. The way to do this is to maintain three separate files of the format of OUTPUT REQUEST, i.e., one for commands, one for modules and one for library. Each time a routine is changed, place a PUNCH entry in the appropriate request file. At the end of each month, uncrunch the necessary files as specified in the request files. Then run three separate and special request dump runs which will create input tapes which may be used by FMS FAP UPDATE in conjunction with the current source document tapes in order to produce new document source and listing tapes. Delete the n-2 document tapes and the listings from CUMFIL 3.
Identification
Equipment Configuration

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

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

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

All of these terminals can operate simultaneously with the MAC computer installation by time sharing its central
processor. In order to insure prompt response, the number of terminals active at a given time is limited by the supervisor program to 30. This number has already grown to 30 from 10, and is expected to grow further in the next few months. However, maximization of this number is not a primary objective at this time.

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

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

The fourth data channel provides communication with a disk (IBM 1302) and a low speed drum (IBM 7320). The theoretical storage capacity of the disk is 38 million computer words and the capacity of the drum is 186,400 words. The time required to transfer 32K words in or out of core is approximately one second for both the disk and the drum.

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

The transmission control unit (IBM-7750) consists of a stored-program computer which serves as an interface between the sixth data channel and up to 112 communication terminals capable of telegraph-rate operation (up to 200 bits per second). An appropriate number of these terminals are connected by trunk lines to the M.I.T. private branch exchange and to the TWX and TELEX networks. Higher rate terminals can be readily substituted for groups of these low-rate terminals; for instance, on the present MAC System, three 1200-bit-per-second terminals are installed, one of which provides the communication channel to the PDP-6 computer. All of these terminals are compatible with Bell System data sets. Part of the core memory of the
transmission control unit is used as output buffer, because the supervisor program and its necessary buffer space have grown in size to the point of occupying all of the A bank of core memory. The design intent was and still is to provide sufficient input-output buffer space in the main memory to eliminate unnecessary core-to-core transfers; the present mode of operation is a makeshift made necessary by equipment limitations.
SCHEMATIC OF COMMUNICATION NETWORK
Identification

Clocks

Purpose

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

The supervisor uses this clock both for interrupting programs and for time accounting. Base-time and day-of-the-month information are obtained from the Chronolog clock which is attached as a pseudo tape unit. The supervisor can also simulate the interrupt clock behavior for each user. By supervisor calls, the user can program for nested interrupts and computation time readings.
The C-35 will be the designated OCS Class. The initial training for this class will be at the intermediate level. The C-35 is a high-speed, high-performance aircraft designed for the role of intermediate-level instructor. It is equipped with a comprehensive array of avionics and navigational aids to ensure safe and effective training.

The C-35 will be used to train future instructors in the Air National Guard. It is designed to provide a high level of training for pilots and will be equipped with the latest in-flight simulation technology. The C-35 will be used to train new instructors in the Air National Guard and will be used to provide advanced training to existing instructors.
Identification

Console character sets

Purpose

For routine computer work, especially older applications, the normal 7094 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, erase character, and kill message. This normal BCD set is contained in a 6-bit code and when the character mode switch of a console is set to "normal", the console will transmit and receive in the normal BCD mode. Consoles, however, may be switched to a "full" or 12-bit mode. The normal BCD character set has been extended to a 12-bit character set by allowing the normal six to be preceded by either a zero or a one. The modes of the consoles are set by supervisor calls and the computer words being transmitted are either treated as six 6-bit characters per word or three 12-bit characters per word.

Code tables

The following tables present the character sets for different consoles. The 6-bit character set is listed under case-0. The 12 bit character set is listed under case-1. The case-1 characters which are listed in parentheses are ignored in 6-bit mode. Other case-1 characters (not in parentheses) are received in 6-bit mode as the corresponding case-0 character. The abbreviations used have the following meanings:

ig - ignored
WRU - who are you
P-off - Printing off
P-on - Printing on
erase - erase previous character
kill - kill or delete previous message
VT - vertical tab
CR - Carriage return
FF - form feed
tab - horizontal tabulation
hang up - data phone disconnect
sgnl spac - single space
dbl spac - double space
K.L. - Keyboard lock
K.U. - Keyboard unlock

An octal 57 is guaranteed to be a null character on output.
# INPUT/OUTPUT CODE TABLE FOR 1050 SELECTRIC CONSOLE

<table>
<thead>
<tr>
<th>Case</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>()</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>(±)</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>(¼)</td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>ig</td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>(#)</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>(%)</td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>(@)</td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>(line feed)</td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>(sngl spac)</td>
</tr>
<tr>
<td>09</td>
<td>9</td>
<td>(dbl spac)</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>ig</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>ig</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>ig</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>=</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>′</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>ig</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>ig (hang up)</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>ig (P-Off)</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>+</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>&amp;</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>A</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>a</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>B</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>b</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>C</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>E</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>F</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>G</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>H</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>I</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>ig black</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>ig</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>ig</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>ig</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
<td>)</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>:</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>ig</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>'erase'</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>ig</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>ig</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>-</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
<td>J</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>K</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
<td>L</td>
</tr>
<tr>
<td>44</td>
<td>44</td>
<td>M</td>
</tr>
<tr>
<td>45</td>
<td>45</td>
<td>N</td>
</tr>
<tr>
<td>46</td>
<td>46</td>
<td>O</td>
</tr>
<tr>
<td>47</td>
<td>47</td>
<td>P</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>Q</td>
</tr>
<tr>
<td>49</td>
<td>49</td>
<td>R</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>Q</td>
</tr>
<tr>
<td>51</td>
<td>51</td>
<td>R</td>
</tr>
<tr>
<td>52</td>
<td>52</td>
<td>ig</td>
</tr>
<tr>
<td>53</td>
<td>53</td>
<td>$</td>
</tr>
<tr>
<td>54</td>
<td>54</td>
<td>*</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>CR</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td>'kill'</td>
</tr>
<tr>
<td>57</td>
<td>57</td>
<td>'quit'</td>
</tr>
<tr>
<td>58</td>
<td>58</td>
<td>ig</td>
</tr>
<tr>
<td>59</td>
<td>59</td>
<td>/</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
<td>blank</td>
</tr>
<tr>
<td>61</td>
<td>61</td>
<td>ig</td>
</tr>
<tr>
<td>62</td>
<td>62</td>
<td>S</td>
</tr>
<tr>
<td>63</td>
<td>63</td>
<td>T</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>U</td>
</tr>
<tr>
<td>65</td>
<td>65</td>
<td>V</td>
</tr>
<tr>
<td>66</td>
<td>66</td>
<td>W</td>
</tr>
<tr>
<td>67</td>
<td>67</td>
<td>X</td>
</tr>
<tr>
<td>68</td>
<td>68</td>
<td>Y</td>
</tr>
<tr>
<td>69</td>
<td>69</td>
<td>Z</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
<td>ig</td>
</tr>
<tr>
<td>71</td>
<td>71</td>
<td>ig</td>
</tr>
<tr>
<td>72</td>
<td>72</td>
<td>tab</td>
</tr>
<tr>
<td>73</td>
<td>73</td>
<td>ig</td>
</tr>
<tr>
<td>74</td>
<td>74</td>
<td>ig</td>
</tr>
<tr>
<td>75</td>
<td>75</td>
<td>ig (P-Off)</td>
</tr>
<tr>
<td>76</td>
<td>76</td>
<td>ig</td>
</tr>
<tr>
<td>77</td>
<td>77</td>
<td>ig</td>
</tr>
</tbody>
</table>

*Interrupt is single push of "Reset line"*  
*QUIT is double push of "Reset line"*
## INPUT/OUTPUT CODE TABLE FOR THE MODEL 35 TELETYPewriter

<table>
<thead>
<tr>
<th>Case</th>
<th>Code</th>
<th>Case</th>
<th>Code</th>
<th>Case</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0 ig</td>
<td>25</td>
<td>E</td>
<td>52</td>
<td>FF ig</td>
</tr>
<tr>
<td>01</td>
<td>1 ig</td>
<td>26</td>
<td>F</td>
<td>53</td>
<td>$ (1)</td>
</tr>
<tr>
<td>02</td>
<td>2 (bell)</td>
<td>27</td>
<td>G</td>
<td>54</td>
<td>* ig</td>
</tr>
<tr>
<td>03</td>
<td>3 ig</td>
<td>30</td>
<td>H</td>
<td>55</td>
<td>CR ig</td>
</tr>
<tr>
<td>04</td>
<td>4 (=)</td>
<td>31</td>
<td>I</td>
<td>56</td>
<td>'kill' ?</td>
</tr>
<tr>
<td>05</td>
<td>5 (v)</td>
<td>32</td>
<td>ig ig</td>
<td>57</td>
<td>'quit' ig</td>
</tr>
<tr>
<td>06</td>
<td>6 (^)</td>
<td>33</td>
<td>. ig</td>
<td>60</td>
<td>blank ig</td>
</tr>
<tr>
<td>07</td>
<td>7 (&lt;)</td>
<td>34</td>
<td>) ]</td>
<td>61</td>
<td>/ (K.L.)</td>
</tr>
<tr>
<td>10</td>
<td>8 (&gt;)</td>
<td>35</td>
<td>: ig</td>
<td>62</td>
<td>S S</td>
</tr>
<tr>
<td>11</td>
<td>9 (~)</td>
<td>36</td>
<td>ig ig</td>
<td>63</td>
<td>T T</td>
</tr>
<tr>
<td>12</td>
<td>ig ig</td>
<td>37</td>
<td>'erase' &quot;</td>
<td>64</td>
<td>U U</td>
</tr>
<tr>
<td>13</td>
<td>= (----)</td>
<td>40</td>
<td>- ( _ )</td>
<td>65</td>
<td>V V</td>
</tr>
<tr>
<td>14</td>
<td>' (1)</td>
<td>41</td>
<td>J</td>
<td>66</td>
<td>W W</td>
</tr>
<tr>
<td>15</td>
<td>ig (WRU)</td>
<td>42</td>
<td>K</td>
<td>67</td>
<td>X X</td>
</tr>
<tr>
<td>16</td>
<td>ig (hang up)</td>
<td>43</td>
<td>L</td>
<td>70</td>
<td>Y Y</td>
</tr>
<tr>
<td>17</td>
<td>ig (P-Off)</td>
<td>44</td>
<td>M</td>
<td>71</td>
<td>Z Z</td>
</tr>
<tr>
<td>20</td>
<td>+ ig</td>
<td>45</td>
<td>N</td>
<td>72</td>
<td>tab (VT)</td>
</tr>
<tr>
<td>21</td>
<td>A A</td>
<td>46</td>
<td>O</td>
<td>73</td>
<td>, ;</td>
</tr>
<tr>
<td>22</td>
<td>B B</td>
<td>47</td>
<td>P</td>
<td>74</td>
<td>( [</td>
</tr>
<tr>
<td>23</td>
<td>C C</td>
<td>50</td>
<td>Q</td>
<td>75</td>
<td>ig (P-On)</td>
</tr>
<tr>
<td>24</td>
<td>D D</td>
<td>51</td>
<td>R</td>
<td>76</td>
<td>ig (K.U.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>77</td>
<td>ig (Alt Mode)</td>
</tr>
</tbody>
</table>

Interrupt is single push of "BREAK"
QUIT is double push of "BREAK"
<table>
<thead>
<tr>
<th>Case</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>ig</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>ig</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>(bell)</td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>ig</td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>(#)</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>(%)</td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>(@)</td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>(&lt;)</td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>(&gt;)</td>
</tr>
<tr>
<td>09</td>
<td>9</td>
<td>ig</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>ig</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>'erase'</td>
</tr>
<tr>
<td>12</td>
<td>=</td>
<td>(←)</td>
</tr>
<tr>
<td>13</td>
<td>'</td>
<td>(!)</td>
</tr>
<tr>
<td>14</td>
<td>ig</td>
<td>(WRU)</td>
</tr>
<tr>
<td>15</td>
<td>ig</td>
<td>(hang up)</td>
</tr>
<tr>
<td>16</td>
<td>ig</td>
<td>(P-Off)</td>
</tr>
<tr>
<td>17</td>
<td>+</td>
<td>&amp;</td>
</tr>
<tr>
<td>18</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>19</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>21</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>27</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>28</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>29</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>31</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>32</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>33</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>34</td>
<td>]</td>
<td>]</td>
</tr>
<tr>
<td>35</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>36</td>
<td>;</td>
<td>;</td>
</tr>
<tr>
<td>37</td>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>38</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>39</td>
<td>(</td>
<td>(</td>
</tr>
<tr>
<td>40</td>
<td>)</td>
<td>)</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>43</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>44</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>45</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>46</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>47</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>48</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>49</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>50</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>51</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>52</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>53</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>54</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>55</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>56</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>57</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>58</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>59</td>
<td>J</td>
<td>J</td>
</tr>
<tr>
<td>60</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>61</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>62</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>63</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>64</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>65</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>66</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>67</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>68</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>69</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>70</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>71</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>72</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>73</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>74</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>75</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>76</td>
<td>[</td>
<td>[</td>
</tr>
<tr>
<td>77</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>Case</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
<td>ig</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>ig</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>ig</td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>ig</td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>(#)</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>ig</td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>ig</td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>ig</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>ig</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>ig</td>
</tr>
<tr>
<td>12</td>
<td>ig</td>
<td>ig</td>
</tr>
<tr>
<td>13</td>
<td>i</td>
<td>ig</td>
</tr>
<tr>
<td>14</td>
<td>'</td>
<td>ig</td>
</tr>
<tr>
<td>15</td>
<td>ig</td>
<td>(WRU)</td>
</tr>
<tr>
<td>16</td>
<td>ig</td>
<td>ig</td>
</tr>
<tr>
<td>17</td>
<td>ig</td>
<td>ig</td>
</tr>
<tr>
<td>20</td>
<td>&amp;</td>
<td>ig</td>
</tr>
<tr>
<td>21</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>22</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>23</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>24</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Identification

Special console characters

Purpose

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

Break Character

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

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

Quit and Interrupt Signals

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

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

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

**Erase and Kill Characters**

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

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

Data phone extensions & console ID.

Purpose

The consoles may be connected with the MAC or CC machines by way of telephone lines through a special exchange or switchboard at M.I.T. Because of the difference in transmission rates, certain lines must be used by 1050 consoles and different lines must be used by teletypes. The exchange has the ability of searching several lines to find one which is not busy, and therefore certain numbers are specified as "hunt" numbers, implying that a set of numbers will be searched as a result of dialing the "hunt" number.

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

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

Computer extension numbers:

MAC computer:

TELETYPES:

HUNT 211: 211, 212, 213, 214, 215, 216, 217, 218
HUNT 221: 221, 222, 223, 224, 225, 226, 227, 228
HUNT 231: 231, 232, 233, 234, 235, 236, 237, 238

1050:

HUNT 311: 311, 312, 313, 314, 315, 316, 317, 318, 319, 320
HUNT 321: 321, 322, 323, 324, 325, 326, 327, 328, 329, 330
HUNT 331: 331, 332, 333, 334

Computation Center Computer:

TELETYPES:

HUNT 251: 251, 252, 253, 254, 255, 256, 257, 258
HUNT 261: 261, 262, 263, 264, 265, 266, 267, 268

1050:

HUNT 381: 381, 382, 383, 384, 385, 386, 387, 388
Locations and telephone extensions for teletypes:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12 Oriole St., West Roxbury</td>
<td>511</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26-157</td>
<td>512</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1-109,115 (jacked)</td>
<td>513</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Harvard, 33 Oxford St., Cambridge</td>
<td>514</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>11 Holdenwood Rd., Concord</td>
<td>515</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>33-214</td>
<td>516</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>E19-319</td>
<td>517</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>26-157</td>
<td>518</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>52-454C, 560, 561, 562 (jacked)</td>
<td>519</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>26-265</td>
<td>520</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Brown U, Comp Lab.,</td>
<td>521</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>180 George St.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Providence, R.I.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26-150</td>
<td>522</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>54-616</td>
<td>523</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Boston U, Rm 5 700</td>
<td>524</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Commonwealth</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26-157</td>
<td>525</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>26-157</td>
<td>526</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Lincoln Lab, B-232, A-166</td>
<td>417</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>(jacked)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lincoln Lab, B-276, A-166</td>
<td>418</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>(jacked)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-482</td>
<td>419</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>1-171</td>
<td>420</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>52-076A</td>
<td>421</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Address</td>
<td>Page</td>
<td>Section</td>
<td>AC.3</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>20D-102</td>
<td>422</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>20B-120</td>
<td>423</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>26-352</td>
<td>424</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>14S-316</td>
<td>425</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>32-108A</td>
<td>426</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>T. S. 823</td>
<td>427</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>111 Ivy St., Brookline</td>
<td>428</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>T. S. 831</td>
<td>429</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>295 Harvard St., Apt 811, Cambridge</td>
<td>430</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>T. S. 809</td>
<td>431</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>T. S. Conference Room, 8th floor</td>
<td>432</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>T. S. 901</td>
<td>433</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>96 Montvale Rd., Newton Center</td>
<td>434</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>T. S. 526</td>
<td>435</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>14N-210</td>
<td>436</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>58 Edgemoor Rd., Belmont</td>
<td>437</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>1-171</td>
<td>438</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>33 Dawes Rd., Lexington</td>
<td>439</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>T. S. 908</td>
<td>440</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>35 Alcott St., Acton</td>
<td>405</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>T. S. 908</td>
<td>406</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>3-463</td>
<td>407</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>T. S. 920</td>
<td>408</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>474 Putnam Ave, Apt 1, Cambridge</td>
<td>409</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>T. S. 920</td>
<td>410</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>Address</td>
<td>411</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2-366</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-155 (jacked)</td>
<td>412</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>4 Hilda Rd., Bedford</td>
<td>413</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>T. S. 920</td>
<td>414</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>T. S. 920</td>
<td>415</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>T. S. 920</td>
<td>416</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>T. S. 920 (33KSR)</td>
<td>510</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>T. S. 920 (33KSR)</td>
<td>540</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>T. S. 931 (35ASR)</td>
<td>532</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>6-320</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tufts, Comp Center, Medford</td>
<td>528</td>
<td>141</td>
<td>141</td>
</tr>
<tr>
<td>NS1-310D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W91-223</td>
<td>529</td>
<td>142</td>
<td>142</td>
</tr>
<tr>
<td>26-145</td>
<td>530</td>
<td>143</td>
<td>143</td>
</tr>
<tr>
<td>54-1425</td>
<td>531</td>
<td>144</td>
<td>144</td>
</tr>
<tr>
<td>20A-121</td>
<td>549</td>
<td>145</td>
<td>145</td>
</tr>
<tr>
<td>24-019</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26-263</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26-261</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26-259</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26-157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26-157</td>
<td></td>
<td>154</td>
<td>154</td>
</tr>
</tbody>
</table>
LOCATIONS AND TELEPHONE EXTENSIONS FOR 1050'S:

<table>
<thead>
<tr>
<th>Location</th>
<th>Code</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-171</td>
<td>541</td>
<td>U</td>
</tr>
<tr>
<td>26-142</td>
<td>542</td>
<td>C</td>
</tr>
<tr>
<td>26-143</td>
<td>543</td>
<td>G</td>
</tr>
<tr>
<td>26-139</td>
<td>544</td>
<td>J</td>
</tr>
<tr>
<td>3-269</td>
<td>545</td>
<td>1</td>
</tr>
<tr>
<td>24-421</td>
<td>546</td>
<td>2</td>
</tr>
<tr>
<td>E19-325 (with attachments)</td>
<td>547</td>
<td>H</td>
</tr>
<tr>
<td>T. S. 404 (with attachments)</td>
<td>548</td>
<td>I</td>
</tr>
<tr>
<td>20D-004</td>
<td>554</td>
<td>P</td>
</tr>
<tr>
<td>26-153</td>
<td>555</td>
<td>3</td>
</tr>
<tr>
<td>52-582</td>
<td>556</td>
<td>4</td>
</tr>
<tr>
<td>26-265</td>
<td>557</td>
<td>5</td>
</tr>
<tr>
<td>26-265</td>
<td>558</td>
<td>6</td>
</tr>
<tr>
<td>26-265</td>
<td>559</td>
<td>7</td>
</tr>
<tr>
<td>26-265</td>
<td>549</td>
<td>8</td>
</tr>
<tr>
<td>61 Shattuck Rd., Watertown</td>
<td>550</td>
<td>9</td>
</tr>
<tr>
<td>1-150</td>
<td>551</td>
<td>V</td>
</tr>
<tr>
<td>26-144</td>
<td>552</td>
<td>C</td>
</tr>
<tr>
<td>26-169</td>
<td>553</td>
<td>I</td>
</tr>
<tr>
<td>T. S. 513, 512 (jacked)</td>
<td>560</td>
<td>.</td>
</tr>
<tr>
<td>T. S. 511, 513 (jacked)</td>
<td>561</td>
<td>.</td>
</tr>
<tr>
<td>T. S. 510</td>
<td>562</td>
<td>.</td>
</tr>
<tr>
<td>T. S. 509</td>
<td>563</td>
<td>.</td>
</tr>
<tr>
<td>T. S. 508</td>
<td>564</td>
<td>.</td>
</tr>
<tr>
<td>T. S. 507</td>
<td>565</td>
<td>.</td>
</tr>
<tr>
<td>Description</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>T. S. 506</td>
<td>566</td>
<td></td>
</tr>
<tr>
<td>T. S. 505</td>
<td>567</td>
<td></td>
</tr>
<tr>
<td>26-265</td>
<td>568 T</td>
<td></td>
</tr>
<tr>
<td>1-171</td>
<td>441 U</td>
<td></td>
</tr>
<tr>
<td>Lot 6, Deacon Haynes Rd., Concord</td>
<td>442 W</td>
<td></td>
</tr>
<tr>
<td>T. S. 803</td>
<td>443 A</td>
<td></td>
</tr>
<tr>
<td>T. S. 814</td>
<td>444 B</td>
<td></td>
</tr>
<tr>
<td>T. S. 806</td>
<td>445 D</td>
<td></td>
</tr>
<tr>
<td>T. S. 801, 836, 838 (jacked)</td>
<td>446 E</td>
<td></td>
</tr>
<tr>
<td>T. S. 920 (with attachments)</td>
<td>447 F</td>
<td></td>
</tr>
<tr>
<td>T. S. 920 (with attachments)</td>
<td>448 F</td>
<td></td>
</tr>
<tr>
<td>T. S. 926</td>
<td>449 K</td>
<td></td>
</tr>
<tr>
<td>T. S. 825</td>
<td>450 L</td>
<td></td>
</tr>
<tr>
<td>3-482</td>
<td>451 M</td>
<td></td>
</tr>
<tr>
<td>T. S. 801 (jacked)</td>
<td>452 N</td>
<td></td>
</tr>
<tr>
<td>24-522</td>
<td>453 Ø</td>
<td></td>
</tr>
<tr>
<td>14S-316</td>
<td>454 Q</td>
<td></td>
</tr>
<tr>
<td>4-107</td>
<td>455 R</td>
<td></td>
</tr>
<tr>
<td>T. S. 503</td>
<td>456 +</td>
<td></td>
</tr>
<tr>
<td>T. S. 503</td>
<td>457 +</td>
<td></td>
</tr>
<tr>
<td>T. S. 503</td>
<td>458 +</td>
<td></td>
</tr>
<tr>
<td>T. S. 503</td>
<td>459 +</td>
<td></td>
</tr>
<tr>
<td>T. S. 503</td>
<td>460 +</td>
<td></td>
</tr>
<tr>
<td>24-109</td>
<td>461 $</td>
<td></td>
</tr>
<tr>
<td>T. S. 822</td>
<td>462 S</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Lot 4, Deacon Haynes Rd., Concord</td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>208-226</td>
<td>464</td>
<td></td>
</tr>
<tr>
<td>32-1 08A</td>
<td>465</td>
<td></td>
</tr>
<tr>
<td>Murray Hill, New Jersey</td>
<td>466</td>
<td></td>
</tr>
<tr>
<td>T. S. 920</td>
<td>467</td>
<td></td>
</tr>
<tr>
<td>T. S. 920</td>
<td>468</td>
<td></td>
</tr>
</tbody>
</table>

Note: . (period), (comma)
Identification

Historic file system

Purpose

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

Considerations

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

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

Protection

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

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

File Structure

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

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

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

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

The new file structure and Input/Output system

Purpose

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

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

Structure of the I/O System

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

The File Coordinator:

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

The Buffer Control Module:

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

The Strategy Modules:

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

The I/O Adapters:

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

Operation of the Buffer Control Module

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

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

Reading without a buffer:

If blocks of integral number of physical records are read or if reading goes through the end of file, no buffer will be used even if one is assigned.

If no buffer is assigned and partial records are called for, the physical record will be read for each call in order to extract the logical or partial record from the physical.

Writing without a buffer:

A complete new file of any length can be written by a single call without a buffer being assigned.

An existing file may be written into without a buffer only from the beginning of a physical record through the end of a physical record or through the end of a file.

Appending to a file or writing partial records requires a buffer.

Truncation without a buffer:

Truncation without a buffer can only be accomplished if the truncation word is beyond the end of file or in front of the first word (file made empty).

The BCM selects an appropriate strategy depending on whether a buffer has been assigned or not and returns an error if a buffer is mandatory where none was assigned. A user may switch a file from "no-buffer" mode to "buffer" mode or vice-versa by calls to BUFFER.
File Notation and Structure

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

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

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

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

Modes:

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

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

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

002. SECONDARY- This property defines files which may be deleted by storage collection mechanisms in preference to other files.
004. READ-ONLY- The file can only be read. An attempt
to write into a file of this property will cause an
error condition.

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

020. PRIVATE- The file can only be referenced by the
AUTHOR i.e. the user who created or last modified
this file.

040. LINKABLE- The file may be referenced by other
users, through the use of the "LINK" facility.

100. PROTECTED- The mode of the file may only be changed
by the AUTHOR of the file. Any attempt by another
user to change the mode of this file will result in
an error condition.

File Directories:

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

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

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

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

RCOUNT specifies the number of elements contained in a physical record of the file. NOREC specifies the number of physical records contained in the file. LCOUNT specifies the number of elements contained in the last physical record of the file. The highest element address in a file may be defined as \((\text{NORECS} - 1) \times \text{RCOUNT} + \text{LCOUNT}\). The 3-bit integer \(P\) is normally one.

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

If user "A" wishes to reference a file contained in some other user's file directory (user "B"), he can accomplish this by means of a "LINKED" file. A LINKED file is defined in a user's file directory as a file with a device specification of zero \((F=0)\). When user "A" references a file which is linked to user "B", the MODE of the corresponding file directory entry for user "B" must include the LINKABLE property.

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

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

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

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

WORD ........................ CONTENTS ........................
1. FILE NAME, PART 1 (36 BITS)
2. FILE NAME, PART 2 (36 BITS)
3. DATE AND TIME LAST MODIFIED (36 BITS)
4. DATE LAST USED (18 BITS), AUTHOR (18 BITS)
5. MODE (8 BITS), ILOCK (10 BITS), F (3 BITS), RCOUNT (15 BITS)
6. --- (3 BITS), NORECS (15 BITS), P (3 BITS), LCOUNT (15 BITS)
7. The next "P" words contain specific information for a file of type "F".
1301/1302 Disk and 7320 Drum Strategy

The file directory entry for a 1301, 1302 or 7320 file contains pointers to the first and last tracks. For a file of this type, RCOUNT will be the number of data words in a single track. NORECS will be the total number of tracks in the file and LCOUNT will be the number of data words in the last track.

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

```
   1  LCOUNT 1  TRAKNO 1
   S   35
```

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

Tracks are assigned in a manner similar to that described in memo CC-196 (Disk Control Routine). All track usage tables will be files contained as entries in the Master File Directory. The file which defines the usage of disk tracks will be referred to as "DISCUT (FILE)". The track usage file for the 7320 drum will be referred to as "DRUMUT (FILE)". Whenever possible, successive tracks of a file will be assigned to separate channels. This procedure will allow all available disk/drum channels to operate on a file in parallel.

1301/1302 Disk and 7320 Drum I/O Adapter

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

Tape Strategy Module

Magnetic tapes will be treated as secondary storage in the same manner as disks or drums. Many files can be recorded on a single tape or set of tapes, and a single file may consist of more than one tape. The first physical record of a tape file will be a BCD header label of five words. The first two words of the header record are the original file name, the next two words contain the date and time when the
file was initiated, and the fifth word will be the tape sequence number. The first tape of a file will always have the sequence number \"1\". All subsequent records on the tape will be in blocked binary format.

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

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

PZE RECNO,,LCOUNT

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

If a file consists of more than one physical tape, the physical tapes will be terminated by an end of file followed by a control record, written in BCD. This record will contain information, such as the next tape sequence number to be used.

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

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

Calling Conventions:

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

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

In all of the calls, if an argument is not pertinent, a -0 may be specified. All calls will accept two more arguments than shown. The first is the location of users' error return and the second, if supplied, specifies the location into which the error code will be stored.
Some of the arguments and information items are of special forms which might be noted here.

DEVICE = 1. is low speed drum
         2. is disk
         3. is tape

File status = 1. is inactive
               2. is open for reading
               3. is open for writing
               4. is open for reading and writing

SUMMARY

Administrative and Privileged:

UPDMFD.($ PROB$, $ PROG$)
DELMFD.($ PROB$, $ PROG$)
ATTACH.($ PROB$, $ PROG$)
MOVFIL.($ NAME1$, $ NAME2$, $ PROB$, $ PROG$)
SFTIL.($ NAME1$, $ NAME2$, DAYTIM, DATELU, MODE, DEVICE)
LINK.($ NAME1$, $ NAME2$, $ PROBN$, $ PROG$, $ NMI$, $NM2$, $MODE$)
ALLOT.(DEVICE,-ALLOC-, -USED-)

Reading and Writing:

OPEN.($STATUS$, $ NAME1$, $ NAME2$, -MODE-, -DEVICE-)
BUFFER.($ NAME1$, $ NAME2$, BUFE(432)...432)
RDFILE.($ NAME1$, $ NAME2$, RELOC, A(N)...N,-EOF-, -EOFCT-)
WRFILE.($ NAME1$, $ NAME2$, RELOC, A(N)...N,-EOF-, -EOFCT-)
TRFILE.($ NAME1$, $ NAME2$, RELOC)
FCHECK.($ NAME1$, $ NAME2$, FINISH)
CLOSE.($ NAME1$, $ NAME2$)

others:

SETPRI.(PRIOR)
RESEF.
CHFILE.($ NAME1$, $ NAME2$, -NMODE-, -$NEWNM1$, -$NEWNM2$)
DELFIL.($ NAME1$, $ NAME2$)
FSTATE.($ NAME1$, $ NAME2$, A(8)...8)
UNLINK.($ NAME1$, $ NAME2$)
STORGE.(DEVICE,-ALLOC-, -USED-)
MOUNT.(-CHAN-,UNIT, MESSAG(20)...20)
UMOUNT.(UNITNO, MESSAG(20)...20)
VERIFY.(UNITNO, LABEL(4)...4)
LABEL.(UNITNO, LABEL(4)...4)
TAPFIL.($ NAME1$, $ NAME2$, UNITNO, FILENO)
1ODIAG.(A(7)...7)
TLILOCK.(RETN)
FERRTN.(RETN)
Identification

Library files

Organization

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

The system library is currently divided into files which reside in the system common file directory. TSLIB1 contains all of the standard routines described as library subroutines and library entries in this manual. The loader will normally search TSLIB1 for missing routines unless prohibited by special arguments. TSLIB2 contains the debugging subroutines and core-B transfer commands. The loader will search TSLIB2 automatically only when a core-B transfer command has been given. If the debugging routines are to be loaded with the program before execution the loader should be informed by (SYS) TSLIB2 or, for example, more completely by (NEED) FLEXPM (SYS) TSLIB2. A special library in the system file is KLULIB which contains subroutines for the "KLUDGE" (i.e., ESL scope console) and which may be searched if special arguments are given to the loader. The old library (prior to October 1964) still resides in the system file. It may be used with the (OLD) special argument to the loader. It is being neither maintained nor updated by the system maintenance staff.

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

COMMON Files

Purpose

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

System Files

The four common files associated with the system programmers' problem number took on the special function of servicing all users, regardless of problem number. Their common file 4 became known as the public file and any user could put files there and copy files from there. In order to housekeep the public files, the disk editor, which was run at least once a day, deleted all files in the public file which were in temporary or permanent mode. Only a system programmer could change a file in public to R1 or R2 mode. A further restriction has been placed on the public file, namely, only programs which are adequately documented may remain in public. The documentation is available from the consultants. The system programmers' common file 2 became known as common file S and any user could copy files from there. Common file S contained the binary files of all the commands and the BSS files of the libraries. The system programmers' common file 1 contains the source and binary files of the supervisor and common file 3 contains listing files of the supervisor.

Changes to Common Files

With the advent of the new I/O system and the ability of many users simultaneously to access the same file, the need
for the public file has diminished. Any file may be declared linkable by any or all users and, therefore, only one copy of it needs exist.

The public file under the new I/O system is a file directory of track quota 0 and is merely a directory of links to files in other directories. Files which are needed by commands will exist in the system file (common file S) and linking pointers to them will exist in the public file directory. If a user wishes to place a file in the public file, after proper documentation is available, a link pointer to the file may be placed in the public file directory by a system programmer. The actual file may remain in the user's file directory or a decision may be made to place the file in the system file.

The concept and implementation of common files will continue for compatibility purposes but they are no longer needed.
Identification

Time-accounting files

Purpose

The time-accounting files contain crucial user information such as passwords, time allotments, party group numbers, etc. These files are read and written by LOGIN and LOGOUT, and they can be updated by a few privileged commands.

Definitions

Each person who is permitted to use the time sharing system is assigned a unique programmer number (4 numeric digits). Depending on the number of jobs he undertakes, he will also be assigned one or more problem numbers (1 alpha and 3 or 4 numeric characters). Groups of people working on the same problem may be assigned the same problem number. When a user logs in, he types his problem number and last name. The combination of problem number and last six characters of the last name is neither unique nor secret. A six character secret password is therefore requested by LOGIN so that a check can be made of the accounting files to see if such a unique combination exists. The unique combination defines a single user and a single file directory, with its associated time and space allotments, etc. An administrator allots a certain amount of computer time each month and a quota of secondary storage space to each user. In addition, each user is placed in a party group. Each party group contains some number of users and some different number of slots or lines, which give access to the computer (see Section AH.1.01). One of these lines is assigned to a user as soon as he logs in. When the number of lines is exhausted, no new users may log in. The administrator has the power to give a user a line multiplier which has the effect of assigning more than one line to the user when he log in and thereby reducing the number of users who may be logged in. Each user is also assigned to a unit group. Each unit group specifies which consoles the user may or may not use.

Description of files

There are four time-accounting files: UACCNT, USDTIM, PRTYGP, and GRPUNI with secondary name SECRET.

UACCNT:

The file UACCNT contains identifying information for each user. LOGIN searches UACCNT for the user's problem number, name, and password; this combination must be found before the person can be logged in.
This file consists of 14-word lines. These lines are free-field with blanks as the delimiter of the fields. However, the first character of each line has a particular meaning. If the first character is blank (b), the line is taken as a continuation.

The first line in file UACCNT must be a problem number master line:

* PROBNO DF1 CF1 TF1 DF2 CF2 TF2
  b DF3 CF3 TF3 DF4 CF4 TF4

DFn drum quota for nth common file for this problem number.
CFn nth common file quota on disk.
TFn nth common file quota on tape.

The first character of this line must be '*'.

Following the problem number master line comes all the programmer information for the programmer numbers which belong to this problem number. The first character of a programmer information line must be something other than '*' or blank. Programmer information must be in this order reading from left-to-right.

NAME PASSWORD PROGNO RQDR RQDS RQTP
  b TA1 TA2 TA3 TA4 TA5 PRTY S LMULT
  b UNIT RESTRICTION

RQDR user's quota on drum.
RQDS user's quota on disk.
RQTP user's quota on tape.
TAn time allotted for nth shift.
PRTY user's party group number.
  S standby indicator, if non-zero user can be standby.
LMULT user's line multiplier.
UNIT user's unit group number.
RESTRICTION user's restriction code (privileged calls).

The amount of information on each line is arbitrary, the only restriction being that there can be no more than ten continuation cards.

Following the user information lines is the next problem number master line followed by the user information belonging to that problem number, etc.
USDTIM:
The file USDTIM contains the following information for each user:

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

This file has a fixed format. There are two lines per user.

Format of line 1:
- Characters 1 - 6: PROBNO
- Characters 7 - 12: PROGNO
- Characters 13 - 18: NAME

Format of line 2:
- Characters DATE1, TIME1, DATE2, TIME2, TU1, TU2, TU3, TU4, TU5, TL

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

PTYGP:
The file PTRYGP contains the party group information and the maximum number of users. This file consists of 14-word lines.

The first line of the file contains only the maximum number of users.

The rest of the lines contain pairs of numbers (XXX b YYY). The pairs of numbers must be separated from each other and other pairs by at least one blank (b). The number of pairs per line is arbitrary.

- XXX: group number
- YYY: number of lines for group number XXX

LOGIN reads this file to determine whether a user will be logged in as primary or as standby.

GRPUNI:
The file GRPUNI defines groups of consoles the user may or may not be allowed to use. Each unit group begins with a
line of 14 words, containing the unit group number and a number equal to 14 times the number of lines describing this group. Each following line contains a word indicating whether this list of consoles is allowable or not, followed by the list of console ID's. If the first word of the line is blank, or zero, the consoles on that line are available to the user. If the first word is anything else, the user is not allowed on the consoles whose ID's are in the line. If the unit group contains a list of available consoles, the user must be on one of them. If, however, the group contains only consoles on which he is not allowed, and he is not on any of them, the user is logged in.

An example of the GRPUNI file:

```
Characters  1-6  7-12  13-18  19-24  25-30 ..............
          15   28
nnn  ID1  ID2               ..............
          25   56
           ID5  ID6               ..............
           ID7               ..............
           ID8               ..............
           ID9               ..............
          30   28
           ID10  ID11
nnn  ID12  ID13
```

The dots indicate additional ID's, or blank fields, which are skipped. Each line is 14 words long. The first unit group, group number 15, has 28 words following the first line. Group 15 consists of consoles the user is not allowed to use, as indicated by the first word on each line being non-blank. Group 25 contains four lines of console ID's, all of which are available to the user, as indicated by the first word on each line being blank. The user must be on one of these consoles. Group 30 contains both allowable and non-allowable consoles. Since this group contains a list of consoles the user is allowed on, he must be on one of them.
Identification

Bulk input and output

Purpose

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

Usage

The disk editor program is run several times a day by the operations staff. Request cards may be submitted to the dispatcher by the user or the REQUEST command may be used to create a line-numbered card image file called OUTPUT REQUEST. Each line within the OUTPUT REQUEST file is the equivalent of a control card and may, therefore, specify any of the following requests except INPUT. The format of each line is the same as a control card except the PROB PROG must not be specified. See Method, below.

The control cards for the disk editor are of the format:

\[
XX \text{ PROB PROG NAME1 NAME2 OP \ldots NAME1n NAME2n}
\]

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

- PROB is the problem number. It must not be specified in an OUTPUT REQUEST file.
- PROG is the programmer number. It must not be specified in an OUTPUT REQUEST file.
- NAME1 NAME2 is the file name. Most requests allow more than one file name per card with the restriction that the file name must be complete on one card, i.e., NAME2 may not be on a continuation card.

\[
XX=C \text{ Continuation card}
\]

- XX=INPUT This card must precede a card deck to be input to the disk. The last card of the deck must have *EOF* beginning in column 8. Only one file name may appear on the control card and OP may specify the desired mode in octal for the file. If OP is not specified, a permanent file will be created, or if a file of the same
name already exists, the mode will be preserved.

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

**XX=DPUNCH** The BCD file NAME1 NAME2 is punched off-line. If the file is line-marked, just the first 80 characters per line of data will be punched.

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

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

**XX=DELETE** The file NAME1 NAME2 will be deleted from the current file directory.

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

The file NAME1 NAME2 on this machine is in no way changed. Any previous versions of the file on the receiving machine will be deleted regardless of mode, but the mode will be preserved by the new file.
XX=PRNDEL; DPUDEL; BPUDEL; 7PUDEL:

The file(s) will be PRINTed, DPUNCHed, BPUNCHed, or 7PUNCHed and then the mode will be changed to temporary. The next time the file is read or the user logs out, the file will be deleted. This is safer than a process request followed by a DELETE request. In case of machine or tape failure during the processing of the request, the operations staff will have the opportunity of restarting the disk editor.

XX=CHFILE

Change the name and/or the mode of NAME1 NAME2. Only one file may be specified in one request. OP is MODE NEW1 NEW2. MODE is the octal value of the new mode or * if the mode is not to change. NEW1 NEW2 are the new names; either may be * if it is not to be changed from the original. Both NEW1 NEW2 may be omitted if only the mode is to change.

Method

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

The problem will be printed.

This is a problem to be printed. The next time the file is saved, the user loses all the files.

This is a problem to be printed. The next time the file is saved, the user loses all the files.

This is a problem to be printed. The next time the file is saved, the user loses all the files.

This is a problem to be printed. The next time the file is saved, the user loses all the files.

This is a problem to be printed. The next time the file is saved, the user loses all the files.

This is a problem to be printed. The next time the file is saved, the user loses all the files.

This is a problem to be printed. The next time the file is saved, the user loses all the files.

This is a problem to be printed. The next time the file is saved, the user loses all the files.

This is a problem to be printed. The next time the file is saved, the user loses all the files.
Identification

DAEMON - Disk Dump and Reload
M.J. Bailey - x6006

Purpose

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

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

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

Purpose
Any programming system that is to be run as background in CTSS must observe certain conventions or restrictions. These conventions arise due to two main system requirements: that the background program be interruptible and that changes of state are a CTSS supervisor function illegal for the background to perform. The main area of a program affected is its input and output which must be timing insensitive.

Restrictions

Change of state:
All changes of state are trapped by the protection mode hardware but certain ones are processed by the supervisor and allowed, such as EFTM (enter floating-point mode). The following instructions are not allowed and, if used, will cause an on-line diagnostic:

ECTM  LPI
ESNT  LRI
ESTM  SEA
ETM   SEB

1/O timing:
Input and Output must be programmed so that it is not timing dependent; thus the LCHX (load channel) instruction is prohibited. An RCHX (reset and load channel) instruction, if given, must immediately follow the select instruction. An exception is made for the on-line printer and punch where up to 3 SPR's, SPU's and/or NOP's can come between the Select and RCHX instructions. If an RCHX is given that does not comply with these conventions, it will still be executed but its execution will turn on the 1/O check light since it was not given "in time".

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

The FAP assembler accepts the pseudo-op, BCORE, which automatically includes this bit 20 in all 1/O commands such as I0CD, I0RT, TCH, etc. And flags any illegal instructions used.
I/O units:

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

a. card reader, card punch, and printer
b. tape units A1-A6, A10, B1-B6, B10
c. A7, the chronolog clock

Referencing of other units will cause a diagnostic.

Program stop:

Any planned system stop should be effected by an HTR instruction rather than an HPR. The instruction counter is set differently on the two instructions and due to this difference the HPR if interrupted does not cause a genuine program stop. Example:

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

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

Channel traps:

Data channel traps on channels A and B operate normally except that when a trap occurs, control is transferred to location 13 or 15, which must contain an unconditional transfer. (In normal operating mode, when a trap occurs, the contents of 13 or 15 are executed by an XEC and need not be transfer instructions.)

Console keys:

Operating procedures have been modified to limit operator intervention in such a way that no foreground user or the CTSS supervisor is affected. The address portion of the keys is used by the supervisor for this function and therefore cannot be used by a background system. The keys are used to simulate the following functions:

a. depressing the "Load Cards" button,
b. depressing the "Start" button,
c. depressing the "Load Tape" button,
d. initiating "a standard error" procedure.

A "standard error" procedure is defined as: storing the instruction counter in a prearranged location and transferring control to another prearranged location (normally a transfer to a post-mortem routine or to the background system itself). The background system specifies these two locations to the CTSS supervisor by the following call:
TSX   DEFERR,4
PZE   ERRILC,,ERRTRA

where DEFERR contains: TIA =HDEFERR. ERRILC is the location where the instruction counter will be stored and ERRTRA is the location to which control will be transferred.

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

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

TESTSS   TIA   L
L   TRA   1,4

If running with CTSS supervisor, the TIA is interpreted as a regular supervisor call with a 2,4 return. If running independently, the TIA L operates as if it were a TRA L; thus the 1,4 return.

Timers:
The subroutines for determining the time operate properly whether the background system is running independently or not. The FMS subprogram GETTM can be used to read the date and time of day from the chronolog clock. The FMS subprogram TIMR can be used to determine elapsed time from the interval timer clock but when running with CTSS the operation of the interval timer clock is simulated and only incremented every 200 ms. (as opposed to every 1/60 th of a second when running independently).
The EKTR 2000 is a computer program designed to operate in the context of an operating system. The program's purpose is to perform specific tasks, such as processing data or controlling a system. The program's structure is based on a series of instructions and functions that interact with each other to achieve the desired outcome.

The program's main function is to read and analyze data from various sources, performing calculations and processing the information accordingly. The output of the program is then used to make decisions or perform actions based on the processed data.

The program's interface is designed to be user-friendly, allowing for easy interaction and control. The program is written in a high-level programming language, ensuring that it is flexible and adaptable to different situations.

Overall, the EKTR 2000 is a powerful tool for processing and analyzing data, providing a robust and reliable solution for various computational tasks.
Identification

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

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

Usage

As supervisor or library entries:

\[
\begin{align*}
\text{TSX RDFLXA,4} & \quad \text{optional(TIA =HRDRLXA)} \\
\text{PZZE LOC,,n'} & \\
\text{TSX WRFLXA,4} & \quad \text{optional(TIA =HWRFLXA)} \\
\text{PZZE LOC,,n'} & \\
\text{TSX WRFLX,4} & \quad \text{optional(TIA =HWRFLX)} \\
\text{PZZE LOC,,n'} & \\
\end{align*}
\]

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

WRFLXA will print n words beginning at location LOC. It does not add a carriage return at the end of the line and does not delete trailing blanks.

WRFLX will print through the last non-blank character within the n words beginning at location LOC. Trailing blanks will be deleted and a carriage return inserted after the last non-blank character.

As library subroutines:

RDFLX:

\[
\begin{align*}
\text{TSX RDFLX,4} & \\
\text{PZZE LOC,,n'} & \\
\end{align*}
\]
RDFLX will read a line from the console using RDFLXA. It will then strip the break character from the line, pad any remaining characters up to \( n \) words with blanks, and move the \( n \) words into core beginning at location LOC. If \( n \) is less than the number of words read, the characters in excess will be lost (n.LE.14).

RDFLXB, RDFLXC:

\[
\begin{align*}
\text{MAD:} & \quad A = \text{RDFLXB.(LOC,K)} ; \quad A = \text{RDFLXC.(LOC,K)} \\
\text{FORTRAN:} & \quad A = \text{RDFLXB(LOC,K)} ; \quad A = \text{RDFLXC(LOC,K)} \\
\text{FAP:} & \quad \text{TSX RDFLXB,4 or TSX RDFLXC,4} \\
& \quad \text{PZE LOC} \\
& \quad \text{PZE K} \\
& \quad \text{STØ A}
\end{align*}
\]

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

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

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

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

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

Set the console character mode switch.
SETFUL, SETBCD

Purpose

To set the console character mode switch.

Usage

As supervisor or library entry:

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

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

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

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

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

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

Restrictions

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

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

Console output
PRNTP, PRNTPA, PRNTPC

Purpose

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

Usage

As library subroutine:

MAD:

    EXECUTE PRNTP. (MESS)
    ...
    VECTOR VALUES MESS=$hollerith string$,7777777777777K

FORTRAN:

    CALL PRNTP (nH hollerith string)

PRNTP, the hollerith string up to the fence prints, on the user's console, 14 words per line. The string may be of any length. If the fence is (7777777777777)8, there will be no carriage return at the end of the message. The fence which Fortran automatically supplies is (7777777777777)8.

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

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

Inter-user communication
WRMESS, RDMESS, ALLOW, FORBID

Purpose

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

Method

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

Usage

To send a message:
As supervisor entry:

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

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

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

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

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

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

1 - The specified receiver is not a current user of CTSS. (I.e. logged in).
2 - The receiver's input buffers are full.
3 - The receiver has not given permission for the sender to send messages to his input buffer.

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

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

```
TSX  RDMESS,4
PZE  LOC,'n'
ALPHA OPN  EMPTY
Normal return
```

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

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

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

```
TSX  ALLOW,4
PZE  =HPROBN
PZE  =HPROGN
```

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

If PROGN is zero, all programmers on problem number PROBN may send messages.
If PROBN is zero, programmer PROGN may send messages, whatever his problem number.
If both PROBN and PROGN are zero, any programmer may send messages.
To lock everyone out:
   As supervisor entry:

   TSX FORBID,4 (TIA =HFORBID)

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

Slave remote consoles
ATTCON, RELEAS, SNOLIN, SNDLNA, REDLIN, SLAVE, SET6, SET12

Purpose

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

Definitions and Conventions

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

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

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

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

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

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

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

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

To attach a console:
As supervisor entry:

TSX ATTCON,4 (TIA =HATTCON)
PZE CONSL

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

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

To release a console:
As supervisor entry:

TSX RELEAS,4 (TIA =HRELEAS)
PZE CONSL

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

To send a line:
As supervisor entry:

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

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

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

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

CONSOL If CONSOL is '(HOME)', the line is sent to the user's home console output buffer. If the designated console is not attached to the user, return is to the normal return with the AC non-zero.
ALPHA If the output buffers at the designated console are full and ALPHA is zero, the user is placed in OUTPUT WAIT status. If ALPHA does not contain zero, control is immediately returned to ALPHA.

To read a line:
As supervisor entry:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSX</td>
<td>REDLIN,4</td>
</tr>
<tr>
<td>PZE</td>
<td>CONSOL</td>
</tr>
<tr>
<td>PZE</td>
<td>LOC,,'n'</td>
</tr>
<tr>
<td>ALPHA</td>
<td>OPN EMPTY</td>
</tr>
<tr>
<td></td>
<td>normal return</td>
</tr>
</tbody>
</table>

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

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

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

To create a slave:
As supervisor entry:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSX</td>
<td>SLAVE,4</td>
</tr>
<tr>
<td>PZE</td>
<td>CONSOL</td>
</tr>
<tr>
<td>PZE</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>normal return</td>
</tr>
</tbody>
</table>

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

MODE There are three distinct slave modes (11,00,10) providing eight combinations for any single console. The word at MODE is interpreted as three pairs of letters. If any of the pairs is recognized, the console is made to slave accordingly. If MODE does not contain a recognizable pair, the console is unslaved.
To set the character mode:
As supervisor entry:

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

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

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

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

Purpose

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

Usage

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

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

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

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

.READ and (CSH) read lines from the console according to the format FMT and LIST.

SWITCH if non-zero indicates that the format is enclosed in parentheses and stored forward.

OPS may be indexing or other instructions.

LIST is the beginning location of the LIST.

ENDLST is the final location of the LIST.
DIR if zero the format is stored forwards. If 1, the format is stored backwards.

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

BUF is the first (lowest) location of an array into which data will be read.

n is an integer indicating the number of words to be read into the array BUF.

.LOOK reads one line from the console according to the format specified by FMT. The next time a read statement is encountered, the same input will be processed. If more than one line of input is requested by the format, the same line will be used.

.SCRDS reads a line from the console and stores the number of words requested into the buffer.
Identification

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

Purpose

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

Usage

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

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

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

(SPH) and (SPHM) are synonymous.

.PRINT and .COMNT are synonymous.

.PRINT and (SPH) type on the console the output as requested by the format FMT and LIST. The maximum line length is 22 words.

SWITCH if non zero indicates that the format is stored forward.

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

OPS may be any indexing instructions.

LIST(,t) is the beginning location of the list.
ENDLST is the final location of the list.

DIR if zero, the format is stored forwards. If 1, the format is stored backwards. If anything else, a symbol table is implied. See MAD manual for details.

BUF is the first (lowest) location of an array containing BCD information.

n is the number of words in the array BUF.
Identification
Print a comment
.PCOMT

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

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

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

n is the number of BCD words to be printed.
Identification

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

Purpose

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

Reference

MAD Manual, Chapter II, Section 2.16

Usage

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

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

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

A refers to a single element.

LIST1 refers to the block of data.

LISTN refers to the end of a block of data.

TXH 0 marks the end of the list.

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

X = -12.4

Blocks are labeled as such and are printed using a block format. Elements of three and higher dimensions will be labeled with the equivalent linear subscript. If dummy variables are included, the specific values assigned to such variables and expressions during execution will be preceded by '...'.
PRINT RESULTS (.PRSLT) causes the output to be numeric (that is, integer or floating point).

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

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

Read without list or format
.RDATA, .RPDTA

Purpose

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

Reference

MAD Manual, Chapter 11, Section 2.16 and 1.1

Usage

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

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

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

READ DATA reads information from lines typed on the user's console. The values to be read and the variable names are typed in a sequence of fields of the following form
  V1 = n1, V2 = n2, ......., Vk = nk *
where the V are variable names and the ni are the corresponding values. Reading is continued from line to line until the terminating mark '*' is encountered.

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

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

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

Purpose

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

Usage

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

To write a file on the disk:
Supervisor and library entry:

TSX .DUMP,4 optional(TIA =H.DUMP)
OPN FILNAM
PZE LOC,,'n'

OPN establishes the mode of the file; PZE is
temporary, PON is permanent, PTW is R1, PTH is
R2.

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

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

To read a file:
Supervisor and library entry:

TSX .LOAD,4 optional (TIA =H.LOAD)
PZE FILNAM
PZE LOC,,'n'
SLW n

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

m will contain the number of words actually
loaded, as a full word integer.
Corresponding library subroutine:

MAD:        EXECUTE DSKDMP. (FILNAM, FIRST, N)
           EXECUTE DSKLOD. (FILNAM, FIRST, N)
           M = DSKLOD. (FILNAM, FIRST, N)

FORTRAN:   CALL DSKLOD (FILNAM, FIRST, N)
           CALL DSKDMP (FILNAM, FIRST, N)
           A = DSKLOD (FILNAM, FIRST, N)

Core will be loaded or dumped from FIRST-n+1 through FIRST. If the number of words \( m \) in the file is less than \( n \), the file will load into the block of core through FIRST-n+m.
Identification

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

Purpose

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

Method

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

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

Restrictions

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

Reading by calls to the supervisor entries instead of the library subroutines means that buffers are not automatically assigned and errors cause execution of supervisor error procedures rather than the library error procedures.

Usage

To open a file:

as supervisor or library entry:

TSX .SEEK,4 optional (TIA =H.SEEK)
PZE FILNAM
PZE BUF1

as library subroutine:

FAP, MAD, or FORTRAN,
EXECUTE SEEK. (FILNAM,-BUF1-)

BUF1 is initial location of 432 word block of core to be used as buffer. If BUF1 is not specified to the library subroutine, a buffer will be assigned by extending the memory bound if core space permits. If no buffer space is available, the library error procedure will be initiated.

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

.SEEK does not call SRCH.

To read a record:
as supervisor or library entry:

```
TSX .READK,4  optional (TIA =H.READK)
PZE FILNAM
PZE LOC,t,n
PZE EOF
...
EOF SLW WC
```

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

t of non zero means skip n words without transmission.

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

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

as a library subroutine:

FAP or MAD

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

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

FORMAT is the location of the format by which the variables in LIST will be edited by (10H).

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

DREAD reads logical records and edits them through (10H). Each call to DREAD reads at least one logical record; however, the format may require the reading of more than one logical record. If the file is line-marked, the line marks delineate the logical records. If the file is not line-marked, the logical records are 14 words. If fewer words are requested than are available in the record, the excess of the record is lost. The format may specify the reading of more than one record; however, if more words are requested from a specific record than are available within that record, the library error procedure is initiated.

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

To close an input file:

as supervisor or library entry:

    TSX .ENDRD,4 optional (TIA =H.ENDRD)
    PZE FILNAM

as library subroutine:

    EXECUTE ENDRD. (FILNAM)

ENDRD will delete the file from the active file table and release the buffer.
EXECUTE END0. (LIMITED)

END0 will execute the LEAVE from the section till

** Rapport, 6:53 **

** Rapport, 6:53 **
Identification

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

Purpose

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

Method

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

Restrictions

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

If the supervisor entries (ASSIGN, .APEND and .WRITE) are used and the program is terminated without going to FILE, the file will be lost. Any disk errors initiate supervisor disk error procedures.

Usage

To open a new file:

as supervisor or library entry:

TSX .ASIGN, 4 optional (TIA =H.ASIGN)
OPN FILNAM
PZE BUFL

as a library subroutine:

FAP, MAD or FORTRAN

EXECUTE ASSIGN. (FILNAM, -BUFL-)

UPN defines the mode: PZE is temporary, PON is permanent, PTW is R1, PTH is R2. The library subroutine will define the mode as permanent.

BUF1 is the initial location of a 432 word block of core to be used as a buffer. If BUF1 is absent from the library subroutine call, a buffer will be assigned by expanding the memory bound if core space permits. If no buffer space is available, the library error procedure will be initiated.

ASSIGN calls SRCH which assigns a buffer if necessary and maintains an active file table and buffer assignment table. This allows terminal subroutines to close active files properly.

.ASIGN does not call SRCH.

To open an old file in order to add information: as supervisor or library entry:

```plaintext
TSX .APEND,4  optional (TIA =H.APEND)
PZE FILNAM
PZE BUF1
```

as a library subroutine:

```plaintext
FAP, MAD or FORTRAN
EXECUTE APPEND. (FILNAM,-BUF1-)
```

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

To write a file:

as supervisor or library entry:

```plaintext
TSX .WRITE,4  optional (TIA =H.WRITE)
PZE FILNAM
PZE LOC,,'n'
```

n is the number of words to be written into file FILNAM beginning at location LOC.
as a library subroutine:
FAP, MAD or FORTRAN

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

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

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

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

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

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

WRITE will write a fixed length record without line-marks. If the LIST is shorter than the fixed length, blanks will be filled in. If the LIST is longer than the fixed length, only the first words are written and the excess is lost. The fixed length is assumed to be 14, unless set by SETV8(F). WC will contain an integer equal to the number of words written, the sign will be minus.

WC when WC is returned, it is the proper integer format for the language of the calling program. Fortran, however may have some difficulty as a result of the mode of the function convention. Fortran users should equivalence WC with an integer variable.
To close an output file:
as supervisor or library entry:

```
TSX .FILE,4 optional (TIA =H.FILE)
PZE FILNAM
```
as a library subroutine:
FAP, MAD, or FORTRAN

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

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

.FILE should be used if the file was written by the .WRITE supervisor entry. .FILE is the same as FILE except for the safety feature in the terminal subroutines, i.e. files written by .WRITE will be lost if not .FILE'd before EXIT is called.
Identification

Addressable disk files
.RELRW

Purpose

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

Usage

To open an addressable file:
As supervisor or library entry:

```
TSX .RELRW,4 optional (TIA =H.RELRW)
PZE FILNAM
PZE BUF1
```

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

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

To read or write an addressable file:
As supervisor or library entry:

```
TSX .READK,4
PZE FILNAM,,reladr
PZE LOC,,n'
PZE EOF
optional(TIA =H.READK) (TIA =H.WRITE)
```

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

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

EOF Location to which control will be transferred upon encountering an end of file.
To close an addressable disk file:
As supervisor or library entry:

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

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

Set the length of fixed length records
SETVBF, SETVB

Purpose

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

Usage

As a library subroutine:

MAD, FAP, or FORTRAN

\[ B = \text{SETVBF}(N) \]

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

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

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

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

Purpose

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

Usage

To search active file table:

TSX SRCH,4
PZE FILNAM
      not found
      found

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

found returns with the status of FILNAM in the address of the AC and the buffer number (1-10) in the decrement of the AC. If the file is not using an assigned buffer, the buffer number is zero. Write status is 1; read status is 2.

To assign a buffer:

TSX BLK,4
      error return
      normal return

BLK searches the buffer assignment table. If there are no free buffers and there are fewer than 10 assigned buffers, an attempt is made to extend the memory bound.

error return is taken if there are already 10 buffers assigned or the attempt to extend the memory bound was unsuccessful.

normal return is taken with the address of the buffer in the address of the AC and the number of the buffer (1-10) in the decrement of the AC.
To enter a file in the active file table:

TSX FLK,4
PZE FILNAM
PZE status,, number
    error return
    normal return

status is 1 if writing, 2 if reading. The status word is stored in the first free space in the active file table.

number is the buffer number. If number is non-zero, a pointer to the file in the active file table is placed in the assigned buffer table.

error return is taken if there are 10 active files already.

To close one file:

TSX ENDF,4
PZE FILNAM

FILNAM is closed by either .ENDRD or .FILE, the buffer is free, and the file is removed from the active table.

To close all files:

TSX CLOUT,4

All the files in the library's active file table are closed by calls to either .ENDRD for the read files or .FILE for the write files.
Identification

Generate file of zeros
.END

Purpose

To create a new file which contains n zeros.

Usage

As supervisor entry:

TSX .CLEAR,4 (TIA =H.CLEAR)
OPN FILNAM,'n'

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

OPN specifies the mode of the file: PZE is temporary, PON is permanent, PTW is R1, PTH is R2.
Identification

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

Purpose

Files may be opened on any I/O storage device for reading, writing or reading and writing. A buffer may be assigned if needed and priorities may be set for different files.

Method

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

Usage

OPEN:

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

STATUS may be 'R' for read, 'W' for write or 'RW' for read-write. (justification is not significant).

MODE specifies the mode of a new file to be created and may be the inclusive logical or of any of the following octal values. If MODE is not specified, a permanent file will be created.

000 - Permanent
001 - Temporary
002 - Secondary
004 - Read-only
010 - Write-only
020 - Private
040 - Linkable
100 - Protected

DEVICE is pertinent only when creating a new file and it specifies which I/O device is desired. If DEVICE is zero or not specified, the system will assign a device.
1 - Low speed drum
2 - disk
3 - Tape

Error codes:

03. File is already in active status
04. More than ten active files
05. $STATUS$ is illegal
06. 'LINKED' file not found
07. File to which link is made is not 'LINKABLE'
08. File in 'PRIVATE' mode
09. Attempt to write a 'READ-ONLY' file
10. Attempt to read a 'WRITE-ONLY' file
11. Machine or System error
12. File not found in U.F.D.
13. Illegal device specified
14. No space allotted for this device
15. Space exhausted for this device
16. File currently being restored from tape
17. Input/Output error, see AG.4.06

Assign a buffer:

BUFF.$( NAME1$, $NAME2$, BUF(N)...N)

BUFF In general a buffer should be assigned to an open file for reading or writing.

BUF The buffer space should be specified in block notation as the beginning location of the buffer and the size. The size must be large enough to accommodate a physical record from the I/O device.

N is the buffer size and 432 seems to be the going size.

Error codes:

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

Set priority:

SETPRI.(PRIOR)

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

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

Error codes:

Standard error codes. See section 4.06

Read:

**RDFILE**($ \text{NAME1}$,$ \text{NAME2}$,$ \text{RELLOC,A(N)}...\text{N,EOF,EOFCT}$)

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

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

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

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

Error codes:

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

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

WRFILE initiates the I/O necessary to move N words from the array A(N) thru A(1) into the file NAME1 NAME2.

RELLOC is the relative location within the file where writing is to begin. If RELLOC is zero, writing will begin after the last word written in the file. If RELLOC is zero on the first call, writing will begin at the location following the last word of the file. RELLOC may not be larger than the current length of the file.

EOF is the location to which control will be transferred if the N words to be written would have to be written through the end of file (i.e., if part of the record could be contained within the file and the other part would extend to outside the file). This does not occur when appending to the file with a RELLOC of zero where entire records are placed at the end of the file.

EOFCT is an integer variable into which the I/O system will store the number of words actually to be written when control was transferred to EOF. An FCHECK is necessary as with any WRFILE.

Error codes:

03. File is not an active file
04. File is not in write status
05. No buffer assigned to this file
06. Allotted space exhausted for this device
07. Previous I/O out of bounds (membnd changed)
08. Input/Output error, see AG.4.06

Truncate:

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

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

03. File is not an active file
04. File is not in write status
05. No buffer assigned to this file
06. Previous I/O out of bounds (membnd changed)
07. RELLOC larger than file length
08. Input/Output error, see AG.4.06

Check:

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

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

Error codes:

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

Close:

CLOSE($ NAME1$, -$ NAME2$)

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

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

Error codes:

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

Load a file into a free area of core
LDFIL

Purpose

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

Usage

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

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

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

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

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

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

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

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

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

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

Change the mode or the name of a disk file.
CHMODE, RENAME, .RENAME

Purpose

To be able to change the mode or the name of a disk file.

Usage

To change mode:

as library subroutine:

FAP:   TSX CHMODE,4  
PZEO FILNAM
PZEO MODE

FORTRAN:  A = CHMODE (FILNAM, MODE)
MAD:    A = CHMODE. (FILNAM, MODE)

MODE is 0 for temporary, 1 for permanent, 2 for read only R1, 3 for read only R2.

A will be zero if successful or will contain the disk error code if the file cannot be found or is R2 read-only.

To change name and/or mode:

as supervisor or library entry:

FAP:   TSX .RENAME,4 optional (TIA =H.RENAME)
       OPN NEWNAM,,FILNAM

To change name:

as library subroutine:

MAD:   A = RENAME.(NEWNAM, FILNAM)
FORTRAN: A = RENAME (NEWNAM, FILNAM)

.RENAME replaces in the current file directory the file name specified by FILNAM by the new name located at NEWNAM. R2 read only files can not be changed in any way by this entry. All other classes may be changed by .RENAME. The standard supervisor error procedure may be followed.

OPN specifies the mode of NEWNAM. PZE is temporary, PON is permanent, PTW is R1, and PTH is R2.
RENAME calls .RENAM so that the AC or A will be zero
if successful or will contain the error code
if unsuccessful. Any deletable files named
NEWNAM will be deleted (except if NEWNAM =
FILNAM). The old file is given the new name
and the mode is unchanged.
Identification

Delete file from file directory
DELETE, ERASE, .DELETE, .ERASE

Purpose

To delete a file name from a file directory and either make the tracks available for some other use or leave the tracks in the "in use" state until the next system loading.

Usage

To delete a file:
- as supervisor or library entry:

FAP: TSX .DELETE,4 optional (TIA =H.DDELETE)
    PZE FILNAM

- as library subroutine:

MAD: EXECUTE DELETE,(FILNAM) or A = DELETE,(FILNAM)
FORTRAN: CALL DELETE (FILNAM) or A = DELETE (FILNAM)

.DDELETE does not require buffer space but it does read each track of the file specified in order to delete it and update the track usage table. The FILNAM is removed from the current file directory and the tracks are made available for other use. Read only R1 and R2 files may not be deleted by this routine. Any error will invoke the supervisor error procedure.

DELETE calls .DDELETE. If FILNAM is R1 read-only mode, a message will be typed asking if the file should really be deleted. The user must type yes or no and a carriage return. R2 mode files may not be deleted.

Upon return, if the file is not deleted, the AC and A will contain a 1, otherwise the AC and A will be zero.

To erase just the name:
- as supervisor or library entry:

    TSX .ERASE,4 optional (TIA =H.ERASE)
    PZE FILNAM

- as library subroutine:

    MAD: EXECUTE ERASE,(FILNAM) or A = ERASE,(FILNAM)
    FORTRAN: CALL ERASE(FILNAM) or A = ERASE(FILNAM)
ERASE is the same as DELETE (.ERASE = .DELETE) except that the tracks are not made available for other use and the user's track count is not updated until the next time the disk is loaded. Note, this description was true for the old file system but is no longer true. ERASE is no longer different from DELETE but the write-up remains for compatibility.
Identification

Switch current file directory
COMFIL, COMFL, TSSFIL, USRFIL

Purpose

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

Usage

As supervisor or library entry:

```
CAL N
TSX COMFIL,4 optional (TIA = HCOMFIL)
PZE BUSY
```

N contains the integer of the common file
directory desired. Zero is the user's file
directory.

BUSY is the location to which control will be
transferred if the requested file directory is
busy. The logical AC will contain the problem
number and the MQ, the programmer number in
BCD of the user currently using the requested
file. Note, this is no longer true but the
calling sequence is preserved for
compatibility. Control will always return to
2,4.

Note: temporary files in the current file
directory are deleted by directory switching
if the current file directory is not the
user's file directory. All files in the
current file directory are reset.

As library subroutine:

```
MAD:  A = COMFL.(N)    or EXECUTE COMFL.(N)
FORTRAN: A = COMFL(N)    or CALL COMFL(N)
```

A is zero if switching has been performed or A
will contain the programmer number of the user
currently using the requested file directory.
To switch to the system library:

As supervisor or library entry:

```
TSX  TSSFIL,4  optional (TIA = HTSSFIL)
```

TSSFIL switches the user to the system library file directory. This directory is composed of links to certain files in the system file directory which are in read-only, protected mode. The track quota of the TSSFIL directory is 0, so that the user may not create files after a call to TSSFIL.

```
TSX  USRFIL,4  optional (TIA = HUSRFL)
```

USRFIL switches from the system library file back to the current file directory.

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

A call to TSSFIL causes all active files in the current file directory to be RESET (i.e., active write files are lost and temporary read files are deleted).
Identification

Query file status
FSTAT, .FSTAT

Purpose

To obtain the mode and word count of a specified file.

Usage

As supervisor or library entry:

TSX .FSTAT,Y optional (TIA =H.FSTAT)
PZE FILNAM

As library subroutine:

MAD: A = FSTAT.(FILNAM)
FORTRAN: A = FSTAT(FILNAM)

.FSTAT If the file is not found, the supervisor disk error procedure is initiated.

Upon return from FSTAT, the AC or A will contain zero if the file was not found. Otherwise, it will contain a word of the form OPN WDCNT.

OPN is the mode of the file, PZE is temporary, PON is permanent, PTW is R1, PTH is R2.

WDCNT (the address and tag) is the word count of the file.
To obtain the image, I work count on a special file

The

AX, SUPPLEMENT OR ORDINARY ETC.

TABLE I

AX, ESTAT, ETC.

AX, ESTAT, ETC.

AX, ESTAT, ETC.

AX, ESTAT, ETC.

AX, ESTAT, ETC.
Identification

Get the name of next file
GTNAM

Purpose

If a program creates an unknown number of files, assigns them sequential primary names, and uses them in a push down list, it is necessary to be able to determine the next available primary name. GTNAM performs the search for the next available name.

Usage

As library subroutine:
FAP, MAD or FORTRAN

\[ A = GTNAM.(\text{CLASS}) \]

GTNAM searches for the first file which does not exist in the series of primary names \( ...001 \) thru \( ...999 \) with secondary name \( \text{CLASS} \); then tries to delete the following file, if any; and returns in \( A \) the first BCD primary name available in the series.
Identification
Drop files from active status
.RESET

Purpose
To remove all user's files in active status from the supervisor's list of active files.

Usage
As supervisor or library entry:

TSX .RESET,4 optional (TIA =H.RESET)

.RESET will remove all the user's active files from the active status. All files in active write status will be lost. All temporary files in active read status will be deleted. This call will not remove the user's active files from the library subroutines' list of active files.
Identification

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

Purpose

With the new I/O system, as with the old, it is possible to change the mode or name of a file, to delete a file, or query the system about the status of a file.

Usage

Change:

CHFILE.($OLDNM1$, $OLDNM2$, $NEWMOD$, $NEWM1$, $NEWM2$)

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

NEWMOD is the desired mode of the file.

NEWM1 NEWM2 is the desired name of the file.

Error codes:

03. Attempt to change M.F.D. or U.F.D. file
04. File not found in U.F.D.
05. 'LINKED' file not found
06. File to which link is made is not 'LINKABLE'
07. Attempt to change 'PRIVATE' file
08. Attempt to change 'PROTECTED' file of another user
09. Temporary file would overflow space allotted for device
10. File already exists with name 'NEWM1 NEWM2'
11. Machine or System error
12. File in active status

Delete:

DELFIL.($NAME1$, $NAME2$)

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

Error codes:

03. File not found in U.F.D.
04. 'LINKED' file not found
05. file to which link is made is not 'LINKABLE'
06. File is 'PROTECTED'
07. Machine or System error
08. File in active status
Status:

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

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

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

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

DEVICE is
1 Low speed drum
2 Disk
3 Tape

Error codes:

03. File not found in U.F.D.
04. 'LINKED' file not found
05. File to which link is made is not 'LINKARLE'

Size:

STORGE.(DEVICE,ALLOC,USED)

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

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

Error codes:

03. Illegal DEVICE specified
04. Machine or System error
Identification

Historic File System Error Procedure

Purpose

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

Usage

Standard:

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

ILLEGAL CALL TO XXXXXXX. NO ERROR RETURN SPECIFIED

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

User's option:

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

<table>
<thead>
<tr>
<th>Illegal calling sequence</th>
<th>PZE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too many active files (.G.10)</td>
<td>PZE 2</td>
</tr>
<tr>
<td>User not found in Master File Directory</td>
<td>PZE 3</td>
</tr>
<tr>
<td>Available space on module exhausted</td>
<td>PZE 4</td>
</tr>
<tr>
<td>File not found</td>
<td>PZE 5</td>
</tr>
<tr>
<td>Allocated track quota exhausted</td>
<td>PZE 6</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

h. Tried to rename read-only class 2.

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

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

Library disk error procedure
SETERR, SNAP, RECOUP

Purpose

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

Method

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

Usage

SETERR:

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

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

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

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

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

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

If no argument is provided to SETERR, the standard error procedure will be reinstated.
Every call to SETERR supercedes the previous one.

RECOUP:

CALL RECOUP (ERCODE, IR4,-IND-)

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

ERCODE contains the logical AC from the offended disk routine, or the error code from (10H).

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

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

IND contains the contents of the sense indicators at the time of the error in the disk routine. This argument is not present in the call from (10H).

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

SNAP:

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

End-of-file procedure for library subroutines
EUFXIT, SETEUF, WRDCNT

Purpose

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

Method

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

Usage

EUFXIT:

The library routines call EUFXIT by:

```
TSX EUFXIT,4
PZE FILNam
```

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

SETEUF:

```
FAP: TSX SETEUF,4
-PZE EUF-
-PZE FILNam1-
-PZE FILNam2-
```

MACH: EXECUTE SETEUF.(-EUF-, -FILNam1-, -FILNam2-)
FORTRAN: CALL SETEUF(-N-, -FILNam1-, -FILNam2-)

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

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

N must be set by an ASSIGN statement in Fortran
i.e. ASSIGN 1 TO N
    GO TO N, (1,2)
    1 ASSIGN 2 TO N
    
    2 eof procedure

FILEM1, FILEM2 are the locations in which NAME1 and
NAME2, respectively, will be stored by EOFXIT. If
FILEM1 is missing, the logical tape number
will be stored in FILEM1. If both FILEM1 and
FILEM2 are missing, a single argument will be
assumed to be EOF or N.

WRDCNT:
    FAP: TSX WRDCNT,4 OR TSX WRDCNT,4
    PZE LOC STØ LOC

MAD OR FORTRAN: CALL WRDCNT (LOC)

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

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

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

Purpose

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

Usage

EXIT, CLKOUT and ENDJOB are synonymous.

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

The message "EXIT CALLED. PM MAY BE TAKEN" will be printed.

EXECUTE DUMP.
EXECUTE PDUMP.

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

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

EXECUTE EXITM.

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

Error Exit from Math Library Routines
LDUMP

Purpose

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

Usage

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

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

ARG1 contains the first argument to the math library subprogram.

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

NAME contains the BCD name of the offending routine.

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

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

dump

Note: This error message is printed to the screen if the program encounters an error. The version of DUMP used in the printer is a call to EXIT but the error may prevent the program from recovering action.

The calling sequence to DUMP which is given on the form:

TO EXIT FROM ROUTINE

TO ACCEPT ROUTINE

NAME continue the BEO name of the offending routine.

ARG1 continuing the second argument. If any. In the
ARGS continuation the first argument.

LIBRARY continuation of the list of libraries.

LIBRARY subprogram.

MAIN continue the main routine.

exit from program.

include file containing the control macro.

calculation.
Identification

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

Purpose

There are three different ways that errors from the I/O system can be handled. First, if the user does nothing, the I/O system will print a standard message and call DORMNT. Second, the user may call FERRTN to establish a standard error return for all I/O system errors. Third, every call to the I/O system will accept two additional arguments which specify an error return and a location into which the error code will be stored.

Usage

Standard:

If an error is encountered by the I/O system and the user has not supplied an error return, the I/O system will type a standard message and call DORMNT so that debugging tools may be used. The typed message will include the information available from IODIAG.

Single return:

OLDERR = FERRTN(ERRLOC)

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

ERRLOC is the location to which control should be transferred if the I/O system detects an error. Upon entry to ERRLOC, index register 4 will contain the value set by the call to the I/O system that caused the error to be detected. To continue execution by ignoring the I/O call, transfer to 1, 4. To continue execution by repeating the I/O call, transfer to 0, 4.

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

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

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

Diagnostic:

IUDIAG. (A(7) ... 7)

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

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

PRINTER, or PRINTER command

PRINTER The routine PRINTER or the command PRINTER may be called after an error in the I/O system in order to print the information that is available from IUDIAG. In other words, PRINTER is a small routine which calls IUDIAG and formats and prints the information.

Error codes:

Standard error codes:

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

001. Illegal calling sequence or Protection violation
002. Unauthorized use of privileged call
100. Error reading or writing U.F.D. or M.F.D.
101. U.F.D. or M.F.D. not found, Machine error
Input/output error codes:

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

1. Parity error reading or writing file
2. Fatal error reading or writing file, cannot continue
3. Available space exhausted on this device
4. Tape file not mounted or not available
Identification

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

Purpose

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

Usage

MAD:  PUNCH FORMAT FMT, LIST
       WRITE BCD TAPE N, FMT, LIST
FORTRAN:  PUNCH FMT, LIST
          WRITE OUTPUT TAPE N, FMT, LIST

The FAP calling sequence compiled for MAD programs is of the form:
  TSX .TAPWR,4
  STR N
  STR FMT,,DIR
  OPS
  STR LIST,,ENDLST
  OPS
  STR 0

The FAP calling sequence compiled for FORTRAN programs is of the form:
  CAL N
  TSX (STH),4
  PZE FMT,,SWITCH
  OPS
  LDQ LIST
  STR SWT
  OPS
  TSX (FIL),4

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

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

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

  OPS may be indexing instructions.
SWITCH is zero if the format is stored backwards and non-zero if the format is stored forward.

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

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

SWT if zero with 1 format, the value is taken from the decrement of location LIST. If non zero with 1 format, the value is taken from the address of location LIST.

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

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

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

Purpose

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

Usage

MAD: READ BCD TAPE N, FMT, LIST
FAP: TSX .TAPRD,4
STR N
STR FMT,.DIR
OPS
STR LIST,.ENDLIST
OPS
STR 0

FORTRAN: READ INPUT TAPE N, FMT, LIST
FAP: CAL N
TSX (TSH),4
P2E FMT,.SWITCH
OPS
STR
STQ LIST
OPS
TSX (RTN),4

(TSH) and (TSHM) are synonymous.

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

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

OPS may be indexing instructions.

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

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

LIST,.ENDLIST are standard LIST processing (see MOVEI).
Identification

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

Purpose

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

Restrictions

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

Usage

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

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

N contains in the decrement the number of pseudo tape.

OPS may be indexing instructions.

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

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

Purpose

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

Restrictions

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

Usage

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

MADTRN: BACKSPACE N
         ENDFILE N
         REWIND N

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

FORTRAN: BACKSPACE N
          END FILE N
          REWIND N

FAP: CAL N CAL N CAL N
     TSX (BST),4 TSX (EFT),4 TSX (RTW),4

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

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

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

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

Program status
DEAD, DORMNT, GETILC, FNRTN

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

Usage
DEAD:
as supervisor or library entry:

    TSX DEAD,4 optional (TIA =HDEAD)

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

DORMNT: as supervisor or library entry:

    TSX DORMNT,4 optional (TIA =HDORMNT)

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

GETILC: as supervisor entry:

    TSX GETILC,4 (TIA =HGETILC)

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

FNRTN: as supervisor entry:

    TSX FNRTN,4 (TIA =HFNRTN)

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

Periodic Dormancy
SLEEP

Purpose

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

Usage

As supervisor or library entry:

\[ \text{CAL } = n \]
\[ \text{TSX SLEEP,4 } \text{ optional (TIA } = \text{HSLEEP)} \]

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

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

Interrupt levels
GETBRK, SETBRK, SAVBRK

Purpose

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

Restrictions

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

Method

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

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

Usage

SETBRK:
as supervisor or library entry:

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

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

SAVBRK:
as supervisor or library entry:

TSX SAVBRK,4 optional (TIA =HSAVBRK)

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

TSX GETBRK,4 optional (TIA =HGETBRK)

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

Storage Map
STU MAP

Purpose

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

Usage

As library subroutine:

    TSX STU MAP, 4

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

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

Purpose

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

Method

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

Usage

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

TSX .SETUP,4

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

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

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

as supervisor or library entries:

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

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

(LFTM) leaves the floating-point trapping mode.

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

Memory allotment
GETMEM, SETMEM, GMEM, SMEM, EXMEM

Purpose

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

Method

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

Restrictions

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

Usage

As supervisor or library entries:

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

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

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

As library subroutines:

MAD or FORTRAN:

```
A = GMEM.(1)
A = SMEM.(J)
```

FAP: TSX GMEM,4 TSX SMEM,4

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

J contains an integer giving the memory bound desired.

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

SMEM sets the memory bound to the value desired.

To extend memory bound:

As library subroutine:

MAD, FORTRAN or FAP:

\[ A = \text{EXMEM}(\text{INC}) \]

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

A Upon return, A will contain the new memory bound which is the sum of the old memory bound and the increment in INC. If the sum is greater than \((777777)_8\) or if the prefix of the argument is not PZE, TSX or TXH, return is made with A and the AC set to zero and the memory bound is not extended.
Identification

Free or erasable storage management
FREE, FRER, FRET

Purpose

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

Usage

As a library subroutine:

AED: \[ X = \text{FREE}(N) \], \[ X = \text{FRER}(N) \], \[ X = \text{FRET}(N, X) \],

FAP: \[ \text{TSX FREE}, 4 \] \[ \text{TSX FRER}, 4 \] \[ \text{TSX FRET}, 4 \]
\[ \text{PZE N} \] \[ \text{PZE N} \] \[ \text{PZE N} \]
\[ \text{STA X} \] \[ \text{STA X} \] \[ \text{STA X} \]

\( N \) contains an integer specifying the size of the block of storage.

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

FREE will find a block of storage either from free storage or by extending memory bound. If more space is requested than can be found, the following message will be printed, and EXIT is called:

'nnnnn LOCATIONS OF FREE STORAGE ARE UNAVAILABLE
(nn.nn is an octal number.)

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

FRET returns storage to free storage. If a block of storage being returned overlaps memory bound or any block previously returned, the following message is printed and EXIT is called:

** ILLEGAL CALL OF FRET, BLOCK rrrrr SIZE nnnnn'
(rrrrrr is a pointer to the block, nnnnn is size; both in octal)
The problem of optimizing the amount of data space
needed for the program is a common problem. The
program must temporarily store data in a common pool and
then store it back when it is no longer needed.

As a simple example:

```
AREA X-FREE(N), Y-FREE(N) X=FREE(N), X=FREE(N)
PARM X, FREE(N) X=FREE(N) Y=FREE(N)
FREE: X FREE(N)
FREE X Y FREE(N)
```

In concatenating the integer and floating components of the
block of storage.

It is important to locate the start of the block of
storage (FREE) and the end of the block of storage. If a
storage location is a block of FREE, the block could be
concatenated with the previous block of FREE and the
storage location would not be available.

```
FREE   X Y FREE(N)
FREE   X Y FREE(N)
FREE   X Y FREE(N)
```

LBRK serves the same function as FREE space.

If not enough space is available, return with
```
```

LRTK returns a flag to indicate if the storage provided
is not enough space in storage, return with
```
```

If you provide enough space but a fixed area is
available, return with
```
```

If you provide enough space but a block of
storage is not available, return with
```
```

If you provide enough space but a fixed area is not
available, return with
```
```

If you provide enough space but a block of
storage is not available, return with
Identification

Reset file-wait return
TILOCK

Purpose

A field called ILOCK exists within the UFD for each file. This field contains the number of users who currently have the file open for reading. Any user who tries to write in a file when ILOCK is greater than zero, will automatically be placed in file-wait status until no more users are reading the file. Any user who tries to open a file, which is open for writing or is waiting for a file-wait user to be able to open the file, will also be placed in file-wait status. A routine has been provided which allows the user to avoid file-wait.

Usage

OLDRTN = TILOCK. (RETRN)

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

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

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

Purpose

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

Restrictions

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

Usage

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

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

As library subroutine:

MAD:  WORD = GLOC.(LOC)

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

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

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

As library subroutine:

MAD: EXECUTE SLOC.(WORD, LOC)

Upon return, the A-core location whose address
is in LOC will be set equal to the contents of
WORD.
Get a supervisor parameter:
As library subroutine:

FAP:  TSX SYPAR,4
      PZE CODE
      ST0 PARAM

MAD:  PARAM = SYPAR.(CODE)

SYPAR  returns a supervisor parameter in the AC.

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

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

Get common file number
GETCF, GETCFN

Purpose

GETCF will find out the current common file directory number from the supervisor and return this information to the user.

Method

A supervisor entry has been provided which returns the common file directory number in the AC in a packed way. A library subroutine has been provided which will unpack and convert this information so that it can be used by MAD or Fortran programs.

Usage

As a supervisor entry:

TSX GETCF,4     (TIA =HGETCF)

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

As a library subroutine:

FAP: TSX GETCFN,4
  PZE CFN
  STO CFS

FORTRAN: CFS = GETCFN(CFN)

MAD: CFS = GETCFN.(CFN)

Both CFN and CFS will be set to the current common file directory number (0,1,2...). In Fortran, the file directory number is returned as an integer. This same value may be used later to call COMFL(CFN).
Detect the current Common Life Allocation number from the supervisor and return this information to the user.

Method

A supervisor entry has been provided which returns the common Life Allocation number to the AS in a packet may. A supervisor entry has been provided which will return this information to the user. It can be used by the M0 or correction program.

A common Life Allocation number is defined as the number of the common Life Allocation to which the data is matched.

As a correction program:

TA 0 DETECT (LIA - HECTOR)

Look for the data which is matched to the common Life Allocation number of the common Life Allocation to which the data is matched.
Identification

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

Purpose

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

Method

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

1 User may use common files
2 User may use privileged calls to the I/O system.
4 User may modify "PROTECTED" files of other users.
10 User may refer to "PRIVATE" files of other users.
20 User may modify the supervisor and I/O system.
1000000 User is background system
2000000 User is foreground
4000000 User is FIB
10000000 User is incremental dumper
20000000 User is privileged command.

A privileged command sets the low order digits "on". The bits which occupy the decrement may be moved left six bit-positions to indicate the .not. condition.

Usage

Update MFD:

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

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

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

Error codes:

03. User already in M.F.D.
04. Machine or System error

Delete from MFD:

DELMFD.($ PROBN$, $ PROG$)

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

Error codes:

03. User not found in M.F.D.

Attach to UFD:

ATTACH.($ PROBN$, $ PROG$)

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

Error codes:

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

Quota allotment:

ALLOT.(DEVICE,QUOTA,USED)
ALLOT may be used to allot a quota of records for each user, for each device by first ATTACHing to the user's file directory and then calling ALLOT.

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

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

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

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

Error codes:

03. Illegal device specified

Move a file:

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

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

Error codes:

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

Link to a file:

LINK.( $NAME1$, $NAME2$, $PROBN$, $PROG$, $-SNAM3-,-SNAM4-,-MODE-$)

LINK establishes a link in the current file directory to a file in some other file directory. Links may be established to any depth.

NAME1 NAME2 is the name which will be used to refer to the file in the current file directory.
PROBN  PROG specifies the file directory to which the
link is being made. This file directory may
contain the actual file or it may contain a
link to some other directory.

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

MODE is an integer or integer variable which will
be 'OR'ed with all the modes through all the
links to the actual file. The resulting
'OR'ed mode will be used as the mode in the
current file directory. If any one of the
modes (except MODE) in the chain of links does
not have the linkable mode bit, the file is
not linkable and the error return will be
taken. MODE need not be specified.

Error codes:

03. Machine or system error
04. 'PROBN PROG' not found in M.F.D.

Date a file:

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

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

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

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

Identification

General discussion of MACRO command programs

Purpose

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

Discussion

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

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

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

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

On the more detailed level, chains may be constructed within the supervisor, the command location counter may be set or interrogated, and the chains may be executed and chained by calls to supervisor entries. On this programming level many of the housekeeping details must be handled by the user.
The principle of control is to transfer all functions of a given function to a given program. This is done through a program called the "Control Command Program". The Control Command Program is a software program that allows the user to control the system through a series of commands.

The Control Command Program allows the user to:

1. View the status of the system.
2. Control the system's operations.
3. Monitor the system's performance.
4. Make changes to the system's settings.

The Control Command Program is designed to be user-friendly and easy to use. It includes a help system that provides detailed information about each command.

The Control Command Program is available for all users of the system. It can be installed on any computer that has the necessary software installed.

The Control Command Program is regularly updated to include new features and improve performance. The latest version can be downloaded from the system's website or from the software vendor's website.
Identification

Single command
XEOM, NEXCOM, NCOM

Purpose

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

Usage

NEXCOM:

as supervisor entry:

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

as library subroutine:

NCOM.(COMAND,ARG1)

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

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

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

XEOM:

as library subroutine:

MAD, FORTRAN, FAP:

K = XEOM. (COMAND,LIST)
EXECUTE XEOM. (COMAND,LIST)

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

LIST is any legal list specifying locations which
contain the BCD names of the arguments
appropriate to the command. Right justification is not necessary but the number of items in the list must be \texttt{.LE. 18}.

\texttt{K} will be zero if execution was successful; non zero if failure.

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

MACKO command
SCHAIN

Purpose

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

Reference

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

Usage

MAU, FORTRAN or FAP:

\[ A = \text{SCHAIN. (FILNAM,-ARG1,ARG2,...,ARGN-)} \]
\[ \text{EXECUTE SCHAIN. (FILNAM,-ARG1,ARG2,...,ARGN-)} \]

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

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

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

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

The purpose of this program is to provide a comprehensive educational experience that will prepare students for future opportunities in the fields of education and related careers. The program is designed to be flexible and adaptable to the needs of individual students.

The program includes a variety of courses and activities that are intended to develop knowledge and skills in areas such as early childhood education, special education, and other related fields. Students will have the opportunity to engage in hands-on learning experiences and to participate in field placements that will provide real-world application of their coursework.

The program also includes opportunities for students to participate in research and service learning projects, and to develop leadership and communication skills. Through these experiences, students will be prepared to enter the workforce or pursue advanced degrees in education.

Overall, the program is designed to provide a well-rounded educational experience that will prepare students for a successful career in the field of education.
Identification

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

Purpose

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

Method

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

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

Restrictions

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

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

TSX SETCLS,4 optional (TIA =HSETCLS)
PZE TAB, 'n'

...  
TAB BCI 1,command
BCI 1, arg1

...  
OCT 777777777777

As library subroutine:
MAD or FORTRAN:
EXECUTE SCLS. (TAB,N)

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

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

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

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

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

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

As library subroutine:
MAD or FORTRAN:
EXECUTE GCLS. (BUFF,N)

GETCLS moves 20 words from the nth command buffer of the command list into locations beginning at BUFF.
GCLS interprets MAD or FORTRAN calling sequences, calls GETCLS and stores the command buffer backwards in BUFF. Only the words to and including the fence are moved into BUFF.

BUFF must be at least 20 words long for GETCLS.

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

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

As a library subroutine:

```
MAD or FORTRAN:
EXECUTE SCLC. (M,N)
```

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

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

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

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

```
TSX GETCLC,4 optional (TIA =HGETCLC)
ST0 A
```

As library subroutine:

```
MAD or FORTRAN
```

A = GCLC (M,N)

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

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

A will be set to a word of the form PZE m,,n.
To initiate or continue a chain:

As supervisor entry:

\[
\text{TSX CHNCOM,4 (TIA =HCHNCOM)}
\]
\[
\text{PZE 'j'}
\]

As library subroutine:

MAD or FORTRAN:

\[
\text{EXECUTE CHNCOM (J)}
\]

FAP: \text{CAL = 'j'} \quad \text{or TSX CHNCOM,4}
\[
\text{TSX CHNCOM,4 PZE 'j'}
\]

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

\( \text{CHNCOM} \) determines whether or not another command exists in the chain. If one exists, it is initiated. If no chain exists; DORMNT is called if \( j = 1 \), DEAD is called if \( j = 0 \).
Identification

Fetch a current command argument
GETCOM, COMARG

Purpose

To extract the Nth argument from the current command buffer.

Usage

As supervisor or library entry:

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

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

As library subroutine:

MAD, FORTRAN or FAP:

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

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

Trace of Subroutine Calls.
ERROR

Purpose

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

Restrictions

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

Each subprogram executed must have at least one argument.

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

TRACE FAILURE IN 'sub'
EXIT FROM ERROR

Usage

MAD, FORTRAN, or FAP:

ERROR.(MESS)

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

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

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

BCD or spread-octal to binary
BCDEC, BCOCT

Purpose

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

Usage

BCD to binary:
As library subroutine:

FORTRAN: EQUVALENCE(XNUM, NUM)
XNUM = BCDEC(X)

MAD: NUM = BCDEC.(X)

FAP: TSX BCDEC, 4
     PZE X
     STO NUM

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

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

Spread-octal to binary:
As library subroutine:

FORTRAN: EQUVALENCE(XNUM, NUM)
XNUM = BCOCT(X)

MAD: NUM = BCOCT.(X)

FAP: TSX BCOCT, 4
     PZE X
     STO NUM

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

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

Binary to BCD
DEFBC, DELBC, DERBC

Purpose

To convert a binary integer to BCD with leading zeros.

Usage

As library subroutine:

MAD or FORTRAN:

\[
\begin{align*}
A &= \text{DEFBC}.(K) \\
A &= \text{DELBC}.(K) \\
A &= \text{DERBC}.(K)
\end{align*}
\]

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

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

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

DERBC converts the right half of \( K \) into a decimal BCD number.
Identification

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

Purpose

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

Usage

As library subroutine:

MAD or FORTRAN:

\[
\begin{align*}
A &= \text{OCABC}(X) \\
A &= \text{OCDBC}(X) \\
A &= \text{OCLBC}(X) \\
A &= \text{OCRBC}(X)
\end{align*}
\]

X contains the binary number to be converted

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

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

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

OCLBC converts the left half of X to 6 digits.

OCRBC converts the right half of X to 6 digits.
Identification

Justification and padding
BZEI, ZEL, LJUST, RJUST

Purpose

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

Usage

Justification library subroutines:

FAP:  TSX  LJUST, 4  TSX  RJUST, 4
      PZE  WORD             PZE  WORD
      STØ  X                STØ  X

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

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

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

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

Interchange leading zero and blanks, library subroutine:

MAD, FORTRAN or FAP:

A = BZEI (B)  A = ZEL (B)

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

BZEI replaces leading zeros with blanks. If B is
zero, "bbbbbb0" will be returned.

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

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

Purpose

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

Reference

CC 186 FORTRAN and MAD Format Specifications Spall

Method

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

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

Usage

Output, binary to BCD:

Fortran:

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

MAD:

```
TSX USRSTH,4
PZ£ FORMAT,,SWIT
RTN .
STR FIRST,,LAST
STR 0
USRSTH Set upper core locs
TRA* (IOH)
OUT .
TRA 2,4
```
Input, BCD to binary:

**Fortran:**

- TSX USRTSH,4
- PZE FORMAT,,SWITCH
- RTN .
- STR
- STQ SYMBOL,t
- TSX (RTN),4

**MAD:**

- TSX USRTSH,4
- PZE FORMAT,,SWITCH
- RTN .
- STR
- STR FIRST,,LAST
- STR 0

**USRTSH** Set upper core
**TRA* (10H)**
**IN**
**TRA 1,4**

**FORMAT** is the beginning location of the desired format.

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

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

**SYMBOL,t** locates the variable to be converted. A loop may be included here for arrays or a series of LDQ, STR. After each variable is converted by (10H), return is made following the STR in order to find the next variable to be converted.

**FIRST** is the starting location of the list.

**LAST** is the final location of the list. **LAST** may be lower in core than **FIRST**. If the list is of length one, **LAST** is zero.
(FIL) is called to indicate that all the output data 
has been converted and the current buffer 
should be truncated.

STR 0 terminates the list in a MAD call.

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

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

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

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

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

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

IOHSIZ:

MAD, FAP, or FORTRAN

TSX IOHSIZ,4
PZ E N

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

MAD, FAP, or FORTRAN

TSX STQUO, 4

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

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

(77737)8 address Location of subroutine that (IOH) calls
for input or output. This address corresponds to INPUT or OUTPUT.

    Tag 0
    decrement +1 if format stored backwards -1 if format stored forwards
    prefix TXL if FORTRAN type call TXH if MAD type call

(77740)8 address location of first word of format statement.

    tag 0
    decrement user's index register 4 on initial entry to the input-output subroutine.
    prefix TXL for on-line printer TXH for all other I/O

(77741)8 address scratch area for (IOH) to use for output. The number of words in the output record is stored here.

    tag 0
    decrement maximum number of columns available in input or output record (may not exceed 132).
    prefix TXL for output (binary to BCD), TXH for input (BCD to binary).

(77742)8 The beginning of a 22 word buffer from which BCD data is converted to binary or into which BCD data is placed after binary to BCD conversion.

(77771)8 address location of symbol table (if any)

    0 address the address of RTN as RTN is the location to which programs should return after calling (IOH).
Pack and unpack, single characters to full words
PAKR, PAKL, UNPAKR, UNPAKL

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

Usage
As a library subroutine:

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

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

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

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

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

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

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

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

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

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

Purpose

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

Usage

As a library subroutine:

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

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

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

I is a full word (MAD) integer.
J is a decrement (FORTRAN) integer.
A is equivalent to J.

FINT converts from full word to decrement integer.
If the integer is too large, the following message will be printed and the integer will be taken modulo 32768.
MAD INTEGER EXCEEDS 32767

MINT converts from decrement integer to full word.
Identification

Variable length calling sequence processor
COLT, SELAR, MDL

Purpose

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

Usage

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

COLT, as a library subroutine:

    TSX  COLT,4
    PZE  IR4

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

AC upon return, will contain, in the decrement,
the number of arguments in the calling
sequence to the program and, in the address, a
code specifying the type of the
calling-program. The codes are:
  0 unknown, or no arguments
  1 FAP
  2 FORTRAN
  3 MAD

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

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

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

RETURN is the location to which SELAR is to return.

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

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

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

MDL, MAD list processor:

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

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

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

Determine type of calling program and FILNAM GNAM

Purpose

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

Usage

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

As library subroutine:

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

OPN may be PZE, TXH, or TSX.

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

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

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

0 unknown
1 FAP
2 FORTRAN
3 MAD
Detecting incorrect type of calling argument

**Purpose**

To provide a testing vehicle to ensure the purpose of an argument matches the purpose of the calling routine. A call to a routine with the type of calling argument and a little wrong in one or both parameters.

**General Purpose**

This program is a testing vehicle for calling CMLR.

Calling Procedure

The macro can put one MCMK or MCMK of TEST.

**Library Subroutine**

TEST CMK

- READ

Can read the TEST of TEST or TEST.

**Conclusions**

In the case of the general purpose, the purpose of the arguments is not ascertained in the CMLR. The function of the arguments in the CMLR routine is not ascertained in the calling sequence to the CMLR.

**AC will contain a copy of your-destination-line**
Identification

List transmission
MOVE1, MOVE2, MOVE3

Purpose

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

Usage

As library subroutine:

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

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

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

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

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

ENDLST (specified only when OP is TIX or STR) is the ending location of the argument list.
ALPHA is the return from MOVE2 at which time the AC contains the first data item as specified by BEGLST.

STR causes the storing of the AC in the data block specified by BGDATA. If this fills the data block, return is made to DATOUT and the AC is meaningless. The next data item from the list is then placed in the AC and return is made to BETA. If there is no next item in the list, return is made to LSTOUT.

If BEGLST was specified as an array base, successive STR's will cause the transmission of successive elements of the array. The number of elements thus transmitted must be controlled by the user.

DATOUT is the return location if the data block is full. The AC is meaningless. MOVE1 may now be called again to initialize another data block.

LSTOUT is the return location if the list is exhausted MOVE2 may be called to specify another list or another STR may be executed if moving an array.

OPN may be any programming to establish loops and use or modify the AC if desired.

MOVE1 initializes addresses and indexing for the data block and also initializes the STR trap cells to entries to this routine.

MOVE2 initializes addresses and indexing for the list, initializes the trap cells if not already done, and gets the first data item in the AC.

MOVE3 restores the trap cells.
**Identification**

Name a format or file name
SETFMT, SETNAM

**Purpose**

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

**Usage**

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

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

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

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

These two routines allow the library disk routines to be called with FILNAM and FORMAT as arguments instead of the actual BCD information. i.e., CALL DWRITE (FILNAM, FORMAT, LIST)

instead of, CALL DWRITE (12H NAME1 NAME2, nH(....), LIST)
Identification

Get the date and time of day
GETIME, GETTM

Purpose

To provide the user with the current date and time of day.

Method

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

Usage

As supervisor or library entry:

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

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

As library subroutine:

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

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

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

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

Purpose

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

Method

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

Restrictions

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

Usage

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

To initialize or reset the stop watch to zero:
EXECUTE RSCLOCK.

To read the elapsed execution time since the last call to RSCLOCK:
EXECUTE STOPCL. (J)

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

To read the elapsed execution time since the first initialization of the clock:
EXECUTE JOBTM. (J)

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

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

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

MAD: EXECUTE TIMER. (J,S)

FAP: TSX TIMER,4
PZE J

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

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

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

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

EXECUTE RSTRTN.

To void the last setting of the alarm clock:

EXECUTE KILLTR.

To provide foreground/background compatibility to job time remaining:

EXECUTE TIMLFT. (J)

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

CLOCK function
CLOCON, CLOCOF

Purpose

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

Method

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

Usage

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

TSX CLOCON,4 optional (TIA =HCLCON)

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

TSX CLOCOF,4 optional (TIA =HCLCOF)
To cease (the interrupter to terminate the internal timer for

the wired clock function 7-an (the A core internal) timer cell.

will be updated at the interrupter at each time burst (100
milliseconds) it will be updated on the elapsed time

running time (at least 1 s) in steps of 64 seconds. You

and the reference in your calculation (this will be interpreted as

the time of the update).

For interrupter of 384000 Hz)

In the clock function all

in the interrupter or 384000 Hz)

not interrupter or 384000 Hz)
### Identification

List of miscellaneous library subroutines:

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

<table>
<thead>
<tr>
<th>DFAD</th>
<th>DFSB</th>
<th>DFMP</th>
<th>DCEXIT</th>
<th>DFDP</th>
<th>SFDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOSET</td>
<td>IOPAR</td>
<td>IOEND</td>
<td>IOSCP</td>
<td>IOITR</td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>ORA</td>
<td>ANA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLØ</td>
<td>SLI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.01300</td>
<td>.01301</td>
<td>.01311</td>
<td>.03310</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX0</td>
<td>MAX1</td>
<td>XMAX0</td>
<td>XMAX1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN0</td>
<td>MIN1</td>
<td>XMIN0</td>
<td>XMIN1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>XMOD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSIGN</td>
<td>SIGN</td>
<td>XLOC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIM</td>
<td>XDIM</td>
<td>INT</td>
<td>XINT</td>
<td>XFIX</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>EXP(1)</td>
<td>EXP(2)</td>
<td>EXP(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACOS</td>
<td>ASIN</td>
<td>ATAN</td>
<td>ATN</td>
<td>COS</td>
<td>SIN</td>
</tr>
<tr>
<td>LOG</td>
<td>SQRT</td>
<td>SUR</td>
<td>TAN</td>
<td>COT</td>
<td>TANH</td>
</tr>
<tr>
<td>SIMCS</td>
<td>XSMEQ</td>
<td>XSIMEQ</td>
<td>XDETRM</td>
<td>XDTRM</td>
<td>DETCS</td>
</tr>
<tr>
<td>FLIP</td>
<td>RANNO</td>
<td>SETU</td>
<td>INDV</td>
<td>DPNV</td>
<td></td>
</tr>
</tbody>
</table>
Identification

Log in
LOGIN

Purpose

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

Usage

user: LOGIN probno name
response: W TIME, PASSWORD
user: private password
response: probno progno LOGGED IN date1 time1
message of the day
CTSS BEING USED IS id

SHIFT MINUTES
ALLOTTED USED SINCE date2 time2
1 at1 ut1
2 at2 ut2
3 at3 ut3
4 at4 ut4
5 at5 ut5
LAST LOGOUT WAS date3 time3

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

probno is the user's problem number.
name is the user's last name of which only the last six characters are used.
TIME is the current time of day. HHMM.M
password after typing PASSWORD, the system turns off the printing mechanism, if possible, so that the user may type his password with privacy.
progno is the programmer number equivalent to probno name.
date1 is today's date. MM/DD/YY
time1 is current time of day. HHMM.M
message of the day (optional) gives some pertinent information about the system.

id is the name of the version of CTSS being used.

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

ut1-5 total time used in each shift.

date2 time2 are date and time when used time began accumulating.

date3 time3 are date and time of user's last LOGOUT.

DRQ number of records on drum allotted to user.

DRU number of records on drum used.

DIQ number of records on disk allotted to user.

DIU number of records on disk used.

TPQ number of records on tape allotted to user.

TPU number of records on tape used.

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

YOU HAVE URGENT MAIL

before the response:

probo no progno LOGGED IN date1 time1

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

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

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

3. probn0 name ALREADY LOGGED IN.
   LOGIN COMMAND INCORRECT.
   User is already logged in on another console.

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

5. PARTY LINE IS BUSY, PLEASE TRY AGAIN LATER.
   This is the result of one of the following conditions.
   a. Primary lines in user's party line group are filled, and user is not allowed to be standby.
   b. User's party line group number is zero, and user is not allowed to be standby. In other words this user may not be logged in at all.
   c. Maximum number of users for time-sharing has been reached, the user's party line group is either zero, or the primary lines for that group have been filled.
   d. Maximum number of users for time-sharing has been reached, and there are no standby users to be logged out.

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

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

8. ALLOTTED TIME EXCEEDED FOR THIS SHIFT - NO LOGIN.
The user's allotted and used time will follow this printout, same format as that of the normal login, except that the printed number of tracks used will be zero.

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

**Party Groups**

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

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

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

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

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

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

Log out
LOGOUT, AUTOMATIC LOGOUT

Purpose

LOGOUT allows the user to terminate his use of the console without saving any machine conditions and also to express his opinion about the quality of the time-sharing service he received. The opinions thus collected are statistically processed to gain information about user satisfaction with different scheduling techniques. The automatic logout allows the system to terminate an inactive user or terminate all users for a system shut down in such a way that each user may later RESUME his program from the point of interruption.

Usage

LOGOUT:

user: LOGOUT -N-
response: W time
probno progno LOGGED OUT date1 timel
TOTAL TIME USED = XX.X MIN.
time and timel are the current time of day. HHMM.M
date1 is today's date. MM/DD/YY
probno is the problem number
progno is the programmer number
XX.X is the time used in minutes.

N (optional) if N is omitted, no opinion of the service is expressed. If N is included, it should reflect as closely as possible the users satisfaction with response for this session for his particular application.

N=X The user has no opinion to express, (e.g., he logged in, then logged out without doing anything).

N=0 The system response was totally unusable.

N=1 Response was intolerably slow.

N=2 Response was sluggish but not intolerable.
N=3  Response was typical of what was expected.
N=4  Response was better than expected.
N=5  Response could not have been better.
N=Anything else. A one sentence description of the choices of parameters is desired.

AUTOMATIC LOGOUT:

response: WAIT,
AUTOMATIC LOGOUT

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

RESUME LOGOUT

or CONTIN LOGOUT
Identification

Foreground Initiated Background
FIB, DELFIB

Purpose

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

The FIB facility allows the user to specify files which are to be executed by RUNCOM when the user is not logged in. The supervisor schedules a FIB job in the same scheduling queues as regular foreground jobs.

Restrictions

No FIB job will be run while its donating user is logged in. A user who logs in during execution of one of his FIB jobs will cause that job to be automatically logged out. A user may initiate any number of FIB jobs. As one might expect, there is no way for FIB jobs to receive console input.

Usage

To initiate a FIB job:

\[
\text{FIB NAME1 TIME}
\]

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

TIME is the maximum execution time limit in minutes which the user wishes to place on the job. If TIME is not specified, a time limit will be set by the supervisor. No FIB job will be allowed to exceed a certain maximum time; which is currently set as 5 minutes. A FIB job which exceeds its time limit will be "automatically logged out", and it may be restarted by the user.

To delete a waiting FIB job:

\[
\text{DELFIB NAME1}
\]

Method

FIB jobs are run one at a time on a first-come-first-served basis. A FIB job is run in the same scheduling queues
as foreground jobs but as the result of no console interaction, it moves to the lower priority queues. The FIB job is logged in; the list of commands are executed by RUNCOM; and when the list is exhausted or time limit is exceeded, the job is logged out. Calls to WRFLX cause writing into a file, $$$$$FIB OUTPUT, in the user's file directory. Calls to DEAD, DORMNT, RDFLX, COMFIL, and ATTCON cause FIB jobs to be automatically logged out. If a FIB job is logged out for any reason, it must be restarted by the user. The FIB job running at system shutdown time will be run to completion or until it exceeds its time limit. If a FIB job is logged out because it exceeded its time limit it is logged out by ENDLOG so that as much as possible is saved.

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

IF YOU LOG IN YOUR FIB JOB WILL BE LOST.
DO YOU WANT TO LOGIN,

If the user types YES, his FIB job will be automatically logged out, and login will continue to log him in. If he types NO, he will not be logged in, and his FIB job will continue to run.

The user may initiate any number of FIB jobs.
Identification

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

Purpose

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

References

MAC 146 AED-0 Programmer's Guide Feldmann, Ross
MAC 154 Warnings & Restrictions in AED-0 Feldmann
MAC 169 "LOADER: A New Version of the Wolman
BSS Loader"
MAC 198 PLEX-DUMP & Relocation in AED-0 Fox
MAC 199 Stack manipulation in AED-0 Coe
MAC 207 "Internal Memos for AED Users" Feldmann
MAC 208 "Flash No. 10 - New CTST2 Feldmann
Command"
MAC 213 "Flash No. 11 - AEDBUG Usage" Fox
MAC 225 Argument Checking for AED Walsh
MAC 226 Availability of AED Jr. Systems Ross
A "readable" output of a CED program written to an ABAP V.0, including a comment, that was generated by a computer. The comment is a description of the CED program's purpose and the ABAP instructions that were used to create it. The ABAP program is written in a language called ABAP, which is a structured programming language used for business and financial applications. The ABAP program is designed to be read and executed by ABAP compilers, which translate the ABAP code into machine-readable instructions that the computer can execute.
Identification

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

Purpose

BEFAP is a version of FAP with a more powerful macro compiler and with the ability to handle compressed source decks directly (see CRUNCH). Its advantages are the abilities to edit larger files (via the alter feature with CRUNCH decks) and to produce more readable listing files. An immediate benefit is the ability to use and modify languages under CTSS which were developed and written in BEFAP. (e.g., BLOJ/1, ALWAC, SNOBOL)

References

IBM C28-6235 FORTRAN II Assembly Program (FAP)
MAC 179 BEFAP command within CTSS R. U. Bayles

Usage

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

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

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

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

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

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

Purpose

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

References

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

Usage

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

Modifications

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

READ/DISK NAME1 NAME2

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

DELAY/PRINT N

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

COMIT - Symbol manipulating and string processing
Bob Fabry - X2525

Purpose

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

References

MIT Press Introduction to COMIT Programming
ACM Comm. Mar. 1963 "COMIT" V.H. YNGVE
MAC 156, CC 237 COMIT operation in CTSS V.H. YNGVE
CC 178 Availability of COMIT V.H. YNGVE
CC 246 COMIT subroutines for generating
   Fortran programs D.C. MATIATOF
CC 248 COMIT system under MIT's FMS FABRY, YNGVE
Identification

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

Purpose

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

References


Usage

DYNAMO NAME1 P R

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

P-Page Skip

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

R-Rerun

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

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

**PLEASE TYPE CHANGES IF RERUN DESIRED**

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

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

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

**QUIT**

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

**Differences in Input**

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

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

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

Purpose

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

References

MAC 122 DEMON: ESL Display Console Demonstration Program Polansky
MAC 125 ESL Display console Time Studies Polansky
MAC 166 B-core system for programming ESL in CTSS Lang
MAC 201 ESL Display console system manual Bayles
MAC 202 Proposal to improve rotation matrix of ESL Stotz
MAC 217 Operating Manual for the ESL Display Console Stotz, Ward
To develop a comprehensive input and output facility with a limited real-time capability, two IBM type 727 computers, one from IBM's Data Processing Division, two Apple microcomputers, and other means of information retrieval, real-time login, transaction validation, and protocol under program control.

References


MAD 12/8: Code-Order Codes to Present an Environment for the CDD. Table 1.

MAD 205: Acquisition of DEC equivalent from CDD. Table 1.

MAD 77: Operating Manual for the DEC Design Computer System.
Identification

FAP - IBM 7094 machine language Programming staff

Purpose

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

References

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

Usage

FAP NAME1 -'(LIST)'-

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

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

INSERT pseudo-operation

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

LSTNG pseudo-operation

The pseudo-operation LSTNG controls console printing of the assembly listing. The first LSTNG card causes printing, beginning with the next line. Alternate appearances of LSTNG turn this feature on and off.
NOSEQ pseudo-operation
The pseudo-operation NOSEQ prevents sequence checking of serialization in columns 73-80; i.e., no "SOURCE ORDER ERROR" diagnostics appear in the NAME1 BCD listing file.

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

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

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

GPSS - General Purpose System Simulator
M. M. Jones - X5670

Purpose

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

References

MAC 140    On-line Version of GPSS II    M.M. Jones
IBM 820-6346 General Purpose System Simulator II
Identification

LISP - List Processing Language
Wm Martin - X5879

Purpose

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

References

MIT Press
MIT Information
International
Information
International 'LISP'

| MAC 128 | Secondary Storage in LISP | Edwards |
| MAC 129 | LISP Garbage Collector | Minsky |
| MAC 134 | LISP Exercises | Hart |
| MAC 142 | LISP as the language for an Incremental computer | Raphael, Lombardi |
| MAC 153 | Time Sharing LISP | Martin, Hart |
| MAC 206 | CTSS LISP NOTICE | Hart |

Usage

LISP - NAME1-

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

LISTEN NIL - while executing data from the DATA file, if the doublet LISTEN NIL is executed, subsequent S-expressions will be read from the console. When the atomic symbol STOP is typed, reading and executing pairs of S-expressions will continue from the DATA file.

Example: EXAMPLE DATA contains -
DEFINe (( ()
LISTEN NIL STOP )))))))))

When the command LISP EXAMPLE is given, the functions are defined from the file and pairs
of S-expressions are read from the console as test cases. Typing STOP at the console will terminate the command.

Example: EXAMPE DATA continues

DEFINE C

(type out)
Identification

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

Purpose

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

References

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

RESTRICTIONS

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

Usage

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

MAD NAME1 -(LIST)"- -(SYMB)"-

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

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

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

CHANGES:

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

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

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

   ../RTN. This symbol, which is obviously invalid in a statement, stands for the operation of placing the appropriate value(s) in the arithmetic register(s) and then returning from a function to its calling program. It is analogous to the right hand side of a substitution statement (the B operand) and then a transfer to a given address (there is no designation for this address within the triple). As such, there is no result. As an example, if the result of a function were a double precision number, say mode 5, the following would be a reasonable definition.

   MODE STRUCTURE 4../RTN.5
   JMP **3,BT,**1
   CLA B
   LDQ B+1
   TRA FF
   OUT ACQ
   END

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

MADTRN - Fortran II to MAD translator
Programming staff.

Purpose

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

References

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

Usage

MADTRN NAME1 OP

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

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

OP=(SYMB) The argument (SYMB) will be passed on to the MAD compiler and the file NAME1 SYMTAR will be produced to be used by MADRUG.
MADTRM - Fortran II to MAD Translator

Programming Style

FORTRAN II programs are compiled into MAD programs, which are then translated into machine language. The FORTRAN II source program is converted to MAD and then translated into machine code. The resulting MAD program is then loaded into the MAD computer system. The MAD program is executed, and the results are returned to the user via output devices. The MAD program is designed to be portable and can be run on any computer system that supports the MAD language.

Special Features

DC 10? Pattons and MAD Fortran Specifications

MADTRM NAME? OF

MADTRM NAME? OF THE COMPUTER

MADTRM NAME? OF THE COMPUTER

MADTRM NAME? OF THE COMPUTER

MADTRM NAME? OF THE COMPUTER
Identification

SNOBOL - A String Manipulation Language
M. Daggett, X4116

Purpose

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

References

Journal of the ACM, January 1964, pp. 21-30
CC 235 SNOBOL Available as a Daggett, Pouzin
CTSS command
SMOOTH - A STRING MANIPULATION LANGUAGE

SMOOTH is a programming language designed for the manipulation of strings of characters. It allows for the definition of procedures, the manipulation of strings, and the use of string operators. The language is particularly useful for string processing and for applications requiring the manipulation of characters and strings.
Identification
STRESS Command
R. Logcher

Purpose
STRESS is a problem-oriented language for structural engineering. In its present form it performs primarily structural analysis, although it is developing into a computer-aided design system.

Reference

Implementation
STRESS is currently a command in the CTSS system. It consists of a short starter program and a number of links. For normal operation, the links are accessible through the PUBLIC file with the names STR(n) LINK.

For development purposes, the linking program has been written so that a user can be developing one or more links while using others from the PUBLIC file. If a copy of a needed link exists in the user's file directory, that copy will be used rather than the one in the PUBLIC file.

Modifications
Two statements have been added to the language in order to facilitate its use with time-sharing.

READ (FROM ) (FILE) NAME1 NAME2
READ CONSOLE

READ (FROM) (FILE) reads the input from a line-numbered file (which may have been created by INPUT or ED). A single file should not contain more than a single problem specification (not beyond a SOLVE statement because, presently, all active files are reset to the beginning whenever a new link is loaded).

NAME1 NAME2 are the names of the disk file to be read. The names must begin with non-numeric characters other than commas, signs, or periods. The file is renamed .TAPE. 1 for reading, but will be restored to NAME1 NAME2 by a different READ, (i.e., READ NAME3 NAME4) or by the statement STOP which calls EXIT.
NAME2 is assumed to be DATA if not given.

READ CONSOLE transfers to read the input from the console.

Return to input from any part of the process, any error detection, or a single interrupt causes control to be returned to the console. The word TYPE is given to indicate that a line of input is expected from the console.

NOTE:
The statements INPUT and EDIT are now available. They will function in the same way as the supervisor commands but the existing data structure will be saved and control will return to the console after the FILE command.
Identification

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

Purpose

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

References

Identification

Generate line-numbered files
INPUT, EDIT, FILE, TFILE

Purpose

Allows the user to create a line-numbered, 14 words per line, BCD file and edit such a file by specifying the line numbers to be modified. These files are appropriate input for many of the card-oriented subsystems i.e., FAP, MAD, MADTRN, STRESS, etc.

Method

In the INPUT mode, the command generates and prints the line numbers (sequential and incremented by 10) so that the user need type only the data. The line number becomes part of the line as columns 75-80. In the EDIT mode, lines may be deleted, replaced or inserted by manual typing of the line numbers. At FILE time, the EDIT requests are sorted by line number and the later requests override any previous requests for the same line number.

Usage

Input:

user: INPUT
response: 00010

The user then types a card image per line, according to the format appropriate to the programming language being used. Tabs may be used to specify fields. The carriage return terminates a line and the next line number will automatically be printed (incremented by 10).

A line consisting of only a carriage return will terminate the automatic or INPUT mode and enter the manual or EDIT mode. In the manual mode the following conventions may be followed:

1) A line number (all numeric) followed by a space or tab followed by the desired line allows the deletion, insertion, or replacement of a line of that sequence number. If the line number is followed by a tab, the first field is blank.

2) DELETE, 'n1'-',n2'- where n1 and n2 are previous line numbers. Lines n1 thru n2 will be deleted or just n1 will be
deleted if n2 is not specified. (Note that the commas in this request are optional.)

3) SEQUENCE, 'n1'-',delta'-. The automatic mode is resumed starting with line number n1 and subsequent incrementing by delta. If delta is omitted, the previous increment is retained. (Note that commas are optional)

4) A line consisting of only a carriage return resumes the automatic mode with the next line number after the last line of the file.

5) FILE NAME1 NAME2 terminates EDIT and initiates the FILE command.

6) QUIT signal terminates INPUT and EDIT and the input lines may be lost. (See next page.)

Edit:

user: EDIT NAME1 NAME2
response: XXXXX

XXXXX is the next line number following the last line of file NAME1 NAME2.

EDIT establishes the automatic or INPUT mode and appends to the file NAME1 NAME2. See conventions under INPUT.

File:

user: FILE NAME1 NAME2 -MODE-

NAME1 is the primary name of the file

NAME2 is the secondary name and should be the class of language used during INPUT.

MODE is the mode of the file: 0 is temporary, 1 is permanent, 2 or R1 is read only R1, 3 or R2 is read-only R2. If MODE is not specified, permanent is assumed.

FILE creates a sequenced line-numbered file with the line numbers as right-adjusted sequence numbers in the corresponding card images of the file. If an older file of the same name exists in the user's directory, it will be replaced by the new file. Any corresponding NAME1 SYMTB and BSS files will also be deleted. No message is given regarding this.
TFILE is also recognized as a command in the manual mode. TFILE is used by system programmers for checking-out new versions of FILE.

Error procedure for FILE:

In most cases the following procedure should succeed:

- Comments are printed explaining the cause of the error, and FILE calls DORMNT.
- Type a SAVE command. If track quota is exhausted, give the extra parameter 't' to create a temporary mode saved file.
- Fix up whatever was indicated as the cause of the error (file mode, track quota, etc.)
- CONTIN the SAVED file.

LOGOUT Peculiarities:

The FILE command deals with files which usually are of temporary mode. Consequently, they are lost on a LOGOUT. There is some possibility of losing both the old and the new files. This does not apply in the case of automatic LOGOUT where temporary files are saved.

The following tips may be of some help to restart a FILE command after a LOGOUT break.

- Before LOGOUT, if possible, change the mode of (EDIT FILE) and (FILE FILE) to permanent.
- If (INPUT FILE) is not found, all the input lines typed from the console have been lost, and must be retyped.
- If (EDIT FILE) is not found, rename the old file (EDIT FILE).
- If (FILE FILE) is not found, try a FILE command, without going through INPUT.
Erase preceding 46 lines.

In order to make the following sequence flow successfully,
Command are utilized and an equivalent chain of
actions and line code follows.

Type the SAVE command.
Type the code sequence "ls" to create a temporary work readable file.
Set the variable with information in the change of
the order (lower order) (case change, etc.)

LGOOFT record initialized

Type the LFC command with a life which contents are of
previously work. Command are given but not set to a
constant. These are some preliminary information that may be
printed and the new LIFE. This does not, of course, set the change
of the temporary work readable file. These are reading
of temporary work. Command after a LGOOFT (lower order) is
written. The sequence given may be used but not a
large command and reading

If (COME FILE is not found) then the next.
If (COME FILE is not found) then the next.
If (COME FILE is not found) then the next.
If (COME FILE is not found) then the next.
If (COME FILE is not found) then the next.
Identification

Context editor for card image files
ED

Introduction

ED is a command for editing 14-word BCD card image files within CTSS. The command is based on TYPSET (CC-244, MAC-M-193 by J. H. Saltzer) and many of the conventions of TYPSET are used by ED. Tabs are automatically interpreted for FAP, MAD, MADTRN, LISP, COMIT, and ALGOL (i.e., AED) programs. Tabs may also be set by the user for other purposes. Although line numbers may be generated by the ED command, editing is done entirely by context. The ED command is offered as an alternate to the present INPUT, EDIT and FILE commands.

Usage

The ED command is initiated with the following CTSS command.

```
ED -NAME1- NAME2 -NAME3-
```

NAME2 is the secondary name of the file to be edited or created and must be provided. NAME1 is the primary name of the file to be edited. If NAME1 NAME2 is not specified, ED will assume that a new file is to be created and will start in the high-speed INPUT mode. If NAME1 is provided, the command will look for the file NAME1 NAME2. If the file is not found, the high-speed INPUT mode will be entered. If the file is found, the EDIT mode will be entered.

If NAME3 is specified and the file NAME1 NAME2 is found, the subsequent FILE will create a file NAME3 NAME2 and NAME1 NAME2 will remain unaltered. Any arguments to the FILE request, however, will take precedence.

HIGH-SPEED INPUT MODE:

When the user enters this mode, the ED command will type "INPUT:" on the user's console. While the user is operating in this mode, the ED command will accept input lines from the user's console. Tabs will be interpreted automatically for each input line. Backspace characters may also be used to move back one character position in the input line. No response is typed for input lines and as a result, the user may type successive lines as fast as he wishes. When the user types a line consisting only of a single carriage return, the ED command will place the user's console in the EDIT mode.
EDIT MODE:

When the user enters this mode the response "EDIT:" will be typed on the user's console. At this time the user may type requests to the ED command. All changes made to a file become effective immediately and as a result, the user is able to make recursive modifications to his file. We may think of a pointer which is positioned at a line in the edited file. When the user enters the EDIT mode from the INPUT mode, this pointer will be positioned at the last input line typed by the user. When the user starts the ED command in the EDIT mode, the pointer is positioned before the first line in the old file. If the end of file is reached by an EDIT request, the comment "END OF FILE REACHED" is typed on the user's console followed by the request which caused the end of file to be reached. At this time the pointer will be positioned after the last line in the file. When in the EDIT mode, any line which is not a legitimate EDIT request will cause the comment "NOT A REQUEST:" to be typed on the user's console followed by the line which caused the error. In many cases it is possible for the user to stack EDIT requests. If one of the requests causes an error message to be typed, any stacked requests will be ignored. This is done in case one of the stacked requests depended on the successful completion of the request in error.

Any number of initial tabs or spaces (including 0) may occur in a request line. Arguments and the request must be separated by at least one space or any number of tabs or spaces. Wherever the argument is line image, however, tabs and spaces retain their normal significance.

Warning:

One interrupt level is set which returns the user to EDIT mode. Use it sparingly, as it can result in lost lines in files if used during any request which moves the pointer (FIND, LOCATE etc.).
EDIT REQUESTS:

REQUEST: FIND LINE
ABBREVIATION: F
RESPONSE: none
ERRORS: END OF FILE

The FIND request is used to move the pointer forward from its present position to the line specified by LINE. LINE is a normal input line and may contain tabs and backspaces. This line is used as a mask for selecting the desired line in the edited file. Matching is done only on the non-blank characters specified in LINE. For example, the request,

F (tab)-(tab)-ALPHA,1

might be used to find the line,

LOOP TIX ALPHA,1,4

REQUEST: LOCATE string
ABBREVIATION: L
RESPONSE: none
ERRORS: END OF FILE

The LOCATE request is used to move the pointer forward from its present position to the first line which contains the entire character string specified by "string". The full line of 84 characters is scanned, so that "string" may specify line numbers. It is recommended that "string" include the leading zeros of the line numbers to avoid any undesired match with program constants.

REQUEST: NEXT 1
ABBREVIATION: N
RESPONSE: none
ERRORS: END OF FILE

This request is used to move the pointer forward from its present position in the file. "1" specifies the number of lines to be skipped over. If 1 is "0" or not specified, it is assumed to be "1" and the pointer will be moved to the next line in the file. If the NEXT request is given after the end of file has been reached, the pointer will be reset to the beginning of the file and moved "1" lines from there.

REQUEST: DELETE 1
ABBREVIATION: D
RESPONSE: none
ERRORS: END OF FILE

The DELETE request will delete "l" lines from the file starting with the line at which the pointer is currently positioned. The pointer is left at the position vacated by the last line deleted by this request. If l is "0" or left unspecified, only the current line will be deleted.

REQUEST: PRINT lL-
ABBREVIATION: P
RESPONSE: printed lines
ERRORS: END OF FILE

The PRINT request will print "l" lines from the file starting with the line at which the pointer is currently positioned. Upon completion of this request, the pointer will be left pointing to the last line printed. If l is "0" or left unspecified, one line will be printed. Normally lines are printed without line numbers. If the character "L" is present in the PRINT request, line numbers will be printed to the right of the printed lines.

REQUEST: RETYPE LINE
ABBREVIATION: R
RESPONSE: none
ERRORS: none

This request will cause the line at which the pointer is currently positioned to be replaced by LINE. LINE is a normal input line and may contain tabs and backspaces. The pointer is not moved by this request.

REQUEST: TOP
ABBREVIATION: T
RESPONSE: none
ERRORS: FILE WORD COUNT ZERO
NOTHING IN FILE
INPUT:

This request will cause the pointer to be reset and positioned before the first line in the file. If the file is empty, high speed input mode is entered.

REQUEST: BOTTOM
ABBREVIATION: B
RESPONSE: Input:
ERRORS: none

This request will cause the pointer to be positioned after the last line in the file. Upon completion of this request
the user's console will be placed in the high-speed INPUT mode. All subsequent lines will be treated as input and added to the end of the file.

REQUEST: INSERT or (C.R.)
ABBREVIATION: l
RESPONSE: INPUT:
ERRORS: none

This request will cause the user's console to be placed in the high-speed INPUT mode. All subsequent lines will be treated as input and inserted after the line at which the pointer is currently positioned. If the INSERT request is given immediately following a TUP request, the inserted lines will be placed at the beginning of the file.

REQUEST: INSERT line
ABBREVIATION: l
Response none
Errors: none

The INSERT request may be used to insert a single line without changing to the high-speed input mode. Line is a normal input line. It is inserted following the line at the present pointer position.

REQUEST: CHANGE "string1" "string2" 1 G
ABBREVIATION: C
RESPONSE: none
ERRORS: END OF FILE

This request will examine "1" lines starting at the line at which the pointer is currently positioned. Upon completion, the pointer will be left positioned at the last line examined by this request. If "1" is "0" or left unspecified, it is assumed to be "1" and only the current line will be examined. The character "Q" is taken to be the delineator or "Quote character" and may be any character in the 6-bit BCD set. "string1" and "string2" are arbitrary BCD character strings and may be of different lengths. If the character "G" (GLOBAL) is present, every occurrence of string1 will be replaced by string2. If "G" is not present, only the first occurrence of string1 will be replaced by string2 in each examined line. EXAMPLES:

line: ALPHA= ALPHA+ALPHA
request: C 'ALPHA' 'BETA'
new line: Beta= ALPHA+ALPHA
request: C 'ALPHA' 'DELTA' 1 G
new line: Beta= DELTA+DELTA
request: C 'DELTA+1'
new line: BETA = DELTA

REQUEST: BLANK line
ABBREVIATION: BL
RESPONSE: none
ERRORS: none

The BLANK request will put blanks in the current line wherever non-blank characters appear in "line". For example 'BL XXXXX' will clear the label field of a line in a FAP file.

REQUEST: OVRLAY line
ABBREVIATION: O
RESPONSE: none
ERRORS: none

The OVRLAY request will place the non-blank characters of "line" into the corresponding position of the current line. Notice that only non-blank characters of "line" replace what was in the current line. For example in a FAP file, if the current line is

TXI     **1

then

0 EUF(tab) bbH (tab)(tab) comment

will produce

EUF    TXH   **1  comment

REQUEST: VERIFY
ABBREVIATION: VE
RESPONSE: none
ERRORS: none

The VERIFY request sets the verify mode. In the verify mode, completion of any of the requests FIND, NEXT, LOCATE, OVRLAY, BLANK and CHANGE will cause the printing of the current-pointer line. In addition, CHANGE will cause the printing of all changed lines. Requests may not be stacked while in the verify mode.

REQUEST: BRIEF
ABBREVIATION: BR
RESPONSE: none
ERRORS: none
The BRIEF request sets the brief or normal mode. Within the brief mode, the FIND, NEXT, LOCATE, OVRLAY, BLANK, CHANGE requests will give no response.

REQUEST: CLIP 'ON' or 'OFF'
ABBREVIATION: CL
RESPONSE: none
ERRORS: TRUNCATED

The request CLIP ON sets a mode such that any input line which exceeds column 72 will cause the message "TRUNCATED:" followed by the faulty line image. Any waiting input lines will have been deleted. Requests on which this may occur are FIND, INSERT, RETYPE, OVRLAY, BLANK and high-speed INPUT. The request CLIP OFF resets the mode. The normal mode is CLIP ON for all files except FAP files which are normally CLIP OFF.

REQUEST: SERIAL N
ABBREVIATION: S
RESPONSE: none
ERRORS: none

This request is used to change the increment between line numbers of successive lines to the increment specified by the decimal integer "N". Initially, this increment is set to 10 by the ED command. If N is "0" or not specified, it is assumed to be "10". Lines inserted after a line with the line number "L" will be sequenced L+N, L+2N, L+3N, etc. If the lines following the inserted lines have line numbers which are less than or equal to the line number of the last inserted line, as many lines as necessary will be resequenced to ensure that all line numbers are unique and in ascending order. For example, assume that "N" is 2 and the user wishes to insert 9 lines after line 25 in a file that was previously sequenced by fives. The inserted lines would be numbered, 27, 29, 31 ... 43. The lines previously numbered, 30, 35, 40, 45 and 50 would be renumbered to, 45, 47, 49, 51 and 53 respectively. The remaining lines in the file would be unchanged.

REQUEST: COLON a
ABBREVIATION: CU
RESPONSE: none
ERRORS: Illegal argument

A colon (or backspace on 1050) is a logical backspace anywhere eg., 'ABC :: Db(C.R.)' is interpreted as 'DbC'.
The colon moves the character pointer back one but does not erase the characters over which it has moved. One should be careful in using this convention that the total number of characters does not exceed 84, as any extras will be added to the next line during INPUT, or result in a request during EDIT.

The COLON request allows the colon character to be inserted as text. (They may also be 'CHANGE'd in as desired.) If 'a' is T or TEST, all ':' will be treated as text except for the ':' as the first character after a tab. If 'a' is B or BACKUP the normal mode will be reinstated and all ':' will be backspaces.

REQUEST: TABSET T1 T2 ... TN
ABBREVIATION: TA
RESPONSE: none
ERRORS: ILLEGAL TAB SETTING

Ti specify the columns at which tabs are to be set. Tabs must be set in ascending order and may not exceed column 84.

REQUEST: FILE -NAME1-
ABBREVIATION: FL
RESPONSE: Ready message from CTSS
ERRORS: NO FILE NAME GIVEN
          or FILE WORD COUNT ZERO
          NOTHING IN FILE
INPUT:

This request is used to terminate the editing process and write the new edited file on the disk. NAME4 specifies that the new file will be created as NAME4 NAME2. If NAME4 NAME2 is not specified, the old file will be replaced by the edited file or a new file NAME3 NAME2 will be created. If no name was given by the initial ED command or by the FILE request, an error message will be printed and the FILE request will be ignored.

If a file to be deleted is in R1 mode, confirmation of deletion will be requested. If confirmation is denied or if file is R2 mode, the EDIT mode will be reentered with the pointer at the top of the file.
Identification

Save present dormant program.
SAVE, MYSAVE

Purpose

Allows the user to preserve the current program and condition of the machine so that it may be continued at some later time by use of the RESUME, CONTIN, or RECALL or RESTOR commands.

Usage

user: SAVE NAME1 -'t'-
error: MEMORY BOUND ZERO, NO SAVED FILE CREATED.
       NAME1 SAVED BUSY. SAVE NOT EXECUTED.

NAME1 SAVED is the name of the file to be created.

MEMORY BOUND ZERO means that the user was in DEAD status and there was no program to save.

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

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

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

't' if present, the mode of the file will be temporary.
Identification

Saving and renaming temporary file generated by RUNCOM
SAVFIL, RERUN

Purpose

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

Usage

SAVFIL NAME1
RERUN NAME1

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

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

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

MYSAVE NAME1 to save a RUCOM job.
SAVFIL NAME1

.
.
.
RERUN NAME1 to continue the RUNCOM at
CONTIN NAME1 any future time

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

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

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

Purpose

It is possible for a user to define or create a file in linkable mode so that other users may have access to it without having to copy it into their own file directories. The author of a linkable file may permit as many users as he wishes to link to his file. In order for one user to access another user's file, the author must first grant permission to the user and the user must establish a link to the file. Once a link has been established, the user has complete access to the file within the restrictions of the mode permitted by the author.

Usage

Grant permission:
PERMIT NAME1 NAME2 MODE PROB PROG ...-PROBn PROGn-

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

MODE (octal) is the mode which the author wishes to permit for the file. During the linking process, this mode will be 'or'ed with any other modes in the chain of links to determine the final mode. Any mode within the chain of links which does not contain the 'LINKABLE' mode will terminate the chain.

PROB PROG specifies the problem number and programmer number of the user to whom the file NAME1 NAME2 is being permitted. If PROB is * or 0, all problem numbers are implied. If PROG is * or 0, all programmer numbers are implied.
Withdraw permission:
REVOKE NAME1 NAME2 PROB PROG ... -PROBn PROGn-

REVOKE withdraws the linking permission for file NAME1 NAME2 of the current file directory from the user PROB PROG. Any link which may have been formed is also removed.

Form a link:

LINK NAME1 NAME2 PROB PROG -NAME3- -NAME4-

LINK establishes a link in the current file directory to the file NAME1 NAME2 in the file directory of PROB PROG. The file may be given a different name in the current file directory, if desired, by specifying NAME3 NAME4. If NAME4 is not specified, NAME2 will be used as the class name. NAME3 NAME4 is the name of the file in the other directory and NAME1 NAME2 is the name of the file in the current file directory.

If permission has not been granted or if the file does not exist or is not of linkable mode, the link cannot be established. Links may be established through any depth of file directories.

Remove a link:

UNLINK NAME1 NAME2 .... NAME1n NAME2n

UNLINK will remove the link from the current file directory for file NAME1 NAME2. This in no way affects the permission.

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

Method

Two files in the system file directory (M1416 CMFL02) contain the permission and linking information. The PERMIT FILE contains the information of NAME1 NAME2 PROB and PROG of the author and MODE as the full file description with the PROB PROG of the user to whom permission is being granted. The LINK FILE contains the same information about files to which the links have actually been formed. These two files are line-marked BCD linkable files which may be LINKed and PRINTed by any user.
Identification

Combine seldom-used files
ARCHIV

Purpose

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

Restrictions

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

Line-marked files should not be ARCHIVED.

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

Usage

ARCHIV KEY NAME1 NAME2 FIL1 ... FILn

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

KEY=P: Print file(s) FIL1 ... FILn which is (are) contained in archive file NAME1 NAME2. Only card image BCD files may be printed.

KEY=T: Print a table of contents of archive file NAME1 NAME2.

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

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

KEY=R: Replace FIL1 in the archive file NAME1 NAME2 with a copy of the file FIL1 NAME2. This involves creating a new archive file and deleting the old one. If no FIL1 exists within the archive file, a message is printed
and the command is executed as ARCHIV C NAME1 NAME2 NAME1 FIL1.

**Method**

In combining files, the last line of each file is augmented by enough blank words (6060606060606060)8 to make the number of words in the file a multiple of 14. Between the files is inserted a 14-word line consisting of six carriage returns (555555555555555)8 followed by the name of the file (12 characters) followed by 66 blanks. Thus if an archive file is printed with PRINTF, the component files will be separated by six blank lines.

Whenever ARCHIV creates a new file, it is first named "...... ......" and is of temporary mode. After this file is created, the file which it replaces, if any, is deleted and file "...... ......" is renamed and the mode changed to permanent (or R1 if it is replacing an R1 file).

Files to be deleted are handled in the standard DELETE manner, i.e., verification is requested for R1 mode files and R2 may not be deleted. If a file cannot be deleted, ARCHIV accomplishes nothing.
Identification

Compress BCD files
CRUNCH

Purpose

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

Usage

Crunch:

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

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

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

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

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

Uncrunch:

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

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

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

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

MAJ if specified in conjunction with 'NUMBER', the first three non blank characters are placed in
columns 73-75 of the source file NAME1 NAME2.

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

Print:

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

PR directs the printing of NAME1 CRUNCH

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

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

NUMBER SEQ begins the printing with alter number SEQ

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

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

List contents of User's File Directory
LISTF

Purpose

To provide the user with a table of contents of his files including date last used, mode, and the number of physical record used. Information may be requested for specific files or files used before a certain date.

Method

Among the data available in the file directory for each file are the dates last used, the file name and mode, and the number of records used. LISTF will type this data, selectively, as well as type a heading line giving the total number of files and records used.

Usage

Table of Contents:

LISTF -'REV'-

LISTF will list, in reverse chronological order by date last used, the available data for each file in the current file directory.

REV (optional) directs the listing to be in chronological order rather than reverse chronological order.

Specific file:

LISTF NAME1 NAME2

LISTF will list the available data for file NAME1 NAME2. Either NAME1 or NAME2 may be '*' in which case all files with the specified primary and secondary name will be listed.

Before a date:

LISTF MM DD YY

LISTF will list the available data of all files last used on or before the date MM/DD/YY.

MM DD YY are numeric digits expressing the desired date.
PUBLIC INFORMATION

The purpose of these guidelines is to provide information on the layout of accounts for each life insurance plan.

A graphic diagram is provided for each plan to illustrate the layout of accounts. The layout includes the number of accounts and the number of policies assigned to each account. Each account has a specific identification number assigned to it. The identification number is used to reference the account in the system.

Special features such as bonus and surrender values are also illustrated. The layout includes the number of bonus values and the number of surrender values assigned to each account.

The guidelines also include a table of sample accounts for each plan. This table provides a visual representation of the layout of accounts for each plan.

The guidelines are designed to be a comprehensive resource for anyone interested in understanding the layout of accounts for each life insurance plan.
Identification

Print BCD card image files
PRINTF

Purpose

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

Usage

PRINTF NAME1 NAME2 -SEQ-

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

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

Print a BCD file
PRINT

Purpose

To print the contents of a BCD file, either line-numbered or line-marked. Specific lines and special format may be requested.

Restrictions

The maximum record length is 132 characters.

Usage

PRINT NAME1 NAME2 -LINES- -FIELDS-

PRINT will normally print line-numbered files as characters 73-80, blank, then characters 1-72. Line-marked files will be printed characters 1 to the last with 132 characters per line of type.

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

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

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

Line-numbers are assumed for card image files and line-marked files of 14 words of information. The mode is switched to record number upon encountering any line-marked record different from 14 words (no switch back possible). Using THRU instead of TO forces the record number mode.

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

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

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

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

Print contents of a file in octal
PRBIN

Purpose

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

Usage

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

O (optional) the presence of 0 indicates that the following arguments are expressed in octal rather than decimal.

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

R (optional) Printing will be grouped into blocks of R words (5 words per line) where each block is labeled with its rank, i.e., N, N+R, N+2R etc. Naturally if R is to be specified, N may not be omitted. If R is omitted, it is assumed 10 (or 8 if 0 is specified). R may not exceed 40.

PRBIN terminates by calling CHNCOM.
Identification

Print summary of BSS files.
PRBSS

Purpose

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

Usage

PRBSS NAME1 -PROG-

PRBSS prints the summary of information about the programs in file NAME1 BSS, possibly three or more lines per program.
1st line: Entry names and their relative locations
2nd line: Common, program, and transfer vector lengths
3rd line: Names in the transfer vector, if any.

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

BREAK A single interrupt terminates the command. PRBSS terminates by calling CHNCOM.
TRANSLATION

FIGURE 1

To obtain a summary of information from the program in the life of the operator, fill out the form on your sheet of paper. Insert the name of the program in the "NAME" column and the number of information requests in the "QUANTITY" column. Each line in the program sheet represents the number of the program and its number in the order of execution. Insert your name in the "L.E.R.S." column, along with any comments or remarks you may have regarding the program. This will help in identifying the operator and the program for future reference.
Identification

Print SAVED file
SDUMP

Purpose

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

Usage

SDUMP NAME1

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

SDUMP NAME1 LOC N

The contents of N consecutive locations beginning at octal location LOC, of the core image contained in file NAME1 SAVED will be printed on the user's console. All registers are typed in octal with mnemonics. Lines of all zero are omitted. The decimal integer N may not exceed 1000.
To print the machine configuration under location within a

The machine configuration of the KAMER 2 A 9320
will be printed on the user's computer.

The computer or its coordinating locations
beginning at zero location LOC of the core memory containing the KAMER 2 A 9320 will be

The machine configuration of the KAMER 2 A 9320 consists of

The user's computer. All references to the user's computer.

The computer is accessed by the access numbers

...
Identification

Combine files
COMBIN

Purpose

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

Usage

COMBIN SEQ NAME1 NAME2 FILL1...FILn

COMBIN will combine files FILL1 through FILn of secondary name NAME2 into one file NAME1 NAME2 within the current file directory. If any FIL cannot be found, the NEED-USE convention will be followed (see Section AH.7.01). Within the USE process, an * for a corresponding FIL means that FIL should be ignored. The combining will not begin until all FIL's are accounted for. FIL's are not deleted.

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

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

EXAMPLES:

If characters 73-80 of the first line of FILL1 are ABC123GH and SEQ = 1, the new sequence for NAME1 NAME2 will begin with ABC00010.
If the first line contains Abbbbbb and SEQ = 1, the new sequence will begin with Abb00010.

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

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

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

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

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

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

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

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

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

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

Mode:
There are three kinds of files which may be SPLIT:

1) Line-numbered - BCD card images (14 words) with numeric sequence number in column 76-80.
2) Line-marked or variable length records preceded by an extra word which contains the word count of the record.

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

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

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

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

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

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

Purpose
To change the mode or the name of a file or to delete a file.

Usage
Delete:

DELETE NAME1 NAME2 .... NAME1n NAME2n

Delete all versions of files NAME1i NAME2i from the current file directory, if possible. R2 mode files may not be deleted and the following message is printed:
*ATTEMPT TO DELETE FILE IN READ ONLY MODE.

Attempts to delete R1 files results in the following message:
NAME1i NAME2i R1 MODE. DO YOU WANT TO DELETE IT,
Type either yes or no after the comma and hit carriage return. If for any other reason a file cannot be deleted, a message is printed.

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

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

Change mode:

CHMODE NAME1 NAME2 MODE1 ....NAME1n NAME2n MODEn

MODE must be 0, 1, 2, 3, T, P, R1 or R2. The mode of the file in the current file directory will be changed to MODE, if possible. R2 mode files may not be changed and the following message is printed:
**TRIED TO RENAME READ ONLY CLASS 2.

If the mode can not be changed for any reason, a message is printed.
NAME1i NAME2i - The same * convention is used as in the DELETE command.

Rename:

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

RENAME change the file name NAME1 NAME2 to the name NAME3 NAME4. All other files NAME3 NAME4 will be deleted before renaming NAME1 NAME2. The deleting of NAME3 NAME4 has the same options and messages as under DELETE, if NAME3 NAME4 cannot be deleted or if NAME1 NAME2 is R2 mode, no names are changed and a message is printed.

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

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

NAME4n missing, is assumed to be NAME2n.
Identification

Common files
COMFIL, COPY, UPDATE

Purpose

A group of common files is assigned to programmers working on the same problem number. The COMFIL command allows the user to switch the current file directory to be one of the common file directories or to switch back to his own. The UPDATE command allows the user to transfer one of his files into one of the common file directories. The COPY command allows the user to copy a file from a common file directory into his own file directory.

Usage

Comfil:

    COMFIL -N-

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

    COMFIL switches the current file directory to N so that all subsequent commands will refer to directory N. Temporary files created in common files 1, 2, 3, or 4 will be lost on switching back to the user's directory or to another common file directory.

Copy:

    COPY N NAME1 NAME2....NAME1n NAME2n

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

    N may be 1, 2, 3, 4, S. S allows copying from the system file directory which contains the BSS foreground library and the saved version of public commands (with secondary name TSSDC.)
Update:

UPDATE N NAME1 NAME2...NAMEIn NAME2n

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

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

Library file
EXTBSS, UPDBSS

Purpose

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

Usage

Extract:

```
EXTBSS NAME1 -NAME3- PROG
```

EXTBSS will extract from the library file NAME1 BSS the first BSS routine with the entry PROG and create a file NAME3 BSS. Older files of NAME3 BSS are deleted, if possible. NAME1 BSS is unchanged.

NAME3 If NAME3 is omitted (only 2 arguments) the newly created file will be PROG BSS. Omission of both arguments causes the printing of instructions for the use of the command.

Update:

```
UPDBSS NAME1 NAME3 -PROG-
```

UPDBSS searches the library file NAME1 BSS for the first BSS routine with the entry PROG and replaces that routine with the entire file NAME3 BSS. This is accomplished by creating a new file NAME1 BSS and deleting the old, if possible. If NAME1 BSS can not be deleted, no updating is accomplished.

PROG If PROG is omitted, the entry name is assumed to be NAME3. A main subprogram is designated by (MAIN). If no entry PROG can be found the message "PROG NOT FOUND IN LIB NAME1" will be printed and no updating will be done.

NAME3 If NAME3 is '/*', the routine with entry name PROG is deleted from NAME1 BSS.
Identification

Off-line processing
REQUEST

Purpose

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

Usage

REQUEST XX NAME1 NAME2 OP

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

XX=DPUCH: The BCD file NAME1 NAME2 is punched off-line. If the file is line-marked, just the first 80 characters per line of data will be punched.

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

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

XX=DELETE: The file NAME1 NAME2 will be deleted from the current file directory.

XX=CHMODE: The mode of file NAME1 NAME2 will be changed to OP. UP may be 0, 1, 2, or 3 for temporary, permanent, R1 or R2, respectively. If the new mode is temporary, the track count will be adjusted. (Temporary files are not counted against the user's track quota.)
The file NAME1 NAME2 will be carried to the other computer and will be loaded onto the disk during the next load or update. It will be loaded as permanent mode, as the same name NAME1 NAME2, within the same problem-programmer file directory. If a different problem-programmer specification is desired for the receiving file directory OP may be PROB PROG, i.e., the desired problem programmer numbers. If a different file name is desired, OP may be PROB PROG NAM1 NAM2, where PROB PROG must be the problem programmer numbers for the receiving file directory and NAM1 NAM2 may be the name given to the input file. The file NAME1 NAME2 on this machine is in no way changed. Any previous versions of the file on the receiving machine will be deleted regardless of mode.

The file(s) will be PRINTed, DPUNCHed, dPUNCHed, or 7PUNCHed and then the mode will be changed to temporary. The next time the file is read or the user logs out, the file will be deleted. This is safer than a process request followed by a DELETE request. In case of machine or tape failure during the processing of the request, the operations staff will have the opportunity of restarting the disk editor.

Method

The REQUEST command creates or appends to a file in the user's file directory called OUTPUT REQUEST. This file contains control card images which will be processed by the disk editor program. After processing, the disk editor program will change the mode of OUTPUT REQUEST to temporary. This change to temporary allows the operations staff to rerun the disk editor if any difficulty was encountered in the first run. Note that OUTPUT REQUEST contains only the control cards which point to the actual files to be processed. The disk editor program, upon processing the request files, will generate three different tapes: printer, punch, and carry. These tapes are then the responsibility of the operations staff.
Identification

Relocatable program loading
LOAD, LOADGO, VLOAD, NLOAD, L, USE

Purpose

There are five different types of loading available for relocatable programs i.e., SSS files. The first (LOAD) will load a program into core without destroying the loader or MOVIE) table, place the program in dormant status and return to the user for the next command. The second (LOADGO) is the result of the chain of commands LOAD and START. The third (VLOAD) will load the program; move all of the program and COMMON down in core to destroy the loader and MOVIE) table (thereby making the available core larger); place the program in dormant status and return to the user for the next command. The fourth (NLOAD) is the same as VLOAD except that erasable COMMON is also destroyed so that no library routines which use erasable COMMON may be used. The fifth (L) is a separate command which allows any one of the previously mentioned four to be used with larger loading tables (see Restrictions).

Programs or files may be loaded (or searched as library files) from the user's file directory, from his common files and from several system files.

If needed routines cannot be found by the loader, the USE command may be used to specify which routines may be used instead.

Restrictions

Normal table sizes are: MOVIE) table is 500 words and the table of missing entries is 100.

The tables for the L command are: MOVIE) table of 1200 words and missing entries of 250 words.

Usage

Any of the load commands (LOAD, LOADGO, VLOAD, NLOAD) may be used in place of LOAD; all special arguments are optional and order is significant by meaning or where specified. Special arguments are those beginning and ending with parentheses as shown. They cause the loader to behave in a special manner. The non-special arguments are either file names or entry points, depending upon the preceding special arguments.

Upon completion of loading, the current file directory is switched to its initial status.
LOAD (OLD) (ORG) (CFLn) (LIBE) (SYS) (NEED) (NLIB) NAMES (MORE)

(OLD) The presence of (OLD) as the first argument instructs the loader to replace itself with the old loader and to use the old library version of 11/18/64. These old versions will be maintained in the system file for some short time for compatibility with old BSS files.

(ORG) The presence of (ORG) instructs the loader to set the starting address to the entry name specified by the next non-special argument following (ORG). The non-special argument must be a primary file name.

(NEED) The presence of (NEED) instructs the loader to treat the next non-special argument as a program entry point as though it had been an entry in a transfer vector.

(MORE) may be the last argument before the carriage return (because only one line can be interpreted by the command) to indicate that more arguments will be specified. In this case the loader will not print the NEED list; will restore the common file switching to the initial setting; and return to the user (by way of CHNCOM) so that the USE command may be used.

(CFLn) directs the loader to switch the current file directory to common file directory n which may be 0, 1, 2, 3, or 4. The current file directory is initially the user's file directory or a directory set by a COMFIL command. There may be any number of these switches in the argument list and each one supercedes the previous one.

(LIBE) directs the loader to use the next non-special argument as a file within the current file directory to be searched as a library file to find any missing routines.

(SYS) directs the loader to use the following non-special argument as a file from the system file directory to be searched as a library for any missing routines.

(NLIB) directs the loader not to search the system library (i.e., TSLIBL) for missing routines after the argument list has been processed.
(LIB) supercedes (NLIB).

NAMES may be the primary names of BSS files to be loaded or BSS files to be searched as libraries following certain special arguments or routine entry points following other special arguments.

NEED Following the processing of the argument list, the system library TSLIB1 will be searched for any missing routines (unless prohibited by (NLIB)). If routines are still missing, the current file directory is switched to the user's directory, a list of needed routines (by entry names) is typed by the loader and DURMNT is called so that the user may type the USE command. Upon completion of loading, the current file directory is switched to its initial status.

USE will reinstate the last common file switching and go back to the loader. All of the arguments available to the loader are therefore available to USE.

LOAD sets the origin of the first program at (5200)8. The MOVIE) table and the loader are left inviolate below this origin. COMMON addresses are relocated with the same parity as on the assembly listing. FAP coded subprograms, which contain the EVEN pseudo op, will be loaded with relative location 0 in an even core location. Upon completion, all loaders call CHNCOM with an available core image specified.

LOADGO is equivalent to the sequence of commands LOAD and START.

VLOAD After the entire program and library subroutines have been LOADED the program is moved down so that the origin is (30)8, covering the loader and the MOVIE) table. The (31b)8 words of erasable COMMON are included with the program. The MOVIE) table will be preserved if MOVIE) occurs in the transfer vector of any routine loaded.

NCLLOAD is the same as VLOAD except that the (316)8 words of erasable COMMON are not included and, therefore, if library subroutines which use erasable common are included, a COMMON assignment error message will be printed.
L  The L command may be used if larger loading
tables are needed (see Restrictions). The L
precedes any one of the LOAD commands as: L
LOAD ARGUMENTS. If the loader name is
omitted, it is assumed to be LOAD. All of the
regular loader arguments are available except
(OLD). The program loading by LOAD starts as
(7000)8 instead of (52000)8. There may be more
than 250 missing entry names if this does not
occur during a library search. L always calls
CHNCOM, regardless of the outcome of the
loading. No core image is kept if loading
failed.

MOVIE) table is created by the loader to provide a
storage map of all entry points of routines as
they are loaded. It is always written as a
file (MOVIE TABLE) in the user's directory in
temporary mode. If the entry MOVIE) appears in
the transfer vector of any routine loaded, by
VLOAD or NLOAD, the MOVIE) table will be
preserved by moving it to the top of the load.
The MOVIE) entry points to location (27)8
which contains the MOVIE keyword which
contains the number of words in the movie
table in the decrement and the location of the
lowest word in the movie table in the address.
The format of the MOVIE) table, starting with
the lowest location, is:

1. fence
2. Lowest common break (address)
3. SVN prefix
4. Memory bound (address)
5. BCD entry name
6. Entry point for previous name
   (address)
   .. Pairs of words 5 and 6 for each
   entry to the subprogram.
7. SVN prefix or PZE 0,,n, where there
   are n words in this program.
8. Origin of this subprogram (address)
   .. Repeat groups 5 thru 8 for each
   subprogram loaded.
### STORAGE MAP

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>8</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>ZERO</td>
<td>VLOAD: PROGRAM, NCLOAD: PROGRAM</td>
</tr>
<tr>
<td>10-23</td>
<td>BOOTSTRAP for NCL\text{OAD} and VLOAD or TSX LOAD,4</td>
<td>TO (77461)8</td>
</tr>
<tr>
<td>24</td>
<td>TSX (ORG),4</td>
<td>ERASABLE COMMON TO</td>
</tr>
<tr>
<td>25</td>
<td>TSX (ORG2),4</td>
<td>(77461)8</td>
</tr>
<tr>
<td>26</td>
<td>TSX (ORG3),4</td>
<td>PROGRAM</td>
</tr>
<tr>
<td>27</td>
<td>MOVIE) Keyword</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>30</th>
<th>LOADER FOR LOAD, LOADGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>4661</td>
<td>VLOAD: PROGRAM, COMMON TO</td>
</tr>
<tr>
<td></td>
<td>(77461)8</td>
</tr>
<tr>
<td>5200</td>
<td>PROGRAM COMMON</td>
</tr>
<tr>
<td></td>
<td>ERASABLE COMMON</td>
</tr>
<tr>
<td></td>
<td>TO</td>
</tr>
<tr>
<td></td>
<td>(77777)8</td>
</tr>
<tr>
<td></td>
<td>PROGRAM</td>
</tr>
<tr>
<td></td>
<td>MB3</td>
</tr>
<tr>
<td></td>
<td>MB2</td>
</tr>
</tbody>
</table>

\[ \text{MB3} = 30 + 77461 - \text{COMMON BREAK} + \text{PROGRAM LENGTH} \]
\[ \text{MB2} = 316 + \text{MB3} \]
\[ \text{MB1} = 4632 + \text{MB2} \]
Identification

Absolute Program loading
LDABS

Purpose

To load a program from a file containing absolute column binary card images.

LDABS NAME1

NAME1 is the primary name of the file NAME1 ABS which contains absolute column binary card images with full word checksum.

LBABS occupies the upper (1271)8 locations of core and it will load a program into lower core with an upper limit of (77503)8. The memory bound is set, upon completion of the load, to the highest location loaded. Loading terminates with a transfer card and execution may be started at the transfer-location by issuing the START command.

Error Conditions:

a) If a check sum error occurs, the comment: CHECK SUM ERROR IN CARD XXXXX is printed, where XXXXX is the location in which the first word on the card is to be stored. After this comment is printed the card is ignored and loading continues.

b) If an attempt is made to store in a location greater than (77503)8, the comment: CARD YYYYYY OVER BOUND is printed, the card ignored, and loading continued.

c) If a transfer card is missing, i.e., an end of file is reached, the comment: TRANSFER CARD MISSING, TYPE OCTAL STARTING LOCATION is printed. The last five characters typed are converted to an octal location and the transfer location for starting is set up.
Identification

Start or continue execution
START, RESTOR, RESUME, R, RSTART, CONTIN, RECALL

Purpose

A program which has been LOADED may be STARTed or a program which has been SAVED may be RESUMEd. In other words, programs may be stopped, saved, and subsequently restarted.

Usage

START
RESTART

The START command may be used to begin a program which has been loaded by one of the LOAD commands, or it may be used to continue a dormant program from the place of the last interruption.

RSTART is equivalent to START, except that it is transparent to the current command buffer and command location counter. It should be used when restarting a chain of commands.

RESTOR NAME1
RECALL NAME1

The RESTOR and RECALL commands will restore the core image from NAME1 SAVED complete with active files, if any. The program is placed in dormant status so that it may be STARTed in order to continue from its last interruption.

RECALL also restores the command list and common file switching from NAME1 SAVED, and sets aside the command location counter and current command buffer in case of a subsequent RESTART.

RESUME NAME1 P1 P2 ... Pn

The word RESUME may be abbreviated by the letter R. The RESUME command is effectively the same as RESTOR and START. The parameters are placed in the current command buffer so that it contains NAME1 P1 P2 ... Pn. This is a technique for writing and checking out a new command.

CONTIN NAME1

The CONTIN command should be used to resume a program involving a chain of commands. It restores the program and machine conditions from NAME1 SAVED, together with any active files, the common file switching, and the contents of the command list, command location counter, and current command buffer. In other words, it is exactly equivalent to the chain of RECALL and RSTART.
Identification

General discussion of debugging commands.

Method

There are three different kinds of commands within CTSS, one of which is of no importance in this discussion. The first kind is often referred to as a "disc-loaded" command. The distinctive property is that the supervisor loads the command from a core image SAVED file and thereby eliminates any previous core image the user might have had. The second kind is often referred to as a "core-B transfer" command. Here the distinctive property is that the supervisor does not load the command, but instead, transfers to the relocating loader which is already in core-B. The loader then determines which command is specified and proceeds to load the command from a standard BSS library file (TSLIB2) into the area of core above the current core image. If the command has already been loaded, the loader merely transfers to the desired entry point.

Some of the present debugging commands are core-B transfer commands. The earliest routine available to CTSS was called FLEXPM which includes the commands PM, PATCH, STOPAT, and TRA. More sophisticated commands have been written more recently, such as FAPDBG and STRACE. These routines are able to make use of the tables created by the translators and the loader, such as the MOVIE table and symbol table files. The use of these commands imposes some restrictions on the user, namely that the vanishing and absolute loaders not be used and that the symbol table files from the translators be available and of the proper format.

Programs which extend the memory bound during execution create some problems in connection with the debugging routines. Note that the core-B transfer commands are relocatable BSS subroutines with normal entry points. If the debugging routine is loaded after the program has started execution, there may be a conflict about the space acquired by expanding memory bound. Therefore, the solution is to force the debugging subroutines to be loaded with the program before execution. This may be accomplished either by placing one of the entry points in a transfer vector of one of the loaded programs or by use of the special arguments to the LOAD command.

The SP command is a disc-loaded command which may be used only by the system programs for patching core-A.

The SD command may generally be used for examination of locations in core-A.

The MADBUG command is a disc loaded command which serves as an intermediate supervisor between the user and the CTSS
supervisor. MADBUG allows the user to specify a MAD source file rather than BSS file. MADBUG manages all the calls to the MAD translator and the appropriate loader so that the restrictions implied by the core-B transfer routines are not as evident to the user.
Identification

FAPDBG - A symbolic debugging aid for FAP program
R. H. Campbell - x

Purpose

FAPDBG, as a symbolic debugging aid for FAP programs, was produced as an experiment with typing conventions and formats. FAPDBG acts upon requests typed by the user on the console and performs such functions as examining and typing or changing the contents of specified registers and allowing a subprogram to be run in controlled segments.

Reference

CC-216 FAPDBG, a symbolic debugging aid R. H. Campbell

Usage

LOAD NAME1 ARGUMENTS
FAPDBG ALPHA
requests

The FAPDBG command can be issued anytime a program is dormant and the loader is available, i.e., may not have been loaded by a self-erasing loader. If the program extends memory bound or damages the loader, FAPDBG should be called before execution. The FAPDBG command calls the loader to load the FAPDBG subprogram from the debug library, "TSLIB2". FAPDBG uses the loader's symbol and loading tables to build its own symbol table (800 symbols maximum) for the subprograms which the user wishes to debug. FAPDBG is approximately (12400)8 locations in length.

If the line-numbered file ALPHA DEBUG can be found, requests are taken from there. When ALPHA DEBUG is exhausted, not found, or not specified, requests will be taken from the console.

Conventions:

1) A request is a single letter request name followed by arguments, all separated by blanks.
2) A blank is a string of any number (not zero) of spaces or tabulations.
3) Any number of requests may be concatenated on one line by typing an apostrophe or an equal sign between successive requests. Concatenation is recommended since FAPDBG will be brought into core less often and will generate less output.
4) If a request cannot be accomplished, FAPDBG will so inform the user and return to process the next request.
5) Syntax - The location, address, tag, and decrement parts of a request argument may consist of strings of symbols and octal numbers separated by plus and minus signs to denote the desired algebraic manipulation. The indicated operations are carried out, any negative result is converted to two's complement form and the right fifteen bits saved (in the case of the tag field, only the right three bits are saved). Symbols, which must be defined, may consist of any number of characters, at least one of which must be non-numeric (i.e., not 0 through 7), and none of which may be one of the special characters plus, minus, comma, space, or tabulate. If the number of characters is greater than six, only the last six will be used. Any string consisting only of the digits 0 through 7 will be considered an octal number of five digits, with left zeros if necessary. If more than five digits are typed, only the last five will be used. The line typed in is scanned from the left and each field is evaluated when encountered. If an undefined symbol is discovered, or a deviation from an understandable format is discovered, an appropriate comment is typed and processing of the request is terminated. If one or more requests cannot be interpreted, any go or proceed requests following them on the same line will be ignored.
There are four classes of requests: set up, register examination and modification, subprogram control, and FAPDBG control.

**SET UP REQUESTS:**

The set up requests are necessary to tell FAPDBG which subprograms are to be debugged and allow FAPDBG to build the necessary symbol tables. These requests are Load address, symbol Table, Work, and Equals.

**LOAD ADDRESS:**

ENTRY is an entry point of the subprogram to be debugged. The origin of the subprogram will be typed out and will be used as the relocation constant for all symbols within that subprogram.

**SYMBOL TABLE:**

T -NAME1-

All the symbols from the file NAME1 SYMTB will be relocated by the origin printed from the last L request and placed in the FAPDBG symbol table. Note that this means absolute symbols and COMMON (except for the first-loaded) will be incorrect.

Successful completion is signaled by "SYMBOLS LOADED". If the FAPDBG symbol table becomes full, the last symbol entered will be typed out. Note that the symbols in the SYMTB file are in alphabetic order.

If NAME1 is omitted, all of the symbols will be deleted from the FAPDBG symbol table.

**WORK:**

W ENTRY -NAME1-

W is the combination of L and T requests.

NAME1 need not be specified if ENTRY and NAME1 are the same.

**EQUALS:**

E FE FS

FS is the symbol to be entered in the symbol table with the value of the expression FE.

FE is a FAP expression involving constants and/or symbols already entered in the symbol table (see convention 6.)
Register Examination and Modification

The register examination and modification requests permit the user to examine and change the contents of core locations as well as the live registers. They are look (floating point, Hollerith, full word integer, decrement integer, octal, symbolic), deposit, compare, signed and logical accumulator, and storage map.

LOOK:  

-request- -LOC1- -LOC2-

request sets the output conversion mode and if an argument is specified, prints the specified locations. Request may be one of the following:

F Floating point
H Hollerith
I Full word integer
J Decrement integer (Fortran)
O Octal
S Symbolic

LOC1 LOC2 are FAP symbolic expressions specifying a block of core from LOC1 through LOC2.

LOC1 specifies a single location.

The contents of a single location in the current output mode may be obtained by typing just the location expression without the look request with the restriction that the first symbol in the expression may not be a single letter. The contents of "* + 1" may be obtained by an empty request (just a carriage return or concatenation character).

DEPOSIT:  

D LOC FW

FW is the FAP word to replace the previous contents of location LOC.

This request may be abbreviated by omitting the request name, provided that the location expression does not begin with a single-letter symbol. The FAP word may be a symbolic machine instruction such as CAL ALPHA-10,4 or one of the data generating pseudo instructions OCT, BCD, FLU, INT (full word decimal integer), or JNT (decrement integer) followed by a blank and one word of data.
A symbolic machine instruction consists of a symbolic operation code, an optional asterisk to indicate indirect addressing, and an optional variable field in the same format as accepted by FAP, except that all numbers are interpreted as octal and that multiplication and division are not allowed. No blank may intervene between the operation code and the indirect flag; a blank must, however, precede the variable field. Note that since the address field is truncated to fifteen bits, the left three bits of the address part of type D instructions (left and right half indicator operations) will be considered by FAPDBG as the tag field, both for input and for output. Thus to insert the instruction

\[ \text{RFT 300105} \]

it is necessary to type

\[ \text{RFT 105,3} \]

The OCT pseudo instruction accepts a signed or unsigned octal integer of magnitude less than or equal to 377777777777. Thus, to insert the traditional fence, it is necessary to type

\[ \text{OCT -377777777777} \]

The FLO pseudo instruction accepts a signed or unsigned floating point number with optional decimal point and optional E modifier to denote multiplication by the indicated power of ten. The B modifier is not allowed.

The INT and JNT pseudo instructions accept signed or unsigned decimal integers of sufficiently small magnitude to fit into the number of bits available (34359738367 for INT and 131071 for JNT).

The BCD pseudo instruction accepts any string of characters preceding the request terminator and assembles the last six into one word. If fewer than six characters are typed, spaces will be inserted on the left. Note that this pseudo instruction uses the input line image after FAPDBG has edited and "normalized" it. Therefore a string of spaces and tabulations will be interpreted as a single blank.

\[ \text{COMPARE and VERIFY: C ENTRY -NAME1-} \]
ENTRY is the entry point of a subprogram already loaded in core.

NAME1 BSS is the name of the file which is to be compared with the core image of ENTRY. NAME1 need not be specified if it is the same as ENTRY.

By using the origin value of the ENTRY subprogram, it will read and relocate each word in NAME1 BSS and compare it with the corresponding word in core. If a discrepancy is found, FAPDBG will type in the current mode the location, the word from NAME1, and the contents of the memory location for which there is a discrepancy. "EXAMINATION CONCLUDED" will signal the completion of the request. The request may be terminated by a single interrupt; FAPDBG will close the BSS file and return to process the next request.

**ACCUMULATOR:**

A-FW- or K-FW-

A places the FAP word 'FW' in the signed accumulator and clears the P and Q bits.

K places the FAP word 'FW' in the logical accumulator and clears the sign and Q bits.

A (or K) without argument types out, in the current mode, the contents of the signed (logical) accumulator followed by the P and Q (sign and Q) bits.

**STORAGE MAP:**

M

M requests the typing of the storage map with subprograms listed in order of loading. The map includes the origin and entry points with their locations.

**Subprogram Control**

The requests which have to do with subprogram control allow the user to run his subprogram in controlled segments. They are break, go, and proceed.

**BREAK:**

B-Loc-

Conditions FAPDBG to insert a "breakpoint" at location LOC. FAPDBG will save the location
and set an indicator to signal that a breakpoint instruction, specifically a transfer into FAPDBG, is to be inserted into that location. No subprogram modification occurs at this time. An examination of the breakpoint location will reveal its original contents and changing the contents (via a deposit request) will not remove the breakpoint. The breakpoint must not be placed at a subprogram-modified instruction or where it would be used for indirect addressing. Only one breakpoint at a time may be inserted.

The omission of LOC in the request causes the breakpoint to be removed.

**GO:**

```
G LOC
```

This allows the user to start execution of the subprogram at location, LOC. FAPDBG will examine the breakpoint flag and, if a breakpoint exists, will save the contents of the break location and insert the necessary transfer instruction. It will then restore the machine conditions, and transfer to the specified location.

**PROCEED:**

```
P
```

This allows the user to continue executing his subprogram from the state it was in just before control last entered FAPDBG. Upon encountering the breakpoint transfer instruction, control will be transferred to FAPDBG, which will save the machine conditions and restore the temporarily-removed instruction at the break location. FAPDBG will then type "BREAK." and wait for requests.

Proceed will cause FAPDBG to perform all the steps performed by go, except that after restoring the machine conditions, FAPDBG will execute the above-mentioned instruction and transfer to the appropriate location following its location as governed by any skipping which might occur. If the instruction is location-dependent, namely TSX, STR, STL, or XEC, FAPDBG will interpret it as if it were being executed from its normal location. Thus a breakpoint may be inserted at a subroutine call. A chain of XEC instructions will be interpreted to a maximum depth of ten. A subprogram in operation may be interrupted at
any time by pressing the interrupt button.

**Internal Operation**

The request which controls the internal operation allows the user to return to CTSS. It is **QUIT**.

**QUIT:** Q

Returns control to the Time Sharing Supervisor in such a way that a **START** command will transfer control to the place in the user's subprogram where it last entered dormant status.
Internal Symbols

The following symbols are permanently defined in FAPDBG as locations where the machine conditions are stored.

$\mu\nu$  The multiplier-quotient register.
$\zeta$  The signed accumulator
$\kappa$  The logical accumulator
$\pi$ The sense indicator register.
$\chi_{0}$  Index register one.
$\chi_{2}$  Index register two.
$\chi_{3}$  Index register three.
$\chi_{4}$  Index register four.
$\chi_{5}$  Index register five.
$\chi_{6}$  Index register six.
$\chi_{7}$  Index register seven.
$*$  The current location.

This symbol is defined as the last location referred to by either the user or FAPDBG. It is redefined as the location of the next instruction to be executed in the user's subprogram by encountering a breakpoint or by a manual restart.

$\lambda\varsigma$  Lights and switches.

This location contains the state of the machine conditions in the right-most eight octal digits as listed below; the off status is represented by zero, on status by one. Reading from left to right:

<table>
<thead>
<tr>
<th>DIGIT</th>
<th>CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Floating point trap mode.</td>
</tr>
<tr>
<td>6</td>
<td>Divide check light.</td>
</tr>
<tr>
<td>7</td>
<td>Overflow light.</td>
</tr>
<tr>
<td>8</td>
<td>Multiple tagging light.</td>
</tr>
<tr>
<td>9</td>
<td>Sense light one.</td>
</tr>
<tr>
<td>10</td>
<td>Sense light two.</td>
</tr>
<tr>
<td>11</td>
<td>Sense light three.</td>
</tr>
<tr>
<td>12</td>
<td>Sense light four.</td>
</tr>
</tbody>
</table>

$\sigma\omicron$  The instruction location counter.

This location contains the address of the next instruction to be executed in the user's subprogram. It is set by encountering a breakpoint or by a manual restart. It is examined by the proceed request in order to determine the location to which to transfer control.
### Summary of Requests in Alphabetic Order

<table>
<thead>
<tr>
<th>Request</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A FW</td>
<td>CLA =FW</td>
</tr>
<tr>
<td>A</td>
<td>Type out signed AC with P and Q</td>
</tr>
<tr>
<td>B LOC</td>
<td>Insert breakpoint at LOC</td>
</tr>
<tr>
<td>B</td>
<td>Remove breakpoint</td>
</tr>
<tr>
<td>C EP FN</td>
<td>Compare and type discrepancies of subprogram EP and FN BSS.</td>
</tr>
<tr>
<td>D LOC FW</td>
<td>Deposit =FW in LOC</td>
</tr>
<tr>
<td>E FE FS</td>
<td>Define symbol FS equal to expression FE</td>
</tr>
<tr>
<td>F LOC</td>
<td>Set output mode to floating point</td>
</tr>
<tr>
<td>F LOC1 LOC2</td>
<td>Set floating point and type C(LOC1 thru LOC2)</td>
</tr>
<tr>
<td>G LOC</td>
<td>Go to LOC</td>
</tr>
<tr>
<td>H LOC</td>
<td>Set output mode to Hollerith</td>
</tr>
<tr>
<td>H LOC1 LOC2</td>
<td>Set Hollerith and type C(LOC1 thru LOC2)</td>
</tr>
<tr>
<td>I LOC</td>
<td>Set output mode to decimal integer</td>
</tr>
<tr>
<td>I LOC1 LOC2</td>
<td>Set integer and type C(LOC1 thru LOC2)</td>
</tr>
<tr>
<td>J LOC</td>
<td>Set output mode to decrement integer</td>
</tr>
<tr>
<td>J LOC1 LOC2</td>
<td>Set decrement integer and type C(LOC1 thru LOC2)</td>
</tr>
<tr>
<td>K FW</td>
<td>CAL =FW</td>
</tr>
<tr>
<td>K</td>
<td>Type logical AC with S and Q.</td>
</tr>
<tr>
<td>L EP</td>
<td>Find and type origin of subprogram EP</td>
</tr>
<tr>
<td>L EP FN</td>
<td>Type storage map</td>
</tr>
<tr>
<td>M LOC</td>
<td>Set output mode to octal</td>
</tr>
<tr>
<td>O LOC</td>
<td>Set octal and type C(LOC)</td>
</tr>
<tr>
<td>O LOC1 LOC2</td>
<td>Set octal and type C(LOC1 thru LOC2)</td>
</tr>
<tr>
<td>P</td>
<td>Proceed (location in $LC) or interrupt after break</td>
</tr>
<tr>
<td>Q</td>
<td>Quit and return to CTSS</td>
</tr>
<tr>
<td>S LOC</td>
<td>Set output to symbolic</td>
</tr>
<tr>
<td>S LOC1 LOC2</td>
<td>Set symbolic and type C(LOC1 thru LOC2)</td>
</tr>
<tr>
<td>S LOC1 LOC2</td>
<td>Add symbols from FN SYMTD to symbol table (relocated by last origin typed out)</td>
</tr>
<tr>
<td>T FN</td>
<td>Remove all symbols from symbol table</td>
</tr>
</tbody>
</table>
Identification

MADBUG - A MAD Debugging System
Robert S. Fabry

Purpose

MADBUG is a system under which the user can create and debug programs written in the MAD programming language. MADBUG allows the user to input and edit symbolic programs and to execute in a controlled way and interrogate the derived machine language programs. The most important consideration in the design of MADBUG was ease in learning and using, both for the beginner and for the advanced programmer. MADBUG is unusual in that it utilizes information which has been previously ignored. This information comes from: (1) the sequence in which the user types his requests, (2) the files available in the user's file directory, (3) the expanded information content of the new MAD symbol table files developed for MADBUG, and (4) the information inherent in the very limited, stylized set of coding sequences generated by a compiler. The use of this additional information manifests itself in two ways: (1) the user need provide very little information to accomplish a given task, and (2) the user does not have to understand assembly languages, machine languages, octal numbers, relative or absolute addresses, symbol tables, machine representations of constants, or any of a host of similar items. The MADBUG requests of CHANGE, DELETE, INSERT, and APPEND demonstrate the influence of the "Expensive Typewriter" program written for the PDP-1 by Steve Piner. The "DDT" program written for the PDP-1 by Robert Saunders and the "FLIT" program written for the TX-0 by Jack Dennis and Thomas Stockham have influenced the OPEN, VERIFY, BREAK, and KILL requests.

Reference

CC-247 and Mac-M-205 MADBUG: A MAD DEBUGGING SYSTEM R.S. Fabry
A DESCRIPTION OF MADBUG

MADBUG is instructed by requests, typed one per line. A request line is made up of the name of the request followed by its arguments, with one or more blanks for separation. Request names may be abbreviated by their first letter. In request lines, tabulation characters are equivalent to blanks. There may be blanks before the request name and after the last argument; blank request lines are ignored. Since blanks are used as delimiters, the arguments, which may be as complicated as "a(1)+1...b-3", must be typed without internal blanks. A request which operates on variables will operate on single variables or on blocks of variables, specified in the usual MAD manner as "alpha...beta"; a request which operates on cards will operate on single cards or on blocks of cards. For example, "verify alpha beta(1)...beta(3) k(1,1,1)" would verify, in a sense described later, the variables ALPHA, BETA(1), BETA(2), BETA(3), and K(1,1,1).

MADBUG requests can be classified into four groups: the edit requests which are PRINT, DELETE, INSERT, CHANGE, APPEND, MANIPULATE, and TRANSLATE; the core requests which are GO, OPEN, VERIFY, LINKAGE, BREAK, KILL, SAVE, and RESTORE; the requests for returning to CTSS which are QUIT and EXECUTE; and the declarations which are WORK, USE, and FORCE. These requests will be discussed in the next few sections.

The Work Request:

The MADBUG requests are carried out in the context of a single MAD subprogram. The WORK request allows the user to declare which subprogram is of interest. For example: "work prog" sets up MADBUG to work on the program in file PROG MAD. The file PROG MAD does not have to exist. As illustrated in the sample session, if the user adds lines to a non-existent file, MADBUG will create the file. Thus, if the user is working in the context of a subprogram PROG, and wishes to print a subprogram ROOT, he must first request "work root" and then may request "print".

Edit Requests:

MADBUG uses a different technique for editing than the CTSS EDIT command. Neither the user nor MADBUG supplies a line number for a card image. Instead of indicating a card image by giving its associated line number, the user has three options: (1) the statement label on the card, if any; (2) the card's position relative to another card which has a statement label (the third card before ALPHA is ALPHA-3; and (3) the number of the card in the deck (the 17th card in the deck is simply 17). In counting for (2) or (3), the user must count all physical card images including remark and continuation cards. MADBUG interprets the arguments of a
request before executing the request; thus, if a deck consisted of three cards, "delete 1 2" would leave the third card, but "delete 1" followed on another line by "delete 2" would leave the second card.

In unusual situations there may be a long section of program with no statement labels. The user is free to insert remark cards with statement labels in such a case. MADBUG, but not the MAD translator, will allow references to statement labels on remark cards.

Three special conventions exist for specifying statement labels: (1) the "+" is always taken to mean the previous card referred to by the user, so that a "print *3" after a "print 6" would print the 9th card, and so that a "print alpha...*2" would print three cards starting with ALPH. (2) the "/" is always taken to mean the last card in the deck, so that, in a five card program, "print 1 3 5" is identical to "print 1 3 ". (3) Requests which operate on cards will operate on every card in the subprogram if no cards are specified, so that "print" is identical to "print 1.../".

MADBUG observes the standard conventions of horizontal spacing: the characters after a tab will be moved to column 12 and the characters after a tab-backspace will be moved to column 11.

The description of several of the editing requests will refer to input line blocks. An input line block consists of all the lines the user types before typing a blank line. The editing requests are defined as follows:

PRINT will print all cards mentioned as arguments. Thus, "print a(1)+1...b-3" would print a block of cards starting with the card after the card labeled A(1) and ending with the third card before the card labeled B.

DELETE will delete all cards mentioned as arguments. Thus, "delete" would delete all of the cards of the subprogram being worked, and "delete 1 3...6" would delete the first and the third through sixth cards.

INSERT will insert successive input line blocks before successive cards mentioned as arguments. Thus, one might see the following sequence:

U:print
M:ONE
M:
U:insert
U: zero
U:
U: print
M: ZERO
M: ONE
M:
U: insert 1 one
U: a
U:
U: b
U:
U: print
M: A
M: ZERO
M: B
M: ONE
M:

CHANGE will replace successive cards or blocks of cards, given as arguments, by successive input line blocks. A block containing any number of cards may be replaced by an input line block of any length.

APPEND with no arguments will append the input line block which follows the request line to the subprogram being worked. On the other hand, if the request has arguments, they are taken to refer to MAD subprograms which will be appended, in order, to the program being worked.

APPEND is also useful for creating a modified version of a subprogram while keeping the original. To do this, WORK the new name, APPEND the old name, and then make modifications.

MANIPULATE is a request for character manipulation within a card image. The first argument specifies the manipulation. Arguments after the first specify cards within which the manipulation will be performed. The first argument has the form: /***/***/ where the slash stands for any separation or delimiter character which must occur exactly three times, and the strings of asterisks stand for any pair of character strings. The manipulation consists of replacing all occurrences of the first string by the second string. Any character except a tab or space may be used as the delimiter; it is recognized by its being the first character of the argument. The two character strings may include any characters except the
delimiter and the carriage return, and they may be of different lengths. If the first string is empty, it will be taken to match a null string before column one on the card, thus allowing a simple way of inserting a statement label on a card. As a confirmation to the user, MADBUG will print a list of cards on which the manipulation is performed. If the manipulation is performed more than once on a card, the card will be included in the list once for each time the manipulation occurs. MADBUG does not consider replacing a string by itself to change the symbolic program. Thus the user can replace a string by itself to locate all occurrences of the string.

TRANSLATE has no arguments, and causes the subprogram being worked to be translated into machine language by the MAD compiler. From the user's point of view MADBUG is performing the translation. It is not necessary to translate any subprogram before using it. MADBUG will request any translations that are needed at load time. The TRANSLATE request is a convenience to the user who is changing several subprograms at one time, and who would like to catch any syntactic errors in one before turning his thoughts to another.

The Use Request:

The core requests, which will be discussed in the next section, operate in the context of a core image. MADBUG must have some way of knowing what subprograms to load when creating a core image. The arguments of the USE request are the subprograms to be used. Thus a user writing a subroutine ROOT and a test program MAIN might "use main root". There are provisions for using FAP programs, special libraries, and special loader parameters; these provisions are described later.

Core Image Requests:

Some core requests require cards for arguments, and their arguments observe the same conventions as those of the edit requests. A core request which refers to a declaration or remark card will operate on the first executable statement following the referenced card. Other core requests require variables for arguments. A variable is given as an argument in standard MAD notation, including multi-dimensional arrays and COMMON and ERASABLE variables, but not the dummy arguments of functions. Three special conventions exist for variables: (1) the "*" is always taken to mean the
previous variable referred to by the user; (2) if no variables are specified, the request will operate on every variable in the program; and (3) the block notation can be used to include several arrays or variables at once. Variables are taken to be ordered alphabetically (with a blank coming after R, alas.) and then by linear subscript.

The first time the user gives a core request, a core image must be created by MADBUG. This is accomplished by translating each of the needed subprograms into machine language, if necessary, loading the subprograms into core, and finally modifying some of the subprograms in order to intercept illegal references to an array. If an error is detected in this process, the core image will not be formed, and the core request will be terminated. The user should correct the error and try the core request again. The core image will be destroyed when the user issues the quit request or edits a program occurring in the core image. The core requests are defined as follows:

GO will start the user program. A single card given as an argument for GO will cause the user program to be started at the named card. If no argument is given, the user program will be started wherever it stopped last. A fresh core image will start at the beginning of the main program.

The user program will remain in control until (1) it terminates by calling DEAD, DORMNT, ENDJOB, ERROR, or EXIT; (EXIT can be implicitly called by letting control reach an END OF PROGRAM or END OF FUNCTION card.) (2) a "breakpoint" is encountered by the user program; (3) the user interrupts by pushing the break button once; or (4) an array is referenced with subscripts pointing outside of the dimensioned array. (Some array dimension violations are not caught; this is discussed in a later section.) On any of these occasions, control returns to MADBUG, and the user is informed of the reason.

Infrequently, the user program may have an error which causes control to return to CTSS. In this case, the user should type two CTSS commands, first "save (user)" to save his own core image and second "resume (mdbg)" to return control to the core image on which MADBUG saved itself. Even if the first of these commands results in an error comment from CTSS, the user should type the second. This procedure is called a manual restart.
OPEN will print the contents of variables mentioned as arguments, one by one, and after each, wait for the user to type a new value for the variable. If the user wishes the old value to remain, he just types a carriage return. In typing out the value of a variable, MADBUG makes use of the declared mode of the variable and of the current value to decide whether the value should be presented to the user in integer, alphabetic, floating-point, Boolean, statement label, or function mode. The user must type a constant for the new values in a form compatible with the declared mode of the variable. It is possible to change the input/output form associated with a declared mode permanently or to override the normal associations for a single request. This is discussed later.

One special note: because of the way the MAD compiler works, one may change the effect of a transfer statement by changing the value the variable which has the same name as the statement label to which the statement transfers. One may not, however, change the scope of a THROUGH loop in this fashion, even by changing the value of the variable with the same name as the THROUGH scope.

VERIFY will cause the values of variables mentioned as arguments to be compared with the values of the same variables in a fresh, unexecuted version of core. Each variable whose value has changed will be printed with its present value. Its value in the fresh version of core will also be printed if it is non-zero.

An option is available with verify; the user may specify any core image saved with the SAVE request to be used instead of the fresh copy of core discussed above. This is done by giving the name of the saved image following the request name and before the list of variables to be verified. As the user will discover below, this name must begin with an asterisk, and can thus be recognized by MADBUG.

The discussion of output forms used for the values of variables, which was given under the OPEN request, also holds for the VERIFY request.
LINKAGE causes MADBUG to tell the user which statement made the most recent call to the external function subprogram currently being worked.

BREAK will modify the machine language program in the current user core image so that control will return to MADBUG if one of the cards given as arguments is to be executed. When MADBUG regains control from the user program, the name of the statement which is about to be executed will be printed for the user. At this time the user will usually examine variables in his program to determine what his program is doing. "Breakpoints", as these points in the user core are called, belong to a given core image, and can vary from one saved core image to another. (See the SAVE request.)

KILL will remove any breakpoints at cards mentioned as arguments. It is not an error to insert a breakpoint where one already exists nor to remove one which does not exist. For example, to kill all the breakpoints in the subprogram being worked, "kill".

SAVE has a single name as its argument and causes a copy of the current user core image to be saved as a CTSS file with the primary name given as an argument and the secondary name SAVED. The name given by the user must begin with an asterisk. The current user core image was produced by loading, and has been modified by execution and by MADBUG requests. One may save the current core image under a name which has already been used for a save request. In this case, the current core image will replace the previous core image. All the core images saved using the SAVE request will be destroyed when the user's current core image is destroyed. This is because the saved files created by MADBUG are not normal CTSS saved files, and are useless out of the context of MADBUG.

RESTORE will replace the current user core image with a copy of the image whose name is given as an argument. The core image name must be a name under which the user has saved a core image using the SAVE request, or it must be *FRESH. *FRESH is a byproduct of the loading process. It is a completely unexecuted version of core with no breakpoints and with all variables at their initial values. Except for the special
way in which it is created, *FRESH is like any normal core image saved by the SAVE request.

Getting Back to CTSS:

When the user is finished with MADBUG, and desires to return to CTSS, he should use the QUIT request. The QUIT request will destroy all the files created during the session, except for the modified MAD programs and their associated BSS and SYMTAB files.

The EXECUTE request allows the user to return to CTSS for a single command, without ending his session with MADBUG. For example, the user could effect the CTSS command "listf aa mad" by requesting "execute listf aa mad". These commands are executed using the command chaining technique with the sequence: "save (mdbg)", the user's command, and "resume (mdbg)". No provision is made for saving a core image which might result from the user's command.
SPECIALIZED FEATURES AND TECHNIQUES

Two error comments that the user may get from MADBUG have special significance. One is "TRY AGAIN.," which always means that the current request has been terminated. The other is "CONSULT LISTINGS." which can only occur as a result of a bug in MADBUG. Any user getting this comment will please retain as much information in the way of output, files, etc. as he can and call Bob Fabry, x2524, so the bug can be removed promptly. The user can often continue with more requests in spite of a "CONSULT LISTINGS." error.

Two types of improper array references are not caught. First, references with a constant linear subscript are not checked. For example, one might DIMENSION A(10) and A(20)=100. Second, references to arrays which are given as arguments to functions are not checked. For example, one could have called for ROOT.(A(K)) where K is 20. This situation can sometimes be avoided by placing arrays in COMMON, and not passing them back and forth as arguments.

In unusual cases, the user core image may "blow-up" in such a way that the information about control and about the values of variables is gone or meaningless. In this case the user will still find MADBUG a useful tool, and may approach the problem by an exponential search through time for the point at which the blow-up occurs. Stated another way, this amounts to performing a series of tests in which each test is designed to cut by a half the uncertainty about when the blow-up occurs. When the user knows the exact point of the blow-up, he can then step through very cautiously, looking for clues. Such an approach relies heavily on BREAK, KILL, SAVE and RESTORE. At the start, the user moves a core image as close to the blow-up as he knows he can, SAVEs the core image, and guesses the half-way mark, in terms of opportunities for bugs, to a place by which the blow-up must have occurred. He then uses BREAK and KILL to step his current core image to the half-way point he guessed. (1) If the core image blows-up in this process, he guesses a new half-way point, half way between his saved image and his old half-way mark, RESTOREs his saved core image, and tries his new guess. (2) If the core image doesn't blow-up in the process, he SAVEs his current core image for a new starting point, guesses a new half-way mark between his new core image and the blow-up, and tries this new guess. This process is fairly simple to carry out using MADBUG, and most blow-ups can be readily solved this way.

When loading is performed, MADBUG will normally load a program named (MDBG), which MADBUG provides, immediately following the files specified by the USE request. Then MADBUG will process the core images of all programs loaded into core before (MDBG) and insert patches, using an area reserved in (MDBG), to attempt to catch any user subprogram
when it accesses an array with an illegal subscript. If the user wishes to load programs which were written in FAP, MAD programs for which the symbolic programs are not available, debugged MAD programs which he does not wish to protect, or library files, he may specify the position of (MDBG) by typing (MDBG) in place of a file name in the USE request. All the files before this parameter will be treated normally, and all things after it will be ignored by MADBUG and just passed on to the loader. Any loader parameters, such as (CFLP) or (LIBE), can also be used after (MDBG). If the user needs more than eighty characters for his USE request, he may type a hyphen as an argument of use. When the hyphen is encountered, MADBUG will immediately read the next input line for more arguments for the USE request. This may be done for several successive lines.

The FORCE request forces certain internal registers in MADBUG to new values, picked by the user. To FORCE a parameter, give the name of the parameter as the first argument of FORCE, and give remaining arguments as required by the parameter being forced:

FORCE PATCH will set the amount of patch space available in the user core images to the decimal number given as the argument. Initially PATCH is set to 500. The patch space is used during loading and whenever breakpoints are inserted. FORCE PATCH does not change the available patch space immediately, since the internal register is examined only during loading. A user would reduce the patch space if he was squeezed for core space. He would increase it if MADBUG complains, during loading, that there is not enough patch space, or if he exhausted the patch space inserting breakpoints. If the patch space is exhausted by breakpoints, however, it is usually sufficient to KILL some of the less necessary breakpoints to get space for new ones.

FORCE FORMAT will set the normal input/output form associated with each of the possible modes for variables. After the word FORMAT, the arguments are taken in pairs, the first item of the pair indicates a mode and the second indicates a form. The modes are indicated by a digit from 0 to 7, standing for floating-point, integer, Boolean, function, statement label, mode 5, mode 6, and mode 7, in that order. The form designation is one of the following: "Gn" for floating point with n significant figures on output, "I" for integer, "A" for alphabetic, "P" for either
integer or alphabetic with MADBUG picking for
output, "O" for octal, "B" for Boolean, "S" for
statement label, and "F" for function.
Initially, FORMAT is set to: 0 G3 1 P 2 B 3 F
4 S 5 Ø 6 Ø 7 Ø. (In this section, "Ø" is
used to denote the letter "0".)

FORCE MODE allows the user to predetermine whether
MADBUG saves itself as a permanent mode file
or as a temporary mode file. The values of
MODE are, correspondingly, "P" and "T". MODE
is originally set to "P". The user will want
to FORCE MODE to temporary if he is not
interested in extreme reliability as much as
in conserving his track allotment.

It is also possible to override all the normal I/O forms for
the duration of one OPEN or VERIFY request. To do this, use
one of the form designations listed above, but preceded by a
slash. Insert it after VERIFY (and the saved file name, if
present) or OPEN and before the arguments. For example,
"open /o alpha".

MADBUG observes the convention that the first statement of a
main program starts after the call to .SETUP which the
compiler always inserts as the first executable machine
instruction. Another convention at this level is imposed by
the compiler. A breakpoint on an ENTRY TO statement will
not be encountered when the entry is called, but will be
encountered if control is transferred to the statement or
fails to the statement.

MADBUG creates and destroys special files as it processes
the user's requests. They are destroyed during the
processing of the same request for which they are created.
Normally, the user will not have to worry about them, but
occasionally he may be made aware of their existance.
(MDBG) SAVED is the name under which MADBUG saves itself
when it chains to other commands. This file will vary in
length during a session, but will be on the order of 30
tracks long. Its mode depends on the value of MODE, as
described earlier. (TEMP) (MDBG) is used during file
modification. When a word in a file must be modified, the
modified file is first created as (TEMP) (MDBG), and then
the original file is deleted and (TEMP) (MDBG) is renamed.
The length of this file depends on the length of the file
being modified. The file has permanent mode. (MDBG) BSS is
created by MADBUG whenever loading is required. Its
position in the new core image was discussed earlier. It
contains the bootstrap for MADBUG and the patch area. It is
one track long and has temporary mode. (MDBG) SAVED is a
very short program which processes the input line blocks the
user types while editing. It processes all the input line
blocks associated with one edit request and reads in the
following request before chaining back to MADBUG. It is usually one track long and is permanent mode.

A user core image may use the command buffers. A call to CHNCOM will not return control to MADBUG. MADBUG saves the command buffers and counter initially and restores them when the user gives the QUIT request. MADBUG also treats the command buffers and counter as pseudo-machine conditions associated with each core image. The buffers are only lost on manual restart. A fresh core image has empty buffers.

By editing, the user modifies the MAD subprogram on which he is working. By inserting and removing breakpoints and by changing the values of variables, the user modifies the current user core image, (USER) SAVED. MADBUG does not change external files until the changes are logically needed. If the user uses EXECUTE to ask CTSS to process these files, he may want to insure that these logical modifications are made physically. To insure that the MAD subprogram being worked is modified physically, give a redundant WORK request using the name of the subprogram already being worked. Whenever a WORK request is given, the logical modifications associated with the subprogram previously being worked are made physically. To insure that the current user core image is modified physically, use a SAVE request. A user who cannot afford the added tracks can give an "execute delete" on the created SAVED file. This variation between the physical and logical modifications provides some degree of safety to the user who carelessly makes gross incorrect modifications to one of his programs. If the user should accidently type a "d" as a request line for example, he should quit by hitting the break button twice in succession. This will prevent MADBUG from actually deleting the file in question.
SUMMARY OF MADBUG REQUESTS

<table>
<thead>
<tr>
<th>Request</th>
<th>Arguments</th>
<th>Additional Lines (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>work</td>
<td>subprogram name</td>
<td>none</td>
</tr>
<tr>
<td>print</td>
<td>card names (1)</td>
<td>card images by MADBUG</td>
</tr>
<tr>
<td>delete</td>
<td>card names (1)</td>
<td>none</td>
</tr>
<tr>
<td>insert</td>
<td>card names (1)</td>
<td>card images by user</td>
</tr>
<tr>
<td>change</td>
<td>card names (1)</td>
<td>card images by user</td>
</tr>
<tr>
<td>append</td>
<td>none</td>
<td>card images by user</td>
</tr>
<tr>
<td></td>
<td>(or) subprogram names</td>
<td>none</td>
</tr>
<tr>
<td>manipulate</td>
<td>special, then cards</td>
<td>card names by MADBUG</td>
</tr>
<tr>
<td>translate</td>
<td>none</td>
<td>comments by MADBUG</td>
</tr>
<tr>
<td>use</td>
<td>subprogram names</td>
<td>none</td>
</tr>
<tr>
<td>go</td>
<td>card name or none</td>
<td>comments by MADBUG (4)</td>
</tr>
<tr>
<td>open</td>
<td>variables (1,2)</td>
<td>values by both (4)</td>
</tr>
<tr>
<td>verify</td>
<td>variables (1,2,5)</td>
<td>values by MADBUG (4)</td>
</tr>
<tr>
<td>linkage</td>
<td>none</td>
<td>linkage by MADBUG (4)</td>
</tr>
<tr>
<td>break</td>
<td>card names (1)</td>
<td>none (4)</td>
</tr>
<tr>
<td>kill</td>
<td>card names (1)</td>
<td>none (4)</td>
</tr>
<tr>
<td>save</td>
<td>save-name</td>
<td>none (4)</td>
</tr>
<tr>
<td>restore</td>
<td>save-name</td>
<td>none (4)</td>
</tr>
<tr>
<td>quit</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>execute</td>
<td>command and arguments</td>
<td>depends on command</td>
</tr>
<tr>
<td>force</td>
<td>parameter, special</td>
<td>none</td>
</tr>
</tbody>
</table>

Notes: (1) If none, all are implied.
(2) Optional form forcing first argument.
(3) Any request can get error comments from MADBUG.
(4) Comments by MADBUG if core image is created.
(5) There is an optional save-name argument.
Identification

Post Mortem Debugging
PM

Purpose

Produce post-mortem information about the user's last dormant program (loaded by the relocatable program loader).

Restrictions

The program should be loaded by LOAD or LOADGO so that the loader and movie table are available.

Usage

The PM command may be followed by one of several requests.

PM 'ILC' Gives the stop location or ILC (1 line).
PM 'LIGHTS' Gives machine conditions and ILC (4 lines).
PM 'STOP' Gives ILC and contents of two locations on either side of the stop (5 lines)
PM 'AUTO' Corresponds to LIGHTS plus STOP (9 lines.)
PM 'STOMAP' Gives origin and entry of all subprograms loaded.
PM NAME Gives contents of four initial locations of subprogram NAME (5 lines).
PM NAME LOC1 LOC2 -MODE- -DIRECTION-
Gives contents of all locations from relative location LOC1 through LOC2 of subprogram NAME in the specified mode and direction. NAME is '(MAIN)' for the main program. LOC is assumed to be decimal; if the number is preceded by a slash, '/', it is taken as octal. MODE specifies the form of printed output and may be 'FIX', 'FLØ', 'DEC', 'ØCT', 'BCD', or 'ALL'. DIRECTION specifies the order of printing and may be 'FWD' or 'REV'. If MODE is omitted 'ALL' is assumed; if DIRECTION is omitted, 'FWD' is assumed. LOC1 and LOC2 may be replaced by 'ENTIRE' to cause printing of the entire program.

PM LOC1 LOC2 -MODE- -DIRECTION-
Gives the contents of absolute locations LOC1 thru LOC2.
References to COMMON must be the high core locations which appear in the assembly listing, not the lower core area actually used for COMMON. (Caution: illegal requests, either outside the program range or improper requests for COMMON, cannot be interpreted correctly.)
Identification
Relocatable program patching
PATCH, STOPAT, TRA

Purpose
To allow break points to be set in a program after it has
been loaded, to allow transfer of control to a specified
location, and to allow modification of the loaded program.

Restrictions
These service routines are normally loaded after the program
is loaded and so the loader must be available in core.
Therefore LOAD or LOADGO should be used for loading the
program.

Usage
Set a break point:

STOPAT ENTRY RELLOC

ENTRY is an entry point in the desired subprogram.
If ENTRY is omitted, the main program is
assumed.

RELLOC is the relative octal location in the
specified subprogram at which the break point
stop is to occur.

STOPAT replaces the instruction at RELLOC with a
transfer. When the transfer is executed, the
original contents of RELLOC is restored and
the program is placed in dormant status. The
START command may then be used to continue
with the execution of the original contents of
RELLOC.

Transfer:

TRA ENTRY RELLOC

Same argument specifications as STOPAT. The
issuance of the START command will cause a
transfer to RELLOC. This may be used to
restart the program from different locations
during debugging sessions.
Modify the program:

**PATCH ARG**

ARG=entry: ARG may be the entry point of a subprogram which is to be patched by referring to relative locations within the subprogram. If ARG is omitted, (MAIN) is assumed.

ARG='(ABS)' allows patches to absolute locations.

ARG='(COM)' allows patches to relative locations within the COMMON region.

ARG='(PAT)' allows patches to be entered into locations above the user's current memory bound. This patch space is referenced by relative locations and is shared by all subprograms.

After a response from the PATCH command, the user enters lines of the form:

**LOC, TYPE, VALUE, RELOC**

LOC is the octal address to be patched. This octal number may be immediately followed by a special letter if it is desirable to override ARG for this response. The special letter may be A for absolute location, C for relative location in common, or P for a relative location in the patch space.

TYPE is the type of value to follow i.e.,
- 'OCT', octal word (used for instructions)
- 'FLO', fixed or floating-point number (E or F notation)
- 'INT', fortran integer
- 'DEC', MAD integer

VALUE is the number to be patched into LOC.

RELOC is the relocation specification for VALUE if TYPE is 'OCT'. It consists of two letters, the first for the decrement and the second for
- A: absolute
- R: relocatable
- C: common
- P: Patch space

If RELOC is omitted, AR is assumed. Successive VALUES and appropriate RELOCs may be specified in any line.

Exit from PATCH is the QUIT signal.
Identification

Absolute program patching
SPATCH

Purpose

Programs loaded with LDABS, NLOAD, or VLOAD may be patched using some supervisor routines which do not require special loading and movie tables. This is accomplished by patching their SAVED file, rather than the core B program directly.

Usage

SPATCH NAME1 LOC A1 B1 A3 B2 ... An Bn

SPATCH NAME2 ILC L

SPATCH patches the file NAME1 SAVED beginning at absolute octal location LOC for n locations. If LOC is 'ILC', only the ILC of NAME1 SAVED will be patched, causing a transfer of control to absolute location L when NAME1 is RESUMED.

Aibi are the octal left and right half words respectively.

L is the location at which control should be RESUMED.
Implementation

Adaptive Program Parricing

The program, however, with the module 'NCLAD', which is used to calculate
procedures involved, can now be executed. The program is designed to handle
inflation and wage changes. It is also adaptable to new rules and
criteria. A new feature is added to the program for

MATCH HaR 17 17 PAL 17 
MATCH NAME 1 IC J

MATCH parameters in the NAMF Y גד
beginning of

MATCH parameters for the NAMF Y גד
end of

MATCH parameters for the NAMF Yגד
end of

MATCH parameters for the NAMF Yגד
end of
Identification

Supervisor debugging
SD, SP

Purpose

To allow for printing and patching the supervisor (core A).

Usage

The printing routine has several options:

SD ENTRY RELOC N

N consecutive locations starting at relative octal location RELOC in subprogram ENTRY in the supervisor will be typed on the user's console in unrelocated (i.e., relative) octal with operation code mnemonics. If N is omitted, it is assumed to be 1. If ENTRY is omitted, the request is taken to be absolute. Lines of zero are not suppressed.

SD ENTRY 'TRACE'

The name of the calling subprogram and the relative location from which subprogram ENTRY was last called will be printed on the user's console. The user may continue tracing back by typing a carriage return. The trace may be terminated by the QUIT signal.

SD 'STOMAP'

A storage map of all subprograms loaded into the supervisor's core (core A) will be printed.

SD ENTRY

The contents of the specified entry will be printed on the user's console in appropriate form (RCD for LDNAME, all others in 5 octal digits).

Patching:

SP ENTRY RELOC A1 B1 C1 A2 B2 C2 ... AN BN CN

Patching will begin in relative octal location RELOC within the subprogram ENTRY. A1 B1 are the relocatable octal left and right half-words, respectively. The Ci contain two
characters indicating how the left and right half-words are to be relocated. The characters may be A for absolute or R for relocatable. If a Cl is omitted, it is assumed to be AR. If ENTRY and Cl are omitted, the patching is absolute.
Identification

STRACE – A trace debugging routine
B.L. Wolman, x6022

Purpose

STRACE (Subroutine TRACE) is a debugging program which allows the user to monitor the calls of selected subroutines. A set of conditions may be specified for each subroutine to be traced. At each call of the subroutine, STRACE checks to see if all the conditions are met. If they are, STRACE prints a message identifying the subprogram called, how many times it has been called, the absolute location of the call, the program in which the call occurred, and the relative location within the program making the call.

The user may request STRACE to STOP execution before executing a subroutine or to HALT after the subroutine has been called. If either of these options are used, STRACE will print an identifying message before going to dormant status. PM or OCTLK may be used to inspect the machine conditions. Issuing the START command will cause execution to continue.

The user may also specify a debugging subroutine which is to be called before executing a subroutine. This debugging subroutine may perform any function the user desires; the call issued by STRACE is of the form

DEBUG(LOC, ARG)

where DEBUG is the debugging subroutine name, LOC is the location of the call to the subroutine being traced, and ARG is a parameter previously specified by the user.

Options are also available which allow the user to obtain octal snapshot dumps of the machine registers, the subroutine calling sequence, and the value returned by the subroutine in the accumulator.

Usage

STRACE may be entered by issuing the CTSS command STRACE. Because of the method of implementation, the loader must be present in memory. The STRACE command may be issued immediately after loading, after a QUIT signal, or after a trace stop. (In general, STRACE may be entered any time the user's program is in dormant status). At the end of the input phase, STRACE will return to dormant in such a manner that the START command will cause execution to be resumed at the point where it was interrupted.
TRACE is an alternate entry which may be called as a subroutine. In this case, TRACE returns to 1,4 in the calling sequence. The calls are of the following form:

AED TRACE($) $,
MAD EXECUTE TRACE.
FORTRAN CALL TRACE
FAP TSX $TRACE,4

When STRACE is ready for input or more input, it prints the word TYPE. and waits. After receiving this response, the user may enter a series of commands. Each command consists of a subroutine name followed by one or more requests. Within a command, blanks are used to separate requests and their parameters. Since a carriage return is completely equivalent to a blank, commands may be split across one or more lines of input. Each command is terminated by a comma. The last command is terminated by an asterisk which signifies the end of the input phase.

The following requests are currently recognized by STRACE (N and M are positive decimal integers less than 32768, DEBUG is the name of a subroutine):

AFTER N - Begin tracing after the Nth call of the subroutine.

EVERY N - Trace every Nth call. N should be non-zero.

UNTIL N - Trace until the Nth call. The AFTER condition should be less than the UNTIL condition.

STOP N - Go to dormant before every Nth call. If N is zero, the STOP condition will be removed.

HALT N - Go to dormant after every Nth call. Execution will be interrupted after the specified subroutine has been executed and before it has returned to the program making the call. This request should not be used if the subroutine being traced has an error return or does not always return to the same point in the calling sequence. If N is zero, the HALT condition will be removed.

ARGS N - Every N times it is called, print the arguments of the subroutine. Each word in the calling sequence is assumed to specify a single variable. The absolute and relative addresses of these variables and their contents will be printed in octal. The
relative location will be "****" whenever the specified location is in COMMON or is in turn an argument of the subroutine making the call. Whenever N is zero, the ARGS condition will be removed.

VALUE N - Print the value of the specified subroutine. The value of the subroutine will be obtained by interrupting execution in the same manner as the HALT request; the same restriction applies. The VALUE condition will be removed whenever N is zero.

PM N W1 W2 ... Wn - Every N times the specified subroutine is called, print an octal snapshot dump of the machine registers specified by the parameters W1 to Wn. The W1's may be any of the following words.

AC  Accumulator, Q and P bits
MQ  Multiplier-quotient register
SI  Sense indicators
MB  Memory bound
X1  Index register 1
X2  Index register 2
X3  Index register 3
X4  Index register 4
X5  Index register 5
X6  Index register 6
X7  Index register 7
L1  First location in subroutine calling sequence
L2  Second location in calling sequence
L3  Third location in calling sequence
C1  First argument of subroutine
C2  Second argument of subroutine
C3  Third argument of subroutine
XS  Equivalent to the sequence X1 X2 X3 X4 X5 X6 X7
ALL  Equivalent to the sequence AC MQ SI MB XS

If any of the above words appears with an initial minus sign in the request, the PM of the corresponding register(s) will be removed. Because the PM request has a variable number of parameters, it must be the last request of any command. The PM print occurs after any call of a debugging subroutine and before any stop. The request PM 0 will suspend all PM requests for the particular subroutine.

CALL N DEBUG M - Before every Nth call, execute the debugging subroutine DEBUG with parameter M. If N is zero, the CALL condition will be removed; in this case the debugging subroutine name and the parameter M should not appear. If M is zero, the parameter used in the call of DEBUG will be the number of
times the subroutine being traced has been executed. If both the STOP condition and the CALL condition are simultaneously satisfied, the CALL of the debugging subroutine will occur before the STOP.

COUNT N - Reset the execution count of the subroutine to N. This request may be used to continue tracing after the UNTIL limit has been reached.

REMOVE - Remove the subroutine from the internal trace table. After this request has been given, STRACE will have no record of control over calls to the subroutine.

OFF - Turn off tracing of this subroutine. All succeeding calls will be ignored until tracing is restored via the ON request.

ON - Restore tracing of this subroutine.

FIND - Print the entry point of the subroutine. Any requests after the FIND will be ignored. FIND should only be used if no tracing is desired, since entry points are automatically printed the first time a subroutine name is encountered during the input phase.

If no request is given following the subroutine name, the standard requests

AFTER 0 EVERY 1 UNTIL 32767 STOP 0
CALL 0 HALT 0 ARGS 0 VALUE 0 PM 0

are assumed. Any requests given by the user override the corresponding standard value. Any of the tracing parameters of a subroutine may be changed by the user in a later entry to STRACE.

Method

When STRACE is asked to trace a subroutine, it saves the name of the subroutine in an internal table. STRACE searches the MOVIE) table for the named subroutine. If it is found, STRACE obtains the entry point. STRACE then uses the MOVIE) table to find the origins of all programs in core. When it finds a program that has a transfer vector, it searches this transfer vector for a TTR to the subroutine entry point. If a TTR is found, it is changed to a TXL TRAP, TABLE where TRAP is the address of the trace processing section of STRACE and TABLE is the index of the subroutine being traced in the internal trace table.
The REMOVE request causes essentially the inverse operation to be performed. All TXL TRAP, TABLE instructions are changed to TTR ENTRY and the subroutine is removed from the internal table.

During execution of the user's program a call to a traced subroutine will result in a TSX to the TXL instruction in the transfer vector. The TXL instruction will transfer to the appropriate section of STRACE. Using the contents of index register 4, STRACE obtains the TXL instruction and checks to see if it is legal (i.e., does the table position indicated by the decrement actually correspond to a subroutine name?). If the TXL is legal, STRACE retrieves the tracing conditions for this subroutine and checks them. Depending on the conditions and the number of executions of the subroutine, STRACE may print the trace message before transferring to the subroutine.

When the HALT or VALUE requests have been specified, STRACE examines the subroutine calling sequence to determine where the subroutine will return. It then saves the instruction at the return point and the instruction immediately following in the trace table and replaces them with a transfer back to STRACE. When STRACE obtains control following the execution of the subroutine it restores the two instructions. If the subroutine does not return correctly the breakpoint will not be removed and the two instructions which were saved will be destroyed the next time the HALT or VALUE condition(s) are satisfied.

The call of the debugging subroutine and the execution stop occur just before the transfer to the traced subroutine. In both cases the user's machine conditions (with the exception of index register 4) are restored.

Restrictions

Only 20 subroutines may be traced at one time. This limit is somewhat arbitrary and may be increased in the future.

STRACE will correctly handle any subroutine that is called by an instruction of the form TSX SUB, 4. A subroutine such as (10H) which is entered by the instruction TRA* (10H) cannot be traced. A subroutine should not be traced if there is any indirect reference to it through the transfer vector.

ERROR MESSAGES

The following error messages are currently implemented

TKACE TABLE FULL - No more subroutines can be traced until the REMOVE request is used.
NAME IS NOT IN TRACE TABLE - The user has attempted to use the ON, OFF, or REMOVE requests for subroutine NAME which is not in the internal trace table.

NAME IS NOT USED - Subroutine NAME has been loaded but is not called by any program. All requests for this subroutine are ignored.

NAME IS NOT IN MOVIE TABLE - Subroutine NAME has not been loaded. All requests pertaining to this subroutine will be ignored.

NAME IS NOT A REQUEST - STRACE does not recognize the request NAME. This word and the next word of input (most requests have a parameter) will be ignored. If the command line seems to be fouled up, the user can recover by typing a comma to terminate the command and then retype the entire command.

NAME PARAMETER MISSING, REQUEST IGNORED. - The user has typed a sequence such as AFTER, or UNTIL. The parameter for the request NAME is missing, since the command was terminated by the comma, the user must enter another command. Note that the command

```
SIN AFTER UNTIL 2,
```

will result in the comment 2 IS NOT A REQUEST.

BAD CALL OF TRACE FROM LOC - There has been a spurious transfer into STRACE or else location LOC (the word pointed to by the instruction at 0,4) contains a TXL instruction which has an illegal decrement. The decrement of a legal TXL instruction should be less than 201 (for the current limit of 20 entries) and a multiple of 10. The user's machine conditions will be restored, and STRACE will go to dormant.

NAME IS NOT A LEGAL PM - STRACE does not recognize the word NAME as a legal PM parameter, it will be ignored.

NO DEBUGGING SUBROUTINE, CALL IGNORED. - The user has forgotten to supply the name of the debugging subroutine. The CALL condition will be removed.

Disclaimer

This version has been checked and debugged, although no claims are made in this respect. Any comments, suggestions, and/or modifications will be welcomed. In case of trouble, contact Barry Wolman, Room 532 T.S., Ext. 6022.
Identification

Manuscript typing and editing
TYPSET, RUNOFF
J. Saltzer, X6039

Purpose

The command TYPSET is used to create and edit 12-bit BCD line-marked files. This command permits editing and revising by context, rather than by line number. The command RUNOFF will print out a 12-bit BCD line-marked file in manuscript format. RUNOFF contains several special features which were not available with the DITTO command, including type-justification.

References

This work represents one more iteration in the arduous task of creating an "ultimate" editing scheme. As such, it is primarily a synthesis of techniques which have been proven valuable in several separate problem areas. It is felt that this particular synthesis brings to bear on the editing problem an easy to use package of techniques, and might provide a model for an editor on a "next generation" time-sharing system. Here is a list of some of the sources of ideas for these commands:

- J. McCarthy
- S. Piner
- P. Samson
- Comp. Center staff
- M. L. Lowry
- M. P. Barnett
- V. H. Yngve
- R. S. Fabry
- A. L. Samuels
- F. J. Corbato

(Colossal typewriter)
(Expensive Typewriter)
(Justify)
(Input, edit, and file)
(Memo, Modify, and Ditto)
(Photon)
(Comit, Vedit)
(Madbug)
(Edits)
(Revise)
An Edit-by-Context Program

**Program Name:** TYPSET

**Description**

TYPSET is a command program used to type in and edit memorandum files of English text. TYPSET, along with the command RUNOFF, is an alternative to the commands MEMO, MODIFY, and DITTO. Editing is specified by context, rather than line number, and input is accomplished at high speed since the program does not respond between lines.

**Usage**

TYPSET name

"name" specifies the primary name of a file to be edited, or of a file to be created; it may be absent, in which case a file is to be created, and must be named later by the "FILE" request.

When TYPSET is ready for typing to begin, the word "Input" or "Edit" is typed, and the user may begin. If he is creating a file, he begins in high-speed input mode; if he is editing a file, he begins in edit mode.

**High-Speed Input Mode**

In high speed input mode, the user may type lines of up to 360 characters in length (e.g., 120 underlined characters) separated by carriage returns. He does not wait for response from the program or the supervisor between lines, but may type as rapidly as desired. The full character set of his keyboard may be used.

The user leaves high-speed input mode and enters edit mode by typing an extra carriage return. When switching modes, the program acknowledges the switch by typing the name of the new mode, "input" or "edit".

**Edit Mode**

In edit mode, the program recognizes "requests" of the form given below. All requests take effect immediately on a copy of the file being edited. Except where a request is expected to cause a response, such as "PRINT," successive requests may be entered immediately on successive lines without waiting for a response from the program. Each separate request must begin on a separate line. Responses from the program are generally typed in red.
Requests

Editing is done line by line. We may envision a pointer which at the beginning of editing is above the first line of the file. This pointer is moved down to different lines by some requests, while other requests specify some action to be done to the line next to the pointer. All requests except FILE may be abbreviated by giving only the first letter. Illegal or misspelled requests will be commented upon and ignored.

For purposes of description, the requests have been divided into two categories, those necessary for effective use of the command, and special-purpose requests which are not so generally useful. The first category includes eight requests:

LOCATE character string

This request moves the pointer down to the first line which contains the given character string. Only enough of the line need be specified to identify it uniquely. Since the pointer only moves down through the file, the second occurrence of a line containing a given character string may be located by giving the LOCATE request twice. The line which has been found is printed in its entirety.

It is not necessary to count blank characters exactly. If one blank character appears at some point in the request string, any number of blank characters or tabs at the corresponding point in the file will be deemed to satisfy the request. If 2 blank characters appear together in the request string, there must be at least two blank characters or tabs at the corresponding point in the file, etc.

If the LOCATE request fails to find a line containing the given character string, a message is printed, and the pointer is set to point after the last line in the file. Any requests which were typed in between the LOCATE which failed and the message from the program about the failure are ignored. Another LOCATE request will move the pointer back to the top of the file to begin another scan down through the file.

PRINT n

Starting at the pointer, n lines are printed on the typewriter console. The pointer is left at the last line printed. If n is absent, 1 line is printed and the pointer is not moved. If the pointer is not at a
line (e.g., above or below the file, or at a line just deleted) only a carriage return is typed.

**NEXT n**

This request moves the pointer down "n" lines. If "n" is absent, the pointer is moved to the next line.

**DELETE n**

This request deletes "n" lines, starting with the line currently being pointed at. The pointer is left at the last deleted line. If "n" is absent, the current line is deleted and the pointer not moved.

**INSERT new line**

The line "new line" will be inserted after the line by the pointer. The first blank following the request word is part of the request word, and not part of the new line. The pointer is set to the new line. To insert more than one line, give several INSERT requests, or just type a carriage return to switch to high-speed input mode. All lines typed are inserted after the line being pointed at. When the user returns to edit mode by typing an extra return, the pointer is set to the last inserted line. If the very first edit request given is an INSERT, the inserted lines are placed at the beginning of the file. If an INSERT is given after the pointer has run off the bottom of the file, the inserted lines are placed at the end of the file.

**CHANGE "string 1"string 2" n G**

In the line being pointed at, the string of characters "string 1" is replaced by the string of characters "string 2". Any character not appearing within either character string may be used in place of the double quote character. If a number, "n", is present, the change request will affect "n" lines, starting with the one being pointed at. All lines in which a change was made are printed. The last line scanned is printed whether a change was made or not. The pointer is left at the last line scanned. If the letter "G" is absent, only the first occurrence of "string 1" within a line will be changed. If "G" is present, all occurrences of "string 1" within a line will be changed. Blanks in CHANGE-request strings must be counted exactly.

**Example:**

line: It is a nice day in Boston.
request: CHANGE "is"was
new line: It was a nice day in Boston.
request: CHANGE xwasxisx
new line: It is a nice day in Boston.
request: CHANGE '' g
new line: It is a nice day in Boston.
request: CHANGE ''
new line: It is a nice day in Boston.
request: CHANGE "tis"t is"
request: CHANGE "" G
request: CHANGE 'on 'on.'
new line: It is a nice day in Boston.

FILE name

This request is used to terminate the editing process and to write the edited file on the disk. The edited file is filed as "name (MEMO)". If "name" is absent, the original name will be used, and the older file deleted. If no name was originally given, the request is ignored and a comment made. When this request is finished, the user returns to command level, and the supervisor will respond by typing "R" and the time used.

TOP

This request moves the pointer back to above the first line in a file.

The following seven requests are handy for special purposes, but will probably not be used as often as the ones previously described.

BOTTOM

This request moves the pointer to the end of the file and switches to input mode. All lines which are then typed are placed at the end of the file.

ERASE c

The character "c" becomes the erase character. Normally, the character "#" is the erase character. (The erase character is used to delete the previously typed character or characters.)

KILL c

The character "c" becomes the kill character. Normally, the character "z" is the kill character. (The kill character is used to delete the entire line currently being typed.)
VERIFY p

If the parameter, "p" is "OFF", the following program responses are not automatically typed:

"INPUT" or "EDIT" when the mode is changed.
Lines found by the FIND or LOCATE requests.
Lines changed by a CHANGE request.

If the parameter "p" is "ON", the responses are restored. The command begins in "ON" mode.

RETYPE new line

The line "new line" replaces the line being pointed at. The first blank following the request word is part of the request word and therefore is not part of the new line.

FIND character string

This request moves the pointer down to the first line which starts with the given character string.

SPLIT name

All the lines above the pointer are split into a file named "name (MEMO)". Any old copy of "name (MEMO)" is deleted. The remainder of the file may still be edited, and filed under another name. The SPLIT request may be used several times during a single edit, if desired. Unless at least one "TOP" request has been given, "name" must be different from the original name of the file being split.

Backspacing

The backspace key may be used to create overstruck or underlined characters. All overstruck characters are stored in a standard format, independent of the way they were typed in. CHANGE-, LOCATE- and FIND-request strings are also converted to this standard format, so it is not necessary to remember the order in which an overstruck character was typed in order to identify it. For example, suppose the line:

    Normal Mode statement of Mad

had been typed in by typing the letters NORMAL, five backspaces, a slash, and four forward spaces. The slashed $\emptyset$ can be changed to a normal $0$ by typing
Restricted Names and Recovery Procedures

Two special names are used for intermediate files by TYPSET. They are:

(INPUT (MEMO)
(INPT1 (MEMO)

Following a quit sequence (or a CTSS system breakdown) one of these files may be found. (Whenever a quit sequence has been given, a SAVE command should be issued to save the status of all files.) Since the intermediate file generally contains a complete copy of the file at the end of the last pass, it may be renamed and used as a source file, and may permit recovery of lost requests. The original file is never deleted until the new, edited file has been successfully written and closed.

The user’s disk status and file directory are updated at the end of each pass through the file, thereby providing additional insurance against accidental loss.

The intermediate files are normally written in permanent mode. If the user's track quota becomes exhausted while editing, TYPSET will switch to temporary mode intermediate files. If it is necessary to leave the edited file in temporary mode, a comment will be made.
Summary of TYPSET requests.

<table>
<thead>
<tr>
<th>abbrev.</th>
<th>request</th>
<th>response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic requests:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>LOCATE string</td>
<td>line found *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>end-of-file</td>
</tr>
<tr>
<td>D</td>
<td>DELETE n</td>
<td>end-of-file</td>
</tr>
<tr>
<td>N</td>
<td>NEXT n</td>
<td>end-of-file</td>
</tr>
<tr>
<td>I</td>
<td>INSERT line</td>
<td>none</td>
</tr>
<tr>
<td>P</td>
<td>PRINT n</td>
<td>printed lines, end-of-file</td>
</tr>
<tr>
<td></td>
<td>CHANGE QxxQyyQ n G</td>
<td>changed lines *</td>
</tr>
<tr>
<td>T</td>
<td>TOP</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>FILE name</td>
<td>Ready message</td>
</tr>
<tr>
<td>Special purpose requests:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>BOTTOM</td>
<td>&quot;Input&quot; *</td>
</tr>
<tr>
<td>V</td>
<td>VERIFY ON (or OFF)</td>
<td>none</td>
</tr>
<tr>
<td>S</td>
<td>SPLIT name</td>
<td>no name given</td>
</tr>
<tr>
<td>R</td>
<td>RETYPE new line</td>
<td>none</td>
</tr>
<tr>
<td>E</td>
<td>ERASE x</td>
<td>none</td>
</tr>
<tr>
<td>K</td>
<td>KILL x</td>
<td>none</td>
</tr>
<tr>
<td>F</td>
<td>FIND string</td>
<td>line found *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>end-of-file</td>
</tr>
</tbody>
</table>

* These responses will not occur if VERIFY mode is off.
A Right-Justifying Type Out Program

Program Name: RUNOFF

Program Description

RUNOFF is a command used to type out memorandum files of English text in manuscript format. Control words scattered in the text may be used to provide detailed control over the format. Input files may be prepared by the context editor, TYPSET.

Usage

RUNOFF NAME1 -'NOSTOP'- -N-

where "NAME1" is the primary name of a file "NAME1 (MEMO)" to be typed out. If a number, "N" is present, typing starts with the page numbered "N". The optional parameter "NOSTOP" is explained below, under "Non-Stop Typing."

Control Words

Input generally consists of English text, 360 or fewer characters to a line. Control words must begin a new line, and they begin with a period so that they may be distinguished from other text. RUNOFF does not print the control words.

.append A

Take as the next input line the first line of A (MEMO).

.line length n

Set the line length to "n". The line length is preset to 60.

.indent n

Set the number of spaces to be inserted at the beginning of each line to "n". Indent is preset to 0.

.undent n

In an indented region, this control word causes a break, and the next line only will be indented n spaces fewer than usual. This control word is useful for typing indented numbered paragraphs.

.paper length n

This control word is used for running off a memorandum file on non-standard paper. The number "n" is a line
count, figured at 6 lines per inch. If this control word is not given, "n" is assumed to be 66, for 11-inch paper.

.single space

Copy is to be single spaced. This mode takes effect after the next line. (The normal mode is single space.)

.double space

Copy is to be double spaced. This mode takes effect after the next line.

.begin page

Print out this page, start next line on a new page.

.adjust

Right adjust lines to the right margin by inserting blanks in the line. The next line is the first one affected. (This is the normal mode.)

.nojust

Do not right-adjust lines.

.fill

Lengthen short lines by moving words from the following line; trim long lines by moving words to the following line. (This is the normal mode.) A line beginning with one or more blanks is taken to be a new paragraph, and is not run into the previous line.

.nofill

Print all lines exactly as they appear without right adjustment or filling out.

.page -n-

Print page numbers. (The first page is not given a page number. It has instead a two-inch top margin.) If "n" is present, print out the present page immediately, and number the next page "n".
.space -n-

Insert "n" vertical spaces (carriage returns) in the copy. If "n" carries spacing to the bottom of a page, spacing is stopped. If "n" is absent or 0, one space is inserted.

$header xxxxxxxxxxxxxxxxxxxx

All of the line after the first blank is used as a header line, and appears at the top of each page along with the page number, if specified.

.break

The lines before and after the .break control word will not be run together by the "fill" mode of operation.

.center

The following line is to be centered between the left and right margins.

.literal

The following line is not a control word, despite the fact that it begins with a period.

All control words may be typed in either upper case or lower case. Illegal control words are ignored by the RUNOFF command. A comment may appear to the right of a control word, as long as it is on the same line.

Manuscript Conventions

The RUNOFF program assumes a page length of 11 inches, with 6 vertical lines per inch. The top and bottom margins are 1 inch, except for the first page which has a 2-inch top margin. If page numbers are used, they appear flush with the right margin, 1/2 inch from the top of the page. If a header is used, it will be on the same line as the page number. The first page is not numbered, or given the header line unless the control words ".header" and ".page 1" appear before the first line of text.

Customary margins are 1 1/2 inches on the left and 1 inch on the right, implying a 60-character line. This is the standard line length in the absence of margin control words.

The program stops between pages and before the first page for loading of paper. The paper should be loaded so that after the first carriage return typing would take place on line 1 of the paper. The left margin stop of the typewriter
should be placed at the point typing will begin, and the right margin moved as far right as possible. If you now type the first carriage return, the program will continue typing.

Tabs

Tabs included in the text are faithfully reproduced in the final copy. Since there will be interactions between inserted tabs and the right-adjust and fillout procedure, the control word ".nofill" should precede any use of tabs. If you wish to indent, say, a whole paragraph, with right-adjustment and filling, change "indent" and "line length" rather than using a tab at the beginning of each line. In order to type out a memo which uses tabs, the typewriter tab stops and left margin must be set up properly.

Backspacing

Underlining or overtyping may be accomplished with the aid of the backspace key, even in a line that is subject to right adjustment.

Non-stop Typing

If continuous form paper is used, RUNOFF can be instructed not to stop between pages by inserting the parameter "NOSTUP" after the file name (and before any initial page number) when the command is typed, e.g.

    RUNOFF ALPHA NOSTUP

Abbreviations

All control words may be abbreviated if desired. A list of abbreviations is given in the summary. In most cases, a single word is abbreviated by giving its first two letters; two words are abbreviated by giving the first letter of each word.
Summary of RUNOFF Control Words

<table>
<thead>
<tr>
<th>abbrev.</th>
<th>control word</th>
<th>automatic break</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ap</td>
<td>.append A</td>
<td>no</td>
</tr>
<tr>
<td>.ll</td>
<td>.line length n</td>
<td>no</td>
</tr>
<tr>
<td>.pl</td>
<td>.paper length n</td>
<td>no</td>
</tr>
<tr>
<td>.in</td>
<td>.indent n</td>
<td>no</td>
</tr>
<tr>
<td>.un</td>
<td>.undent n</td>
<td>yes</td>
</tr>
<tr>
<td>.ss</td>
<td>.single space</td>
<td>yes</td>
</tr>
<tr>
<td>.ds</td>
<td>.double space</td>
<td>yes</td>
</tr>
<tr>
<td>.bp</td>
<td>.begin page</td>
<td>yes</td>
</tr>
<tr>
<td>.ad</td>
<td>.adjust</td>
<td>yes</td>
</tr>
<tr>
<td>.fi</td>
<td>.fill</td>
<td>yes</td>
</tr>
<tr>
<td>.nf</td>
<td>.nofill</td>
<td>yes</td>
</tr>
<tr>
<td>.nj</td>
<td>.nojust</td>
<td>yes</td>
</tr>
<tr>
<td>.pa</td>
<td>.page (n)</td>
<td>yes, if n</td>
</tr>
<tr>
<td>.sp</td>
<td>.space (n)</td>
<td>yes</td>
</tr>
<tr>
<td>.he</td>
<td>.header xxxx</td>
<td>no</td>
</tr>
<tr>
<td>.br</td>
<td>.break</td>
<td>yes</td>
</tr>
<tr>
<td>.ce</td>
<td>.center</td>
<td>yes</td>
</tr>
<tr>
<td>.li</td>
<td>.literal</td>
<td>no</td>
</tr>
</tbody>
</table>

If "automatic break" is yes, the lines before and after the control word will never be run together, and the previous line will be printed out in its entirety before the control word takes effect.
Identification

Documentation of program changes
LOG

Purpose

The LOG command has been designed primarily to provide a convenient way of bookkeeping information among users working on the same programs.

Another related usage is to supply any user with a permanent way of obtaining information about the latest modifications in CTSS.

Format

LOG -CF- -NAME1- -NAME2-

CF is a common file number, (0,1,2,3, or 4)

NAME1 and NAME2 are the names of a log file to which all new information is added, and kept in the specified common file CF. If NAME2 is missing it is assumed BCD. If NAME1 and NAME2 are missing, they are assumed SYSLOG BCD. If CF is also missing it is assumed 2.

Execution

The LOG command begins by printing the contents of the file NAME1 NAME2, if it exists, so that the user is reminded of the latest modifications which he or other users might have made in the same programs. An interrupt will terminate printing. LOG then chains to the INPUT command so that the user may type whatever information he wishes to add to the log file.

This information, along with a problem and user number, system code, date, and time, is recorded at the beginning of the previous log file. Due to this procedure, the latest information is always available at the beginning of the file.

CTSS Modification Bookkeeping

Modifications to the CTSS system are usually recorded in the file SYSLOG BCD kept in the system files. Any user may copy this file by using the following command

COPY S SYSLOG BCD

The file may then be printed.
LOG Structure

The LOG command uses other system commands, viz. COPY, UPDATE, EDIT, FILE, COMBIN, PRINTF. There is no restriction in using LOG in a chain of commands.

Some imperfections in the commands called on by LOG may result in meaningless messages, or in an abnormal exit breaking the chaining set up by LOG. Such flaws will be eliminated gradually by improvements in each particular system command.

Suggestion

Every new item recorded in the log file starts with *b*b*b* heading the line containing the user's identification and date. It is suggested that no line typed in by users begin with this particular pattern. This would provide an easy way to process log files automatically for information retrieval, sorting by user, etc.
Identification

Users talk to GOD
REMARK

Purpose

Users may address themselves to "whom it may concern". The users' remarks file is printed off-line each day and the operations staff directs the printed copy to the appropriate members of the systems programming staff.

Usage

REMARK NAME1 NAME2

The 6-bit BCD file NAME1 NAME2 which contains the user's remarks is appended to a PUBLIC file called USER REMARK. This file is printed each day by the operations staff and delivered to the addressees. If NAME2 is omitted, it is assumed to be BCD. If NAME1 and NAME2 are omitted, instructions for using the command are printed.
Identification

INFO - An on-line documentation system
J. Winett - X81-301

Purpose

In order to better provide information about the programs associated with the Compatible Time-Sharing System an on-line system for documenting computer programs has been developed. The design of this on-line system attempts to satisfy the following objectives:

a) Have up-to-date information available to the user on request, thus eliminating the delays which occur in any memo distribution system.
b) Have the ability to get specific information on request, e.g. the author of a routine, as well as the complete documentation of a routine.
c) Give textual output in steps, i.e., printed according to information item types, and indicating the amount of printout that will result.
d) Provide the facility to search through the library of programs to determine the programs which satisfy particular conditions.
e) Standardize the format of the program description by requiring that when a new program is added to the system all information of interest be provided.
f) Permit editorial control of the program documentation that is to be available on-line.

The information describing a program is divided into information items. Each item of information is associated with an item name and is referred to as the item value of the associated item name. For example, the item value "WINETT" is associated with the item name "AUTHOR". The following items of information indicate what is required as documentation of a computer program:

1. Program NAME (N) - A single word.
2. Program TYPE (T) - One of the following: COMMAND, ENTRY, LIBRARY, or PUBLIC.
3. DESCRIPTORS (D) - Key words used to classify the programs in the information files.
4. PURPOSE (P) - A short abstract or sentence description indicating the context in which a program might be used.
5. USAGE (U) - The instructions of how to use the program, e.g., the calling sequence.

6. Programming LANGUAGE (L) - The language in which the program is written.

7. REFERENCE (R) - A bibliography of the program.

8. AUTHOR (A) - The name of the person who is responsible for the program.

9. DATE (DA) - The date the information was last entered or modified.

Additional information items may also be defined, e.g., program size, transfer vector, etc., but the above items are considered to be required to document any program.

Usage

A model of this information system has been implemented as a CTSS command program with command name INFO. The system may be initiated as a console command or chained to from another program. If, when the INFO system is called the NAME of a program is given as a command parameter, the documentation on that program will be printed after which the system will call CHNCOM. This procedure allows other command programs to access their own documentation. For example, when no parameters are supplied with a command which requires at least one parameter, the command should chain to the INFO command with the command name as a parameter. This technique provides a means of tying the documentation of a command program to the command itself.

If only the command name INFO is typed the system will respond

TYPE REQUEST, OR C.R. FOR INSTRUCTIONS...

whereupon a carriage return will initiate the request to describe the INFO command.

Alternatively, requests can be typed to the INFO system. There are three classes of requests: a) Retrieval requests to obtain information from the system, b) Storage requests for adding, changing, or deleting information from the system, and c) System requests which affect the operation of the system. The Retrieval requests - DESCRIBE (D), LIST (L), and FIND (F) are to be used by all CTSS users. The Storage requests - STORE (S), EDIT (E), ALTER (A), and REMOVE (R) - are to be used by the people responsible for the information stored within the system. The System request - QUIT - is used to terminate communications with the system, and the requests - END, TSSFIL, and USRFIL - are
used for changing the operation of the system. Whenever the
INFO system prints a comment followed by two periods, it is
the user's turn to type. After processing each request, the
system types

OK.

To obtain a description of a Storage or Retrieval request
the user types the request name only. A request to the INFO
system indicates three types of information: 1) an
imperative request to the system, 2) a list of single
information words, and 3) information items specified by
item names together with the item values associated with the
item names. A request to the system is assumed to be
indicated by one of the first few words typed. Other words
following the request name may be item names and are added
to a list of "information words" or may specify the values
of information items and are added to a list of pairs
consisting of an item name and its value. When the word
"IS" or "ARE" is encountered in a request it is assumed that
the previous word is an item name and that the following
words form the item value. The input specifying the item
value must be terminated by a comma (or the carriage return
at the end of the request) and may be followed by other item
names and their values or by item names alone. If the word
"THEN" appears as an information word the input scanned so
far is assumed to constitute a request. After the request
is processed, the input following the word "THEN" is scanned
for the specification of another request. Thus the word
"THEN" indicates the termination of a request and allows
multiple requests to be typed. Words other than item names
or item values or the word "THEN" may be typed but are
ignored.

Requests and item names may be abbreviated by their first
letter (except the item name DATE which is abbreviated DA).
If an item value is specified more than once in an input
request the value last specified takes precedence. Thus,
the on-line user may change or correct the specification of
an item value by retyping the item name together with the
item value in the same input request.

To continue the typing of a request on another line precede
the carriage return (C.R.) by a dash (-). When in doubt of
what to do, type a carriage return.
RETRIEVAL REQUESTS

1. The DESCRIBE (D) request:

   DESCRIBE NAME IS name, i(1)...i(n)

This request is used to obtain the documentation of a
program whose name is known. The input with this request
gives the program name and the names of the desired items of
information. If no item names are specified, the
information on all items will be printed. For example,

   DESCRIBE THE COMMAND WHOSE NAME IS INFO

produces all the documentation associated with the INFO
command, and

   D N IS INFO, USAGE

prints just the item USAGE for the INFO program.

When more than five lines of text are to be printed, the
INFO system informs the user of the number of lines which
follow. After realizing how much information will be
printed, the on-line user may terminate the request by
pressing the CTSS interrupt or quit button.

If the interrupt button is pressed the user may type
"CONTINUE (C)" to resume printing or "RESTART (R)" to type
another request. Printing will be resumed approximately ten
lines after the line at which printing was interrupted.
(This is due to the fact that the CTSS output buffers are
cleared on interrupt.) Since a number of lines are lost on
interrupt, the process of interrupting and continuing
provides a means of skipping lines of documentation.
Unfortunately, this procedure gets very poor response from
CTSS.

If the quit button is pressed, the on-line user may type
another command or type the CTSS command "START" to continue
as described above. This procedure gets very much better
response from CTSS.

2. The LIST (L) request:

   LIST i(1)...i(n), TYPE IS type

This request is used to obtain a list of all available item
names, a list of the available values of certain information
items, or a list of the names of all CTSS programs of a
particular type. The request may ask for the values of one
or more of the following items to be listed:
or may also request a list of all CTSS programs of a particular type by typing either or both of the types

```
COMMAND or ENTRY
```

after the words: 'TYPE IS' . The list of programs of a particular type is obtained directly from CTSS and thus is automatically provided the most up-to-date list of programs available. A request to

```
LIST NAMES
```

causes a list of commands and entries to be printed. A list of descriptors may be obtained by typing

```
LIST THE DESCRIPTORS
```

or just

```
L D
```

3. The FIND (F) request:

```
FIND i(1) IS v(1), . . . . , i(n) IS v(n)
```

This request is used to perform a search for the program or programs which have particular information item values. The items to be matched are given by typing the item names together with their item values. Acceptable items for searching on are:

```
TYPE, DESCRIPTORS, AUTHOR, DATE, and LANGUAGE.
```

A date value must be given in the form "DATE IS mm/dd," where mm is a numerical month and dd is a numerical day. All programs whose date is greater than that given will be printed. Note that a year is not specified and hence January is less than December (This bug will soon be eliminated). Descriptors are single words typed in any order and separated by spaces or the word "AND".

For example, to find the commands which were documented since January 1 and have at least the descriptors UTILITY AND EDITING, type -

```
FIND TYPE IS COMMAND, DATE IS 1/01, DESCRIPTORS - ARE UTILITY AND EDITING
```

or

```
F T IS C, DA IS 1/01, D ARE UTILITY EDITING.
```

(note the use of the dash to continue the input request on the next line.)
When a search results in more than twenty matching items the system asks whether the user wants to continue the search. The user may then type "YES" or "NO". For each twenty more matching items the user is given the option of continuing.
STORAGE REQUESTS

4. The **STORE** (S) request:

```
STORE NAME IS name, FILE IS file, i(1) IS v(1), -
    ..., i(n) IS v(n)
```

This request enables one to enter information about a new program into an information file. This request requires that information values be provided for each required item, as previously listed, in the form:

```
item name IS/ARE item value
```

The **NAME** of new information items may be defined by typing the new item name and its value. When the **INFO** system prints an item name followed by two periods, the user is to type the value of that item. Item names and item values of other items may be supplied following the item value which was requested, by typing a comma after each item value and thus anticipating the required input and reducing on-line interaction.

If the word **FILE** is specified in the input specification, a file with primary name the same as the program name (if specified) and secondary name **INFO** is read. This file is assumed to contain item values for this program where each item value is preceded by a line giving the item name prefixed by a period and beginning in column one. If the primary name of this input file is not the same as the program name, the file name may be specified by typing the item

```
FILE IS file name
```

If a file name is specified and a program **NAME** is not specified, the **NAME** of the program may be read from the input file. A program **NAME** is indicated in an input file by the presence of two periods before the program **NAME**. An input file may specify the documentation of many programs by preceding the documentation of each program with a line giving the program **NAME** prefixed by the **two** periods (for example, **..INFO**). The priming of the command documentation was done from an input file (with name **COMMAND INFO**) of this type by typing

```
STORE FILE IS COMMAND
```
5. The EDIT (E) request:

EDIT NAME IS name

This request re-creates a BCD file (as a line marked file) from the information in the system for use in making changes to information items using some CTSS editing procedure. The EDIT request requires that the program NAME be specified. The file created contains all information items except those items which can be used with the find request. Each item value is preceded by a line giving the item name prefixed by a period (e.g. .PURPOSE) and consequently no line of an item value should begin with a period. The primary name of the file created is the same as the program name and the secondary name is INFO.

6. The ALTER (R) request:

ALTER NAME IS name, i(1) IS v(1), . . . , i(n) IS v(n)

This request allows one to change item values in the information documenting a program or to store additional information items. The ALTER request requires that the program name be specified and is used like the STORE request. The ALTER request is different from the STORE request in that it does not require that values for all information items be specified. That is, the user-system interaction is different.

7. The REMOVE (R) request:

REMOVE NAME IS name, D IS d, A IS a, ITEM IS i

This request is used to delete an AUTHOR, DESCRIPTOR, or ITEM name from the appropriate list, or to delete the documentation of a program from an information file when a program is deleted from CTSS. To REMOVE the documentation of a program give the program NAME. To REMOVE an AUTHOR from the list of AUTHORS or a DESCRIPTOR from the list of DESCRIPTORS, specify the item value to be removed. Verification of each request to remove the documentation of a program is required.
SYSTEM REQUESTS

8. The QUIT (QU) request:

This request causes the INFO system to call CHNCOM and may be used to terminate the INFO command or to chain to other commands.

9. The END (no abbreviation) request:

This request causes the INFO system to terminate through the standard COMIT termination sequence. (The INFO command has been written in the COMIT language.) The amount of unused free storage, i.e., the number of WORKSPACE registers, is printed.

10. The TSSFIL (no abbreviation) request:

This request causes the INFO files to be obtained from one of the CTSS system file directories and is required before the INFO system is included as a CTSS command.

11. The USRFIL (no abbreviation) request:

This request causes the INFO files to be obtained from the user's file directory rather than the system file directory. This request may be used by a user to indicate that the documentation files are to be obtained from the user's file directory. In this way a user may keep documentation on his private programs.
Summary of INFO requests

RETRIEVAL REQUESTS

1. DESCRIBE NAME IS name, i(1)...i(n)
2. LIST i(1)...i(n), TYPE IS type
3. FIND i(1) IS v(1), ... , i(n) IS v(n)

STORAGE REQUESTS

4. STORE NAME IS name, FILE IS file, i(1) IS v(1), ...
   ... , i(n) IS v(n)
5. EDIT NAME IS name
6. ALTER NAME IS name, i(1) IS v(1), ... , i(n) IS v(n)
7. REMOVE NAME IS name, D IS d, A IS a, ITEM IS i

SYSTEM REQUESTS

8. END
9. QUIT
10. TSSFIL
11. USRFIL

Required information items are:

NAME (N), TYPE (T), DESCRIPTORS (D), PURPOSE (R),
USAGE (U), LANGUAGE (L), REFERENCE (R),
AUTHOR (A), and DATE (DA).

Note: i stands for item name and v stands for item value.
Identification

Macro Command
RUNCOM, CHAIN

Purpose

Public and private commands may be linked or chained together in order that the chain may be executed by merely issuing one command. This is convenient if the same series of commands is to be executed more than once and the user does not wish to retype the series each time. Arguments to the commands may be specified at execution time.

Reference

Section AG.8 gives further information about macro command programs.

Usage

Command Chain:

The command chain, or macro-command, must first be prepared as a BCD line-marked or line-numbered file, with one command per line. Blank lines are ignored. Command arguments are separated by one or more spaces; if an argument is more than six characters long, it will be truncated from the left. Arguments may be command names, actual argument values or dummy symbols. If dummy symbols are used, there must be a list of the dummy symbols specified by the pseudo-command CHAIN somewhere before the first executable command.

Example of a macro-command:

CHAIN ALPH BET TRANSL
ED ALPH TRANSL
PRINTF ALPH TRANSL
TRANSL ALPH
LOAD ALPH BET (LIBE) OWNLIB
etc.

Comments may be included in the command chain as lines which have as the first character an '*' or a '$'. Comments introduced by '*' will be ignored during execution. Comments introduced by '$' will be printed on the user's console at the point of execution corresponding to their position in the chain.
Execution of Command Chain:

RUNCOM NAME1 ARG1 ARG2 ARGn

NAME1 is the primary name of the BCD command chain file NAME1 BCD.

ARG1 are the arguments to be substituted for the dummy symbols (if any) in the same order as specified in the pseudo-command CHAIN. If any ARG1 is '(NIL)', the corresponding dummy argument will be ignored; if it is substituted for a command name, the whole command is ignored. If any ARG1 is '(END)', it will be replaced by a fence (all 7's). Any additional arguments will be ignored by commands in which this substitution is performed. If (END) is substituted as a command name, the chain is terminated at this point. If there are fewer ARG1 than dummy symbols in the CHAIN specification, the rightmost dummies will retain their literal values.

RUNCOM will interpret the file NAME1 BCD, substitute the explicit arguments for dummy arguments, if any, and perform the execution of the specified commands by appropriate use of the supervisor command chain buffers and subroutines. RUNCOM contains a list of public commands indicating whether or not each command assumes a current core image; RUNCOM can then properly intersperse the SAVE and RESUME commands. Nesting and recursion are possible.

Core image management:

Some more details may be necessary to understand the mechanism whereby RUNCOM takes care of core images between commands.

As a general rule, a core image is kept over two consecutive commands if, and only if, the first one is supposed to leave a core image, and the second one is supposed to expect a core image.

e. g. LOAD - SAVE - FAPDBG
Use the same core image created by the LOAD command.

Whereas LOAD - SAVE - LISTF does not keep the core image from SAVE to LISTF. Commands which are supposed to leave a core image are:
CTEST1 to CTEST9
LOAD VLOAD NCLOAD LOADGO LOADBS USE START
PM TRA STOPAT PATCH
FAPDBG STRACE L
SAVE RESUME R RESTOR
MYSAVE RECALL CONTIN RSTART
RUNCOM

Commands which are supposed to expect a core image are:

PM TRA STOPAT PATCH
USE START
SAVE MYSAVE
FAPDBG STRACE

(NIL) arguments as command names, and $ headed lines do not alter the saving of a core image.

As one may notice, RUNCOM itself may yield a core image, if the last command in the chain does. e.g.,

LOAD ALPHA BETA
SAVE ZETA
LISTF ZETA SAVED
RESTOR ZETA

may be used as a macro-command, and followed by a START command.

Common file switching:

The only commands which are allowed to begin and terminate in different file directories are:

COMFIL COPY UPDATE REMARK LOG

Indeed, COMFIL switches to whatever directory is specified, and the others switch to the user's file directory when completed.

Any other command must be initiated and terminated in the same file directory. On the other hand, there is no restriction on the various switching which may be performed during the execution of the commands, as long as the initial setting is restored before the end.

One should notice that the present implementation of the common files results in losing the temporary files created in the common files, as soon as one switches to some other directory. However, temporary files always stay in the user's file directory, regardless of the common files switching.
RUNCOM may be initiated in any common file, but the RUNCOM command will switch back to its initial file directory whenever it needs to load a new set of commands for execution. Consequently this may result in losing temporary files, if the execution was in a different common file.

It should be noted that a $ headed line produces a major break in the RUNCOM command. The following commands in the chain will then be loaded together in the supervisor's buffers, up to a maximum of 3 at a time. This peculiarity may be used to deal with temporary files in the common file directories.

COMFIL 3
LISTF ALPHA BETA
PRINTF ALPHA BETA
$ EDIT
EDIT ALPHA BETA
FILE ALPHA BETA
BETA ALPHA
....

Will insure that the (EDIT FILE) is not lost between EDIT and FILE, because the previous line headed by a $ has automatically forced the beginning of a new sub-chain in the supervisor's buffers.

Some examples of macro-commands:

We shall assume here that the name of the BCD file containing the chain is MACRO BCD.

1. CHAIN FILE (NIL) (END)
   ED FILE MAD
   MAD FILE (NIL)
   (END) FILE ... (LIBE) ...

   may be called in the following ways:
   RUNCOM MACRO FILE
   Whence: ED FILE MAD
           MAD FILE
   
   RUNCOM MACRO FILE (LIST)
   Whence: ED FILE MAD
           MAD FILE (LIST)
   
   RUNCOM MACRO FILE (SYMB) VLOAD
   Whence: ED FILE MAD
           MAD FILE (SYMTB)
           VLOAD FILE ... (LIBE) ...

2. CHAIN FILE BCD FILE1 N1 N2
* This macro inserts the file FILE BCD
* into the file FIL1 BCD, after line number N1
* and deletes the initial part of FIL1 BCD
* until after N2.

SPLIT FIL1 BCD (A) N1 * N2 (B)
CHMODE (A) BCD T (B) BCD T
COMBIN * FIL1 BCD (A) FILE (B)

May be called by:

RUNCOM MACRO ALPA FAP BETA 1030 1040
inserts ALPA FAP after line 1030, and deletes until after 1040

RUNCOM MACRO ALFA FAP BETA 1030 1030
same thing, but does not delete anything from BETA FAP

RUNCOM MACRO * FAP BETA 1030 1050
deletes in BETA FAP lines after 1030 until after 1050

3. This chain allows stacking commands typed on the
*console, and then starts the execution
SPLIT MACRO BCD MACRO N
* N is the number of the line containing 'EXECUTION'
EDIT MACRO BCD
FILE MACRO BCD
$ EXECUTION
Identification

Supply arguments in octal to any command
GENCOM

Purpose

If for some reason, the desired arguments for any command cannot be expressed in BCD, the command may be used with the arguments expressed as pairs of six-digit octal arguments.

Usage

GENCOM COMAND OCT11 OCT21 OCT12 OCT22...OCT1n OCT2n

COMAND is the BCD name of the desired command.

OCT1i OCT2i are pairs of six-digit octal equivalents of the desired arguments for COMAND. GENCOM checks to be sure they are octal.

GENCOM will combine the pairs of six-digit octal arguments, OCT1i OCT2i, into single twelve digit octal arguments, ARGi, and by the use of XECOM will initiate the command.

COMAND ARG1 ARG2...ARGn