OCS INTERACTIVE SERVICES

November 1968

DIRECTORATE OF SCIENCE AND TECHNOLOGY

Office of Computer Services

WARNING

This document contains information affecting the national defense of the United States, within the meaning of Title 18, sections 793 and 794, of the US Code, as amended. Its transmission or revelation of its contents to or receipt by an unauthorized person is prohibited by law.

Approved For Release 2000/05/08 : CIA-RDP78-03948A000100080002-8

OCS INTERACTIVE SERVICES

November 1968

DIRECTORATE OF SCIENCE AND TECHNOLOGY

OFFICE OF COMPUTER SERVICES

TABLE OF CONTENTS

I. Scope

This report gives an overview of the hardware and software interactive time-sharing system which OCS plans to implement in February 1969. User services are described in more detail. The report intends to provide information on the interactive system and subsystems which are being planned for Agency analysts during the 1969 period.

In January 1969, OCS is replacing the IBM 360/50 hardware of its present time-sharing system with an IBM 360/67. OCS is planning a system in which all of the present Model 50 software programs will run on the Model 67 in the same way they have been running in the past. Any changes are intended to be transparent to the customer or user. The official IBM Time-Sharing System (TSS) is not the system which OCS plans to implement. OCS is implementing a CP (Control Program) system, developed jointly by Lincoln Laboratories of MIT (Massachusetts Institute of Technology) and IBM at the Cambridge Scientific Center. This system is not an official IBM system and has not received wide publicity. OCS is altering the CP software and is adding several subsystems for Agency users. The total, the CP system and its subsystems, is described in this report. It is suggested that the reader select those topics from the outline which are pertinent to his interests rather than read the total report.

II. General

A. Philosophy of Interactive Processing

Computer processing systems can generally be grouped into one or two categories; either batch processing systems or interactive processing systems. In the former case the system is designed to utilize the processing power of the computer most efficiently; whereas the latter system is designed to give the most effective use to the remote terminal user. Thus in one, optimization techniques are oriented towards hardware performance and in the other they are oriented towards the performance of the remote terminal user. Most systems are not so clearly defined and comprise blends of both types of systems. As an example, the most efficient use of a batch processing system would create maximum throughput, but poor turnaround time (time between job submittal and time of product return to customer). Typically, arrival time of jobs at the computer are quite aperiodic and peaks and troughs of load occur. To avoid the troughs of computer idleness, the input queue would necessarily be overloaded and consequently most output results would be delayed. In practice, such a method creates unhappy customers. Usually, as a compromise, customers are given better service at the expense of less than optimum throughput by various techniques, such as "light loading", scheduling priority shuffling, etc. In the interactive system, the user optimum performance objective is some times compromised by other techniques, one example being the "shaving" of a few milliseconds from each response to each of the multiple users.

In all processing systems some input-output processing must be performed. This may vary from very minimal in a large scientific application to a very large proportion of total time in an operation (such as that of payroll) where a large data file is passed through the machine. During input-output processing there is mechanical motion which consumes large increments of time relative to that time which is consumed by processing in the central processor. Modern computers are doing central processing and input-output processing concurrently. Thus there

can be an effective overlap of processing within a time increment if the software provided is so designed. If only one program is running in a machine and this program is largely input-output and is using very little central processor power, in effect minimum and inefficient use is being made of the machine.

In a general purpose processing environment, such input-output jobs occur at random and by their very nature allow some overlapping of processing. It is beneficial to have more than one such program operating in the computer at one time. Theoretically, the more programs that can be run in the computer at a given time, the more probability there is that at least one of them will have completed enough input-output so that it can use the central processor efficiently. However, obviously there is a trade off here in that core memory size is a limitation; the number of programs that can be run in the computer at a given time is constrained by the size of the programs and the size of memory. Core memory is expensive and yet it takes much core memory to run many programs concurrently. Decreasing the size of core memory may mean a throughput loss per dollar in the sense that not enough programs can be run in core to utilize the total system in the most efficient manner. In summary, in the batch mode the objective is to run as many programs as possible concurrently, so that the probability of using the central processor time effectively is optimal.

In the design of an interactive system an attempt is made to give quick response time to every user who is normally located at a remote terminal. The user at a remote terminal generally enters data by keyboarding and receives data by relatively slow input-output operations such as by character by character printing, which operates at about 12 characters a second, or by a simple cathode ray type display unit which also receives data at a relatively low data rate. When the user is at a terminal, the time between the finger strokes as he punches keys is long in relation to the cycle time of a computer. A computer can receive the impulse from a user's key, cycle around and do a little bit of processing for each of one-hundred users, then return back and wait for the remote operator to push the next key.

In practice most times the computer will make several iterations through its whole list of users giving each a slice of time before the remote terminal operator makes his next stroke. Since the operator observes almost immediate computer response to his manually entered commands, the operator is given the impression of being the sole user of a large computer system. The remote user cannot use more power than the system has available. Thus the remote terminal is being effectively utilized and the human performance is optimized.

In the design of an interactive system, prime consideration must be given to the bookkeeping operations which are involved in keeping track of online remote terminal users. This bookkeeping has a very direct effect on the machine since a significant proportion of the CPU power is spent doing this bookkeeping. Many interactive systems use from 10 to 70 percent of the total machine power in doing such bookkeeping. One of the more famous, but now obsolete, systems utilized 90 percent of CPU power for overhead operations. On the other hand some studies of user program performance at remote terminals have been conducted and show that performance of the users may be increased threefold since they are able to have immediate response to their queries and computations.

In summary: There are two types of systems: The batch system, which is oriented to the efficient use of the computer but which in a practical environment generally requires some compromise in order to give a turnaround time of one to two hours for most users. In interactive systems, the design objective is to give the remote user optimum utilization of computer power to solve his problems. In practice this generally requires some compromise to give instantaneous (less than one-second response) to all requests which in turn requires more overhead. Many interactive system designs tolerate 20 second response times but in some cases times of 10 to 15 minutes may be tolerated. These systems typically reduce the proportion of the central processor power spent on overhead operations.

B. IBM 360/67 Hardware

IBM 360/67 is a large size general purpose computer which is equivalent to an IBM 360/65 in

processing power. Basically, the Model 67 is a Model 65 which has been extensively modified. Three hardware components, a DATBOX (Dynamic Address Translator Box), eight associative registers, and special channel hardware, have been added to the Model 65 to make it a Model 67. Special circuitry has also been added to allow three additional hardware components, storage switch, one or more channel controllers and a configuration console to be added as options. The optional hardware is used when multiple processors are assembled as a duplex multiprocessing unit.

The DATBOX and the eight associative registers were added specifically to facilitate dynamic address translation which allows a dynamic relocation of memory pages (a page equals 4,096 bytes). The dynamic relocation hardware is used by special software to implement the concept of a virtual memory. In the virtual memory concept, independent multiple users imagine that they have large memory sizes available to them. One user may imagine and effectively use more memory than the central hardware has available, and multiple users in aggregate may use many times the total real memory available. The chief controlling software manages memory resources and allocates these resources to users as per the need of each.

1. The Virtual Memory Concept

Virtual memory is a software implementation. It is beneficial to have hardware which easily implements this type of software. The Model 67 is the only computer in the IBM 360 line which has hardware to aid in the implementation of the virtual memory concept.

Considering the cost of core, storage cost-per-byte increases in ascending order in the following media: magnetic tape, magnetic strip, magnetic disk, magnetic drum, and magnetic core. However, the access time decreases in the same sequence from milliseconds to nanoseconds. Theoretically if magnetic core were inexpensive, a large amount of it would be available to each installation, and the thinking related to system

programming and design would probably change. However, since cost is a prime factor, the concept of virtual memory was developed in order to distribute large areas of core among many users.

In the ordinary machine a great deal of the core storage has been pre-empted for use by either the hardware or by the resident portion of the operating system. Lower core is used for machine oriented cells such as program status words, timers and diagnostic scan areas. The resident portion of the operating system may take from 50,000 to 100,000 bytes. The remainder of core is available to the problem program but even much of the remaining portion must be used for input-output buffers and for the access routines which use the buffers.

Management of core allocation in a multi-programming environment is a problem under the best generalized multiprogramming systems. Several independent programs reside simultaneously in core and make the request for additional core on a reservoir or pool of core available to all of these programs. However, even in this case, core size is finite and the sum of the hardware-required core, the resident system core, and that of all application programs must not exceed this finite limit. As an additional problem, as variable sizes of programs are loaded and terminated, interspersed sections of core are vacated and cannot be used because of fragmentation conflicts.

The ideal situation from a hardware viewpoint is to have a very large amount of core for each user and, of course, this implies that core would have to be relatively inexpensive. This is not the case. A large amount of core can be simulated (if the proper hardware is available) to the extent that no user need know he does not have real core at his disposal. A good virtual memory system should be transparent to the user. In order to implement or simulate such a system; 1) adequate external direct access storage is needed to hold all

of the programs, and 2) a monitor system is needed to handle the mechanics of swapping sections of programs from direct access device to machine and vice versa. Also, the monitor system must provide the necessary relocation of addresses.

In the IBM 360 there is a 24 bit addressing structure which theoretically allows memory sizes of 16 million bytes (2^{24} = 16,777,216). With the virtual memory concept on the Model 67, the control program allows each user to think that he has 16 million addressable bytes of memory at his disposal. The control program, with the aid of the dynamic address translation feature, translates these addresses into real memory addresses; if the addressed code is not available in real core, the control program directly retrieves the addressed pages of code from the direct access storage devices. In this system there are obvious advantages to the user.

a. He does not have to plan and implement overlay structures. He can code his program strictly on addressing and not within the constraints of the storage capacity of the system.

b. He does not need to be concerned with infrequently used coding. The monitor will make it available only when it is required.

c. He can act and react as though he alone had the entire computing system under his jurisdiction.

Each reference to virtual memory must be translated to an address which occupies real storage. The time cost of each address translation on the Model 67 is 150 nanoseconds. In some cases, the cost may be higher if the code does not exist in core and must be moved from direct access storage. Despite a considerable overhead, the system effectively distributes blocks of code internally to many users over dynamic increments of time. (For further information on the virtual memory concept, see "Virtual Memory Concept" dated 12 April 1968, attached as Appendix I.)

2. DATBOX (Dynamic Address Translator Box)

The interface between virtual memory and real core is the DATBOX. Only the supervisor deals with real core addresses. All other programs deal with virtual memory addresses which the DATBOX translates into real core addresses.

3. Associative Registers

In order to quicken the translation process, frequently arising addresses are translated with the use of eight associative storage registers containing logical addresses and results of the most recent address conversions. The eight registers can perform all references in parallel in a total of 150 nanoseconds. A relocate instruction counter is also provided to lessen translation time; therefore, instructions executed sequentially within a page do not require an associative reference.

4. Channel Hardware

Special channel hardware circuitry has been integrated to interface properly with multiple CPU's or multiple channel controllers.

5. Storage Switch Units (optional)

This unit is used as a tie breaker which controls the acceptance of memory requests in a predetermined priority sequence.

6. Channel Controller (optional)

The channel controller is functionally an input-output processor. Upon receipt of a channel address word, the channel controller can oversee the continuation of channel commands without intervention from the CPU. The channel controller is used to control both the multiplexor and the selector channel. Multiple channel controllers can be used on the system to facilitate reconfigurations. No channel controller is needed on a simplex system.

7. Configuration Console (optional)

Memories, channel controllers, CPU and control units may be switched at this unit. Any of these units may be partitioned to an operating subsystem or switched offline for operational reliability reasons.

C. Software Options for the IBM 360/67

Several software options are available for use on the IBM 360/67. The Model 67 does have additional instructions which are used to implement the dynamic relocation address features. Thus Model 67 software has a larger instruction set than that on standard 360 hardware. Several universities have written time-sharing systems for the Model 67 which utilized its special hardware features. However, all of these which were analyzed were somewhat limited in nature and at best were "experimental" systems. Two systems are worthy of mention; TSS (Time-Sharing System) which is IBM's official system for the Model 67, and the CP System developed jointly by Lincoln Laboratories of MIT and the IBM Cambridge Scientific Center.

1. TSS

TSS was developed by IBM and when first implemented in 1967 performed so poorly that IBM withdrew it from marketing consideration. In fact, IBM redesigned TSS and in early 1968 implemented the new version on several Model 67's. Recent evaluations of this software by OCS suggest that it is still not an acceptable software package for Agency usage. It is a workable system and long range plans suggest a powerful potential; however, its present performance per dollar is low by any competitive evaluation.

TSS, conceptually, is a very ordinary operating time-sharing system; the system resides on the machine and multiple users of this one system process their problem programs subordinate to the TSS. Each user has available up to 16 million bytes of memory and TSS allocates memory to each user in turn by the implementation of the virtual memory and its paging techniques.

2. Control Program (CP-67)

Control Program (CP-67is a super control program; it is not an operating system. Operating systems such as OS, BOS, CMS, and even the OS based TSMON time-sharing system run at a lower hierarchical level. All are equal and subordinate to CP. The usual operating system performs two chief functions: resource (time, memory, device) allocation and data management. CP performs only resource allocation. Thus, multiple combinations of resources such as memory, tapes, and DASD (Direct Access Storage Device) space can be defined to CP. A package of the above resources (memory, tapes and DASD) in effect is the equivalent of a computer; the only missing ingredient is central processor time. CP allocates this time by giving time increments to CPU power to each defined combination of resources.

In effect, we see a small control program (CP) controlling multiple combinations of resources, each combination having its own software or operating system. Thus, we say that CP controls imaginary computers (virtual computers in the trade terminology). Each virtual computer can run any IBM 360 system and program. When CP must allocate more memory than it has, it uses the dynamic relocation hardware features of the Model 67, and uses DASD space to simulate the real memory requirements.

In the OCS planning, as soon as TSMON (Time Sharing Monitor, the system written and used by OCS) is certified as correctly running subordinate to CP, the combination of CP and TSMON will be used. This transition from TSMON to CP/TSMON should be transparent to users. One objective in using CP is to be able to use other systems such as OS and CMS concurrently while TSMON is operating.

III. OCS Interactive System

A. Structure

The conceptual structure of the interactive system shows CP at the top hierarchical level (see Appendix II). All usual operating systems such as CMS (Cambridge Monitor System), OS (the Operating System), DOS (Disk Operating System), and TSMON (Time-Sharing Monitor Operating System, developed by OCS) are one level below CP and are subordinate to CP. Each user of a terminal may request the operating system and the hardware configuration he requires. CP will allocate the hardware and software operating system required. As seen in Appendix II, even though CP may have only 524K bytes of real memory, each of the two CMS systems may have 256K bytes, one OS system may have 512K bytes, the other OS system may have two million bytes, the ATS system may have 64K bytes and TSMON may be allotted 524K bytes. Thus, the aggregate core allotted to operating systems far exceeds the real core which is available to CP. The "marginal" core is in virtual memory, i.e., it is on the direct access storage.

In Appendix II, the first CMS system with an IBM/2260 as a console could be creating and executing FORTRAN programs, and displaying the computed results. The second CMS system which, in this case, is using an IBM/2741 might be creating and editing a data file. On the OS system which, in this case, has a 2741 console, the user may be compiling and executing a large FORTRAN program. On the ATS (Administrative Terminal System) which uses a DOS system, a secretary may be using a 2741 to create and edit a report or a technical document. On the other OS system the user at the 2741 may be receiving data from an RJE (Remote Job Entry) device such as a PDP-8, IBM 2780, or a IBM 1130. Most RJE devices for reading cards, punching cards, and printing data can be attached to the system. In the last diagram the TSMON (OCS time-sharing monitor) will be accessed by a 2741 terminal which in effect simulates the main console of the present OCS time-sharing machine. The terminals attached to this system will access TSMON and its virtual hardware just as they have been accessing TSMON and its real hardware components on the IBM 360/50.

Execution times on the Model 67 are not quite equivalent to those on a Model 65. Because of the

dynamic relocation addressing, there is a slight delay in execution time of instructions. Thus, there is some inherent degradation of performance. In addition, there is considerable overhead given to CP in the allocation of other resources. Transferring pages to and from real memory does take time and absorb CPU cycles. Tests show that programs will run on a virtual machine at a speed somewhat less than that of Model 65. If only a few users are using the Model 67, each program may run as fast as it would on a Model 50. However, if many users are accessing the system, some programs will run no faster than they would on a Model 30. However, even with the slower execution speeds, in many cases total turnaround time from job submittal to production return is less than in the typical batch system.

B. Implementation

The system is to be implemented in early February 1969. The Model 50 will be removed; the Model 67 will be moved in. The physical shuffling of these two machines will completely curtail services for several days. Probably, not all necessary software will be implemented on the first day in which the hardware is running; however, there is confidence that the system will run soon after installation since many portions of software have been tested on a Model 67 and are now running. It is quite probable that portions of the system will run in degraded form; however, it is planned that they will run. It may be months before the system is tuned to optimum performance. Many things need to be learned about the interaction of these diverse systems with each other under one control program.

C. Problems

Many of the remote terminal devices which are presently planned do not meet security specifications. As equivalent remote components which do meet security specifications become available, these will be tested and phased into the system. A plan is underway to test and evaluate Raytheon DIDS-400 and Sanders-920 terminals; both appear to have less emanation problems than the components currently planned.

The CP software cannot handle certain types of programs. Up to now, none of these types of programs has been discovered in the Agency or in any of the

software which OCS anticipates to use. Two general types of forbidden software are timing sensitive software and software with data dependent I/O.

1. Timing Sensitive Software

CPU cycle time cannot be guaranteed to any one user. A total amount of CPU cycle time can be guaranteed, but a minimum number of CPU cycles cannot be guaranteed per time increment. For example, this applies to a device at a terminal such as a small computer which bursts (in parallel data paths) large blocks of many bytes of data. A device which transferred 100K bytes of data to the Model 67 in one burst may lose some bits if this transfer exceeds the available time slice.

2. Data Driven I/O

CP handles an I/O interrupt initially generated by a user system, makes the addressed core pages available, and then generates the command code to the real machine to process the I/O. Data which is being transferred can never alter the instruction sequence since CP has already generated the real code. Thus, data driven I/O is not acceptable.

D. Systems Available

Theoretically, any standard IBM 360 software (with the exception of timing sensitive software and data driven I/O described above) can be run subordinate to CP. Implementation of TSMON, the present OCS interactive system, is being given priority. CMS (Cambridge Monitor System), an operating system designed specifically to run well in a paging environment under CP, is being given next priority. OCS will implement OS early since this gives the new system compatibility with present OS programs. Other systems, such as TDMS, ATS, etc. are being given lesser priority.

In order to present an orderly and transparent transition for customers of the interactive system, OCS is giving very careful attention to the TSMON implementation. TSMON under CP in effect is a time-sharing monitor within a time-sharing monitor and

some unusual and unpredictable operating conditions are anticipated. In spite of the unknowns, OCS feels that it is master of the implications and has confidence in producing good results. CMS is specially designed to run well under CP. It offers proven good performance under CP for a variety of services.

Under CP each user is given an USERID (user identification code). For each USERID there is defined a virtual hardware configuration and one or more software systems which are available to this USERID. Thus when the user sits at his console, he has a virtual machine with the operating system of his choice. At present, terminals which are subordinate to TSMON cannot access other systems, but an effort is being made to remove this restriction.

1. Time-Sharing System Monitor (TSMON)

The Time-Sharing System is composed of a control program, service routines, and user programs. The control program TSMON is a problem program resident in core which directs the operation of the System. By replacing interrupt locations in the Operating System, the monitor gains control, generates the time sharing partitions, maintains a pseudo-clock to designate time intervals, and handles all interrupts.

a. System Services

(1) CAM

The Conversational Access Method (CAM) enables the user to "converse" with the computer via remote terminal devices. CAM comprises a package of programs capable of reading from or writing to any terminal.

(2) PAM

The Paging Access Method (PAM) enables the user to easily create and to manipulate files for the Time-Sharing System. All files reside on

a direct access device and are blocked into 1024 bytes which constitute a PAM page.

(3) RINUS

A PAM page contains one physical record per 1024 bytes. The RINUS program efficiently utilizes PAM pages by writing records contiguously and overlapping records between pages. RINUS also provides indexing capabilities.

b. User Services (Programs)

(1) DESKCAL

DESKCAL serves as a super powerful desk calculator. Refer to Appendix IV.

(2) SOLVE

SOLVE is a computational programming language and provides the user with arithmetic operations, elementary mathematical functions, and select programming commands. Refer to Appendix V.

(3) TORQUE

TORQUE (TSAR) is a program which allows rapid remote querying of direct access files and outputting of information from these files in a variety of formats and on a number of devices. Refer to Appendix VI.

(4) LINUS

The LINUS program provides a method of creating files, maintaining files, and retrieving information on a terminal. Refer to Appendix VII.

(5) TRUMP

TRUMP furnishes commands which locate, display, and change fixed fields in TORQUE (TSAR) files via the terminal. Refer to Appendix VIII.

(6) Customized Retrieval

A package of compression, hashing, and indexing techniques is available for the customizing of large files for fast direct access in which the primary search criteria is on one field. By using these techniques, a staff programmer can very quickly put a user file into on-line query operation.

Contact OCS/APS for further information and assistance.

(7) On-line BATCH

With the program BATCH, the user can request a printout of a file from the terminal. The following programming languages are supported under BATCH: Assembly Language, Programming Language/One, and FORTRAN. Two IBM Operating System utilities are available -- IEHPROGM and IEHLIST. Also two OCS utilities have been implemented -- PRINT and PUNCH. A special procedure TSBATCH can be entered to simulate the terminal on the printer.

Data can be directly entered on the terminal or a LINUS or SOLVE file can be loaded for program execution.

BATCH provides some limited batch services from a terminal. The programs per se are not interactive but program control is interactive.

The System schedules high priority batch time for the job and prints the input and output on the printer rather

than on the remote terminal. An option is being written in the system to allow printouts to be sent to the remote terminal.

LIST OF PROCEDURES FOR ON-LINE BATCH

```
PL1LFCLG
FORTGCLG
ASMFCLG
ASMTESTL  (assembles and linkedits the file;
           permanently saves output of linkage
           editor)
PRINT
PUNCH
IEHPROGM
IEHLIST
PGM=TSBATCH
```

SAMPLE

```
ENTER PROGRAM NAME
   ► BATCH
SPECIFY PROCEDURE (AND PARM)
   ► PL1LFCLG
SPECIFY FILENAME FOR EXECUTION (OR'*DATA')
   ► *DATA
ENTER DATA (TERMINATE WITH '/*')
   ► FIRST:  PROC OPTIONS (MAIN);
   ► DCL  AA  FIXED (2)  INIT(00);ON ENDFILE(SYSIN)GOTO EF;
   ► DO  J = 1 to 80;
   ► GET EDIT (AA)(F(2));PUT EDIT(AA)(F(2));
   ► END;
   ► EF; END FIRST;
   ► /*
ENTER DATA FILE NAME, '*DATA', OR 'NONE'
   ► *DATA
ENTER DATA (TERMINATE WITH '/*')
   ► 1245893467
   ► /*
JOB IS NEXT FOR EXECUTION
ENTER PROGRAM NAME
      .
      .
      .
      .
```

2. Cambridge Monitor System (CMS)

The Cambridge Monitor System (CMS) is a conversational operating system designed to provide a wide range of capabilities through relatively simple commands at a terminal.

Users communicate with the system through commands that cause compilation, file creation, and numerous other operations. Familiarity with the entire command set is not necessary in order to use the system; typically, if the system is to be used for an occasional FORTRAN compilation and execution, a knowledge of only two or three commands will be sufficient. Should the user's needs be more varied, however, numerous commands are provided which support many complex operations.

Whether running on a real or a virtual machine, CMS expects the following machine configurations:

device	virtual address	symbolic name	
1052	009	CON1	console
2311	190	DSK1	system disk (read-only)
2311	191	DSK2	permanent disk (user files)
*2311	192	DSK3	temporary disk (work space)
1403	00E	PRN1	line printer
2540	00C	RDR1	card reader
2540	00D	PCH1	card punch
*2400	180	TAP1	tape drive
*2400	181	TAP2	tape drive
At least 256K bytes of core storage.			

* The 2311 for the temporary disk and the two 2400 tape drives are optional devices; they are not included in the minimum configuration. 2314 disks can be substituted for the 2311's.

Under the Control Program (CP), of course, these devices are simulated and remapped to different addresses and/or different devices. For instance, CMS expects a 1052 printer-keyboard operator's console, but remote terminals are 2741's or 2260's, CP handles all channel program modifications necessary for this simulation.

Under CP, all CMS users share the read-only system disk; on it reside the CMS nucleus routines, which the user IPL's, and the other routines and libraries that CMS calls as needed. Since each virtual machine maintains its own copy of CMS, users may modify it as they wish without affecting either the system disk version or the copies in other virtual machines.

a. File Management Under CMS:

Each CMS user is assigned two virtual disks (a third disk is optional), one of which is shared with other CMS users. The shared disk contains the CMS nucleus, which is loaded into the virtual machine by the IPL console function. Also on this disk, referred to as the "system (SY) disk" are CMS commands which are not core-resident, and system libraries of routines and macros. No user may write on this disk as it is read-only. Any attempt to modify any of the system files results in an error message.

The two other disks are known as the "permanent" and "temporary" disks. The user does not share these disks with any other user, as they are accessible only to him after he has logged in with the correct USERID and password. The permanent disk is used for files which are to be saved from one terminal session to the next. The temporary disk, which is optional, provides space for work files which need not be retained between sessions. This disk is erased whenever the user logs out.

CMS uses a three-field file identification to catalog both system and user files. The fields are referred to as the filename, filetype, and filemode of the file. Uniqueness of any one of the fields is sufficient to differentiate a file from other files. CMS maintains a directory (the User File Directory) of each user's files, which includes information on the file format, size and location. This allows the user to specify files by using only the file identification or a portion of it.

All CMS disk files are written in 800-byte physical records. The system I/O routines handle packing of logical records into this format. The record blocks are written onto the user's disk area in random order. CMS maintains chains of disk addresses to keep track of the files. These chains are linked to the User File Directory, which has an entry for each user file. The directory is brought into storage when the user logs in, and is updated whenever files are used. Periodically, and at least as often as once per command, the updated file directory in core is written out onto disk, so that the permanent copy is as current as possible. This insures an accurate directory if it is necessary to re-IPL CMS during a terminal session.

The directory handles files up to 12.8 million bytes in length, which is beyond the capacity of a whole (2311) disk pack. In practice, the user's disk will not normally handle files of that length, since it is usually less than a whole disk. Whenever CMS detects that only a few tracks are left on the user's disk, a warning message is typed, and files currently open are closed. A program or command in execution is halted, so the user may create more free space on the disk by erasing some files, or copying them from disk to other media.

Although most of the CMS commands operate on disk-resident files, the user also has access to the card-read-punch, printer, and tape drives. The commands, in general, create sequential files of fixed-length records; however, the programmer using the CMS I/O support routines is able to use any record format with either fixed-length or variable-length records.

Files are automatically "opened" for reading or writing when the first read or write is issued. Only eight files may be open at the same time. CMS routines automatically close files after every command, and after user programs that complete normally. If

a user program needs to access more than eight files during execution, the FINIS command must be called to close some files. Files must also be closed between writing and reading.

b. CMS Commands:

CMS commands fall naturally into four categories: file manipulation, compilation, execution control, and debugging aids.

The file handling commands allow the user to create, copy, move, combine, and erase disk files. Other commands provide access to the tape units, printer, and card-read-punch. Under the CMS linkage scheme, all of these commands are available to executing programs as well as to the user at the terminal.

The CMS language processors are the same ones used under Operating System/360 (OS); these include Assembler (F), FORTRAN IV (G), and PL/1 (F). The Assembler produces object programs that may be executed under either CMS or OS, depending on the macros used in the source program. Special file-handling routines for macro libraries are included. The FORTRAN and PL/1 compilers also produce OS-compatible object programs. (The FORTRAN execution-time support programs have been modified for CMS.) The SNOBOL compiler and assembler-interpreter were adapted from programs designed to execute under OS and are also available.

The execution control commands allow the user to load his programs from single object decks (the filetype TEXT is reserved for relocatable object programs) or from a library of programs. He can pass a list of parameters to his program from the terminal, and specify the point at which execution is to begin. To avoid relocation (bypass the relocating loader) he can create a file consisting of an image of the portion of core storage containing his program, and load that non-relocatable copy back at any

time. Since the loading commands can be accessed by executing programs, overlay structures may be set up. The user can also create a file which is a series of commands, and then execute these commands by typing a single line (loading and executing that file).

The debugging command in CMS is called DEBUG. It allows the user to stop his programs at pre-determined points and examine his registers, PSW, and storage, and modify these if he desires. This information may be typed out at his terminal or printed offline. A program interrupt gives control to DEBUG, as does the external interrupt caused by the EXTERNAL console function. The user may also employ the program tracing routines, which record all SVC transfers, or will record just those in which an error return is made.

A description of each command follows:

ALTER changes all or part of the identifier (filename, filetype, and filemode) of a file stored on the user's permanent or temporary disk without altering the contents of the file.

ASSEMBLE converts assembler language source code into relocatable object code using the OS/360 F level assembler.

CLOSIO signals the Control Program that I/O to offline unit record equipment has been completed and that the spooling areas for this I/O may be processed. CLOSIO is generally issued automatically by the commands which access unit record equipment.

CLROVER clears overrides set by the SETERR and/or SETOVER commands and causes all recorded trace information to be printed on the offline printer.

COMBINE copies the specified file(s), concatenating them in the order given, into a new file which is placed on the user's permanent or temporary disk and assigned the specified identifier.

DEBUG allows the user to stop and re-start programs at specified points and to inspect and change the contents of registers, core locations, and hardware control words online.

DISK	causes a CMS disk file to be punched out or read in from cards which are in CMS card format.
DUMPREST	dumps the contents of an entire disk to magnetic tape or restores the contents of an entire disk from magnetic tape.
ECHO	tests terminal line transmission by repeating as typeout whatever is typed in by the user.
EDIT	allows the user to create card-image files on disk and to make changes to existing files from his terminal.
ERASE	deletes the entry for a specified file (or files) from the appropriate directory, rendering the file inaccessible to the user, and freeing the disk area containing that file.
EXEC	executes a file containing one or more CMS commands, allowing a sequence of commands to be executed by issuing a single command.
FINIS	closes the specified file (or files) by writing the last record of that file on disk, updating the User File Directory, and removing the entry for that file from the user's table of active files.
FORMAT	prepares the user's permanent or temporary disk area for CMS use by writing blank records over the currently stored information.
FORTRAN	converts FORTRAN language source code into relocatable object code using the OS/360 FORTRAN G compiler.
GENMOD	creates a non-relocatable core-image file on the user's permanent disk which is a copy of the contents of core between two given locations.
GLOBAL	specifies (1) macro definition libraries to be searched during the assembly process or (2) test libraries to be searched when loading files containing relocatable object code.
KE	truncates information currently being typed at the terminal to 72 characters per line. This truncation will remain in effect for the duration of the currently executing command or user program.

KO	clears overrides previously set by the SETOVER or SETERR commands and causes all trace information recorded by these commands to be printed on the offline printer.
KT	stops typeout at the terminal for the duration of the currently executing command or user program.
KX	terminates the currently executing program, updates the User File Directory, and logs out from CMS, transferring control to the Control Program.
LISTF	either types out at the terminal the identifier and size of the specified disk file(s), or creates a file on the user's permanent disk containing information for use by the EXEC and/or $ commands.
LOAD	reads the specified TEXT file(s) -- containing relocatable object code -- from disk, loads them into core, and establishes the proper linkages.
LOADMOD	reads a MODULE file -- which is in non-relocatable core-image form -- from disk and loads it into core.
LOGIN	causes the user's permanent disk files to be either saved or deleted, as specified. If LOGIN is not issued, the files will be saved.
LOGOUT	compacts the User File Directory, executes any CMS command specified as an operand, and logs out CMS, transferring control to the Control Program.
MACLIB	generates or adds to a specified macro library, or types out the contents of the dictionary of that library.
OFFLINE	creates a disk file from card input, prints a disk file on the offline printer, or punches a disk file on cards.
PL1	converts PL/1 language source code into relocatable object code using the OS/360 PL/1 F compiler.
PRINTF	types at the terminal the contents of all or part of a specified disk file.

REUSE	reads the specified TEXT file(s) -- containing relocatable object code -- from disk and loads them into core, establishing linkages with previously loaded files and changing the default entry point of these files to that of the first file specified in the REUSE command.
SCRIPT	either (1) allows the user to create arbitrary alphanumeric text files on disk and to make changes to existing files of this type from the terminal or (2) types out the contents of the specified file, formatting it as indicated by control words contained in the text.
SETERR	sets error overrides which will cause trace information to be recorded for each SVC-called program which returns with an error code in general purpose register 15.
SETOVER	sets normal and error overrides which will cause trace information to be recorded for all SVC-called programs -- both those which are executed normally and those which return an error code in general purpose register 15.
SNOBOL	converts a card-image file in Snobol source language into SPL1 interpreter language and executes SPL1 programs.
SPLIT	copies the specified portion of a card-image file and appends it to a second specified card-image file.
START	begins execution of the loaded program(s) at the specified or default entry point and passes the address of a string of user arguments to the program(s).
STAT	types statistics regarding the amount of permanent and/or temporary disk space used; lists all user-defined commands, or compacts the User File Directory, as specified.
TAPE	writes the contents of CMS disk files of any type or size onto magnetic tape, or restores these files by writing them from tape onto disk.

TXTLIB — either (1) generates or adds to a specified text library, (2) types out the contents of the dictionary for that library, or (3) creates a file containing a list of entry points and control section names contained in that library.

UPDATE — updates the specified disk file with a file containing control cards, where each control card indicates whether the information immediately following it is to be resequenced, inserted, replaced, or deleted.

USE — reads the specified TEXT file(s) -- containing relocatable object code -- from disk and loads them into core, establishing linkages with previously loaded files.

$ — executes a file containing one or more CMS commands, or loads into core a file which is in either core-image form or relocatable object code and begins execution of that file.

3. OS-360 (IBM-360 Operating System)

OS-360, the largest and most comprehensive of the several operating systems furnished with the IBM-360, will be used as a subordinate system to CP-67. OS can be SYSGEN'ed in one of three optional forms, PCP (Primary Control Program), MFT (Multiple Fixed Tasking), and MVT (Multiple Variable Tasking). OCS has been using the latter two options because they perform concurrent multitasks such as SPOOLing and multiprogramming. Both options require additional software and, consequently, more overhead.

The PCP version of OS is planned for SYSGEN in lieu of MFT and MVT because it involves less software. The software of MVT and MFT repeat some services offered by CP-67 which probably would require extra paging and, thus, would degrade performance. The differences between PCP, MFT, and MVT are transparent to users. With any one of the three options, any remote terminal can be viewed as the main console for the virtual computer. Most services of the batch MVT systems on the Model 65's will be available through other Model 67 operating systems.

The use of OS is limited in practice, but not in theory. Certain processes, such as long computer bound jobs or large matrix computations, probably should not be run on the Model 67. However, at this time it is impossible to predict the parameters and attributes of programs which will run better on the Model 65's. Every CP user of OS services must proceed cautiously and must be prepared to adjust to the batch on Model 65 if his particular application is not practical on the Model 67.

4. Time-Shared Data Management (TDMS)

a. INTRODUCTION

The Time-Shared Data Management System (TDMS) is a general-purpose system for managing data in a time-sharing environment. TDMS enables individuals and organizations with large, and often complex, data files to manage their data with the speed and ease provided by a computer-based system. TDMS is oriented to the nonprogrammer user, who, after learning some basic commands, can work directly with the computer to manage his data base. OCS has not solved all problems of placing this system subordinate to CP, but the problems appear solvable. In early 1969, an estimate will be made as to a feasible implementation date.

TDMS permits the user to describe entries in a data base, to load them into the machine, to ask questions about them, to perform calculations on them, to have the data displayed on a cathode-ray-tube, to obtain hard-copy reports, and to update and maintain the data base.

b. MAJOR DESIGN FEATURES

TDMS is "conversational"-- i.e., during the man/machine dialog, the system may ask the user to supply parameters, control-information, file-names, operations to be performed, and format desired, while the user, in turn, may ask the system to define a term, to comment upon a process he does not understand, to tell him what steps of a procedure are available, to explain error messages, or to give the user other tutorial help.

TDMS provides the user with a complete, integrated set of generalized data management tools, including capabilities for manipulating data files and generating reports or graphic displays, performing on-line updating and retrieval, and managing many different kinds of data bases.

The TDMS file organization employs list, table, and inverted tree data-structures which make optimum use of random-access, mass storage devices. This permits the rapid retrieval of isolated entries in the file, while maintaining the hierarchical relationships among various data items.

c. SYSTEM OPERATION

In the typical case, the user will call upon a large data base that he has previously stored, the names and structure of which he has determined himself. He can then request that selected portions of his data base be displayed or printed out for his analysis. He can ask questions about the characteristics of his data (limits, number of items of certain types, average values, and so on); he can change values, perform arithmetic operations on the data, and combine or rearrange groups of data. Finally, he can ask for hard-copy reports of any data -- and in the format he prefers. Thus the user -- that is, the person directly concerned with the problem--receives only the information he really needs, at the time he needs it, and in the form most usable to him.

For example, suppose a personnel administrator had a large file of personnel data. Using the Query program of TDMS, he could search for patterns within selected subsets of this data base or find answers to specific questions.

If, for example, he wishes to determine the number of employees in some critical technical specialty he might need to recruit in the near future due to a sudden increase in the military draft, he could type in (by remote teletype) a request such as this:

PRINT COUNT EMPLOYEE NAME WHERE DRAFT STATUS EQ 1A AND NAME OF TECHNICAL SPECIALTY EQ ELECTRONIC TECHNICIAN AND MARITAL STATUS EQ MARRIED

He would immediately receive a count of the employees whose draft status was 1A, who were electronic technicians, and who were single, divorced, or separated.

If he were then interested in knowing the salary range and qualifications possessed by present beginning employees in this job skill, he could type in the request:

PRINT MINIMUM SALARY, MAXIMUM SALARY, AVERAGE YEARS IN SPECIALTY, AVERAGE TOTAL YEARS EDUCATION WHERE NAME OF TECHNICAL SPECIALTY EQ ELECTRONIC TECHNICIAN AND JOB TITLE EQ JUNIOR ELECTRONIC TECHNICIAN

He might also be interested in knowing the names of potential trainees for projected training classes in that job skill. If so, he might type in:

PRINT EMPLOYEE NAME BY LOW EMPLOYEE NUMBER WHERE JOB TITLE EQ JUNIOR ELECTRONIC TECHNICIAN AND TOTAL YEARS EDUCATION GR 14 AND APT SCORE GR 75

In addition to the retrieval capabilities available to the user through the Query program, TDMS also provides the user with the ability to change selected entries on-line. This is done by simply typing in the desired change, for example:

CHANGE MARITAL STATUS = MARRIED WHERE EMPLOYEE NUMBER EQ 32060 AND SALARY HISTORY = (800, 10/12/66) WHERE EMPLOYEE NAME EQ JOHN BROWN

TDMS also provides various aids for the user who is unfamiliar with (or who has for-

gotten about) the structure of a data base, the steps required to retrieve information, the meaning of terms or operations, etc. For example, to receive a list of commands available to him, the user merely types a question mark; to obtain additional information about an error (after receiving an error message from the system), the user types in the word "MORE"; to receive information on words recognized by the system -- such as command words -- the user types in the word "MORE" and the specific system word. For example, if the user typed in a statement such as

MORE SIGMA

he would receive a reply similar to this:

SIGMA COMPUTES AND PRINTS THE STANDARD DEVIATION OF A DATA VALUE FOR AN ELEMENT, SUBELEMENT, OR ARITHMETIC EXPRESSION

Similarly, if the user is unfamiliar with the structure of a data base, he can type in the word "COMPONENTS" and receive a list of the components in the data base. If he is still in doubt, he can obtain a fuller description of the components, including their legality, format, coding types and so on, by typing in the word "DESCRIBE".

d. SYSTEM COMPONENTS

As shown in Figure 1, the basic data management tools of TDMS include:

- DEFINE - allows the user to create a new data base description or alter an existing one; inputs may be prestored or provided interactively, on-line.
- LOAD - accepts all types of numeric and non-numeric data either batched or interactively, from a console; performs legality checks; permits error correction on-line; includes the capability to convert pre-existing data values to the required TDMS format.

- QUERY - allows the user to request single values, multiple values, or entire entries, printed on-line in a standard tabular format; provides full arithmetic capabilities, plus operators such as sum, minimum, maximum, count, average, and standard deviation.

- UPDATE - allows changing of values and adding or deleting either single values or entire entries.

- DISPLAY - allows the user to define, generate, manipulate, save, and recall a variety of display formats.

- COMPOSE/PRODUCE - allows the user to specify report format and content in a user-oriented language; provides arithmetic, ordering, formatting, and tutorial capabilities; produces reports on the interactive console or line printer.

- MAINTAIN - provides for merging, subsetting, extracting, ordering, restructuring, and updating data bases; accepts one or two different data bases, an on-line description of the desired output data base, rules for selection of data, and the transformations that are required.

e. SYSTEM USES AND IMPLEMENTATION

TDMS was designed to accommodate the needs of the specialists in some technical field -- for example, military intelligence, finance, or personnel management -- who are not professional programmers. They often have large volumes of structured data to deal with (that is, data among which meaningful associations exist), and they are required to manipulate these data to solve pressing, everyday problems.

5. Administrative Terminal System (ATS/360)

ATS/360 consists of a group of application programs operating under IBM/360 Disk Operating System and related computer equipment especially integrated to handle large texts and data files. Data may be entered into the system through the IBM 2741

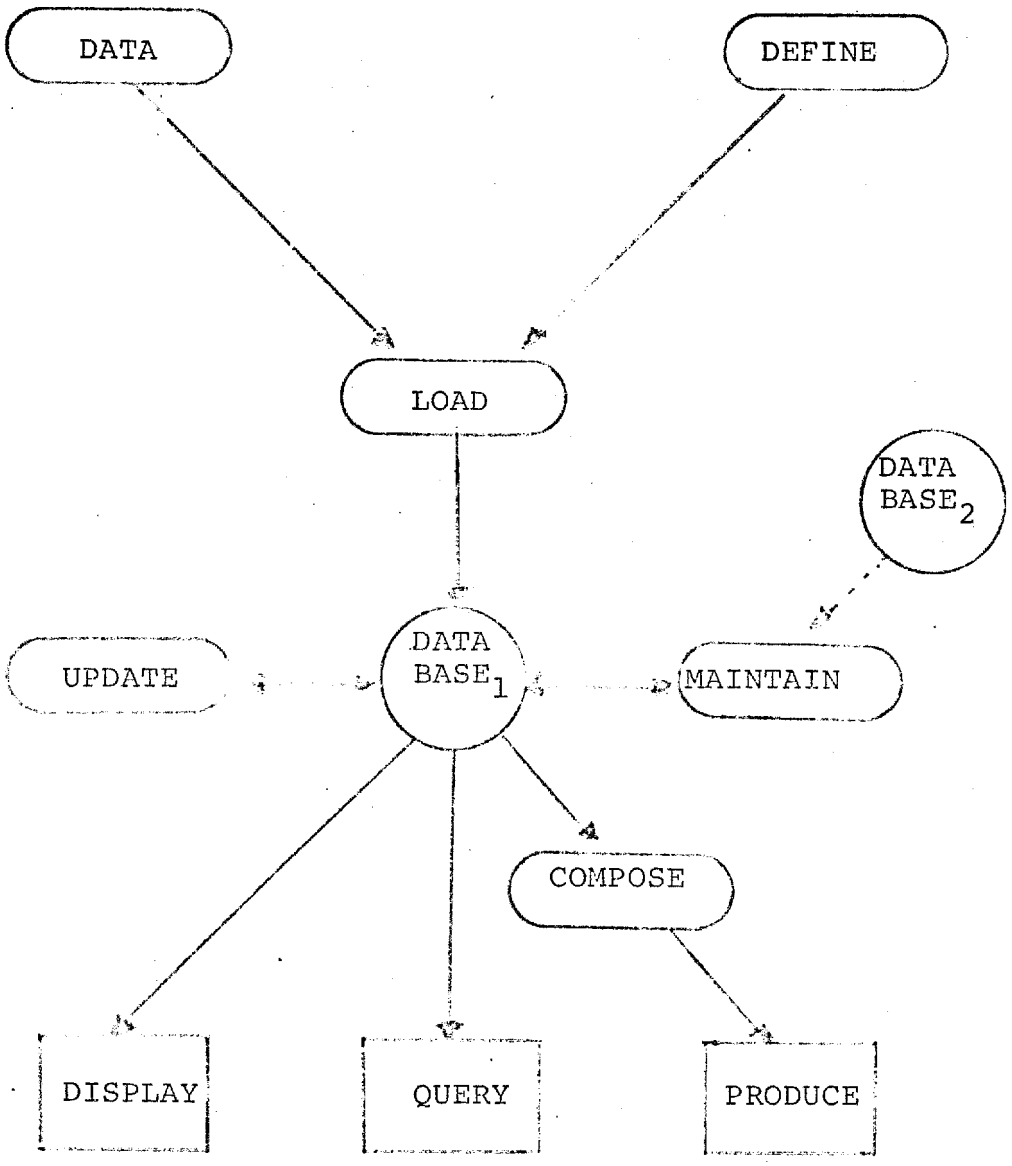


Figure 1. TDMS SYSTEM COMPONENTS

Communications Terminal, punch cards, and magnetic tape. Data is stored in the form of a text stream. A text stream is any combination of spaces, characters, words, sentences, or paragraphs in any sequence and entered at any period of time. It may be assembled at any time for output. The text stream may be outputted directly and a copy of the data reproduced, or it may be read as input to be processed in conventional data processing operations.

This system will be available in early 1969 under CP. It is being used successfully in many commercial installations. For more detailed information on ATS specifications see APS/OCS.

6. A Programming Language (APL 360)

IBM has developed a mathematically oriented language, A Programming Language (APL 360), which is useful to the non-programmer type on a remote terminal. With a minimum of typing, the terminal user is able to perform arithmetic operations and complex mathematical functions and receive an immediate response; operations and responses can be stored in a library for future access.

The processing of the language and the execution of programs in the language presently run on DOS (IBM's Disk Operating System). This system can be run as a virtual system on the CP-67 system in OCS. Only the IBM 2741 remote typewriter device has the special fonts and characters to run the language; and this device is supported by the OCS CP-67 system. The system will be available in early 1969.

Samples

1. In 4 bits, represent "7"

```
2222T7
```

2. Calculate average of any number of numbers

```
(+/Y)    Y
```

"add the reduction of Y, divide the dimensionality of Y"

3. "FICA" replaced by either ".044" multiplied times "G" or "7800", whichever is the lower value.

4. Expressing the equation: Y =

```
Y =     * (  t * 2)  2
```

IV. Terminal Installation

OCS has recently published a report outlining the procedure for acquiring a terminal. Call extension 4000 for a copy.

V. User Training

Courses have been designed to train users in the use of terminals and user services. Beginning in February 1969, courses will be offered through the OTR training bulletin on the use of services such as SOLVE, TORQUE, DESKCAL, on-line FORTRAN and on-line PL/1. Basic orientation courses for persons who will be located in interactive environments will begin in December 1968. The course, which is announced regularly in the OTR Training Bulletin, is five (5) days, one-half time (15 hours).

APPENDICES

12 April 1968

VIRTUAL MEMORY CONCEPT

GENERAL

In multi-tasking systems, including time-sharing systems, memory must be shared by the current users of the system, and the constraints inherent in such sharing partially negate the power of multi-tasking. As an example, computers may have the capability to process a number of programs concurrently but memory may not be large enough to hold all these programs concurrently. Sometimes, the monitor itself, in order to handle concurrent or time-shared programs, uses so much additional memory that the remaining memory can process but one program. Virtual memory is one method to give more transparency to memory size.

In the virtual memory concept a virtual memory size is chosen which may be many times the size of actual core memory. In the example of the 32 bit prefixed IBM 360/67, virtual memory is 4 billion bytes while core memory may be as low as 256K bytes (the 24 bit prefix IBM 360/67 allows 16 million bytes of virtual memory). Virtual memory consists of actual memory and the remaining memory which is usually located on direct access storage devices. For example 16 million of virtual memory may be as follows: .5M bytes in core, 3.5M bytes on a drum, and 12M bytes on a disk. Systems software is designed so that index tables (usually in core memory) can locate all blocks of virtual memory, and bring these memory blocks into core. Usually hardware is specially designed (or modified) to perform special parts of this transfer, such as rapid translation of virtual memory addresses to actual memory addresses. These memory blocks, which are transferred to and from direct access storage devices, are called pages. (In the IBM 360/67, a page is 4096 bytes.)

STORAGE ALLOCATION

Programs which are run in a multiprogramming environment must be written in such a way that they can be adjusted by the system during execution to run at any location within main storage. This

requirement is dictated by the fact that it is impossible to determine prior to running just what portion of main storage will be available. This modification of the program is usually done just before it is to be run. The process of specifying what portion of main storage a program is to be run in, is called relocation. Programs that are written so that they can be run from any location in main storage are called relocatable programs.

Programs are written in modules of 4096 bytes or less. Pages (4096 bytes of memory) contain 1 or more modules and are brought into main memory only when they are to be used. Each page is stored at an available page boundary in main memory. Its actual location, translation of addresses within the page, and translation of addresses beyond the page are performed by a technique called Dynamic Address Translation.

DYNAMIC ADDRESS TRANSLATION

All referenced instructions are computed from the sum of the displacement, the index register, and the base register. This computation normally would consume a prohibitive amount of time if software only were to be the address translation medium. A group of associative registers (8 in the IBM 360/67) is used to hold the most recently referenced instructions. These registers are searched in parallel and thus maintain full CPU speed, unless a new page is addressed; the page is moved into actual memory via the index structure (if it was not already in actual memory), and the new actual address is computed and stored in an associative register.

PAGING

A page is a fixed number of bytes (4096 in IBM 360/67). Systems which use virtual memory concepts generally move pages into memory when they are being used. When they are not being used and main storage space is needed for another task, they are written onto secondary storage which is part of virtual memory. Some advantages of paging are:

1) Only those pages of a user's virtual storage which are being used are actually in main storage.

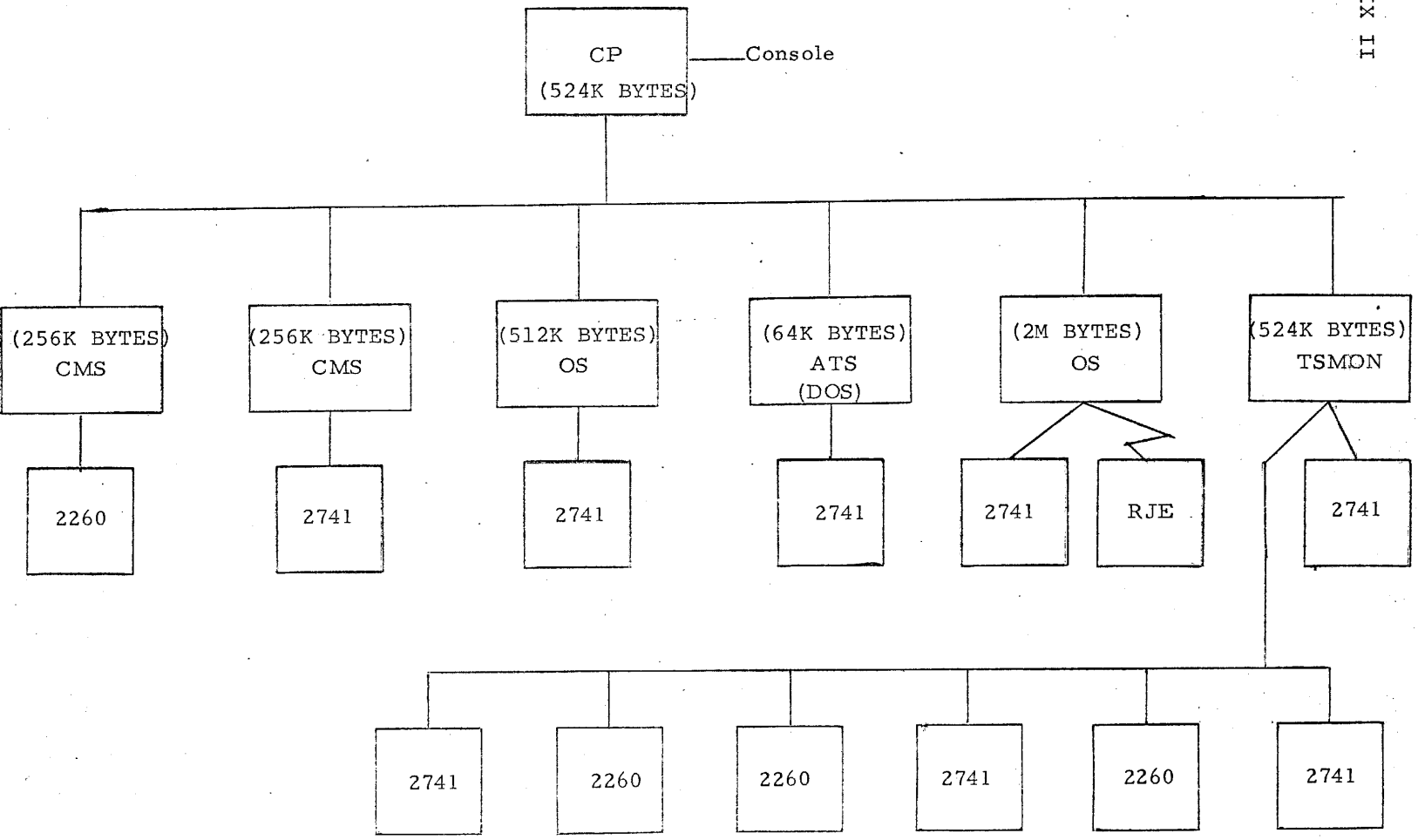
2) Very large programs can be designed without undue concern for the size of the computer's main storage.

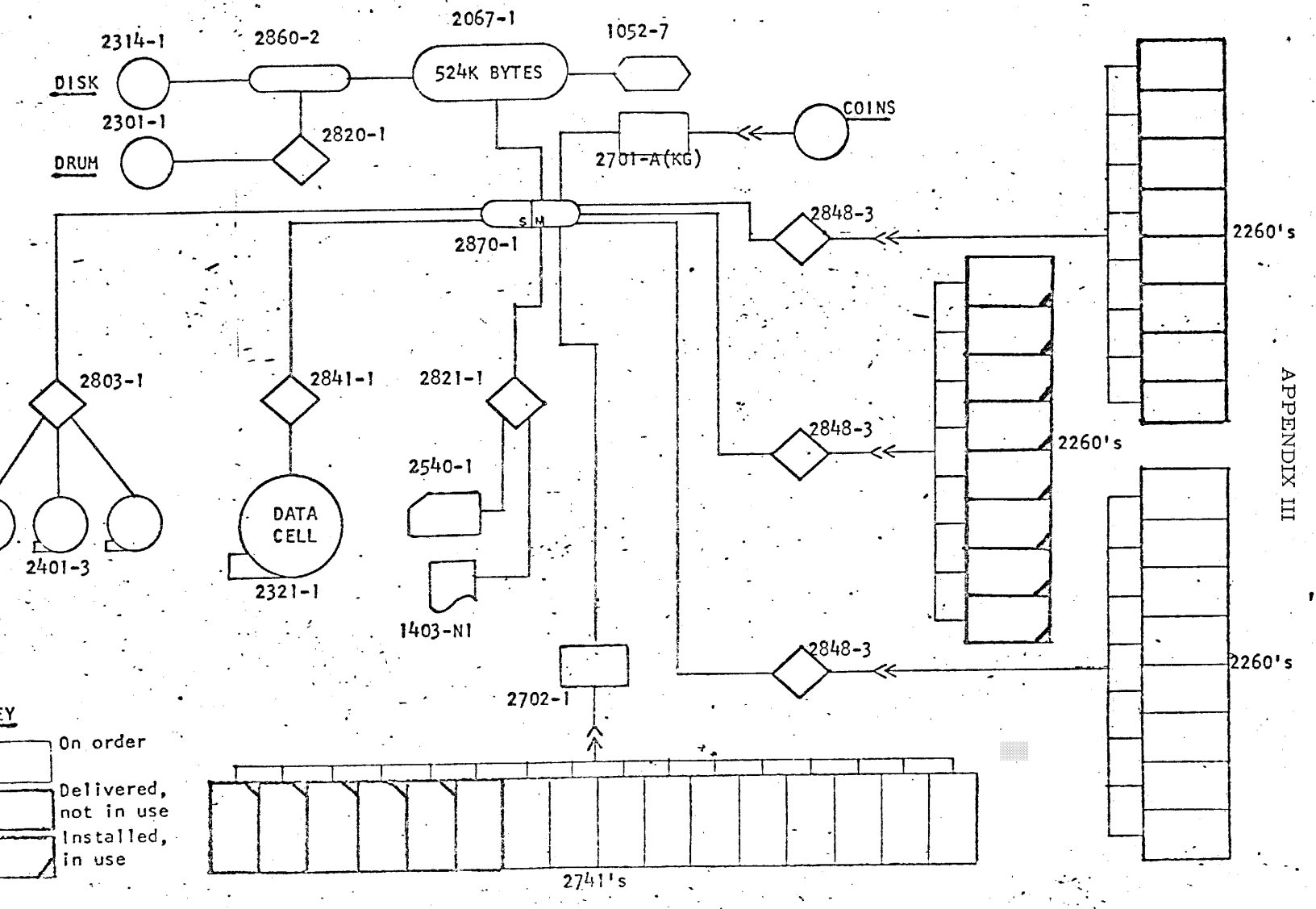
3) Re-entrant subroutines or re-entrant programs, such as the PL-1 Compiler, can be shared easily.

SUMMARY

Virtual memory is conceptual but does give flexibility to memory size restrictions through a combination of hardware and software techniques. This concept combined with paging is one solution to memory size constraints. In practice, paging between actual memory and secondary memory has been a disappointment in that I/O time to perform the paging has been excessive and has significantly degraded performance of most paging systems. Several systems (CDC 6400/6600 and several customer written systems for IBM 360) are using a slow speed core as secondary memory. Some designers believe that paging is an interim technique with no future promise. They hold that large cheap memories will be the future solution to the inherent degradation in all paging systems.

APPENDIX II





APPENDIX III

APPENDIX IV

DESKCAL

This program serves as a desk calculator performing the following operations and functions:

addition:	+	cosine:	COS
subtraction:	-	square root:	SQRT
multiplication:	*	absolute value:	ABS
division:	/	natural log (> Ø):	LOG
exponentiation:	**	common log:	LOGIØ
sine:	SIN	raise "e" to a power:	EXP

Rules:

1. One line is executed at a time.

2. Format of statement:

▶ variable = expression
.....or.....
▶ expression

3. A variable must begin with an alphabetic character and may consist of 7 alphanumerics.

4. A maximum of 8 nested parenthesis are allowed within one expression.

5. The value of a variable is maintained until the variable is reassigned a new value.

6. When using fixed-point notation, four (4) decimal places are obtainable from all operations except for the following functions which provide 5 decimal places: COS, SIN, ABS, LOG, EXP, LOG1Ø.

7. All computations are performed in binary floating point where the high order binary digit of the fifth significant decimal digit is rounded before the result is converted back to decimal.

8. Floating point notation is applicable with an exponential range of ± 74. A constant is expressed as decimal digits followed by "E" and a signed exponent.

Functions:

1. Angles from the sine and cosine functions are expressed in radians.

2. Format of function statement:

```
      ▻ variable = function (expression)
... or ...
      ▻ function (expression)
```

3. EXP raises "e" (the base of the natural logarithm system) to a given power.

a. format:
```
▻ EXP (expression)
... or ...
▻ variable = EXP (expression)
```

b. example:
```
▻ EXP (9)
8103.1
```

c. the value of the parenthesized expression cannot exceed 174.673.

EXAMPLES:

"ENTER USER STATEMENT"

```
      ►B = 9
"B = 9"
      ►C = 5
"C = 5"
      ►A = 2 + SQRT(B) + C**3
"A = 130"
      ►COSA = COS(A)
"COSA = -.36729"
      ►A = COSA + SQRT(B) + C**3
"A = 127.63"
      ►SQRT = SQRT(256.7968)
"SQRT = 16.025"
      ►Z = SQRT(SQRT)
"Z = 4.0031"
      ►A = 2 + A
"A = 129.63"
      ►SIN = SIN(COS(SIN(980)))
"SIN = .83302"
      ►A = 512 + 372 + 416
"A = 1300"
      ►SQRT(B)
"3"
      ►Z = 2.56E+4 + 2.45E-2
"Z = 25600"
      ►XY = 567.3E+3**2
"XY=3.2183E+11"
      ►FINISH
```

NOTE: All the underscored are computer responses.

SOLVE

Table of Contents

SOLVE

"Solve" provides the user with arithmetic operators, built-in functions, a few programming commands, and, in general, the ability to perform various computer functions via the display terminal. In accordance with the language and programming specifications, "Solve" permits the user to type a program into the Mod-50, to enter numeric data, and to execute the total as is the normal procedure.

I. Arithmetic Operators and Built-in Functions

A. Description

1. Addition: + and Subtraction: -

2. Multiplication: * and Division: /

3. Exponentiation: **

4. Relational operations (which always result as "1" (TRUE) or "Ø" (FALSE).

a. greater than: >

b. greater than or equal to: > =

c. less than: <

d. less than or equal to: <=

e. equal to: =

5. Logical operations (yield a value of "1" (TRUE) or "Ø" (FALSE).

   a. AND: &
   b. OR: |
   c. For example:

```
A = 5
B = 4
$C = (A> 4) & (B=7)
EXEC
C = Ø
```

6. Square root: SQRT

7. Absolute value: ABS

8. Sine: SIN

9. Cosine: COS

10. Natural log: LOG

11. Common log: LOG1Ø

12. Raising "e" (2.71828) to a given power:

    EXP

    a. format: ► symbol = EXP (expression)
    b. example:

```
► $A = EXP (9)
► EXEC
  A = 8103.1
```

    c. the value of the parenthesized expression cannot exceed 174.673.

B. Priority of Operations

1. **
2. *, /
3. +, -
4. >, >=, <, <=, =
5. &,

C. Implied groupings (which may be altered by a maximum of 3 nested "parentheses").

1. Addition and subtraction associate from the left -- example: 2+2 - 3+2 = ((2+2)-3)+2 = 3

2. Multiplication and division associate from the left -- example: 2/2*3/2 = ((2/2)*3)/2 = 3/2

3. Exponentiation associates from the left -- example: 2**2**3 = (2**2)**3 = 64

4. Mixed operations associate from the left -- example: 2+2*3**2 - 6 = 2+(2*(3**2)) - 6; 2>3&4>1 = (2>3)&(4>1)=Ø

II. Arithmetic Data Characteristics

A. Precision

1. Five decimal digits are significant.

2. All computations are performed in binary floating point where the high order binary digit of the fifth significant decimal digit is rounded before the result is converted back to decimal.

B. Range: floating point notation is applicable with an exponential range of ± 74.

C. A floating point constant is written as a string of decimal digits followed by the letter E and a signed decimal exponent.

III. Program Commands

A. LIST

1. must be entered as a unique command.

2. lists the complete program.

3. useful as a review of the program in toto.

4. it does not remain as part of your program: it is executed as soon as entered.

5. in order to temporarily halt a listing, depress the SHIFT and ENTER keys which will freeze the current screen contents.

   a. to re-activate the "LIST," type in "GO" and depress the SHIFT and ENTER keys.
   b. to de-activate the "LIST" permanently, type in "END" and depress the SHIFT and ENTER keys.

6. variations of LIST

   a. "► LIST n" this will list your program from statement n to the end of the program.
   b. "►LIST n, m" this will list the program from statement n to statement m.
   c. After execution of LIST, the user may continue to add instructions to his program.

B. EXEC

1. must be entered as a unique command and may not be preceded by a label.

2. allows the user to execute all of his statements (in fact, this command is mandatory in order to do so).

3. provides a listing of the variables and their respective results (refer to "SPECIAL FEATURES, DOLLAR SIGN" for a discussion about displayable variables).

4. in order to temporarily halt execution, depress the SHIFT and ENTER keys which will freeze the current screen contents.

   a. to re-activate the "EXEC," type in "GO" and depress the SHIFT and ENTER keys.
   b. to de-activate the "EXEC" permanently, type in "END" and depress the SHIFT and ENTER keys.

5. after execution:

   a. the user may continue writing a program.
   b. values entered during execution will be purged automatically after each EXEC instruction.

6. "EXEC" does not remain a part of your program; it is performed as soon as entered.

7. variations of EXEC

   a. "EXEC n" this will execute the program from statement n to the end of the program.
   b. "EXEC n, m" this will execute the program from statement n to statement m.

C. FINISH

1. must appear as a unique command.

2. terminates the program SOLVE and allows you to call in another.

D. RESTART

1. must appear as a unique command and may not be preceded by a label.

2. allows the user to purge his program and start writing a new program while SOLVE remains intact.

E. DELETE

1. an integer must appear after DELETE (at least one blank between them) -- the number refers to a previous single line which the user wishes to delete.

2. example: ⇨ DELETE 5Ø

F. VAR

1. may be entered as a unique command while writing your program.

2. applicable after completion of your first EXEC instruction.

3. provides a list of those variables whose values have previously been computed by an EXEC instruction.

G. IF condition THEN statement

1. may be preceded by a label.

2. the outcome of the condition must be a "1" (TRUE) or "Ø" (FALSE).

3. the condition may represent a compound expression -- for example: IF (A > Ø) & (BETA < 1000) | (BETA > 1500) THEN $SWT = 1.

H. ELSE expression

(TO BE IMPLEMENTED)

I. DCL

(TO BE IMPLEMENTED)

J. END

1. may be preceded by a label -- example: [label:]END.

2. used to:

(TO BE IMPLEMENTED)

K. GO TO

1. may be preceded by a label.

2. directs control to a specific label -- for example:
   - INIT: A(J) = 1.255
   - J = J + 1
   - LOOPEND: GO TO INIT

3. format: [label:] TO TO label.

L. DO

(TO BE IMPLEMENTED)

M. SEQ

1. The computer provides a 4 digit sequence number for each line of user responses. SEQ sequences the user's program by the increment of 10.

2. must be a unique command and is executed as soon as entered.

3. variations of SEQ:

   a. SEQ n -- n must be a numeric or a pre-assigned variable and provides re-numbering by the increment of n.
   b. SEQL -- provides the user with a listing of the program while it is being re-numbered.
   c. SEQL n -- same as SEQ n with a listing.

N. STOP

1. terminates processing from a LIST or EXEC (after depressing the SHIFT and ENTER keys).

2. permits the user to continue writing his program.

IV. Special Features

*A. Arrays

1. maximum of three dimensions.

2. must be defined within a DCL statement along with its dimensions.

3. within a DCL statement, the dimensions must be numeric.

4. an array name may contain a maximum of 7 alphanumerics where the first character is alphabetic.

5. when referencing an array within your program, any subscript may be an expression (whose value cannot exceed the integer in the DCL) -- for example:

```
► DCL  A(10,10)
► I = 4
► J = 5
► Y = A(J,I + J)
```

6. initializing an array.

* TO BE IMPLEMENTED

1. a 7 alphanumeric maximum.

2. the first character must be alphabetic.

3. a colon must immediately follow a label.

C. Dollar Sign "$"

1. immediately precedes specific variables (not labels).

2. entered by the user while typing a program.

3. when it precedes a variable on the left side of any equal sign, this indicates that the variable and its value are to be displayed after execution (i.e. EXEC). Therefore, it is essential to include a dollar sign in front of all variables you wish computed; otherwise, no results will be visible upon completion.

4. when it precedes a variable on the right side of the equal sign, the computer is to request a value during execution rather than to assign a value at the offset. Therefore, data values may vary with repeated EXEC statements and several computed results will be easily obtainable.

5. for example:

```
► $A = $B + 7
► EXEC
":B="
► 2
"A = 9"
► EXEC
":B="
► 5
"A = 12"
```

V. USEFUL PROCEDURES

A. Writing a Statement

The computer will provide a 4 digit sequence number

("0010") when ready for your entry. Any "legal" statement may then be entered. Those statements considered "legal" at the offset include: assignment statements, conditional statements ("IF"), repeat statements ("00"), and declare statements ("DCL"). All other commands would be legal as subsequent entries. As a rule of thumb, each statement should begin to the immediate right of the start symbol (►) which appears whenever a user response is required input. A blank should delimit a keyword from a variable.

All assignment statements are recommended as your leading entries. The statement stands as follows:

[label:] variable = expression where the equal sign indicates a command and the expression includes constants and/or functions and/or operations. Only one variable may be assigned per statement; you cannot assign values to two variables on one line by distinguishing them with parentheses and the "AND" sign.

As a helpful reminder, type a dollar sign in front of each variable you wish displayed after execution.

B. Inserting or replacing a Statement

Precede the new statement with your own sequence number. If a statement already exists with this sequence number, the new statement will replace the old; otherwise, the new will be inserted sequentially. For example:

```
0010
    ► $A = 10
0020
    ► $C - B + A
0030
    ► $D = SQRT (C) + A - B
0040
    ► 15 $B = 20
0040
    ► LIST 10, 30
$A = 10
$B = 20
$C - B + A
$D = SQRT (C) + A - B
  0040
  ►
```

Table of Contents

* to be implemented

I. Description

TSAR is a general purpose system which allows rapid remote querying of previously created direct access files and outputting of information from these files in a variety of formats and on a number of devices. *Every user query consists of one or more commands, each followed by information which modifies that command and makes it meaningful. There are two types of commands: search and output; therefore, the two types of statements include either command followed by an expression or a fieldname.

Though similar in many respects to TORQUE, TSAR is faster, more flexible, and incorporates several new capabilities which the user should find helpful. In addition, TSAR can be used with already existing TORQUE (slightly modified FFS) files, so that present TORQUE users need not recreate their files. AEGIS users, after running their files against an existing periodic conversion program, may also use TSAR.

II. Search Commands

A Search command requires TSAR to examine a file and designate certain records as "hits." TSAR must have certain conditional information upon which to make this decision; that is, certain expressions or conditions must follow a search command, the satisfaction of which constitutes labeling any given record as a "hit."

Only one of the three following search commands may appear in any given query:

A. IF - causes an entire file or an indexed subset to be scanned for hits. The information in these hit records is then available for further querying or for outputting. It is advisable to use IF when information about a certain category of records is needed.

*to be implemented

B. FIND - causes a file to be scanned for the first hit record. The information in this single record may then be outputted.

C. COUNT - causes an entire file to be scanned for hits and produces a tally of the select records. A COUNT statement cannot be compounded with any OUTPUT statement; therefore, it alone is a query. COUNT is somewhat faster than IF but only a tally can result and no further investigation of these records is accorded to the user.

D. Syntax

Algebra - like expressions follow a search command. More than one expression may follow a single command if each expression is separated by an "&" (AND) or "|" (OR). Each of the legal expressions takes one of the following forms:

fieldname-operator-fieldname
fieldname-operator-literal
fieldname-operator-function
fieldname-operator-multiple field
fieldname-operator-range

1. Any fieldname in the file can legally be investigated.

2. The four valid operators include:

= (equal)
> (greater than)
< (less than)
¬ (not equal)

None can be compounded.

3. A literal is an alphanumeric data item. It must be delimited by quotes only if blanks are imbedded, special characters are included, or if an alphanumeric begins with a number - e.g., NAME = 99,NAME = S409, NUMBER = '44-S'TE-RUE'

*4. A function consists of a function name followed by one or more fieldnames or literals which are delimited by parentheses and separated by commas. Three probable functions include: SCAN, AVERAGE,SUM.

5. A group of fieldnames, literals, and functions can be enclosed within parentheses and separated by "&" (AND) or "|" (OR); therefore, all refer to the same field - e.g.,

IF NAME = (99 | 'HARRIS,JOHN')

6. Range is a pair of numeric literals presented in ascending order and separated by a hyphen. The operator, in this case, must be an equal sign - e.g.,

FIND DAY = 13-31

7. Parentheses may be inserted freely, and their use is encouraged if any doubt might possibly arise.

E. Output and Subsequent Reference

After entering an IF or COUNT, a tally of the hit records appears. Since FIND scans for only one record, the user is notified when it is found.

*to be implemented

Further investigation of the same hit records is assumed only after an IF or FIND command (i.e., any subsequent statement will reference the same records). Refer to "REFINEMENT of HIT RECORDS."

III. Output Commands

Output commands require TSAR to reproduce information from direct access file records, in either formatted or free form, on a specified output device.

A. LIST - causes the desired information to be displayed on the user's terminal (e.g., IBM/2260, IBM/2741) in a column format.

B. DISPLAY - same as LIST except, when displayed, each data item is preceded by the fieldname.

*C. PUT - causes a new direct access data set to be created, containing data items accumulated from the existing file (i.e., to create a file of your hits).

*D. PRINT - causes the requested information to be reproduced as hard copy. A batch print job is scheduled with your request taking highest priority over the rest of the batch. The PRINT command should be used sparingly since it can extensively tie up a printer.

E. Syntax

The output command is followed by either a list of fieldnames or the keyword RECORD.*

1. In a command with multiple entries, each fieldname is separated by a comma or a blank.

*to be implemented

2. The field is singled out of a record.

*3. RECORD is a request to output the entire record.

F. Output

1. After a LIST command, requested fieldnames appear at the beginning of each listing to form a title line, and data items from each hit appear one under the other. If a record does not contain the data, "NONE" will appear in its place.

2. The DISPLAY output is a keyword sequential listing of all the requested data in the form: fieldname = 'one data item.' A number appearing in parentheses after the fieldname signifies a periodic field where the extent of the periodicity is designated by the number itself. If a record does not contain a specific field, the pound sign (#) replaces the data item. Refer to "SPECIAL FEATURES, POUND SIGN" for further details.

3. Refer to "SPECIAL FEATURES, REFINEMENT of HIT RECORDS" for a discussion on subsequent reference after execution of an Output command.

*to be implemented

IV. Special Features

A. Pound Sign "#"

1. As Input in a Search Command, "No-Care Character" allows the user to issue a search when he is concerned only with locating certain positions in a data item, therefore, ignoring the rest. It also allows him to search when unsure of the spelling of a data item.

   For example, if the user is interrogating a personnel file for a person whose last name is either LANE or LANG, he could enter the query

   ```
   IF  NAME = LAN#
   ```

   and TSAR would search only on the characters LAN. Therefore, names such as LANE, LANG, LANSMAN, LAND, et cetera would be tallied. Similarly, if the user wanted to search an inventory file for a certain type of part, he might issue

   ```
   IF  PART = 'M-###-970-4312-#'.
   ```

   The pound sign can only be used on literals which are not strictly numeric fields. (The RANGE function is best suited to that case.)

2. As Output of a DISPLAY Command -

   a. If a pound sign appears immediately after the fieldname, the field was defined as numeric when the file was being created, - e.g., DATE# = '111345'

   b. When a pound sign appears as a data item, the record does not contain the requested field. - e.g., BIRTHPLACE = '#'

B. Refinement: the ability to investigate a group of previously hit records. Refinement cannot always be assumed since not all commands return a group of hit records.

1. An initial query is one which investigates the file in its entirety. When the file has first been opened and "READY" appears, the statement(s) you enter is an initial query. If this statement consists only of Output commands, no records will be distinguished as hits; therefore, the subsequent entry could also be called an initial query. In fact, "hits" are available only when the first entry contains an IF of FIND statement. **After the COUNT command, "READY" reappears, and the next sequential command again references the entire file (i.e., an initial query).

2. An obvious indication of the subsequent reference is the program message. If "READY" appears, your entry will reference the entire file; and if "ANY FURTHER INSTRUCTIONS?" appears, you are working with a subset of hits.

3. When refinement is assumed, the user redirects a query to the file in toto by entering "N" or "NO" or "NEW Q" in reply to "ANY FURTHER INSTRUCTIONS?". Therefore "READY' should reappear.

4. Refer to the example.

*C. Indices

TSAR allows the user to create a file of hit records for use immediately or at a future time. The index is created as follows: after a query containing an IF search command is completely executed by TSAR, the question

ANY FURTHER INSTRUCTIONS?

appears. At this point, if the user wishes

*to be implemented
** The FIND command only permits subsequent outputting, no further searches.

to save an index to the hits encountered in the latest search he simply enters

INDEX NAME

where "name" is the name the user wishes to give to this subfile. The name must be unique only with respect to other subfiles of the same file. After the subfile has been created the message

PROCEED

will appear. The user may then continue with refinements, outputting, or a new query.

After an index has been created, the user may search his file on this index by simply entering the command at initial query time, i.e., when READY appears. As soon as TSAR has verified that the "name" index exists for this file, it responds with another READY, and the user may then enter his query.

When the user no longer needs an index he simply enters

DELETE name

and the "name" index is flagged for deletion.

1. An index may be created only by executing a query which contains an IF command and which produces some hits (I.E. > Ø).

2. The user is urged to use the indexing capability sparingly, and to delete subfiles as soon as they are no longer needed, since these subfiles can use up large amounts of disk space if created indiscriminately.

D. Special Commands

1. "END" - causes the current phase of execution (e.g., a search or a list) to come to an immediate stop, and the next phase to begin. There is no loss of information when this instruction is

entered. (For example, if a search is ENDed before normal completion, the hits obtained thus far are available for refinement or outputting.)

2. FINISH - causes TSAR to terminate.

3. NEWFILE - causes the current execution to end, and a new file to be requested for querying. The same security procedure must be passed before the new file is made available.

E. Diagnostic Messages

When the TSAR syntax analyzer detects an error in a user query, it pinpoints its location and gives a brief explanation of the error on the line following the query. The user is requested to reenter his query. Only after an error-free query is entered can any processing take place.

V. Useful Procedures

A. Entering a Query

A "statement" is defined as a command followed by a fieldname (with or without conditional information depending upon the "type" of command). A query consists of one or more statements entered simultaneously. The sequence of command statements within a single query has no bearing on the results. A query is entered after "READY" or "ANY OTHER INSTRUCTIONS?" and is limited to 160 characters and to one search statement.

When the file has been opened and "READY" appears, your first query always implores the complete file. An IF or FIND would enable "refinement;" all others would require another initial query to follow.

*B. Naming and Classifying a New Data Set with "PUT"

*to be implemented

VI. Sample

```
                                                                    Line
ENTER PROGRAM NAME                                                   1
   ⊳ TSAR                                                            2
TSAR LOADED                                                          3
THE TSAR IS READY                                                    4
ENTER FILENAME                                                       5
   ⊳ PERSONNEL                                                       6
KEYWORD =                                                            7
   ⊳ BLUEBIRD                                                        8
PEOPLE FILE IS OPEN FOR QUERIES                                      9
READY                                                               10
   ⊳ COUNT SEX = M & GRADE > 9                                      11
47 HITS ENCOUNTERED                                                 12
READY                                                               13
   ⊳ LIST NAME, DEGREE MAJOR IF SEX = F & GRADE > 10                14
6 HITS ENCOUNTERED                                                  15
NAME                DEGREE          MAJOR                           16
ATWELL,SUE          BA              HIST                            17
HATFIELD,ROBIN      NONE            NONE                            18
PERCY,KAY           BS              ECON                            19
                    MS              ECON                            20
END OF HITS                                                         21
ANY FURTHER INSTRUCTIONS?                                           22
   ⊳ IF  GRADE > 12                                                 23
```

```
                                                          Line
2 HITS ENCOUNTERED                                         24
ANY FURTHER INSTRUCTIONS?                                  25
  ⊳ LIST  NAME,GRADE                                       26
NAME              GRADE                                    27
ATWELL,SUE          13                                     28
PERCY,KAY           14                                     29
END OF HITS                                                30
ANY FURTHER INSTRUCTIONS?                                  31
  ⊳ DISPLAY NAME SKILL                                     32
DISPLAY FORMAT                                             33
NAME = 'ATWELL,SUE'                                        34
SKILL (1) = 'COMPUTERS'                                    35
SKILL (2) = 'TYPING'                                       36
NAME = 'PERCY,KAY'                                         37
SKILL = '#'                                                38
END OF HITS                                                39
ANY FURTHER INSTRUCTIONS?                                  40
  ⊳ NEW Q                                                  41
READY                                                      42
  ⊳ FIND  NAME = (BROWN|SMITH|DOE)                         43
BEGIN SEARCH                                               44
RECORD FOUND                                               45
ANY FURTHER INSTRUCTIONS?                                  46
```

	Line
▶ NEWFILE	47
ENTER FILENAME	48
(et cetera)	

APPENDIX VII

LINUS

Table of Contents

I. GENERAL

The LINUS program provides a method of creating files, maintaining files, and retrieving information on a terminal. Files created through LINUS can be used by BATCH and SOLVE.

FILE CREATION

One of the functions of LINUS is to provide file-creating capabilities, besides file-handling. Files created with the LINUS program, referred to as LINUS files, can be investigated by LINUS commands. SOLVE files can also be queried with LINUS.

Each file in one directory must have a unique identifier as a filename. The identifier can be 1 to 8 alphanumerics where the first character is alphabetic. If a new file is created with a filename duplicating an existing file in the same directory, an error message is displayed, and a different filename is requested.

A file can be built on a terminal or run through batch processing:

1. To create a file on the terminal, the user calls the LINUS program, names his new file, and switches the program to the INPUT mode to accept lines of type as records. The new file automatically resides in a directory if one was identified during the log on procedure. If no directory was identified, the new file lies in the public area. A utility program is available to copy the file into a specific directory.

2. To create a LINUS file in the batch, the file is on computer cards preceded by the card:

```
/**filename
```

where "filename" is all that eventually will be required to identify the file. The last card in the deck must be a slash-asterisk:

```
/*
```

The file will appear in the public area and can be moved to a directory by the utility program.

All files are stored on disk and formatted into record. One line of type (or one computer card) constitutes one record. All alphanumerics and special symbols are accepted data.

FILE HANDLING AND INFORMATION RETRIEVAL

Files can be maintained and interrogated with LINUS commands in the EDIT mode.[1] Each command is individually executed and affects only one record at a time. Exactly which record will be affected depends upon the location of a pointer. Whenever the EDIT mode begins, the pointer is above the first record in the file. It points to different records, moving down the file, after execution of most editing commands.

The record currently marked by the pointer is identified by its line number displayed on the IBM/2260 Display Terminal at the upper right hand corner. This line number is maintained internally for the IBM/2741 Communications Terminal and can be obtained by entering "STATUS".

The System limits the number of times any terminal user can INSERT, REPLACE, CHANGE, or DELETE records from an existing file - in the EDIT mode. The maximum generation is set at 50 times, and the latest count is displayed on the terminal whenever the file is opened.

In the INPUT mode, the system has set a maximum to the number of times a terminal user can add to an existing file, distinct from the "maximum generation". The user can add an unlimited number of records at each INPUT session; however, he is restricted to 10 INPUT sessions.

II. COMMANDS

A. Description

The user communicates with the program through commands entered on the terminal. One blank lies between the command and the operand.

---

[1] in contrast to the INPUT mode which allows creating a new file or adding records to an old one.

1. INPUT - signals the System that the user wants to create a file or add to an existing one. It is a mode; the System expects input in the file one line (record) at a time without interruption or response from the program.

   While in the EDIT mode, the user can switch to INPUT; however, the converse is NOT true.

2. EDIT - allows the user to edit a LINUS file and indicates editing commands will follow.

3. NEWFILE - allows the user to change files without terminating the LINUS program. The current file is unloaded, and the user calls up a new one.

4. FINISH - terminates the LINUS program.

5. LIST - a LINUS file is displayed on the terminal.

6. PURGE - causes a file to be erased from the System. Use is restricted to one's own file. A purge must be affirmed (enter "YES" or "Y") in order to execute the command.

7. COPY - duplicates the file on a disk. The duplicate always appears in the same area (i.e., Directory or Public Area) as the original. Originals and duplicates in the public area can be accessed by all terminals.

   A new filename and keyword can be assigned to the copy, besides setting the maximum generation.

8. LOCATE xxx...

   This command moves the pointer down to the record which contains the given combination of characters. Only enough of a character string need be specified to identify it in a record. The second time the given character string appears in the file can be located by giving the LOCATE request twice.

LOCATE searches the file for the requested character string and prints the entire record in which it is contained. The record containing the character string is displayed in its entirety.

If the LOCATE request fails to find the character string, a message is printed, and the pointer is situated beyond the last record in the file.

9. FIND xxx...

searches the file for the first record which begins with the given character string. Therefore, the character string identifies the first few characters or the first word in a record.

This command moves the pointer down to the first record which begins with the given character string.

10. TOP - causes the pointer to move above the first record in the file.

11. BOTTOM - moves the pointer beyond the last record in the file and switches control to the INPUT mode. The command assumes the user wishes to add records to the file.

12. INSERT xxx...

The user can insert a new record while in the EDIT mode. The character string represents an entire record to be inserted after the record designated by the pointer. The new record can be INSERTed between two previously adjoining records. The pointer is set to the newly inserted record.

A maximum of 77 characters can follow the command INSERT; however, the letter "I" suffices as an abbreviation to insert 80 characters.

13. DELETE n

Beginning with the record at which the pointer is set, "n" records are deleted.

The pointer is situated at the last deleted record. The number "n" can never be less than 1.

14. REPLACE xxx...

The given character string replaces the record currently designated by the pointer. A maximum of 77 characters can follow the command REPLACE; however, the letter "R" will suffice in order to enter up to 80 characters.

15. CHANGE /xx/yy/ n G

In the record designated by the pointer, one string of characters is exchanged for another. The record is changed from "xx" to "yy", accepting any alphanumeric or special symbol except slash (/). Each string can vary in length, distinct from the other, and the program will space the characters accordingly -- all characters in the record are moved to the right if more are inserted than originally appeared, and all are moved left if less are inserted.

The CHANGE command affects the first "xx" in the record. If "G" is included with the command, every "xx" in one record will be exchanged for "yy". If a number "n" is indicated, the CHANGE affects "n" number of records beginning with the one distinguished by the pointer. If "G" is inserted, "n" must be at least "1".

16. PRINT n

Beginning with the record designated by the pointer, "n" records (lines) are displayed. The pointer is then set to the last printed record.

If the pointer is set after the last record, the PRINT begins at the top of the file. The "n" must be greater than zero and can exceed the actual number of records, in which case printing will stop at the bottom of the file.

The character string "xxx..." is appended to the line designated by the pointer. The insertion begins immediately after the last character of the record; there must be enough space at the end of the line to allow for the insertion, since the line ends after the eightieth (80) character.

The pointer does not move down after execution of APPEND.

18. NEXT n

The pointer is moved down "n" records (lines) starting with its current location.

19. IGNORE

In the INPUT mode, IGNORE deletes the immediately preceding line of data ONLY while the user is adding to an existing file.

20. VERIFY - determines whether the program will respond to certain commands.

a. VERIFY ON

the default case. The program replies to the following commands:

1) INPUT and EDIT
2) FIND
3) LOCATE
4) CHANGE
5) DELETE

b. VERIFY OFF

the above commands are not visually verified.

c. VERIFY HALF

Of the above list of commands, only LOCATE and FIND commands will be verified. CHANGE, DELETE, INPUT, and EDIT are suppressed.

21. END - terminates "LIST"

B. Chart of Commands and Abbreviations

Command	Abbreviation
CHANGE	C
INSERT	I
DELETE	D
PRINT	P
LOCATE	L
FIND	F
REPLACE	R
VERIFY	V
APPEND	A
TOP	T
BOTTOM	B
NEXT	N
FINISH	none
LIST	L
NEWFILE	none
PURGE	none
COPY	none
INPUT	I
EDIT	E
END	none

III. SPECIAL FEATURES

A. /*

The slash-asterisk signals the end of input data and the end of the edit mode without terminating LINUS. It can be used to abnormally terminate COPY, PURGE, and LIST. The /* is entered exclusively on a line.

B. *filename

When the program requests a filename, "*" preceding a name indicates a new file is to be created with the given name. Later investigation of the file requires the filename only, deleting the asterisk. If a duplication of names arises, an error message is printed and a different name must be entered.

C. The INPUT mode (A, F, or U) describes the data format of the file.
(1)

a. "F" - The file will be a FORTRAN program. Each record will be formatted as a FORTRAN statement.

b. "U" - Data will not be formatted by the System; it will be entered on disks exactly as formatted by the user.

c. "A" - The file will be an Assembly Language program. Each record will be a source statement composed of a label (optional), operation, operands and comments (optional).

The source statement must be formatted as follows:

Label - the first group of characters represent the label and are entered to the immediate right of the Start Symbol. If no label is included, at least one blank column must be allowed after the Start Symbol before entering the operation.

(1) to be implemented

Operation - Assuming a label exists the second group of characters would be the operation. Otherwise, the operation would be the first group of characters preceded by one or more blanks to the right of the Start Symbol.

Operands - the subsequent group of characters separated from the operation by at least one blank, comprises the operands.

Comment - A string of characters can be included as a comment as long as one or more blanks separate the string from the operands.

When the user later queries the file and displays (or prints) a record, the labels start in column 1, the operation in column 10, the operands in column 18, and the comment in column 40. A string extending beyond column 72 is automatically continued beginning in column 16.

For example:

Input Source Statement	Display of Source Statement			
	col. 1	col. 10	col. 18	col. 40
LAB2 LR 6,8	LAB2	LR	6,8	
BCR 6,8		BCR	6,8	
XR 6,8		XR	6,8	
DR 6,8 DIVIDE IF> Ø		DR	6,8	DIVIDE IF> Ø

IV. USEFUL PROCEDURES

A. Creating a file on the terminal

To create a LINUS file on the terminal, the following steps are required:

1. Assuming the log-on procedure complete, the LINUS program is loaded into the computer. If a directory is indicated during LOG ON, the new file will reside in that directory.

2. The new file is given a name when the filename is requested by prefixing the name with an asterisk (*).

3. The user can require a password for this file.

4. The Input Format is determined by A (assembly), F (FORTRAN), or U (Undefined).

5. Input data is typed and entered one line per record.

6. The line (records) tally is displayed on the IBM/2260 Display Terminal at the upper right hand corner. A tally is maintained internally for the IBM/2741 Communications Terminal and can be obtained by entering "STATUS".

The same number identifies the line indexed by the pointer while interrogating the file.

7. A slash-asterisk (/*), appearing exclusively on a line, signals the end of input.

B. Switch from EDIT to INPUT mode

Records can be added to a LINUS or SOLVE file. In the EDIT mode on a new line, the user skips a space between the start symbol and the cursor and depresses the SHIFT and ENTER keys; this step switches the System from the EDIT to the INPUT mode. Input records can be entered a line at a time.

If the input format was described as "A" (Assembly), switch back to EDIT from INPUT mode by skipping a space on a new line and depressing the SHIFT and ENTER keys.

"BOTTOM" also switches control to the INPUT mode after it moves the pointer to the bottom of the file.

Refer to the fourth (4th) paragraph of "FILE HANDLING AND INFORMATION RETRIEVAL" for additional information.

V. Sample

```
SPECIFY DIRECTORY (OR 'NONE')
► DIRECT2
ENTER PASSWORD
► APPLE
      '
      '
      '
      '
ENTER PROGRAM NAME
► LINUS
LINUS LOADED
**LINUS (TYPSET) ONLINE**
ENTER FILENAME
► PERSONNEL
**LINUS PROGRAM**PROCESSING FILE--PERSONNEL
ENTER OPERATION-(INPUT,EDIT,NEWFILE,LIST,FINISH)
► E
EDITING FILE--PERSONNEL
LATEST GENERATION IS 0002
► LOCATE JONES
0140;JOHN PAUL JONES;NOV 13, 1922;PROGRAMMER;B.A.
► L JONES
0150;JOHN S. JONES;MAY 27, 1937;FIN.;M.B.A.
► TOP
► VERIFY OFF
► C /B.A./A.B./ 555 G
► L JONES
0140; JOHN PAUL JONES;NOV 13, 1922;PROGRAMMER;A.B.
► INSERT 0141;JOHN R. JONES;JAN 3, 1945;E. ENG;B.S.
► V ON
► F 0150
0150;JOHN S. JONES;MAY 27, 1937;FIN.;M.A.B.
► APPEND ,B.B.A.;
► PRINT 1
0150;JOHN S. JONES;MAY 27, 1937;FIN.;M.A.B.,B.B.A.;
► /*
LINUS PROGRAM**PROCESSING FILE--PERSONNEL
ENTER OPERATION-(INPUT,EDIT,NEWFILE,LIST,FINISH)
► FINISH
```

(each line preceded by Start Symbol represents a user entry.)

TRUMP

TRUMP furnishes commands which locate, display, and change fixed fields in TORQUE files via the terminal.

I. Commands

A. SEARCH - causes a file to be scanned for the first hit record. The first record containing the data value is designated as a "hit." This record is then available for DISPLAY and SET.

The instruction (composed of command and body) comprises the following format:

SEARCH fieldname = 'data value'
command body

One "data value" is requested per instruction and is delimited by single quotes. The user must be familiar with the form and order of the data. For example, if a file reads "JONES, HARRY" then the query requests "JONES, HARRY," not "HARRY JONES." Only enough characters need be entered to uniquely identify the data value; for example, SEARCH NAME = 'JONES, H' will identify "JONES, HARRY" rather than "JONES, RALPH."

A pointer distinguishes a single hit record. When a file has been initially opened, the pointer is above the first record. The SEARCH command moves the pointer down the file until a hit is made. No hits relocate the pointer on top of the file. The pointer determines which record will be affected by the subsequent DISPLAY or SET command. The pointer only moves after a SEARCH command.

B. DISPLAY - displays the data values, preceded by the fieldnames, on the scope. A record must have been designated as a "hit" by a previous SEARCH instruction.

The fields to be displayed are listed after the command:

DISPLAY $field_1$, $field_2$, . . .

A single field or a string of fields can be listed. If a string of fields exceeds the 80 character maximum for the line,

the last complete field on the line should be followed by a comma. That line should be entered in the computer. After the System has completed its display of the fields and data values, the user should continue to enter the remaining fields. The end of the string is indicated by typing a blank space and by entering the data in the computer.

The DISPLAY command can be entered without listing any fields; the system would display the entire record which was hit in the preceding SEARCH instruction.

C. SET - changes fixed fields in a hit record.

SET $field_1$ = 'data $value_1$',$field_2$ = 'data $value_2$',....

A record must have been designated as a hit by a prior SEARCH instruction. After a field has been SET, DISPLAY can be used to verify the change. Input must conform to the length and character specifications of the receiving field.

Several fields can be included within one SET instruction. If the string of fields exceeds the 80 character maximum for the line, the last "$field_n$ = 'data $value_n$'" is succeeded by a comma, and the line is entered in the computer. The comma signals the system that more fields are to be changed. The user merely continues to list the additional fields and data values.

D. TOP - relocates the pointer at the beginning of the file. The SEARCH command moves the pointer in front of a "hit" record. (Any subsequent DISPLAY or SET instructions would apply to the one hit record.) In order to SEARCH the entire file, it is necessary to start with the pointer at the TOP of the file.

E. FINISH - terminates TRUMP

II. Sample

```
ENTER FILENAME
  ➢ PEOPLE
ENTER KEY
  ➢ MILLIONS
FILE OPENED, CONTINUE ENTER COMMAND
  ➢ SEARCH NAME ='JONES'
```

```
FOUND RECORD
ENTER COMMAND
  ⊳ DISPLAY
NAME = 'JONES, HARRY'
NUMBER = 'ØØ7ØØ'
BIRTHDATE = '111345'
EOD = 'Ø92766'
GRADE = 'Ø9'
DEGREE = 'BA'
ENTER COMMAND
  ⊳ SET NAME ='JONES, HAROLD Q.', BIRTHDATE = 111346
ENTER COMMAND
  ⊳ DISPLAY NAME, BIRTHDATE, NUMBER
NAME = 'JONES, HAROLD Q.'
NUMBER = 'ØØ7ØØ'
BIRTHDATE = '111346'
ENTER COMMAND
  ⊳ TOP
ENTER COMMAND
  ⊳ SEARCH NAME ='CABLE'
FOUND RECORD
ENTER COMMAND
  ⊳ DISPLAY NUMBER, NAME, BIRTHDATE,
NUMBER = 'ØØ6Ø3'
NAME = 'CABLE, GLARK'
BIRTHDATE = 'Ø31339'
CONTINUE
  ⊳ EOD, DEGREE
EOD = '1Ø2668'
DEGREE = 'MA'
ENTER COMMAND
  ⊳ FINISH
END OF TRUMP
```


Approved For Release 2000/05/08 : CIA-RDP78-03948A000100080002-8