REVISIONS				
REV.	DESCRIPTION	CHK	DATE	APPROVED

NO. REQ.	DRAWING NO.	DESCRIPTION	REFERENCE DESIGNATION	ITEM NO.

MATERIAL LIST

NOTES UNLESS SPECIFIED

1. TOLERANCES
   .XX ±.030 ANGULAR
   .XXX ±.010 ±½°
2. BREAK ALL SHARP EDGES .010 APPROX.
3. MACH. SURFACES √
4. ALL DIM. IN INCHES

DRAWN	D. G. Pengery	6/4/68
CHECK	[illegible]	6/4/68
APPR.	E R Shifflett	6/4/68

SDS
SCIENTIFIC DATA SYSTEMS
SANTA MONICA, CALIFORNIA

TITLE

UNIVERSAL TIME-SHARING SYSTEM (UTS)
FUNCTIONAL SPECIFICATION

MODEL NO.

NEXT ASSY.

SIZE A

DWG. NO. 702489

CHANGE LETTER

SCALE

DO NOT SCALE DRAWING

SHEET 1 OF 196

PRELIMINARY

UNIVERSAL TIME-SHARING SYSTEM (UTS)

FUNCTIONAL SPECIFICATION

BY

B. Bruffey
E. Bryan
B. Doeppel
J. Smith

OVERALL TABLE OF CONTENTS

I. OVERALL SUMMARY OF UTS

A. Introduction

UTS is a time-shared computing service consisting of a central computer complex and a collection of remote teletype and other typewriter-like terminals connected to the central complex by full duplex communication lines. UTS gives its users access to all the programming services of the Batch Processing Monitor (BPM), including symbiont and real-time services. These are augmented by tools specifically tailored for remote-terminal users engaged in the on-line creation, modification, debugging and use of programs. The on-line entry of jobs for batched service, in the form of BPM control-card programs, is permitted. Such programs may be composed, filed away and entered in the Batch job stream from the terminal, and on-line users may query UTS about the status of such jobs.

UTS is son, sibling, and parent to BPM, and will be derived from that system by a set of specific changes and additions. For the first version of UTS, these fall into three classes.

1. Processors and associated languages primarily related to on-line users.

a. An executive processor and language (TEL) for handling requests from on-line users. To such users UTS appears to be a single active agent that responds to commands couched in TEL. Most commonplace activities associated with FORTRAN and assembly language programming can be carried out directly in TEL: file management, compilation and assembly, loading, execution and debugging. Lengthier or more involved operations and activities associated with other programming languages must be carried out by requesting the services of a sub-system of UTS. Each sub-system acts as an independent, active sub-agent of UTS, accepting requests in a language tailored to its job and to the expected profile and bents of its users.

b. Simple, fast turn-around compilers and assemblers for batch-compatible subsets of FORTRAN IV and Meta-Symbol (FORTRAN E and Extended Symbol), and a low-cost, one-pass loader (like LOPE). These are available on-line and in Batch.

c. A compile-and-go processor for the extended Basic language, which includes provisions for direct operations on arrays; an on-line sub-system for creating, modifying, running and debugging Basic programs.

d. An editing processor and language for the on-line creation, modification and management of programs and other bodies of text.

e. Debugging processors and languages appropriate to FORTRAN debugging (FDP) and to assembly language debugging (Delta). These processors are always at hand for the on-line user (who can call on them at any stage of execution), and are ideal for carrying out parameter studies.

f. Utility processors and languages for: a) managing files of information and transmitting information between different media (PCL); b) combining and recombining compiled and assembled object programs (Link, Symcon).

2. Distinct bodies of code that regulate and provide information about the activities of UTS and its users. These include routines for: a) scheduling activities; b) managing time and storage; c) measuring and displaying the cumulative and individual behavior of UTS and its users; d) handling information passing to and from remote terminals on an asynchronous basis; e) detecting and recovering from errors.

3. Changes and fixes to BPM and its component processors. These include: a) modifying compilers and assemblers so that they produce information necessary for on-line debugging; b) creating versions of processors and run-time packages (and all other public routines) that are re-entrant and, therefore, capable of being shared among more than one user; c) simplifying input-output interfacing with BPM and speeding-up its file-management services; d) changing BPM and its processors so that they can deal with typewritten lines of information and files of such information produced at a terminal as readily as they now handle card images and card decks; e) fixes required to use the memory map.

B. Behavior and Responses

UTS is supposed to service real-time loads, batch loads and on-line loads simultaneously -- all without batting an eyelash. What these loads are and how they vary from installation to installation are an unknown. On the other hand, some complete statistics have been published for batch loads in aerospace and university environments and for on-line loads in time-sharing systems. These figures share one healthy attribute -- they compute, they compare, they match. These figures and their requirements in terms of UTS capacity are described in a succeeding section. Application of straight-forward queue and traffic-theory techniques to these figures shows that UTS can be designed to strike a balance between on-line and batch requirements. Although some of its facilities will be denied the

batch user and others the on-line user, the two classes of service will be complementary rather than antagonistic. Under typical loads, on-line demands will rarely overwhelm batch processing, nor will batch throughput seriously hamper on-line negotiations. The typical demands of on-line users need less than 50 ms. of processing and constitute 85% of on-line requests. For 30 users, these can be handled comfortably at costs not exceeding 8% of main-frame time. This includes the overhead costs of scheduling, time-sharing and transmitting information to and from consoles. Average delays to typical on-line requests of 30 users will exceed .2 seconds no more than 10% of the time and will exceed 4 seconds no more than .01% of the time. These figures are based on configurations matched to reasonable loads, and should not be considered totally satisfying; they are simply better than anything else on the market, except for dedicated, single-language systems. Delays of .2 seconds are noticeable, particularly to people using processors that maintain intra-line dialogues with their users, when delays can not be blanketed by the carrier-return times associated with typing requests. Although average delays will be just less than .2 seconds, variations will occur frequently. However, users will hardly ever have to wait more than 3 or 4 seconds for a response to a typical request, nor should they observe any halting or stuttering behavior during output situations.

For 60 users, main-frame degradation is doubled, but the distribution of response times remains substantially the same -- delays greater than .2 seconds still occuring about 10% of the time. BPM itself makes demands on main-frame time for symbiont, input-output, and file-management services, for control-card interpretation and simply tooling up to do a batch job, and for processor and monitor overlays. This service cost is 2 to 3 times greater than that required to service the typical demands of 30 on-line users. What is left of main-frame capacity (70% for 30 on-line users) must be devoted to "computing" -- processing batch programs and compute-bound on-line users. The more on-line users and/or the more compute-bound on-line users, the greater the possible impact on raw batch computing power.

C. Requirements

In terms of input/output throughput, the 7202-4 RAD is inadequate if used alone operating at 150% of capacity for typical batch and on-line loads. Under such circumstances,

batch user and others the on-line user, the two classes of service will be complementary rather than antagonistic. Under typical loads, on-line demands will rarely overwhelm batch processing, nor will batch throughput seriously hamper on-line negotiations. The typical demands of on-line users need less than 50 ms. of processing and constitute 85% of on-line requests. For 30 users, these can be handled comfortably at costs not exceeding 8% of main-frame time. This includes the overhead costs of scheduling, time-sharing and transmitting information to and from consoles. Average delays to typical on-line requests of 30 users will exceed .2 seconds no more than 10% of the time and will exceed 4 seconds no more than .01% of the time. These figures are based on configurations matched to reasonable loads, and should not be considered totally satisfying; they are simply better than anything else on the market, except for dedicated, single-language systems. Delays of .2 seconds are noticeable, particularly to people using processors that maintain intra-line dialogues with their users, when delays can not be blanketed by the carrier-return times associated with typing requests. Although average delays will be just less than .2 seconds, variations will occur frequently. However, users will hardly ever have to wait more than 3 or 4 seconds for a response to a typical request, nor should they observe any halting or stuttering behavior during output situations.

For 60 users, main-frame degradation is doubled, but the distribution of response times remains substantially the same -- delays greater than .2 seconds still occuring about 10% of the time. BPM itself makes demands on main-frame time for symbiont, input-output, and file-management services, for control-card interpretation and simply tooling up to do a batch job, and for processor and monitor overlays. This service cost is 2 to 3 times greater than that required to service the typical demands of 30 on-line users. What is left of main-frame capacity (70% for 30 on-line users) must be devoted to "computing" -- processing batch programs and compute-bound on-line users. The more on-line users and/or the more compute-bound on-line users, the greater the possible impact on raw batch computing power.

C. Requirements

In terms of input/output throughput, the 7202-4 RAD is inadequate if used alone operating at 150% of capacity for typical batch and on-line loads. Under such circumstances,

everyone waits. A single 7232 is marginal, while a 7212, high speed RAD would operate at 50% capacity under typical loads -- a comfortable figure, although an extra unit dedicated solely to handling user's files may be required for many installations.

The costs and delays mentioned above can be achieved only through use of the memory map. On systems without this feature, the overhead costs of core management and time-sharing have reached 40% of CPU capacity; this is an unconscionable degradation of computing power. With the feature, uncomplicated, low overhead management and scheduling disciplines can be used, as can re-entrant processors that may easily be shared among many users.

The frequency and extent of variations from the norm are dependent on the hardware configuration chosen and on how effectively an installation can control its own loading patterns: by ad-hoc adjustment of dynamic allocation and scheduling parameters from an on-site console, by education of its user community, or by direct management fiat. UTS is meant to be a large system -- large both in terms of configuration and in terms of its ability to handle variations in load. To this end, space will be traded off to get good responses and to use CPU capacity efficiently. Core residency requirements will be approximately 16K. UTS will be designed for a minimum 64K core configuration with adjustment to 48K possible for reduced requirements.

It must be clearly understood that large transient or installation-systemic variations from the estimated loads around which UTS is built will inevitably call for a retuning of the system, no matter what the configuration. This is an operation that will be possible within reasonable, but fuzzy limits. Beyond these, installation-management techniques must be brought to bear. To test the UTS design, to predict re-designs and to allow installations to tune their own systems, UTS will devote a small portion of its time and other resources to

measuring the cumulative and individual behavior of itself and its users. Installations should be prepared to do the same; it is therefore required that each installation dedicate an on-site console to the job of displaying the results of these metering activities on a minute-by-minute basis.

D. Services and Facilities

UTS provides its users three classes of service.

1. Real-Time Service

Preemptive access to the hardware is provided for programs engaged in simulation, control and other "real-time" activities. Such programs may be permanently resident in UTS's (appropriately enlarged) core store, or may be made temporarily resident on a demand basis, at the user's option.

2. Batch Service

All facilities and processors of BPM are available. Access to, and control over, these facilities is obtained through "programs" written in the control-card language of BPM. Such control-card programs may be submitted to UTS through card readers or they may be composed, filed away and submitted on-line. In addition, the status of previously submitted batch jobs may be interrogated from remote terminals.

Although some facilities and processors are reserved solely for on-line use, while others are available only in batch, the two classes of service are complementary. Generally speaking, anything that can be done in batch can be done on-line, albeit sometimes in a curtailed manner. In particular, compilers and assemblers are compatible across the two classes of service at both source and relocatable levels:

a. processors for FORTRAN IV-H, Symbol and Extended Symbol (X Symbol) are available both on-line and in batch;

b. processors for SDS FORTRAN IV and Meta-Symbol are available in batch only;

c. programs compiled or assembled in batch can be linked with those produced on-line, and can be run and debugged on line;

d. programs compiled or assembled on-line can be linked and run in batch.

3. On-line Service

The summaries given below must be treated as such. Most details of syntax and designation

are glossed over or omitted completely; this is particularly true for sub-systems such as PCL and Delta, whose languages are highly encoded or abbreviated. The names assigned to specific languages and systems will often be used indifferently to refer to the language or to the associated system or sub-system, whichever seems appropriate in context.

a. Communicating with the User

Control of each user's keyboard will be proprietary: either the user has control for purposes of input or UTS has control while carrying out requests and for purposes of output. This holds whether the user is negotiating directly with UTS, one of its sub-systems or his own program. Who has control will be made clear to the user at all times. This is particularly necessary in the case of error reports and task completion reports. In the event of errors the user must know what the error was, who reported it and to whom he may direct any corrective actions he may wish to take. In general, the user must know three things: when he can type responses and requests; to whom he is talking; who last talked to him. These are often clear in context so long as the system adheres to some reasonable rules of behavior.

b. Terminal Executive Language and Processor (TEL)

Requests for the facilities and processors provided on-line users take the form of single-line commands and declarations in UTS's Terminal Executive Language (TEL). Most commonplace programming and accounting activities can be carried out directly in TEL. These include: a) logging-in and out; b) simple file management; c) FORTRAN IV-H compilations, and Symbol and X Symbol assemblies; d) linking and loading of relocatable programs; e) controlling the execution of programs; f) saving intermediate core status for later resumption; g) submitting batch jobs. Other classes of operations, more involved operations, and activities associated with other programming languages must be carried out by calling (in TEL) for the services of one of UTS's sub-systems.

c. Text-Editing Sub-System (EDIT)

EDIT is used to produce FORTRAN and assembly language programs, control card programs for submission to the batch queue, and other bodies of information. Each file produced under EDIT consists of a set of lines of text. Each line is uniquely numbered, and the set is ordered by increasing magnitude of the line numbers. Such files are retained on RAD storage in a format designed to expedite and facilitate their production and up-dating by EDIT and their use by other processors.

d. Peripheral and Information Control Sub-System (PCL)

PCL allows the user to move information between input-output devices and storage media: card and paper tape devices, line printers, disc files, labeled and free-form tape reels. Conversion and re-representation of data, selection of data, and record sequencing and resequencing are allowed. The processor and its language are provided both on-line and in batch. Single-line commands are used for the gross operations of copying, deleting, positioning and for other utility functions.

e. Assembly-Language Debugging (DELTA)

DELTA is specifically designed for the debugging of programs at the assembly-language level. It operates on object programs accompanied by tables of internal and global symbols used by the programs, but does not demand that such tables be at hand. With or without such tables, it recognizes machine instruction mnemonics and can assemble, on an instruction-by-instruction basis, machine language programs. Its main business, however, is to facilitate the activities of debugging.

1) The examination, insertion and modification of elements of programs: instructions, numeric values, encoded information -- data in all its representations and formats.

2) Control of execution, including the insertion of breakpoints into a program and requests for breaks on changes in elements of data.

3) Tracing execution by displaying information at designated points in a program.

4) Searching programs and data for specific elements and sub-elements.

To assist in the first activity, assemblers and compilers of UTS will include in a program's table of symbols, information about what type of data each symbol represents: symbolic instruction, decimal integers, floating point values, single and double precision values, EBCDIC encoded information, and others.

f. FORTRAN Debugging (FDP)

The language is terse, conforming in many respects to the current FORTRAN IV-H console debugging language. If program execution is started under FDP, keyboard control is passed to the user with a notification that execution of the main program is about to begin. During

execution, control reverts to the user whenever he interrupts, whenever an error occurs and whenever FDP reaches a stopping point. When the user is in control he can ask FDP to carry out execution in a variety of modes and then ask FDP to continue execution. He can also request that values assigned to identifiers be displayed, and can reassign new values.

g. Symbol-Control Sub-System (SYMCON)

SYMCON provides programmers the facilities for controlling the global symbols associated with a load module; it may be used either on-line or in batch. When relocatable object modules (ROM) are combined into a load module, the global symbols associated with the ROMs may be required to link the ROMs properly or to link the resulting load module with other ROMs and load modules. In the latter case, it may be necessary to change some of the symbols to avoid conflicts or to eliminate many of them so that the global symbols used for linking the original ROMs become internal symbols for the resulting load module. In brief, SYMCON allows programmers to link ROMs and load modules freely in the face of conflicting naming conventions.

h. Object-Program Linking (LINK)

All operations that can be performed under the LINK executive command can be performed under the sub-system. The notation and conventions for specifying the retention, deletion and merging of internal symbols remain the same. On the surface, the sub-system's main advantage over the executive command is that it allows programmers to link more modules than can be listed in a single executive command line. Its main reason for existence, however, is as a vehicle for incorporating more complicated linkages involving hierarchies of modules.

i. Sub-system for Basic Programmers (BASIC)

Under this sub-system, Basic programs may be composed, edited, executed and debugged. All the appropriate commands of the EDIT and FDP sub-systems are provided. In addition, users of BASIC can indicate an insertion or replacement by typing the desired line number ahead of the line. Basic programs are compiled directly into executable form, and the entire process of compiling and initiating execution is referred to as "running". Detailed descriptions of the sub-system's language and its responses are covered in complete functional specifications for the sub-system.

The above list constitutes a summary description of the initial UTS. Services to on-line users may be expanded in later versions to include a conversational algebraic language, a tutorial service (HELP), etc. The remainder of this specification is devoted to a detailed presentation of the items mentioned in this introductory overview. Any future services or processors will be described in detail when they are authorized and assigned by appropriate departments within SDS.

TABLE OF CONTENTS

II. PREDICTING, MEASURING, TUNING UTS

The effectiveness and quality of each class of service (batch, real-time, on-line) depend on: a) the emphasis and degree of control placed on each class by the installation; transient and systemic variations of load within each class; b) the hardware configuration chosen. Although few absolute assertions can be made, some statements about capacity and responses to typical loads can be offered with reasonable degrees of certitude. These are based on known figures for batch loads in an aerospace and a university environment, and on-line loads for several time-sharing systems comparable to UTS. The effects of such loads on standard UTS configurations are presented in succeeding sections.

A. Demands on Capacity

Figures for on-line systems show that better than 85% of on-line interactions occur infrequently and make only modest demands on the hardware; average demands, however, are much greater than typical ones. The typical on-line user can be characterized as one who is editing, debugging or otherwise interacting with programs at a leisurely rate (in terms of computer speeds), or is observing the line-by-line output of a running program that is highly output-bound by the slow speed of his terminal device. The drain on main-frame capacity for 30 typical on-line users is about 8%; for 60, about 16%. These figures include all processing time required to service requests, including disc transmissions and the transmission of information to and from the terminals. Thus, the typical on-line user does not overwhelm the batch stream, and handling such users must be considered a service of the system for which some overhead is paid. By the same token, most activities associated with BPM must be considered services of that system: symbiont and cooperative processing; control-card interpretation; input, output and file management; fielding and processing of interrupts and monitor calls. It turns out that the overhead costs for such services in BPM are two to three times those needed to handle the typical on-line situation. The remaining capacity of the hardware is dedicated to processing batch programs

and compute-bound (or average) on-line demands. The manner in which this remaining capacity is distributed can be controlled by the installation, in two distinct ways. First, ad-hoc control can be exercised directly from the on-site console, as described in the section on scheduling. Second, education and management control can be applied to the user community to insure that activities appropriate to on-line access (and to the processors provided on-line users) be carried out on-line, while those activities that are best batched be directed to the batch queue. To assist both attacks on the allocation problem, UTS will devote part of its time to measuring the cumulative and individual activities of itself and its users; these are described in the section on metering and performance measures. It is strongly suggested that installations dedicate an extra on-site terminal to the job of displaying the minute-by-minute results of this metering. To assist in the managerial approach, language processors and systems tuned to the on-line user, and to the batch user who does not need the full power of the "big" processors, may be used effectively.

B. Responses to On-Line Demands

As will be discussed in the section on scheduling, typical on-line users will be handled by a straightforward scheduling discipline. In brief, high priorities are given to servicing users whose current behavior portends a short burst of processing followed by a relatively long period of withdrawal when no service at all will be required: users who have just typed a request for service of any kind, users who are output-limited, users who are interrupting UTS or who are entering or leaving the system. The application of this discipline will, in the absence of real-time interference, result in average delays of less than 2/10 seconds for up to sixty users. Delays exceeding 2/10 seconds will be experienced 10% of the time; delays greater than 4 seconds will occur with probability .0001. Many delays will be blanketed by the time required for the typewriter carrier to return to rest point after the user has typed his request and by the time required to type a response. However, delays greater than 2/10 seconds will be felt by users who are debugging, particularly in assembly language. The system and language

provided for this activity is designed to carry on an intraline dialogue with its users, thus providing no carrier-return time for masking delays. This is contradictory, since debugging is an impatient activity that may find stuttering responses a drag; however, lengthy periods of silence (delays greater than two seconds) will be infrequent.

C. Resource Management

In order to achieve the estimates given above for main-frame degradation and for response times, it is essential that UTS manage itself in such a way as to minimize the overhead costs of time-sharing its activities among its batch and on-line users, and organize things so that it can efficiently overlap input and output with main-frame processing. At the same time, it is equally essential that the installation manage itself in such a way as to use UTS most efficiently, and thereby reduce the wide variations that are inherent in the figures given above. UTS's job is complicated by the fact that its core store is not large enough to accommodate simultaneously all possible on-line users. A secondary (High Speed RAD) storage is used to cache those users not of immediate concern, so that time-sharing overhead includes the cost of "swapping" users between core store and disc store. A broad-brush solution to UTS's problems can be characterized by some woodsy-lore precepts: a) keep enough compute bound users in core so that there is always something to do while swapping and other input/output activities are going on; b) keep enough users in core so as to reduce the probability of swapping; c) swap as little as possible. In order to even begin to effect a solution, UTS must strike some compromise in allocating resources, particularly to on-line users. In particular, core and disc storage and input/output devices that are guaranteed to real-time and batch service are de-facto not available for on-line use except by entries into the batch queue. Second, limits must be set on the amount of core storage to be allowed individual on-line users and on the amount of core storage to be given batch users (above that guaranteed); these limits will be controllable within reason from the on-site console. Heavy

use will be made of reentrant processors capable of being shared among many users and residing anywhere in core, thus effectively reducing the average user's core demands. Provisions for handling growing and contracting core requirements for users will be provided. The SIGMA 7 mapping feature is absolutely vital to UTS's operation. In the absence of such a feature, it is necessary at the very least that programs and data reside in contiguous stretches of core store. In systems without mapping features, the overhead involved in compacting, shuffling and swapping core blocks to satisfy the contiguity requirements can reach 40%. By using mappings, this overhead becomes negligible, even under conditions of high loading.

Real-time programs can, of course, bring everything else to an effective halt; such matters are best left to the individual installation. Some real-time programs -- called "resident" -- will be given dedicated core storage and input-output devices at system-generation. Core storage so guaranteed is never available for batch or on-line purposes. Other real-time programs will be given dedicated, input-output devices at system generation time, and will be granted their core requirements on a demand basis. This core is available for batch or on-line purposes until the on-site operator demands it for a "non-resident" real-time program. If released by the operator, it once again becomes generally available. Many programs commonly characterized as "real-time" ones, but who only demand interfaces with terminals, can be operated satisfactorily as an on-line user's program -- one that may be linked to more than one terminal.

Beyond the "resident" real-time guarantee, no more core is frozen than is required to satisfy the residency requirements of UTS itself -- (16K words).

No core is absolutely guaranteed batch programs. Instead, batch programs become "fixed" in core only by virtue of their preferred treatment in the queue for "computation". This may occasion minor delays averaging .2 seconds before some batch operations are carried out, and similar delays during the actual processing of some batch operations. The cumulative delays to batch will probably be equivalent to those caused by the machine operator dropping a tape once or twice during the day.

Allocation of disc resources depends on whether the high-speed RAD is used alone or in consort with a slower one for storage of user's files and system files. It is clear that the high-speed RAD can easily handle swap storage, symbiont and cooperative files as well as dedicated storage for processors and other heavily used components of the monitor.

D. Accounting

The UTS system should provide charging for a variety of usage parameters. The installation manager can assign each of these parameters a separate charge rate and can thus maintain effective control over the use of the system. The price associated with each system commodity will be a dynamic system parameter which can be varied with time of day and user priority, or with other criteria set up by the installation manager. A preliminary list of the system resources which will be separately charged includes:

(1) Central processor time
(2) Line connect time
(3) File I/O activity (e.g. number of pages)
(4) File space used
(5) Core space used
(6) Tape usage (unit holding time and transfer time)
(7) Number of on-line user interactions

An installation manager may thus adjust the usage of his system by modifying the charging structure -- for instance, by raising the price for the use of tapes in order to discourage excessive use.

Such data will be available to the user via direct console requests. At job termination time all accounting information will be output on the accounting log device.

E. Performance Measures

Ability to measure the operation of the system is particularly important during the initial debugging stages and increases in importance as the system is tuned to meet the load of the users' particular environment. These performance measures are built directly into the system as a series of counters; a given area of executive storage will be devoted to counting actions and recording times for completion of various functions. Special code in the form of counting instructions will be provided at critical points within the system to count these events. As such the recording of performance information will be on a routine-by-routine basis throughout the entire system. A user program with special executive privileges will display this information. This program will use a dedicated console to print the contents of the tables which record system performance measures. Appropriate formats and appropriate time intervals for printing will be used. Through a standard monitor feature this program is "awakened", perhaps every minute, to print the current contents of the statistical counters. This mechanism provides a relatively flexible scheme for adding new performance measures to the system and providing for their printout as the gathering of new statistics is indicated. Some items should be measured and displayed frequently, perhaps every minute: others should be measured and displayed at a longer interval -- perhaps every fifteen minutes or every hour. The display frequency will be adjustable so that operational data can be displayed more often if special tests are to be made.

Printing every minute

(1) Total number of users
(2) Inputing
(3) Outputing
(4) CPU time computing
(5) Compute time for users
(6) Compute time for batch

(7) Overhead time

(8) Idle time

(9) Unoverlapped I/O time

(10) Number of swaps

(11) Commands received from users

(12) Number of lines transmitted to users

(13) Number of file I/O actions

(14) Tape errors

(15) Disc errors

(16) Console parity errors

(17) Other errors

Longer-Term Measurements

(1) Distribution of compute time per user interaction

(2) Distribution of time between user interactions

(3) Total compute time per session

(4) Distribution of the size of the programs

(5) Distribution of console input/output frequency on a line basis

(6) Recording of I/O rates to the consoles

(7) Recording of use frequency by processor

(8) Distribution of the number of users simultaneously in core

(9) Distribution of response time to user requests

(10) Restricting all measurements to users of single processor

Continuous monitoring of these quantities and of other central parameters will permit installations to tune UTS to meet local fluctuations in loads, provides installations and SDS an experimental, long-term approach to tuning UTS for general environments, and provides a mechanism for testing the initial design and predicting redesigns.

Adjustable parameters of the system can be adjusted from the on-site operators' console and their effects observed on the console dedicated to displaying

"performance measures". These include:

(1) compute time in short and long time-slice quanta

(2) maximum core size for on-line users

(3) maximum number of on-line users

(4) ratio of CPU time for batch to that for on-line computing, or some other measure of batch priority

(5) I/O buffer limits, and "choke-unchoke" points that determine when a program is to be considered output bound or ready for more computation.

F. System Error Detection and Recovery

In addition to standard error recovery normal to I/O devices the UTS system will take special measures to provide reasonable recovery for detectable machine malfunctions. Assuming that the normal failure mode will be that of intermittent error, the system will effect recovery by immediate restart of the user in question or the whole system if necessary after making records of machine status to aid in error diagnosis. This recovery will be accomplished without operator intervention. This technique will maximize the up time of the system while recording information useful to machine maintenance personnel.

Errors, whether caused by hardware or software, are of concern in any computer system. The consequences of failure in a time-shared system are multiplied because of its multi-programmed operation. When a time-sharing system fails each of the concurrent users of the system is affected, perhaps fatally. The possibility of an operator re-trying a job that has run into a machine problem is no longer an available option. Even symbiont batch systems run into difficult backup problems.

This specification does not offer any complete solutions to the reliability problem. Rather it suggests a number of possibilities of various degrees of implementation

difficulty for use in detecting or recovering from hardware problems. Since truly adequate error recovery depends in large measure on the exact strain put on the hardware by the mode or modes of operation of the software we must continually adjust our approaches to the reliability problem as the effectiveness of the various techniques are proved or disproved through experience. We expect this experience to show both the common failure modes of the hardware and the effectiveness of recovery and detection techniques.

The presumption is made that standard and adequate recovery measures have been taken wherever possible. That is, tape and disc transfers are parity-checked. Critical transfers are check-summed and/or address checked. Detected errors are recovered by re-read or re-write and operator assistance has been used where possible (say card problems). With these standard techniques out of the way we are still left with errors. (For some errors, such as memory parity, we are in trouble immediately and recovery by re-trial is impossible.) The latter category is the one we need to attack.

At least six facets of error handling need to be considered for a comprehensive attack on system reliability:

(1)	Prevention	(4)	Isolation
(2)	Detection	(5)	Recording
(3)	Recovery	(6)	Restart

Prevention of hardware errors is a matter of good machine design and good maintenance. However, we must not eliminate the possibility of identifying weaknesses in the hardware and providing fixes for them. System software has a history of identifying hardware weaknesses. In many cases a hardware fix will be the correct solution.

Detection is also often left to the hardware through parity checks, bounds checks, etc. Often, of course, only the software can tell that a certain signal means a malfunction in one case and not in another. Many software checks are possible --

so many, in fact, that it is often difficult to know where to stop. The usual solution is to check very little and depend heavily on the hardware. This is not good enough in time-sharing systems. Errors must be detected quickly and recovery initiated before total chaos develops. Simple checks for consistency of data should be made when feasible. More elaborate checks should be developed in frequently used codes such as the Scheduler, job control, check interrupt routines, and I/O handlers. A partial list of software error detection techniques which are useful in various situations is listed below. It is certainly not complete and should be added to as we gain experience.

(1) Periodic consistency checks
(2) Checkrunning
(3) One word data comparisons on I/O transfers
(4) Self-addressed RAD records
(5) Range checks on internal data
(6) Double end loop tests in critical routines
(7) Read compare after RAD write
(8) Watchdog timer checks for dropped I/O traps
(9) Software double checks on I/O action (for extraneous interrupts)

Diagnostics have long been used to identify failing machine parts. With the use of margins, weak components can sometimes be detected before they cause trouble in the actual working machine. While diagnostics of many types can be run in a time-shared system, their usefulness is limited because of the difficulty in margining; we have no way of providing marginal voltages or frequencies for just the time slice in use by the diagnostic (and returning to normal after error detection to provide automatic reporting of the error location and type).

Time-shared diagnostic programs are very useful for exercising peripheral units (tapes, card equipment, paper tape equipment, discs, etc.) and their controllers since the equipment can be isolated and separately margined. UTS will provide for such diagnostics allowing them master made operation and

providing for automatic execution of diagnostics during periods of light load.

Recovery of I/O errors of various types is fairly standard practice although it is often a long and difficult task. Many main frame errors are not recoverable at all. In fact, in the case of parity errors in the Sigma 7 it is not even possible to re-try. We may find that hardware help is needed in this and other cases. In certain cases known to the program the error is of little consequence (e.g., if it occurs while cycling in the idle loop) and the remedy will be to ignore the error. These cases will be relatively few. In the time-shared situation a machine error in a user's program may be "recovered" by restarting the job from the last swap image or RAD. This will work if no other I/O has occurred (a fact which can be recorded) and if the accounting information has been updated. Whether it is worth doing depends on the frequency with which we expect machine errors to occur.

Isolation of the area of error is particularly important if recovery is not possible. (Of course if isolation is complete enough we can recover but this is rarely the case). In the time-sharing environment, it is important to isolate the error to a single user if possible. If this can be done then the user and his data can be discarded without injury to other users.

Recording of all detected errors, whether recovered or not, is vital to good system maintenance. Automatic recording is preferable since fewer errors are over looked or ignored. (How many Sigma 7 machine errors went unreported last week?). In addition, the accumulation of records of intermittent failure is valuable in isolating problem areas of the machine which will require both more maintenance attention and better diagnostic and error recovery procedures. It is required that a teletype console be dedicated to recording of errors detected and recoveries made. The console also serves as a performance measurement log.

Total failures of the system should automatically record the vitals of the machine (registers, PSW, etc.) on the log for later analysis and a total core dump of the

machine on RAD to enable a very detailed analysis when warranted. Time and effort required to make this record is paid for on the first error, hard or soft.

A brief summary of the data which should be recorded is:

Recovered errors	Catastrophic failures
user console - sta #; count	type of test which failed
tape - unit #; count	registers
RAD - sector; count	PSD
card	special system temps
other types	core (on RAD)

Restart after a system failure in the shortest possible time is of great importance in a time-shared system. Users understand that machines fail occasionally and are happy if an automatic restart procedure is able to restart quickly from a total but intermittent failure. If all failures were solid ones, automatic restart would not help much but most failures are intermittent and restart serves to get the machine back up for the users quickly. The recording of the failure directs the CEs in their efforts during the next normal maintenance period.

In summary, the philosophy of UTS for machine errors and failures is prevention wherever possible, care in detection at the earliest possible time, recovery from as many errors as possible, isolation of the failures to limit the bad effects, recording of both error and failure situations to aid maintenance and rapid restart in the event of failure to maximize up time.

III. UTS SYSTEM CAPACITY

We have stated above that UTS is intended to handle batch processing operations and real-time programs in addition to on-line terminal users. Clearly the ability of a Sigma 7 to handle all these tasks adequately will depend on the total load submitted, the distribution of this load over the three broad categories of use, and the hardware configuration of the Sigma supplied to the task. Also, the user's satisfaction will depend on his definition of "adequately" -- what job turn-around time is acceptable in batch, and what response delays are tolerable in on-line service.

UTS achieves its responsiveness and efficiency through the application of several hardware and software techniques. The principal additions to the standard techniques embodied in BPM, and the primary gain from their use, have been discussed previously but are listed below for reference:

Multiple users in core - increases CPU utilization by increasing the probability that an executable task is in core. We try to assure that, on the average, four or more executable tasks (on-line, batch, etc.) are in core.

Use of Sigma 7 memory map - provides execution time relocation of user programs by page, thus simplifying bookeeping and reducing overhead in achieving multiple users in core. Since the page parts of a user's program may be placed anywhere in core, scheduling of tasks may be made to depend only on task priority and not be hampered by a need for contiguous memory allocation. Some additional flexibility accrues to the programmer through the availability of a large virtual address space.

Shared common processors - Re-entrant programming, in addition to MAP, allows all users to share commonly used processors such as editors, debuggers, and BASIC. Considerable saving in core space is achieved in comparison to systems requiring a processor copy per user.

But what will be the system's response under some typical loads? How effective will the above techniques be? In the paragraphs below we examine CPU and RAD load, on-line terminal response, and the division of the load among batch, on-line, and real-time uses for various loads typical of the industry. The results are back-of-the-envelope type calculations, but serve to give a general impression of expected UTS operation.

Two critical areas are examined below: RAD usage and CPU usage. RAD usage is examined for total time load; that is, the sum of the time required to service all requests for RAD transfers is estimated and compared with the time available to perform the requests. The calculations are made for three SDS RAD's and average delays are estimated from standard queuing delay curves. The results show that for the "typical" load the 7204 RAD is inadequate, the 7232 is marginal, and the 7212 quite satisfactory in any case. These results are for both files and swap storage on a single RAD. We will discuss later the splitting of these functions onto more than one device. RAD size capacity is not discussed, but BPM capacity can be used as a guide by adding 20-30,000 words for new processors and 120,000 words for swap storage (4,000 words each for 30 users). This would put the UTS RAD size requirement at about 2×10^6 bytes exclusive of file space.

CPU utilization for all non-compute bound and non-batch operations is estimated. Under the assumptions used, 70% of CPU capacity remains to be divided between compute bound batch, real-time, and on-line users after allowing for file I/O, symbiont operation, and service for non-compute bound terminal users.

A. Number of RAD Transfers

Table 1 below summarizes in seven broad categories the number of transfers required of a RAD in a UTS system. In each category the underlying assumptions are noted. Following the table the assumptions are discussed more fully.

Table 1

Disc I/O Transfers

		Transfers/sec
1)	Printer Symbiont & Co-op (800 lpm)	3.3
2)	Card Reader Symbiont & Co-op (200 cpm)	.5
3)	Batch execution I/O - (non-peripheral)	5.0
4)	Terminal user I/O to files -- $\frac{3N}{20}$; not for editing or debugging; 3/user/interaction; 20 sec/interaction; N = 30 users.	4.5
5)	Swaps for interactive users -- N/10 2 transfers/interaction/user; 20 sec/interaction; N = 30 users.	3.0
6)	Swaps for time slicing -- 2/Q 2 transfers per time slice quanta; Q = 300 ms.	6.7
7)	Monitor overlays -- 500 per batch job; 500/j processor fetches, library loading etc; job time j = 1.5 min.	5.5
	TOTAL	28.5

Some notes on the values assumed in Table 1 are appropriate:

1) We assume that the print load generated by all programs in the system will be sufficient to drive the printer at its full speed of 800 lines per minute. This is probably a good assumption for busy periods, but somewhat high as a full time rate. Student problems at a university produce 800-1000 lines of output per minute of execution while scientific-aerospace environments have rates nearer 300 lines per minute.

2) Average card input rates at university and aerospace computing centers seem to be in the range 100-300 cards per minute of computing.

3) File I/O necessary for problem execution naturally depends on the program, and ranges from zero to whatever rate the device is capable of. We guess that 5 transfers/sec will be representative.

4) File I/O generated by terminal users is estimated from JOSS where program loading, JOSS's equivalent of chaining, and data I/O amount to about 3 records transferred per terminal interaction. Three records transferred per interaction also seems to be a reasonable rate for inquiry systems -- say two dictionary look-ups and one data fetch. The assumed figure should be conservative since we presume that most user time will be spent editing or debugging, and in both of these activities the I/O rates should be an order of magnitude smaller than the assumed rate.

5) In servicing terminal users' requests we assume that for every request (interaction) the users program must be brought into core from RAD. Space in core must be cleared by transfer to RAD. The assumed interaction rate of once each 20 seconds is conservative - most time-sharing systems measure an interaction rate of once per 30 seconds.

6) Compute bound users are serviced in round-robin fashion. That is, each time quanta we shift CPU control from the currently executing program to the next program in the compute queue. It is usual that 5-20% of the on-line users are compute bound (both JOSS and SDC systems have 6% compute bound) so it

might often be the case that no swap is required to ready the next compute bound user for execution. We choose the conservative assumption, however, that a swap is always required each compute quanta. (For instance the case of 5-4,000 word compute bound programs operating in 16K of memory.) Note that if some of the users are compute bound then they should not be counted in the swap for interaction, or the terminal file I/O categories. We can either count this as conservatism or say that the number of users served is 5-20% higher.

7) Current measurements on "typical" batch jobs in BPM record about 500 file I/O actions. This includes fetches for all needed processors (FORTRAN, SYMBOL, LOADER, CCI) overlays for the processors, overlays for the monitor, file I/O for ASSIGN's, Debugs, processor intermediate data, programs fetched from the library, etc. The assumption of a constant number of I/O actions per job is rather gross but we know of no better assumption. The average job time of 1.5 minutes is representative of a university-student environment. For scientific-aerospace shops, the job time is more like 3 minutes. We choose the conservative figure.

B. Transfer Times

Transfer time depends on the amount transferred, the RAD used, and the access algorithm. Reasonable transfer amounts for the seven items above are 1) and 2) 256 words, 3) and 4) 512 words, 5) and 6) 4,000 words, and 7) 512 words.

RAD transfer times for 3 SDS RAD's are:

7204	23.6 ms/1000 words
7232	11.3 "
7312	1.7 "

Table 2 below repeats Table 1 but also lists the percent of RAD capacity required for data transfer only -- latency assumed to be zero.

Table 2

Percent of RAD Capacity

	Item	Xfers/sec	Words Xfered 1000's	7204%	7232%	7212%
1)	Print Symbiont	3.3	1/4	2.0	1.0	.1
2)	Card Symbiont	.5	1/4	.3	.14	.02
3)	Batch Execution	5.0	1/2	6.0	2.8	.45
4)	Interactive File I/O	4.5	1/2	5.4	2.5	.4
5)	Swaps for Interaction	3.0	4	28.0	13.5	2.1
6)	Swaps for Time Slice	6.7	4	63.0	30.2	4.7
7)	Batch Overlays	5.5	1/2	6.6	3.1	.5
	TOTAL	28.5		111%	53%	8%
	Latency @ 17ms/Xfer (28.5 Xfers)			49%	49%	49%
	GRAND TOTAL			160%	102%	57%

The table shows clearly that swaps performed for time slicing have a large effect. Since the quanta size is under our control, we change it from 300 ms to 1 second and recalculate. This is "tuning" the system.

Total load on the RAD's are now:

	7204	7232	7212
Transfer Load	67	32	5
Latency Load	40	40	40
Total	107%	72%	45%

C. Interactive Delays

Interactive response time is controlled by our ability to fetch a user's program from the RAD in conflict with all other users wishing response. The situation is similar to single-server queue situations. Average delays have been

calculated and delay curves are shown in Figure 1. The delay is given as a function of the fraction of full load and is plotted in terms of service time.

The four solid curves are plotted according to four different assumptions about the nature of the source of the load. The upper pair of curves represents the exponential service time assumption, meaning that the amount of time required to take care of the request is distributed exponentially. The lower curves assume that service time is a constant for all requests. Our service times are neither, but contain components of each: The compute component and part of the data transfer time are probably distributed exponentially; part of the transfer time and some overhead time is constant; and the RAD latency is uniformly distributed. We hope that our composite case can be estimated to be between the two curves shown.

The upper and lower curves in each pair show the variation with the number of sources supplying the load -- in our case the number of users. Note carefully, however, that the curves are normalized in such a way that the users -- whether 25 or infinite -- are generating the same total load. However, the curves are still useful; when the number of users is doubled, the load is also doubled.

Note that at a load of 1 (100%) that the average delay is equal to the number of users multiplied by the service time. The queue is full; each request finds all the other users already in the waiting line. The average delay is also the maximum delay.

The dashed curves give some idea of the variation to be expected in delay. For the two assumptions of exponential and constant service times these curves mark a level of delay which will be exceeded in 10% of cases. The important thing to note here is that the delay may closely approach the maximum value at RAD loads as low as 80%.

Service time for interactive users is the time to swap his program into core (usually this requires transfer of a currently resident program to RAD to make room) plus the computation time necessary to service the request. An average computation time of 50 ms is sufficient for more than 95% of on-line

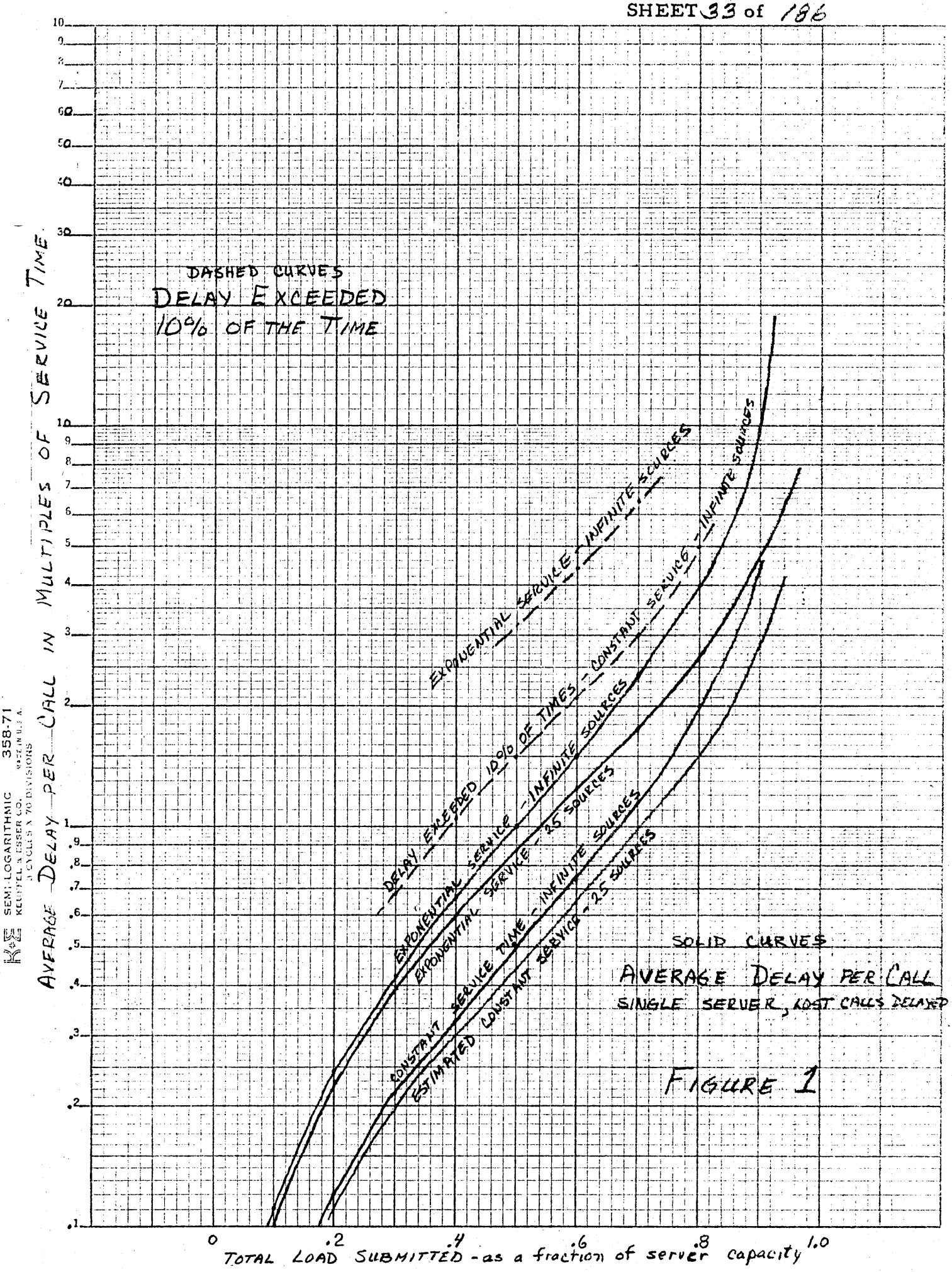


FIGURE 1

interactions. The table below shows the service times for 4000 word programs on the three SDS RAD's, including two way data transfer and 17 ms latency for each unit transferred.

Service Times 4000 Word Full Swap

RAD	Swap ms.	Comp. ms.	Total ms.
7204	222	50	272
7232	124	50	174
7212	48	50	98

The curve indicates that loads of 50% result in an average delay of 1 service time. Thus with the 7212 RAD, responses to users would average about 150 ms, including a reasonable amount of computing time. Clearly this kind of response is good.

On the other hand, the average delay curve rises very rapidly as load approaches 100%. At 100% load the average and maximum delay are equal and may be approximated by the number of users multiplied by the service time -- 7 seconds for the 7204 RAD.

The percent RAD load can be calculated and the delay due to RAD load can be estimated for cases other than that given above in Table 1 from the following formula.

$$L = \frac{d}{10}\left(8.8 + \frac{N}{4} + \frac{2}{Q} + \frac{500}{j}\right) + \frac{1}{10}\left(3.5 + \frac{N}{10}(3/4 + S) + \frac{2S}{Q} + \frac{250}{j}\right)$$

where

N = The number of interactive users.

r = RAD transfer rate - ms/1000 words.

d = latency delay per transfer - ms.

s = average program size (interactive users) - words.

j = average batch job time -- seconds

Q = time slice quanta - seconds.

D. Compute Load on the CPU

Table 3 below shows the breakdown of the major components of load on the CPU not including execution of user programs or batch processors.

Table 3

CPU Load

	% of Sigma 7 CPU
1) Printer Symbiont & Co-op (800 lpm); 2.0 ms/record for Co-op; 1 ms/record for symbiont;	4.0
2) Card reader symbiont & Co-op (200 cpm)	1.0
3) Cycle stealing - memory transfer interference of swap and file I/O with computing. Worst case.	5.0
4) Swap I/O management @ 500 μsec/transfer.	.5
5) File I/O management and transfer @ 7 ms/record.	13.0
6) COC terminal I/O management and conversion - 100 μsec/char; 30 users; 4 char/sec/user.	1.2
7) Computation for interactive response 30 users; 1 interaction/20 sec; 50 ms average processing -- (enough for < 95% of all interactive requests)	7.5
TOTAL	32.2

E. Assumptions

Some notes on the assumptions used in computing the various loads are again given in order.

1 & 2) The loads assumed for the card and printer symbiont are the same as those used for the RAD load. The difference in time required between the symbiont and its corresponding cooperative reflects the fact that the symbiont transfers data directly from buffer to device, while a move of the record core-to-core is required for the cooperative.

3) Worst case interference between a computing program and I/O transfers occurs when both operations use the same memory box. In time-sharing systems we are transferring data and programs between RAD and core a large fraction of time so there is usually a payoff in interference reduction if core is organized into non-interleaved boxes. A first guess would be 1/Nth interference if there are N core boxes.

4) The estimate here of 250 instructions to control each swap should be conservative.

5) Overhead of the BPM file I/O system is currently about 7 ms/record of 100 bytes. Scheduled improvements will reduce the figure by about 2 ms/record and new access methods may add to the improvement.

6) Terminal I/O includes translation between internal and external form and buffering as well as standard checking and facilities for several different kinds of consoles. The rate of 4 characters per second per user is that measured in the JOSS system and others. We have no reason to believe that the rate will be any different in UTS.

7) As before, the interactive rate of one message per user per 20 seconds is a conservative one by standards set in current time sharing systems. The estimate that 50 ms of computing is the average required for over 95% of all requests again comes from JOSS. 85% of requests require less than 50 ms to complete. The figure is lower than that recorded in the SDC and MAC systems but only by amounts that may be explained by the difference in machines. A factor of two increase would not be suprising.

Thus about 70% of CPU capacity remains to be divided among computing for batch jobs, compute bound terminal controlled jobs, and real time responses. Of course a single compute bound program can use all of this time if allowed, and if more than one is in the system, delay must occur sin ce the resource is overloaded. Scheduling of compute bound jobs is controlled by installation management through control parameters discussed in a later section.

IV. SCHEDULING

The routines described in this section control the overall operation of the system. They receive inputs from the I/O systems when certain critical events occur, from the user program when it requests monitor services, and from the Executive language processor reflecting requests of the user. These inputs (or signals) coupled with the current status of the user as recorded by the Scheduler are used to change the position of the user in the scheduling status queues. It is from these queues that selections are made for both swapping and execution. Swaps are set up by selecting a high priority user to come into core and pairing him with one or more low priority users for transfer to RAD. Similarly, the highest priority user in core (and thus ready to run) is selected for execution.

A. Inputs to the Scheduler

The list below records those system activities which must be reported to the Scheduler. The reporting is done variously through a logical signalling table, through direct entry to the Scheduler, and through protected changes to the User Status queues. The Scheduler records the receipt of signals by a change in the user status queues plus other information associated with the user. In general, a table driven technique is used with the received signal on one coordinate and the current state on the other. The table entry thus defined names the routine to be executed in response to the given signal-state combination. Since the number of signals and states is large the table technique aids in debugging by forcing complete specification of all the possibilities.

Inputs from the COC routines

1) Input complete--activation character received
2) Output limit reached--sufficient output for 3-5 seconds
3) Output nearly empty--only 1/2-1 seconds typing left
4) Interrupt (BREAK) character received--request for alternate entry, usually for return of console control to the user.
5) Request for executive control
6) Other special signals as required

Inputs from the swap I/O handler

1) Swap complete--rescheduling and/or another swap may be needed
2) Swap error--a RAD sector cannot be written successfully. Action will be a report to the error log, lockout of the failing sector, and re-trial at a different location
3) Swap error--a RAD sector cannot be read successfully. The user cannot be continued; the error is logged and the user informed.

Inputs from the program (through monitor service calls)

1) Request console input
2) Transmit output to the console
3) Wait a specific time period
4) Program exit (complete)
5) Core request--both kinds provided by BPM plus request at specified virtual address
6) Program overlay--load and link, load and transfer
7) Input-Output service calls.

Inputs from Executive Language Processor

1) Name of system program to load and enter. Implies deletion of any current program
2) Continuation signal
3) Special continuation address
4) File name for submission to batch processing

B. Scheduler Output

The scheduling routine performs two major functions during the times it is in control of the machine: First it sets up swaps between main core memory and secondary RAD in such a way that high priority users are brought into core replacing low priority users who are transferred to RAD. The actual swap is controlled by an I/O handler for the swap RAD according to specifications prepared by the Scheduler. The Scheduler makes up the specifications for the swap according to the priority state queues described below. Given a suitably large

ratio of available core to average user size the Scheduler can keep swaps and compute 100% overlapped.

Secondly, the Scheduler selects a high priority user for execution. According to the single priority state queues and the rules for treating batch. The rule is extremely simple--pick the highest priority user whose data is in core.

C. User Status Queues

The status or state queues form a single priority structure from which selections for swaps and selections for execution are made. The state queues form an ordered list with one and only one entry for each user. Position is an implied bid for the services of the computer. As the events occur which are signalled to the Scheduler as described above individual users move up and down in the priority structure. When they are at the high end they take high priority for swap into core and execution, and when at the low end they are prime candidates for removal to secondary storage. This latter feature--that of a definite priority order which selects users for removal to disc--is an important and often overlooked aid to efficient swap management. It avoids swaps by making an intelligent choice about outgoing as well as incoming users.

In addition to these primary functions the queues are used for other purposes: synchronizing the presence in core of user data and program with the availability of I/O devices, waiting for "wake up" at a pre-established time, queuing for entry and use of processors, and core management problems.

A list of the state queues in decending priority order is given below.

INT	-	Interrupt or break received queue
IAC	-	Input activation queue
UBL	-	Console Output unblocked queue
COM	-	The interactive compute queue
BAT	-	The batch compute queue
CU	-	Current user in execution
DU	-	Current disc file user
DQ	-	Waiting for disc queue
BLK	-	Console output blocked queue
INW	-	Waiting for console input queue
QW	-	Queue of users to be awakened
OFF	-	Queue of users turned off.

The above list serves for the illustration of the operation of the Scheduler below.

D. Scheduler Operation

To select users for execution the Scheduler searches down the priority list for the first user in core memory. Thus, interrupting users will be served before those with an active input message, both will take precedence over users with unblocked console output, next will come compute users and finally the batch job(s). Note that users in any lower states have no current requests for CPU resources. Note also that as each user is selected for execution his state queue is changed to CU, and when his quanta is complete the highest priority queue he can enter is the compute queue. Users who enter any of the three highest priority states receive rapid response, but only for the first quantas of service. Thereafter they share with others in the compute queue.

Two examples of typical interactive use will be illustrative.

The first follows a user with a simple short interactive request. As he types the request he is in the INW queue and his program probably has been swapped to RAD. It remains there until the COC routines receive an activation character. This is reported to the scheduler and cause a state change to IAC. The Scheduler finds a high priority user not in core and initiates a swap to kick out a low priority user (if necessary) and bring in the just activated one. On completion of th swap the Scheduler is again called and it now finds a high priority user ready to run. The users state is changed to CU the program is entered and examines the input command. The cycle may complete by preparation of a response line and a request to the monitor for more input. This would reduce the users state to INW again making him a prime candidate to kick out of core.

The second example illustrates a console output-bound program. This program moves through the state cycle BLK-UBL-CU as output is generated by the program, the COC signals the reaching of the output limit, and finally the output is drained onto the terminal. If the operation is proper, five or six

seconds of typing will be readied in buffers each time the user program is brought into core and executed. During this typing time the program is not required in core and the cpu resources can be given to other programs.

Selection for swapping picks a user to bring into core and the lowest priority user to kick out. Priorities are arranged from high to low, in order of increasing expected time before next activation. This assures that the users who are least likely to be needed are swapped our first, retaining in core always the set most likely to require execution. The swap algorithm will operate so that: a) if there is room in core for three user programs, b) if two users are computing steadily and c) if many other users are doing short interactive tasks, then the compute users will remain in core and use all available compute time while the interactive users are swapped through the third core slot. Of course the non-uniformity of program sizes and request arrival times will cause different action from time to time but on the average it will be substantially as described.

E. Treatment of Batch Jobs

Two ways of scheduling batch are reasonable in this priority structure. They result in quite different fractions of machine time devoted to batch. Both will be provided in UTS and the operator or installation manager will be able to select the desired mode of operation. The first treats the batch stream in a separate queue (BAT) of lower priority than the interactive compute queue as indicated in the queues of Section C. Thus batch only gets service when no interactive user has a request. Crude estimates from current systems indicate that 10-20% of machine time would be available to batch on a system supporting between 20 and 20 concurrent users in prime shift.* That is 10-20% of the time no on-line user is requesting time. During non-prime time 90% or more of CPU time would be available to batch.

*In Part III we estimated that 70% of CPU capacity would be available to batch and on-line compute-bound combined. Here we estimate the on-line will use 50 or 60 of that 70%.

The second discipline cycles the batch user through the interactive compute queue where each job receives an equal fraction of the available time. It is usual in on-line systems that 5-20% of the on-line users are computing at any one time; thus as much as 1/2 of prime time could be devoted to batch background operation plus the 90% + on non-prime time. In this scheme, batch can be biased to get a different quantum than on-line users.

F. Swap Hardware Organization

Users are saved in a dedicated area of the RAD (or a separate RAD in large configurations) during the periods between the turns for execution on the central processor. The minimum system will allocate a portion of file RAD to this purpose and dedicate a special handler to the performance of the swaps.

A bit table is used to keep track of the availability of each sector on the RAD, marking zero for in use (usually assigned to a user) and one for available. Users are assigned a sufficient number of page size sectors to accommodate their current use. The assignment is done in such a way that command chaining of the I/O can order the sectors to be fetched for a single user with minimum latency. That is, each users pages are spread evenly over the set of available sectors so that when the user is swapped data will be transmitted in every sector passed over.

The records of the disc sectors associated with each user will be kept in the users job information table (JIT) which is kept on RAD when the user is not in core. The disc location of the JIT table is kept in core by the Scheduler. The RAD layout is such that sufficient time is available to setup I/O commands for the remainder of a user after his JIT arrives from RAD.

The amount of RAD storage assigned to swapping will be a parameter of SYSGEN The number of on-line users which the system can accommodate is limited by the size of RAD space allocated for swapping and the total size of all active on-line users.

G. Processor Management

Processors will be considered time-sharing processors when they are added in such a way that the processor is read-only and makes no initial assumptions about the user's data area. When these criteria are met the processor is treated in the following special ways:

1) Its name is known to the Executive Language; it may be called on by name.
2) It will have dedicated residency on swap storage established at SYSGEN time.
3) Its use will imply a particular virtual map for the user.
4) A single copy will be used by all requesting users.
5) It will never be swapped out. In fact, we will write lock to portions of RAD dedicated to such processors.

H. System Management Parameters

Effective control over the operation of UTS is provided to the installation manager through the adjustment of the dynamic parameters of system operation. Gross adjustments can be made at SYSGEN time but the fine tuning of the system to the changing demands of a particular compute shop is done by changes to the dynamic system parameters through commands at the operators console. The list of adjustable parameters includes at least the following:

1) Maximum core size for on-line users
2) Maximum core size for batch user
3) Maximum RAD file space allowed on on-line user
4) Number of on-line users allowed
5) Size of time slice compute quanta-milliseconds (Q)
6) Size of minimum compute quanta-milliseconds (q)
7) Batch bias parameter, b: batch gets b·Q compute quanta
8) Batch scheduling dicipline-high or low priority
9) Number of tapes allowed an on-line user.

V. System Requirements and Configuration

Throughout this specification the assumption is made that UTS will be a system designed and built to service batch, real-time, and on-line terminal users. Each installation will have to evaluate its requirements and desired service in order to arrive at a useful machine configuration. A reasonable selection of hardware can only be made with a good knowledge of the characteristics of its intended use, including the portions of computing devoted to real-time, batch, and on-line. Also, the number and usage profiles of on-line users, size of on-line programs, I/O characteristics, etc. must be evaluated.

It is obviously impossible to list the infinite combinations of equipment which would support UTS in some manner. It is also difficult to delineate a minimum configuration (different requirements will have different minimum configurations). Therefore we will describe a configuration for a set of requirements and indicate possible downward and upward adjustments in equipment that could be made for varying requirements.

In attempting to determine what a particular configuration should be, several things must be kept in mind:

1. The UTS resident monitor will require an estimated 16K words.
2. UTS is predicated on and requires a symbiont system.
3. Since real-time requirements are pre-emptive and installation dependent, no allowance is given here to these demands on the machine.
4. References should be made to the sections of this specification dealing with loading, responses, and performance, since these factors will largely determine configuration requirements.

In order to support 32 on-line users (who are not compute-bound) and maintain a high rate of batch throughput (about 70% of BPM rate), the following configuration is appropriate. UTS design optimazation will focus on this system level.

CPU

Model	Description
8401	Sigma 7 CPU
8413	Power Fail-Safe

CPU (Cont.)

Model	Description
8414	Memory Protect
8415	Memory Map
8416	1 Additional Register Block
8421	Interrupt Control Chassis
8422	Priority Interrupt, Two Levels
8418	Floating Point Arithmetic

Memory, 64K

8451	Memory Module, 4 each
8452	Memory Module, 12 each

Basic I/O

8473	MIOP
8475	Four Byte Interface
7012	Keyboard Printer (2)
8485	Selector IOP
7211	Hi-Speed RAD Controller
7212	Hi-Speed RAD
8456	3 Way Access, 4 each
7611	Communications Controller
7612	Format Group Timing Unit
7615	Send Module, 32 each
7615	Receive Module, 32 each
7621	EIA Interface Modules
7613	Line Interface Unit, 3 each
7015	Keyboard/Printer KSR/35, 32 each

Secondary Storage and Peripherals

7140	Card Reader
7160	Card Punch
7445	Line Printer
7321	Magnetic Tape Controller
7322	Magnetic Tape Units (4)
7231	RAD Controller (4 byte interface for 7231)
7232	RAD Storage (24MB)

Comments and Variations

1. CPU

All items listed except Floating Point are minimum requirements for any system. Another option available is the Decimal package.

2. Memory

Some systems may run satisfactorily with 48K; for example, 8 users rather than 32, or a 32 user system with little or no concurrent batch. More memory than 64K will allow more users, larger programs, etc.

3. Basic I/O

The absolute requirements are an MIOP, 2 Keyboard Printers, one RAD (6MB) and controller, and the Communications Controller and associated terminal equipment. However, to maintain reasonable performance, separate RAD's for the system and file storage are recommended for all systems. For a 32 user system, the system RAD should be a Hi-Speed RAD with an SIOP. Smaller systems, 16 users for example, could maintain acceptable performance with two 7232 RAD's, for example.

4. Secondary Storage and Peripherals

The only requirements are a card reader, a tape controller, and at least one tape unit. Any reasonable batch configuration can be expected to also include a line printer, additional tape units, RAD file storage, and a card punch. An additional MIOP may be required, depending on the type and number of secondary storage devices and peripherals. Variations in the number and type of I/O devices will depend on installation requirements, but the configuration listed is reasonable for a batch system with up to 32 on-line users.

4. Secondary Storage and Peripherals (Cont.)

Terminals

The only variation in on-line terminals forseeable now is the addition of different kinds of terminals (Keyboard/Display, IBM 2741, etc.) . Provisions will be made to the extend feasible for supporting future terminal devices. UTS will be limited to a maximum of 64 terminals. To increase this number would require major changes to the COC software (which will support only one 7611 Communications Controller).

VI. Terminal Executive Language (TEL)

TABLE OF CONTENTS

MAJOR OPERATIONS

A. Compilations and Assemblies
    1. Inputs and Outputs
    2. Commands (COMPILE, ASSEMBLE)
    3. Controlling Error Commentary and Output
    4. Error-Handling and End Actions
    5. Entering Program from the Terminal
    6. Debugging Information

B. Linking ROM's and LM's to Form LM's
    1. Simple Linkages (LINK)
    2. Load Module Symbol Tables
    3. Merging Internal Symbol Tables
    4. Searching Libraries
    5. Error Reporting and End Actions

C. Loading LM's Into Core
    LOAD

D. Initiating Execution
    START
    RUN

E. Initiating Debugging Operations
    DELTA, FDP

F. File Management
    COPY
    DELETE
    CALL PCL

G. Editing
    CALL EDIT
    EDIT
    ACCEPT

H. Submitting Batch Jobs
    BATCH
    Requesting Status
    Cancelling Remote Batch Jobs

Page

COMMUNICATION CONVENTIONS

A. Keyboard Control

As previously mentioned, control of each user's keyboard will be proprietary: either the user or the system will have control. The assumption is made that all terminals in use are attended. Terminal communication conventions will be as follows.

1. Whenever the UTS executive processor returns control to the user after an error, an interruption by the user, or after completing a request, it will type an exclamation mark (!) at the left margin of a fresh line before turning control of the keyboard over to the user. This notifies the user that he is talking to the UTS executive processor and must couch his request in that processor's language (TEL).

2. Whenever the services of a sub-system are first requested by the user, that sub-system will identify itself in plain-talk before turning control over to the user.

3. All sub-systems that carry on line-by-line, rather than intraline, dialogues with the user will type an identifying mark at the left margin of the line before returning control to the user. Sub-systems for editing will use an asterisk (*); sub-systems for combining object programs and manipulating their associated symbol tables will all use a colol (:); utility sub-systems for file management and information transfer will use the numerical relation sign (<); all sub-systems for working with other programming languages will use the sign (>). These identifying marks notify the user that one of a class of sub-systems is awaiting a command from him. Which sub-system it is and which language must be used should be in the head of the user. This is possible, since no sub-system of UTS will call on another one, or even on itself. However, some commands in UTS's executive language (TEL) require the services of a succession of distinct processors, as may some commands in other sub-systems. Whenever

such "hidden" processors detect an error, they will, where necessary, precede error messages by a single space followed by an identifying mark appropriate to the processor's function.

4. Users' programs that must return control to the user to allow him to input values and other information are left to their own devices. Such programs should be written so that they display enough information for the user to determine what is expected of him in such situations.

B. Typing Lines

The mechanisms for correcting characters, for erasing messages that may be hopelessly mistyped, for signalling end of message, and for line spacing are uniform. These are given below for users with TTY terminals.

1. The user can erase his last unerased token by depressing the RUBOUT key. UTS will respond by typing a slant-line (/) to indicate that it has effectively backspaced and erased. On terminals that can backspace, backspacing will be non-erasive and users will be able to overstrike tokens as well as erase them. On such terminals, UTS's image of the line being typed by the user is identical to the one the user sees on his printed page -- assuming that he can read his overstrikes and erasures.

2. The user can erase an entire message by depressing two keys simultaneously, CONTROL and X. UTS will type a back arrow (←), return the carrier to the beginning of a fresh line, and return control to the user without further comment.

3. Blank lines are ignored by UTS's executive processor and by all its sub-systems that carry on line-by-line dialogues with users. The appropriate identifying mark will be typed at the left of a fresh line before control is returned to the user.

4. When talking to TEL or any sub-system that carries on line-by-line dialogues with users, the user signals end-of-message by depressing the carrier RETURN or LINE FEED key, or by simultaneously depressing the CONTROL and L keys to signal end-of-page (see 5. below) as well. UTS will shift the carrier to the left margin of a fresh line (after taking care of any pagination that may be called for or required) and take over control of the keyboard. Except for interruptions (see C. below) all subsequent transmissions by the user will be ignored until keyboard control has been returned to him.

5. Pagination and lineation are controlled by UTS so as to provide 8 1/2 by 11 pages with one inch margins at the top and bottom of each "page". This assumes a 9 1/2" platen, giving 85 Gothic characters to the line; 8" platens provide for 72 characters. UTS will count lines to give 54 lines per page. In addition, the user can request pagination directly by depressing the CONTROL and L keys simultaneously. Pagination consists of: a) six blank lines; b) a heading line, containing date, time, user identification, console identification and page number; c) six more blank lines. Thus, the heading line can be scissored off to obtain 11" pages.

6. Some terminal devices have readily adjustable and usable tabbing features, others can tab but make adjustments difficult, others, can't tab at all. To handle the last two cases, UTS permits users to request that tabs be simulated by successive spaces. Tabs are not normally simulated; to turn on tab simulation triply depress the (CONTROL, SHIFT, "OH") keys and then depress the T key. To turn off tab simulation, repeat the procedure. The setting and clearing of tab stops will also be allowed, possibly by preempting two non-printing characters to signal set-tab and clear-tab; an alternative is to provide explicit commands for these actions.

7. Echoing of characters back to the terminal is at the discretion of the user. Normally, UTS will echo; to request no echoing, the user must triply depress (CONTROL, SHIFT, "OH") and then depress the E key. To turn on echoing again, the procedure is repeated.

A complete list of these and other control functions is given in the COC Functional Specification.

C. Interrupting UTS

1. Whenever one of UTS's sub-systems is in control of the keyboard, the user can interrupt and temporarily suspend operations by simultaneously depressing the CONTROL and E keys. UTS will respond by stopping the current operation as soon as it reaches a convenient break-point, and then turning the user over to the executive processor, TEL.

2. Whenever UTS or one of its sub-systems is in control of the keyboard, the user may interrupt whatever is being done for him at the moment by depressing the BREAK key which will give control to that part of the system currently in communication with the terminal (e.g., a sub-system). Since some actions can only be stopped at points of convenience and others have so much inertia that they can not be stopped at all, a succession of BREAK depressions will be treated by UTS as a single interrupt request. It must be emphasized that depression of the BREAK key does not constitute a preemptive request for the services of UTS's executive processor (see 1. above): the precise handling of interruptions by sub-systems will accompany the functional description of the sub-system; handling of interrupts by users' object programs will be covered in the section that describes the calls that programs can make on UTM services. Baldly speaking, however, interruptions of the system or any of its major sub-systems will result in termination of the current operation as soon as possible and a return of keyboard control to the user after the appropriate identifying mark has been typed. Since line noise can generate spurious interrupts, it is also wise to have UTS say something first; e.g., "Stopped by interrupt." Interruptions of object programs will, in the absence of short-stopping actions by the programs themselves, always cause a back-up to the executive processor. Programs being

run under control of debuggers or under control of a programming language sub-system like BASIC will identify the point of interruption as best they can (e.g., "Interrupted at statement 120.") before returning control to the user. By the same token, the execution of so-called "stop" and "pause" commands should result in similar behavior; e.g., "Stopped by statement 120."

D. Typing and Interpreting Commands

Except for a few declaratives, commands take the form of imperative sentences: an imperative verb followed by a direct object or list of objects; indirect objects usually follow a preposition, but may follow the verb (with elision of the implied direct objects). Minor variations on the major theme of a command are expressed as encoded parentheticals following either the verb or one of the objects. Individual elements of a list of objects are set off from one another by commas. Common rules of composition hold: words of the language, numerals, object identifiers and other textual entities may not be broken by spaces; otherwise, spaces may be used freely. For purposes of scanning commands (both by machine and the human eye) this rule has a simple interpretation: in a left-to-right scan for the next syntactic element of a command, skip over leading spaces; treat a trailing space as a terminator for a word, numeral or other textual entity. In terms of machine scanning, tabs (which are represented by a unique encoding) are treated as spaces. In addition, a unique encoding that indicates "end-of-command" must be recognized as a syntactic element; for TEL, this will probably be the carrier-return code. In other words, a legitimate command can't have any trailing garbage -- one could never determine whether it was a spoof on the part of the user or a real error.

E. Error Detection and Reporting

UTS's general philosophy in these areas is made up of two points.

1. Don't mess up the user or his information by carrying out a command or an operation that can't be carried through to completion. This rule must be tempered by considerations of efficiency and speed. For example, in commands that refer to file storage, it may be unfeasible to check for the existence or non-existence of the files mentioned; it is probably unwise to simulate an entire

command to check for storage-limit run-overs before actually carrying out the command, and impossible to anticipate hardware and device malfunctions. However, TEL and all its sub-systems that carry on line-by-line dialogues with users will always parse an entire command before starting an operation to insure that the command is, at the least, formally valid.

2. The majority of errors are readily grasped by the user's eye and head once the fact of an error has been brought to his attention. Accordingly, error messages will be as terse as is possible within the constraints of readability.

The error messages themselves and the specific actions taken on errors will be covered in final UTS documentation. However, many errors and error reports are uniform throughout TEL and some of its sub-systems, and can be listed here.

a) Garbled, malformed or unintelligible commands:

EH?

b) Garbled or invalid file, device, reel, account identifications, and others:

FILE ... ?
DEVICE ... ?
ACCOUNT ... ?
PASSWORD ... ?
JOB ... ?

c) References to (deleting, reading, overwriting) a non-existant file:

NO FILE ...

d) Attempts to write ON rather than OVER an existing file:

ON FILE ... ?

e) Errors, abnormalities, storage-limit over-runs associated with an input-output action or with a specific file:

FILE ... :
DEVICE ... : followed by error message

IDENTIFICATION AND NAMING CONVENTIONS

On-line users are provided a set of uniform conventions for representing information for fiscal accounting, file identifiers, devices, and other objects.

A. Accounting Information and File Identification

An on-line user must identify himself before he can use TEL or any of its sub-systems, Procedures for doing so are described in the next section. Three pieces of information are required:

a) the user's personal identification (id)

b) the user's account identification (account)

c) a password (password)

These, as well as names for files (file name) may be represented by a string of no more than 11 contiguous letters and/or decimal digits. Embedded underscores may be used as separators (they count as characters); these print as left-facing arrows (←) on model 33 and 35 TTY's.

Files are identified by name, account and password; file identifications (fid) are represented by file name, account and password (in that order) separated by hyphens. In the absence of account and/or password, UTS uses the log-on accounting identification. All identified files are permanent.

B. Device Identification

Device identifications are represented by two-letter abbreviations for: card reader (CR); card punch (CP); line printer (LP); on-line terminal (ME); labeled tape (LT); free-format tape (FT).

Tape identifications must be followed by a number sign (#) and a reel number; e.g., LT#727.

INITIATING AND ENDING ON-LINE SESSIONS

An on-line user must establish a connection with UTS and identify himself properly before he can use TEL or any of its sub-systems. When a connection with UTS has first been established, UTS will respond by typing

IDENTIFICATION PLEASE:

and then waiting (on the same line) for the user to identify himself by typing his id, account and password (separated by commas) on the remainder of the line and then depressing the RETURN key. If the identification is valid and consistent with UTS's records, TEL will type an exclamation mark (!) at the left margin of the top line of a new page and then await the user's first command. If the identification is garbled or otherwise invalid, UTS will notify the user, and then repeat the initiation procedure. The messages are:

EH?
ACCOUNT ... ?
ID ... ? (filling in the garbled or invalid item)
PASSWORD ... ?

During the course of a session, the user may close out his current session's accounting and reinitiate under new identification by typing

I'M ID, account, password

or a combination of

ID id
ACCOUNT account
PASSWORD password

To close out and disconnect, the user types

OFF

Whenever a session is closed out, UTS will print at the user's terminal a summary of his charges and other information. This includes total session time, charged CPU time, and the reel numbers of any new tapes that had not been dismounted prior to closing out.

MAJOR OPERATIONS

Most commonplace activities associated with FORTRAN and assembly-language programming can be carried out directly in TEL; others require calling for the services of one of TEL's sub-systems. Figure 3 indicates how such activities are carried out from the console; TEL commands are capitalized, and sub-systems indicated.

1. FORTRAN, SYMBOL and XSYMBOL programs are created, filed away and changed through the EDIT sub-system either by explicitly calling for EDIT or by the EDIT and ACCEPT directives.

2. Programs are COMPILED or ASSEMBLED from the files or from the terminal (line-at-a-time) into relocatable object modules (ROM).

3. ROM's may be LINKed into load modules (LM).

4. ROM's and LM's may be LINKed and may be modified by SYMCON.

5. LM's can be LOADed into core and execution STARTed.

6. Linking, loading and starting of ROM's can be subsumed under the single directive, RUN.

7. Object programs can be run or started under the control of one of the debugging systems DELTA and FDP.

8. Executing programs that have been interrupted or stopped can be CONTINUEd after corrective actions.

9. Core images can be SAVEd on the files, and a user may GET a saved core image at some later date for continuation.

10. Files of information can be managed directly (COPY, DELETE) and through the PCL and EDIT sub-systems.

A. Compilations and Assemblies

1. Inputs and Outputs

One or more source programs can be compiled or assembled into a single ROM. Input identification (sp) may be either a file identification (fid) or the device identification, ME. Whenever it encounters the latter, UTS will request that the user type in his source program a line at a time. To signal end of input, the user depresses the BREAK key, (see 5. below).

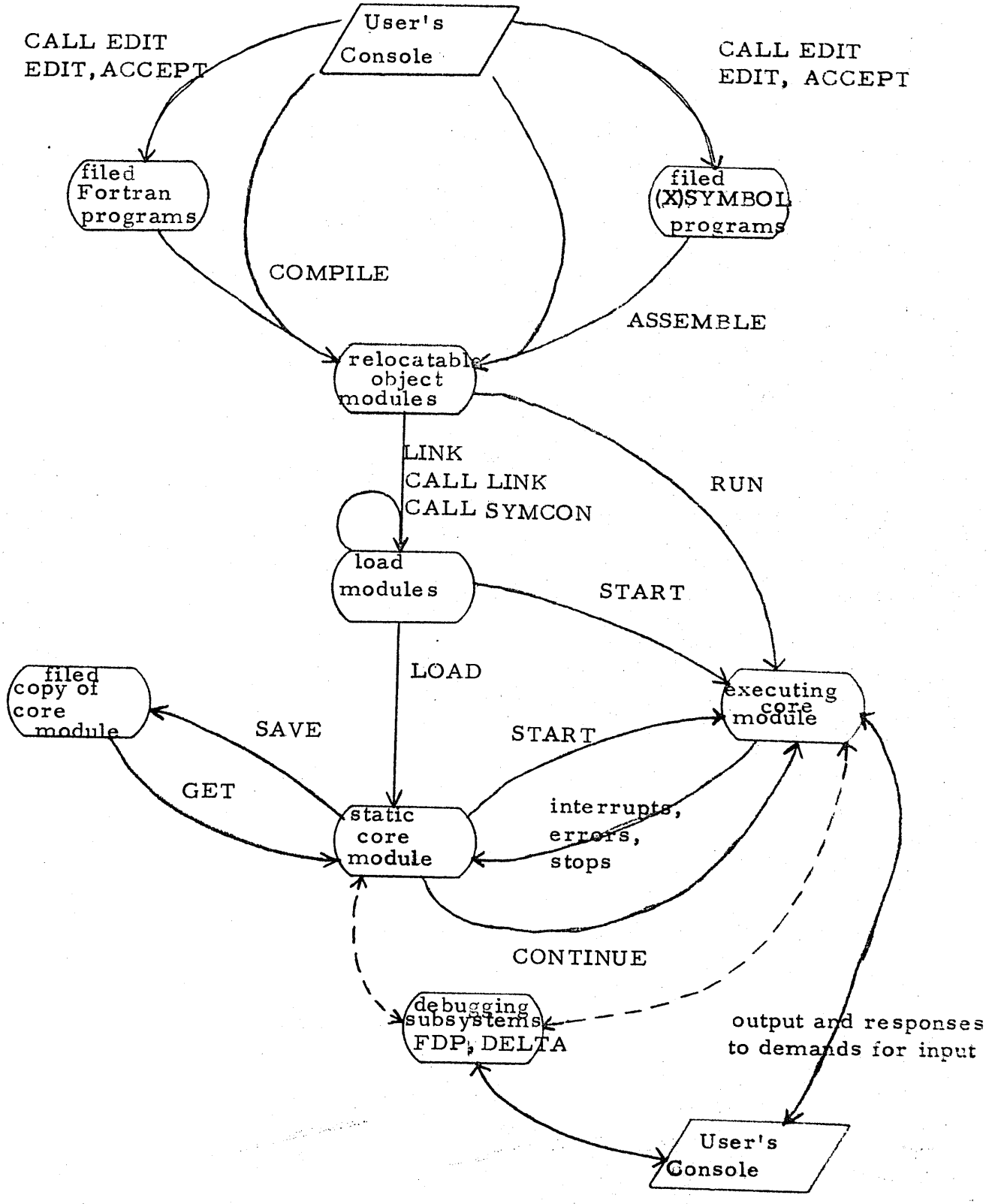


Figure 3. FORTRAN and ASSEMBLY-LANGUAGE PROGRAMMING

Listing output (list) may be directed to a file, the terminal or a line printer (fid, ME, LP). ROM output (denoted rom) may be directed to a file or may be unspecified. In the latter case, UTS caches the ROM on a scratch file, which the user may subsequently refer to by a dollar sign ($).

2. Commands

COMPILE sp, sp, ..., sp ON rom, list

ASSEMBLE sp, sp, ..., sp ON rom, list

Listing and ROM output may be specified beforehand.

LIST ON list (or OVER an existing file)

OUTPUT ON rom

When so specified, the commands can be abbreviated; e.g.,

COMPILE file1, file2

COMPILE file1, file2 ON romfile3

Listing specification hold over all subsequent operations until changed or until specifications accompany a compile or assemble command; specifications that accompany a command hold only for the duration of the command. Beforehand ROM output specifications hold only for the next assembly or compilation, and ROM output files are closed after each operation; that is, no "file-extension" in the BPM sense is permitted. The parenthesized letter "E" after COMPILE or LIST indicates that assembly-language expansions of FORTRAN statements are to accompany the listings; normally, they do not.

3. Controlling Error Commentary and Outputs

Error commentary is always directed to the user's terminal and always accompanies listing output if specified. During the course of a compilation or assembly, the user may interrupt the process to turn output on or off, to turn off error comments to his terminal, and to redirect error commentary.

LIST	or	DON'T LIST
OUTPUT	or	DON'T OUTPUT
COMMENT	or	DON'T COMMENT
COMMENT ON list		
COMMENT OVER fid		(over an existing file)

The facility for turning off and redirecting error commentary is one that can only be appreciated by assembly-language programmers and debuggers who have sat at an on-line console, wringing their hands in desperation while the machine chatters on and on about an error that they could either ignore or repair instantly once they began debugging. Once the user has redirected things to his satisfaction, he can request that processing continue by typing

CONTINUE

In the event that things are hopelessly messed-up, the user can tell UTS to give up on the operation by typing

QUIT

4. Error Handling and End Actions

Whenever UTS aborts an action, either because it cannot be continued or because the user has told it to quit, UTS will always clean up things before reporting and returning control to the user. In particular, after assemblies, compilations, and linkages and loadings (see the sections below):

a) If listing had been specified beforehand, both the listing itself and the specification are retained;

b) Any listing specified with the command itself is erased (unless it has gone to the printer) and specifications return to their default state: DON'T LIST;

c) Any ROM output is erased, and ROM output specifications revert to the default state: DON'T OUTPUT

d) Specifications for error commentary revert to the default state: COMMENT ON ME (but accompany listings if specified).

The same cleaning up is done whenever the user has interrupted an action and asked TEL to carry out a different one before telling it to quit. This holds only for the so-called major commands: ASSEMBLE, COMPILE LINK, LOAD, COPY, and others that require the services of major processors or of sub-systems.

Whenever a major operation has been carried to completion, TEL will notify the user that it is done by printing

DONE

before returning control to him. All specifications and options associated with the operation or command return to their default states except for beforehand listing specifications, which are retained.

5. Entering Programs From the Terminal

Whenever the input designator, ME, is encountered, TEL types:

ENTER PROGRAM

and then returns the carrier to the left margin of a fresh line to await the user's first program statement. Each statement is terminated by a carriage return or line feed. Error commentary, if any, follows immediately to the user's terminal. To indicate the end of his source text, the user depresses the BREAK key. For purposes of formatting, print columns on the terminal's platen are in on-to-one correspondence with card columns, and trailing blanks are assumed for short lines. To facilitate typing of commands and statements, TEL will assume that the terminal's tab stops are set to conform to the programming language being used, and will so simulate them if tab-stop simulation is in effect. For FORTRAN, a single tab stop at print column 7 is used; for assemblies, tab stops are set at columns 10, 20 and 37. The general handling and simulation of tab stops is covered completely in the COC Functional Specifications. Briefly, tabs are simulated so that longer fields can be used: whenever a tab stop is typed-over rather than tabbed-to, the next tab is replaced by a single space. On input, tabs accompany the source statements literally, and assemblers and compilers will treat them properly.

6. Debugging Information

ROM outputs of both compilations and assemblies always contain information required for subsequent debugging at the assembly-language level under DELTA. To debug FORTRAN-produced programs under FDP, further information must accompany the compiled code. In the absence of other specifications, this information will always be produced by the compiler. Such information increases the size of object programs and slows them down. To turn off the production of this information for a specific compilation, the user follows the verb COMPILE by a parenthesized letter "N".

B. Linking ROM's and LM's to Form LM's

ROM's are representations (of programs and data) that are specifically designed for efficient combination with other ROM's; LM's are representations designed for efficient translation into executable programs and loading into core. Both may be pictured as bodies of potential machine code to which are appended so-called symbol-tables. Symbol tables list the correspondences between the symbolic identifiers used in the original source-program and the values or virtual core locations that have been or will be assigned to them. Many of these symbolic identifiers are used and referred to solely within the module itself; these are the so-called internal symbols of the module. Others, the so-called external or global symbols, either identify objects within the module that may be referred to in other modules or are used to refer to objects defined within other modules. Functionally, these modules are black boxes with labelled connectors dangling from them, some pointing out and others in. The labels are the global symbols associated with the module;

the internal connections have all been potted, and are hidden. The process of linking modules together is one of "making big ones out of little ones." In the process, internal symbols associated with the new module's constituent parts are potted and hidden, but all global symbols are still visible. If the resultant module is to be itself recombined with other modules to form yet larger pieces, it is often necessary that it be repotted in such a way that those global symbols used solely for connecting its original constituents either be renamed or be made internal to itself so that conflicts with external symbols of other modules be circumvented. The sub-system SYMCON, described in a separate document,* provides users facilities for such renaming and repotting. These facilities simplify the construction of large programs, since they permit sub-programs to be linked freely in the face of conflicting naming conventions.

Continuing the black-box analogy, if a module is slit open, a jumble of internal connections should be visible. If the module has been tested and deemed fit for production, these connections need not be labeled. However, if the module is still in the debugging stage, the labels may be necessary. To this end, TEL permits users to specify when the internal symbols associated with a module being linked are to be kept with the resulting load module.

1. Simple Linkages

Both ROM's and LM's may be linked. Their identification (mfl) may be a fid or the dollar sign ($) which refers to the ROM produced by the most recent compilation or assembly.

The sub-system, LINK, is specifically designed for linking and is described in a separate document. However, most commonplace linkages can be carried out directly in TEL.

LINK mfl, mfl, ..., mfl ON lm
LINK mfl, mfl, ..., mfl OVER lm (over an existing file)
LINK mfl, mfl, ..., mfl (for subsequent loading into core)

The result of any linking operations is always available for subsequent loading into core whether specified or not (see C. below).

*Drawing Number 702477.

2. Load Module Symbol Tables

A load module can be pictured as being comprised of three parts: a) a body of code; b) a table of global symbols; c) a table or set of tables of internal symbols, each associated with a specific input module and identified by that module's file name. This identification permits users who are debugging under DELTA to define which set of internal symbols are to be brought into play for their debugging activities. What happens to these sub-tables associated with a load module when the module is relinked with other modules is described in 3. below.

The mechanisms for specifying when an input module's internal symbols are to be kept with the resulting load module follow:

a) The parenthesized letters "NI" preceding an input module's file identification in the LINK command specifies that internal symbols for that module are to be left out; the parenthesized letter "I" indicates that internal symbols are to be kept.

b) Once given, a specification holds for all subsequent modules mentioned in the command until the occurrence of a new specification.

c) In the absence of any specifications at all, all internal symbols are kept.

3. Merging Internal-Symbol Tables

Keeping each constituent's internal-symbol table distinct and uniquely identified in a load module makes sense when common naming conventions have been repeated in programming the constituent modules; i.e., when objects internal to distinct modules are frequently identified by the same symbolic identifier. When non-conflicting naming conventions have been used, the user may instruct TEL to merge his specified symbol tables into a single one in the resulting load module. This is done by enclosing the list of input modules named in the command in parentheses. Only one level of parentheses nesting is allowed and either all or none of the input modules may be merged. This convention was

adopted in favor of, say, choosing a distinct command for the process, to maintain uniformity with the conventions of the LINK sub-system. TEL resolves multiple uses of internal identifiers by assigning to them the object that they identify in the last input module with which they were associated (reading from left to right within parentheses). When a load module containing separate internal-symbol tables is itself linked in any way, TEL will merge its sub-tables into a single one before carrying out the linkage.

4. Searching Libraries

To resolve any dangling identifiers, users may indicate the order and identification of libraries to be searched after all input modules have been linked. Libraries are identified by account, and library identification (lid) is identical to account. The list of lid's separated by commas is appended to the list of mfl's in the LINK command, and is separated from that list by a semi-colon. for example:

LINK mlf, mlf, ..., mlf; lid, lid, ...

In the absence of any other specifications, a special UTS library will be searched to resolve dangling identifiers, usually those associated with FORTRAN compilations. This is done after all libraries specified by the user have been searched. To turn off this final library search, the user follows the command verb by the parenthesized letters "NL".

5. End Action and Error Reporting

Options governing error displays are given immediately after the verb, LINK, as a parenthesized code or list of codes:

ND or D mean	do not or do display dangling identifiers
NC or C mean	do not or do display conflicting identifiers
NM or M mean	do not or do display complete loading map

The normal options are D, C, NM. After any displays, TEL types "DONE" and then returns control to the user.

C. Loading LM's Into Core

Two forms of the command are provided:

LOAD lm

LOAD

The second loads the result of the last linking operation, which is always available for loading even when no output file was specified in the LINK command. The user may specify when copies of the internal symbols are to be carried with the loaded LM by following the command verb with a parenthesized letter "S". Normally, internal symbol tables are not "loaded," but global symbols are always "loaded" unless turned off by the parenthesized letters "NG".

LOAD(S) ...	globals and internals
LOAD(NG)...	neither globals nor internals
LOAD(S) NG) ...	internals but no globals
LOAD(NG) (S) ...	

D. Initiating Execution

To start execution of the loaded LM, the user types

START

To load and start execution of an LM, the user types

START lm

To load and start the result of the last major operation (assembly, compilation or linkage), the user types

RUN

To link, load and start execution of a set of modules, the user types

RUN mlf, mlf, ...

All options of the LINK and LOAD commands may be exercised in the RUN command, in exactly the same manners. Normal options are the same except that a RUN under DELTA or FDP (see E. below) always loads internal and global symbols with the load module.

RUN(S)(I) file1, file2, (NI) file3

Requests that three files be linked, loaded and started. Internal symbols for the first two only are to be kept with the resulting load module; all internal symbols kept with the load module are to be "loaded" with it.

E. Initiating Debugging Operations

Execution of programs can be started under control of either of the two debugging sub-systems, DELTA or FDP.

RUN		DELTA	(for assembly-language debugging)
...	UNDER		
START		FDP	(for FORTRAN debugging)

Once the programs have been loaded into core, control passes to the designated debugging package which notifies the user and then awaits his orders.

The debugging sub-systems may also be called when execution has been initiated without them; usually after an interruption by the user or an error comment by the system.

```
CALL DELTA
CALL FDP
```

It must be clearly understood that FDP can do little more than display and re-assign values if applied to programs that have not been compiled under the debugging option, and can't even do that if symbol tables have not been loaded.

F. File Management

A few simple operations on disc files can be carried out directly in TEL. Full file-management and information-transfer capabilities are provided by the PCL sub-system. In TEL, disc files may be copied ON new files the printer or the terminal, may be copied OVER an existing file, and may be deleted

COPY fid OVER fid
COPY fid ON fid or PTR or ME
DELETE fid

Once started, deletions and copies on or over a disc file cannot be interrupted by the user; copies to a printer or to the terminal will be aborted by interruption. TEL will type

REVOKED BY INTERRUPT

in such cases. When an operation is carried through to completion, TEL prints

DONE

before returning control to the user.

G. Editing

Line-at-a-time composition and editing of files of sequentially numbered lines is provided by the EDIT sub-system, which can called in three ways.

CALL EDIT
EDIT fid (an existing file)
ACCEPT fid (a new file)

In the first two cases, the user is connected to EDIT, which identifies itself before returning control to the user. In the second case, EDIT has already been apprised of which file is to be edited, and has opened that file for updating. In the third case, EDIT assumes that the user wishes to type in a new file, a line at a time, beginning with line number 10 and continuing in steps of 10.

EDIT responds by printing each line's number at the left margin and then waiting for the user to type in the line itself. Although EDIT is invisible to the user during this operation, it is explicitly available to him for corrections and other editing operations. To end the operation of accepting a new file, the user must depress the BREAK key to interrupt EDIT, and then type

END

after EDIT returns control to him.

H. Submitting Batch Jobs

Control-card programs destined for submission to the batch queue can be composed and filed away on-line in the EDIT sub-system. These may then be submitted to the batch queue:

BATCH fid

UTS will respond by assigning the batch job an identification (jid) and notifying the user:

JOB jid SUBMITTED date-time

The procedure for assigning priorities to remotely-submitted batch jobs will be defined concurrent with the development of the Remote Batch Functional Specification which is in process. The user can interrogate the status of remotely-entered jobs by typing:

JOB jid ?

At the very least, UTS will be able to tell the user whether the job has completed or whether it is still in queue. The user can cancel an unfinished or unstarted job:

CANCEL JOB jid

I. Calling Sub-Systems

All sub-systems are called by typing the verb "CALL" followed by the sub-systems identification; e.g.,

CALL PCL

All sub-systems will respond by identifying themselves; e.g.,

PCL HERE

and then typing their identifying mark at the left margin of a fresh line before returning control to the user. All sub-systems are described in separate parts of these specifications. The marks are: EDIT (*), PCL (<), FDP (/), SYMCON (:), LINK (:),BASIC (>), none for DELTA.

J. Continuing and Quitting Major Operations

Whenever a major operation, a sub-system or an executing user's program has been stopped or interrupted in any way, the user can:

1. Take any of the minor actions described in the section below, and then request TEL to continue from the point of interruption by typing

CONTINUE

or from a point identified by a global symbol (or a hexadecimal symbol) by typing

CONTINUE symbol

2. Give up completely on the operation by typing

QUIT

In the latter case, TEL cleans things up and then returns control to the user.

3. Initiate a new major operation. In this case, the effect is as if he had told TEL to QUIT before giving the new command. The sole exception to this rule of automatic QUITting occurs when the user calls one of the debugging systems (DELTA, FDP) during execution of his program. In this case the user's program will have to be initiated again under control of the debugging system.

MINOR OPERATIONS

A. Checkpointing Sessions

During interruptions of execution, core images of programs may be saved on the disc files for subsequent recall and continuation. To save and file away a core image:

SAVE ON fid

SAVE OVER fid (over an existing file)

The current status of the user's files is not copied, and the user must be aware of any on-going but interrupted input-output activities. In brief, checkpointing will work well so long as the user knows what he is doing. To recall a checkpointed core image for continuation, the user types

GET fid

At this point, the user is -- to within file changes and input-output activities -- exactly where he was when he SAVEd. TEL will respond to both commands by typing

DONE

when it has finished.

B. Assigning Files and Input-Output Devices

The assign command of BPM is provided in a simplified and restricted form, mainly to allow users to connect files and input-output devices to their running programs. Devices and files are equated in users' programs to so-called data-control blocks (dcb) that make the programs "device independent". Assignment of specific devices and files to dcb's can be made at any stage of the game, even after execution has begun. TEL will notify the user whenever it encounters an unassigned dcb during execution of a program by typing

ASSIGN dcb = ?

The user assigns things to dcb's by typing

ASSIGN dcb = fid or device code or tape reel

INDEX OF COMMANDS - TEL

ACCEPT	Calls EDIT and accepts a new file from the terminal
ASSEMBLE	Assembles specified source program
ASSIGN	Assign file or device to a DCB
BATCH	Enter specified file in batch jobstream
CALL	Make the specified sub-system available
CANCEL JOB	Cancel the designated batch job
COMMENT	Directs error commentary to specified device
COMPILE	Compiles Fortran source program
CONTINUE	Continue processing from point of interruption
COPY	Copy a file to specified device
DELETE	Delete the specified file
EDIT	Calls the EDIT sub-system
GET	Restore previously saved core image
LINK	Form load module as specified
LIST	Directs listing output to desired device
LOAD	Bring designated load module into core
OFF	Disconnects user from system
OUTPUT	Directs object output to specified device
QUIT	Terminate current operation
RUN	Load specified load module and start execution
SAVE	Save current core image on designated file
START	Begin execution of program just loaded

VII. Text Editing Sub-System (EDIT)

TABLE OF CONTENTS

INTRODUCTION

The UTS sub-system, EDIT, is a line-at-a-time text editor designed for on-line creation, modification, and handling of programs and other bodies of information. All EDIT data is stored on disc in a special file structure called Compact. This structure (described in a separate section) permits EDIT to directly access blocks of data EDIT functions are controlled through single line commands supplied by the user. The command language provides for the insertion, deletion, and replacement of lines of text. Selective printing and renumbering commands, and a command to simplify commentary editing for assembly-language programs are included. File maintenance commands are also provided to assist the user.

A. Calling EDIT

An on-line user of UTS may call EDIT using one of three commands provided in the Terminal Executive Language (TEL).

1. CALL EDIT
2. EDIT an existing file
3. ACCEPT a new file

In all cases the EDIT sub-system is brought into play. The first case represents a direct call to EDIT, which in turn, responds by typing "EDIT HERE" and then by typing its identifying asterisk (*) at the left margin of a fresh line. At this point, the user may enter his next command. The second case allows the user to call EDIT for the purpose of updating an existing file. EDIT first opens the specified file and then responds to the user as in the first case.

The third case permits the user to call EDIT for on-line creation of a text file. EDIT opens the specified file and responds to the user by typing the first line number at the left margin of a fresh line. The user is then expected to enter the first line of the new file. "EDIT" and "ACCEPT" are included as part of the EDIT command language and are described further below.

B. Operational States

EDIT, as a processor, operates in one of two states: the command state or the active state. The command state is defined as the time in which EDIT is accepting or processing a command. This state is entered when EDIT types its identifying asterisk (*), returns control to the user, and awaits the next command. On the other hand, the active state is defined as that time in which EDIT is executing commands, processing text, or accepting text from the user. This state is entered when a command starts execution and terminates at the completion of the command. When carrying out a command, EDIT may be processing information while in control of the keyboard, or may have returned control to the user so that he may enter text data. Which of the two situations holds is always clear to the user as described under Text Modification.

C. Summary of Commands

The notation "fid" in the following definitions represent file identification in the form: name-account-password.

1.	COPY fid ON fid (n, k)	Copies an existing file to a new file. The specification (n, k) allows resequencing of line numbers starting at n in increments of k.
2.	COPY fid OVER fid (n, k)	Same as above with the exception that an existing file is copied over.
3.	DELETE fid	Deletes an existing file.
4.	EDIT fid	Specifies an existing file to be updated.
5.	ACCEPT fid (n, k)	Defines a new file for creation. (n, k) indicates desired line number sequencing.

6.	I n,k	Inserts lines of text starting at line number n in increments of k. EDIT displays each line number and awaits text data. The first line n may be a replacement.
7.	I(N) n,k	Same as above except line numbers are not displayed.
8.	D n,m	Delete lines n through m inclusive.
9.	P n,m	Print at console, with line numbers, line n through m inclusive.
10.	P(N) n,m	Print at console, without line numbers, lines n through m inclusive.
11.	R n,m	Re-numbers line n to the new number m.
12.	C n	Insert commentary for an assembly language program starting at line n. EDIT prints each line number and awaits the commentary text.
13.	T F/M/S	Set software tab stops for Fortran (F), Symbol and Meta-Symbol (M), and a short form of Symbol and Meta-Symbol (S).

DESCRIPTION

A. Conventions and Terminal Operation

For purposes of clarification, certain conventions have been adopted throughout this document. These, concatenated with associated terminal operations, are given below:

1. Underlined copy in the example is that generated by the computer. Copy not underlined represents that typed by the user.

2. The line numbers displayed by the Insert (I) or Comment (C) commands are always right justified and blank filled to five characters. Thus, the first input position (card column 1) is displaced to column 7 on the teletype. For purpose of the examples, only the significant (rightmost) portion of the display is shown as computer generated copy.

3. Control characters are represented in this document by an alphabetic character and a superscript c, e.g., E^c. The user simultaneously depresses the alphabetic key and the Control key (CTRL) to obtain this function.

4. Carriage Return. The Cr notation following each line in the example represents a carriage return. Depression of this key informs the computer that an input line is terminated. A carriage return (Cr) will automatically cause the computer to give a line feed. The line feed (Lf) key operates identically to the Cr for the EDIT processor.

5. Escape (E^c). This key enables the user to temporarily escape to the executive command level. Escape may be applied at any time when the user has control of the keyboard. The current status of EDIT is retained and may be re-activated using the executive "CONTINUE" command.

6. RUBOUT The last input character may be deleted with this key. A\ is echoed to the user. N RUBOUTS echo N\'s and delete the previous N character

7. Cancel (X^c) Application of this key cancels the current input line. A← is echoed to the user followed by a Cr and Lf.

8. BREAK This key, indicated by Bk, causes an automatic interrupt in current EDIT activities. When applied during the command state, the current command is ignored as if X^C had occurred. Application during the active state causes EDIT to terminate what it is doing, pass control to the user, and revert to the command state. A Cr response is given if used during input. Effects of the interruption or the termination vary with the command being executed and are discussed in detail with the particular commands. If no mention is made, Bk is assumed to have no effect on the execution of that command.

B. Defining the Edit File

Prior to utilizing text editing commands, the user must first identify a file to be used. The "EDIT" and "ACCEPT" commands perform this function and are available at both the executive and EDIT command levels. When applied at the executive level, EDIT is automatically called and apprised of the command parameters. At this point, the source of the command is irrevelevant to its operation, and the following descriptions are given in this context. An existing edit file is forced closed by either of these commands.

1. ACCEPT To create a new text file through EDIT, the user types a command of the form:

ACCEPT fid (n, k)

Where fid represents name-account-password of a new file, n is any line number, and k is a line number increment. After receiving this command, EDIT checks the file specification against the user's file directory. If a match is found an appropriate diagnostic is given to the user and he must re-enter the command. Otherwise, the file is opened and EDIT executes the implied insert (I n, k). This is indicated when EDIT responds by typing the first line number (n) at the left margin of the next line and awaits the first line of text. For example:

* ACCEPT SOURCE (10, 20) Cr
10

The file named "SOURCE" is opened under the user's account with no password. An insert (I 10, 20) is then simulated giving the line number response (10). The user may now enter his first line of text. The Insert (I) command describes the subsequent steps for file creation in detail.

2. EDIT Updating of an existing file requires a file definition of the type:

EDIT fid

where fid represents name-account-password of an existing file. The file specification here is also checked against the user's directory. An appropriate diagnostic is given if a match is not found and the user must re-enter the command. If a match is found, the file is opened for updating, and the user may type his next command following the identifying * of the command state.

For example:

*EDIT SOURCE--PLEASE Cr
*

After receiving this command, EDIT located the file "SOURCE" under the user's account. The password "PLEASE" was compared with password of the file and "SOURCE" was opened for updating. EDIT then re-entered the command state suggested by the * response.

C. Text Modification

Once the user has defined the edit file, he may desire to make insertions, deletions, or replacements to update his file. Several commands are available in EDIT to assist him in these operations.

1. Tabs (T) In addition to the standard algorithm for setting software tabs in UTS, this command permits the user to set software simulated tab stops for that input or output in Fortran, Metasymbol, or Symbol format. The command is of the form:

T F
  M
  S

where F implies Fortran (column 7), M implies Meta-Symbol or Symbol (columns 10,19,37) and S implies a short form of Meta-Symbol (columns 8,16,30). After receiving this command, EDIT transmits the required settings to COC routines which perform the actual tab simulation. The user can turn on and off tab simulation with special terminal key strokes as described in the Terminal Executive Language specification (TEL). The settings transmitted are displaced by six characters to allow for line number display giving F(13), M(16,25,43) and S(14,22,36). Consider the example:

* T M Cr
*

EDIT transmits the settings for columns 16,25,43 corresponding to actual input columns 10,19,37 for Metasymbol.

2. Insert (I) The insert command may be used to insert or create one or more lines of text; or to replace a line followed by one or more insertions. The general format is of the form:

I n,k

where n is any line number and k is a line number increment. Upon receiving this command, EDIT enters a special "accept text" mode in which one or more lines of text may be inserted or created starting at line n and proceeding in sequential steps of k. EDIT prompts the user to type each line by displaying the associated number at the left margin. The user then enters desired text following the line number display, terminating the line with a Cr. If the first line (n) exists, it will be replaced, however, successive line numbers, created by increments of k, must not overrun the number of the next existing line. Termination of the "accept text" mode is accomplished by replacing the last Cr with Bk. Note also that if tabs are to be used with input text, the settings must take into account the 6 character displacement required for line number display.

Consider the example:

```
*I 30, 2 Cr
30 ONLY THE FIRST LINE CAN BE A REPLACEMENT Cr
32 Cr
34 LINE NO. 32 WAS SKIPPED Cr
36 END INSERT WITH THE BREAK Bk
*
```

The line (if any) with the largest number ≤ 30, say X, is located in the current edit file. Lines 30, 34, and 36 are then inserted following X if X<30 or replacing X if X = 30. The line number 32 was purposely skipped to allow for a future insertion. If the next line in the file had been ≤ 36 an overrun error would have occurred. Note that the insert command was terminated by the Bk key ending 36. Single line insertions or replacements can be accomplished by omitting the increment k as in the example

```
*I 50 Cr
50 SINGLE LINE INSERT OR REPLACEMENT Cr
*
```

Line 50 in the edit file is located. If it exists, it is replaced, if not, the line is inserted in proper sequence. Sometimes it is desirable to suppress line number feedback in order to obtain extra length lines, or just as a convenience when entering relatively error free text. The parenthetical expression (N) following the command letter I suppresses the line number displaying. For example:

```
*I (N) 10, 10 Cr
LINE NUMBER DISPLAY MAY BE SUPPRESSED Cr
USING THE (N) OPTION Bk
*
```

The user is prompted for each line of text by output of a bell character. EDIT operates using the implied line numbers and thus, merges lines 10 and 20 into the edit file.

3. Renumber (R) Use of this command permits the user to renumber a line that would normally be overrun during an insert (I) operation. The command is of the form:

R n, m

where n is any existing line number and m is the desired line number. EDIT locates line n within the edit file and replaces the line number with m. The command must not take the line out of proper monotonic sequence. If n does not exist, the user is notified by an appropriate diagnostic. Consider the example:

```
* R 30, 39 Cr
*
```

The existing line 30 is located and renumbered to 39. This allows nine additional numbers for insertions. If the next existing line after line 30 had been $\leq$ 39, the command would have been ignored and an error diagnostic given.

4. Delete (D) The user may delete one or more lines of text through the following command:

D n, m

where n and m are any line numbers with n $\leq$ m. All lines X with n $\leq$ X $\leq$ m will be deleted in sequence from the edit file. For example:

```
* D 30, 36 Cr
*
```

This command causes lines 30-36 to be deleted from the file. If no lines had existed within this range, the command would have been ignored. Single line deletions may be specified by omitting the second line number, for example:

```
* D 10
*
```

Line 10 located and deleted from the edit file. If line 10 had not existed, the command would have been ignored.

5. Print (P) This command enables printing of one or more lines of consecutive text at the user's console. The command is defined as follows:

P n, m

where n and m are any line numbers with $n \leq m$. After receiving this command, EDIT prints all lines with number X, where $n \leq X \leq m$, in sequence at the user's console. Associated line numbers are printed at the left margin of each line. Execution of P may be terminated following the output of the current line by depressing the Bk key. Assuming the insert (I) example described above still exists, consider the following:

```
* P 30, 36 Cr
30 ONLY THE FIRST LINE MAY BE A REPLACEMENT
34 LINE NO. 32 WAS SKIPPED
36 END INSERT WITH THE BREAK
*
```

Lines 30, 34, and 36 were located within the specified range and were printed with corresponding line numbers. Line numbers may be suppressed from the output as in the following example:

```
* P (N) 30, 36 Cr
ONLY THE FIRST LINE MAY BE A REPLACEMENT
LINE NO. 32 WAS SKIPPED
END INSERT WITH THE BREAK
*
```

In addition, single line printouts may be obtained as follows:

```
* P 34 Cr
34 LINE NO. 32 WAS SKIPPED
* P (N) 34 Cr
LINE NO. 32 WAS SKIPPED
*
```

6. Comment (C) This command allows the user to insert or delete the comment field of program code formatted for Symbol or Meta-Symbol. The command takes the form:

C n

where n is any line number. As in the insert command (I), EDIT enters a special "accept text" mode. Line numbers of existing lines, starting at the first line X, where X = n, are displayed in sequence for possible comment modification. Following each line number display, the user is expected to supply a comment, indicate no change, or delete the comment field. A comment is entered in the same manner as any other line of text. A blank line (one or more blanks) causes the comment to be deleted, and a null line (Cr-only response) indicates no change in the comment field. Termination is implied by depressing the Bk key instead of the Cr on input. Assuming the edit file consists of a Meta-Symbol or Symbol program, consider the example:

```
*C 10 Cr
10 TEST FOR ZERO Cr
20 b Cr
30 Bk
*
```

The comment at the existing line 10 was replaced by "TEST FOR ZERO". Blanks were entered into the comment field of line 20 and the third line 30 resulted with no change. Bk was depressed following the display of the number 30 which flagged EDIT to return to the command state after completion of the current operation.

D. File Maintenance

Two commands are available within EDIT to assist the user in file maintenance. They permit copying, deleting or renumbering of complete edit files (Compact type) retained on disc.

1. COPY To copy an existing file for the purpose of backup or renumbering, the user enters the following command:

COPY fid ON fid (n, k)

where the first fid represents name-account-password of an existing file; the second fid represents name-account-password of a new file; and (n,k) tells EDIT to resequence the line numbers starting at n in sequential steps of k. Depression of the Bk key during execution of this command causes the copy operation to be cancelled. A common example could be:

```
* COPY A ON B (10,10) Cr
*
```

File A was located in the current user's directory and opened for input. File B was opened as a new output file. EDIT then copied the complete file A to B, renumbering each line starting at 10 in sequential steps of 10. Upon completion of the copy, both files were closed and EDIT reverted back to the command state. The command also permits copying over an existing file. This is demonstrated by the example:

```
* COPY C-0986 OVER A Cr
*
```

In this case file C under the account 0986 is written over the existing file A, under the account number of the current job, with no resequencing of line numbers. A Bk character cancels the operation with no effect on the existing file A.

2. DELETE files may be deleted using this command which is of the form:

   DELETE fid

where fid represents name-account-password of an existing file. Following the entry of this command, a con firmation message of the form "DELETE fid?" is typed. The user must then type YES to confirm the deletion or anything else to cancel it. If YES is entered, the file is deleted and the disc space released. Bk cancels the command if applied prior to the confirmation. For example:

```
* DELETE SOURCE--PLEASE Cr
DELETE SOURCE--PLEASE?  YES Cr
*
```

Upon receiving the command, EDIT located the file in the user's directory and responded with the confirmation message. After the YES reply, the file SOURCE was deleted.

E. Termination of EDIT

In order to complete final file updates on disc, it is necessary for the user to indicate when he has finished with EDIT functions. The command END fulfills this requirement and also returns control to the UTS executive level. Prior to exiting the sub-system a termination message is given to the user. For example:

```
* END Cr
EDIT PROCESSING TERMINATED
!
```

This command closed any open files and forced EDIT to return to the executive command level. The Executive responded with it's identifying mark (!) indicating the command state.

F. Language Syntax

The EDIT command language is designed to be free form, with a few restrictions imposed for simplicity in implementation and use. These include:

1. All commands must comply with the general format of the particular command.
2. Blanks are allowed preceding or following an argument field. Imbedded blanks are not permitted.
3. At least one blank must follow each control command verb (COPY, ON, ACCEPT, EDIT, ...) and also proceed an imbedded command verb (ON). Single character commands (I, C, P, ...) do not require this blank delimiter.
4. Continuation between commands is not allowed.
5. Line numbers must be within the range $1 \leq n \leq 99999$

Each input command is edited for format, content and completeness. The user is notified by appropriate on-line feedback of diagnostic messages. At this point the user may re-enter the command or confirm the diagnostic condition if given that choice by EDIT.

EDIT FILE STRUCTURE (Compact)

Files used by EDIT are stored on disc, in a special structure especially designed for compact storage, fast random access, and easy editing of text. The file structure (Compact) has the following features:

1. The file consists of keyed physical records, 512 words long.
2. Within the 512 word block, sub-records (1-64 words long) are preceded by one word of identification. Byte zero of this word carries the byte count for the sub-record and bytes 1-3 are used for a sub-record identifying number (line number).
3. The identification word is kept on a word boundary to assure that sub-records also begin on a word boundary.
4. The blocking buffer is terminated by a zero identification word.
5. Trailing blanks are removed from sub-record text.
6. The key for each 512 word block is the line number of the first sub-record within that block.

This type of file structure enables EDIT to directly access large blocks (512 words) of text data, and thus substantially reduces I/O time for most updates. In addition, line number identifiers attached to each sub-record permit inserts without resequencing.

INDEX OF EDIT COMMANDS

VIII Assembly Language Debugger (DELTA)

TABLE OF CONTENTS

INTRODUCTION

DELTA is specifically designed for the debugging of programs at the assembly-language and machine-language level. It operates on object programs and the tables of internal and global symbols accompanying them, but does not demand that the tables be at hand. With or without symbol tables, it recognizes machine instruction mnemonics and can assemble, on an instruction-by-instruction basis, machine language programs. Its main business, however, is to facilitate the activities of debugging. These are:

1. The examination, insertion and modification of elements of programs: instructions, numeric values, encoded information -- data in all its representations and formats.

2. Control of execution, including a) the insertion of breakpoints into a program, and b) requests for breaks on changes in elements of data.

3. Tracing execution by displaying information at designated points in a program.

4. Searching programs and data for specific elements and sub-elements.

To assist in the first activity, assemblers and compilers of UTS will include in a program's table of symbols information about what type of data each symbol represents: symbolic instructions, decimal integers, floating point values, single and double precision values, EBCDIC encoded information, and others.

The command language of DELTA is cryptic and highly encoded, but easily learned and used by the professional programmer. It is substantially identical to the DDT language family which has been in use on a variety of machines for the last decade.

Two versions of DELTA will be produced:

1. a user version with codes and restrictions appropriate to multiple on-line users operating in the slave mode from teletype consoles, and

2. an executive version for system debugging which will operate in executive mode under control of one of the operator's consoles. This will not be resident when UTS is in service.

A. Calling DELTA

DELTA may be associated with the execution of a user's program either at the time the user loads his program into core for execution or by direct call after execution has begun. The two executive level commands are:

1. To load the user's program in association with DELTA:

RUN program name UNDER DELTA or

START program name UNDER DELTA.

Control goes to DELTA and the user may examine and modify before passing control to the program.

2. To bring in DELTA after a program has been initiated, the user must return to the executive level by the teletype console command E^C (control shift and E key depressed together), and give the executive command

CALL DELTA

DELTA also may be brought in and started without prior program loading for writing and checking of short simple programs and other purposes.

To make it possible to call DELTA in this way, a segment of virtual address space must and has been reserved for DELTA in high virtual addresses. A similar reservation applies to the executive language processor.

B. Symbol Tables

A program consists of one or more individually compiled or assembled units (ROM's) which have been combined by the LINK process into a load module (LM). During linking, a global symbol table consisting of all symbols which have been so declared by a DEF directive is created for the load module and an internal symbol table is created for each unit (mostly ROM's but some LM's). The loader language allows the user to specify which

internal symbol tables should be retained. Internal symbol tables are named by the file name of the source ROM; that is, LINK writes a symbol table for each ROM input under a key identical to the input ROM name. A simple Link Command is shown below:

LINK A, B, C, (NS) D ON E

In this case, the load module E is created for execution, and symbol tables are retained for units A, B, and C, but not for D. For further examples of linking operations and a complete list of options, see the loader specification.

C. Command Summary

The following summary lists the DELTA commands and facilities in eleven broad groupings:

1. Evaluating expressions consisting of symbols, constants, special symbols, and the operators plus and minus (+-).
2. Commands for printing the contents of memory cells and opening them in preparation for change.
3. Format codes which enable the user to control the output format used in the evaluation and display commands of Group 1) and 2).
4. Commands for storing new contents in open memory cells.
5. Format codes which control the conversion of constants typed by the user.
6. Special symbols used to examine machine flags and to control operating bounds in DELTA.
7. Commands to insert in, delete from, change completely, and otherwise control the symbol table used by DELTA.
8. Commands to initiate and continue execution.
9. Commands to insert, delete, and control breakpoints.
10. Commands for searching memory.
11. Mode setting commands.

In outlining the commands, the following conventions are used in depicting the format of the order typed by the user:

- Special characters, numbers and upper case letters stand for themselves. Thus in the command e;G the user actually types the semicolon and the G.

- Lower case letters are placed where the user has a choice of things to type. The letter e alone or postcripted is used to stand for any expression consisting of symbols, special symbols, constants, and the operators plus and minus (+-). At times other lower case letters are used to stand for expressions when some additional mnemonic content seems desirable. Examples are n, loc, val, m.

- The letter f stands for one of the format characters. Abbreviations for user key strokes are:

Letters used in text	User Keystroke
cr	carriage return
lf	line feed
←	shift and O
↑	shift and N
\	shift and L
tab	control and I
bk	BREAK

Most of the DELTA commands are terminated (and thus delivered from UTS I/O to DELTA) by the carriage return (cr) character; however, certain other characters also delimit commands to allow dialog within a single typed line. The command terminating characters of DELTA are cr, lf, ↑, tab, /, and =.

Whenever DELTA gives control of the terminal to the user for input, it sends its "prompt" character, the bell, to the console.

DELTA Commands

1. Expression Evaluation

e= Evaluates and types the value of the expression e in the most appropriate format.

e(f= Evaluates and types the value of e in format f (see 4 below).

2. Displaying and opening memory cells

e/ Displays the contents of a cell e in the most appropriate format. The cell is open; that is, it may be changed.

e(f/ Displays the contents of cell e in format f.

e1, e2/ or e1, e2(f/ Displays the contents of cells e1 through e2 in the most appropriate format or in the specified format. Cell e2 is opened.

e\ Opens but does not display cell e.

/ Slash alone following a display displays the cell addressed by the display. (Displays the cell addressed by the last quantity typed (;Q))

3. Storing in open memory cells

e cr Stores the word specified by e in the currently open cell and closes the cell.

e lf Stores e in the currently open cell, closes it, and opens and displays the next higher addressed cell.

e ↑ Stores e in the currently open cell, closes it, and opens and displays the next lower addressed cell.

e tab Displays and opens the cell addressed by the last quantity typed (;Q). If an expression precedes the tab it is stored in the open cell.

4. Format codes for / and = commands

F symbol table format type

X hexadecimal words

I signed decimal integer

C EBCDIC characters

R	symbolic instructions with symbolic addresses
A	symbolic instructions with hexadecimal addresses
H	half-word addresses
T	binary (base two)
D	double word decimal integer
S	short floating point number
L	long floating point number
f;/	sets the default format for / commands to f
F;=	sets the default format for = commands to f

5. Input conversions and expressions

Expressions for evaluation, display, and storage are formed from the program symbols, explicit constants, and special symbols using the operators plus and minus (+-).

The conversions that may be specified for explicit constants are: 1) hexadecimal when introcuded by a " ("BAD), 2) EBCDIC characters when surrounded by ' ('BAD'), and 3) decimal when the constant consists of all numerics (1234).

6. Special Symbols

Special symbols are recognized by DELTA and may be used in expressions. Used as commands, they set the value of the corresponding symbol table entry.

$ or .	last opened cell
;I	instruction counter
;C	condition code
;F	floating controls
;M	search mask
;1	lower search bound
;2	upper search bound
;Q	last quantity typed

7. Symbol Table Control

s;S Select internal symbol table s.

;U Display undefined symbols.

e(f<s> The symbol s is assigned location.

s(f! The symbol s is assigned the value of the currently open cell ($) and format code f.

s;K Symbol s is removed from the symbol table.

;K Removes all symbols except instruction mnemonics.

8. Execution Control

e;G Begins execution at e.

e;X Executes the instruction e (executive version only).

;P Proceed with execution.

9. Breakpoints

e, n;B Set the n^{th} instruction breakpoint at location e.

e, n, loc;B
Same as above but display contents of loc when the break occurs.

e, n, loc;BT
Same as above but proceed from the break after printing (trace mode).

n;B Remove the n^{th} instruction breakpoint.

O;B Remove all instruction breakpoints.

e, n, val, m;Dr or e, n, val, m;DT<
Causes a data break to occur whenever the contents of cell e (masked by m) are in relation r to val.

The relations are:

A for all changes in e

L e<val

E e = val

G e>val

GE e≥val

NE e≠val

LE e≤val

n;D	Remove the n^{th} data breakpoint.
O;D	Remove all data breakpoints.
;P	Proceed from the break.
n;P	Proceed and do not break until the breakpoint has been passed n times (instruction breakpoints only).
;T	Proceed automatically from the break after printing. (Set 'trace mode!').
bk	Break at the current execution point (analogous to the machine's stop switch).

Output produced when a breakpoint is reached is n;B>loc where n is the breakpoint number and loc its location (or the location of the instruction modifying the data). If a display is specified (data breaks always display), the output produced is:

n;B>loc	addr/contents
n;D>loc	addr/contents

10. Memory Searching

Memory between the bounds specified in ;1 and ;2 (initially set to the lower and upper limits of user memory) are searched under the mask in ;M (initially all ones).

e;W	Search for and display words which match e under the mask ;M.
e;N	Search for and display words that do not match e.
e;1	Set the memory search lower bound to e.
e;2	Set the memory search upper bound to e.
e1, e;2L	Set ;1 to e1 and ;2 to e2.
e;M	Set the search mask to e.

11. Mode Setting and Other

;R	Display locations of displayed cells as symbol plus relative number.
;A	Display locations as hexadecimal numbers.
e1, e2;Z	Zeros memory from e1 through e2.
;Z	Zeros all user memory.

DESCRIPTION

A. Syntax, Symbols, and Such

The language of DELTA follows the DDT formula of simplified expressions and single letter commands, which holds the number of keystrokes required of the user to a minimum. Because every keystroke counts, only a few error conditions are detected. The most common commands have been assigned to lower case keys in order to simplify typing. The space character follows this line of thought in that it is assigned to mean plus in expressions and so eliminates a shift when plus is desired.

1. Command Delimiters

In order to interface efficiently with the time-sharing system, DELTA has been made "message" oriented. That is, only certain characters are recognized as command delimiters or end-of-message characters and cause UTS to deliver the command to DELTA for interpretation. The characters which are command delimiters are:

/	The open and display command
=	The expression evaluation command
cr	The store command and delimiter of other commands
lf	The store and open next command
↑	The store and open previous command
tab	The store and open indirect command

With the exception of / and =, the commands above cause a carrier return and line feed. The slash and equal commands interact within a single typed line.

2. Fixing Typing Errors

Before giving one of the command delimiters, the user may repair typing errors by rubout (the rubout key prints a \ at the console and erases the preceding character; N rubouts print N \'s and erases the preceding N characters) or he may delete the entire current command by using the cancel key (control shift and X keys pressed together). Note that the current command may be a full line or a partial line -- partial if a = or / command is already complete on the line of the cancel character. In the executive version the BREAK key cancels the command.

3. Symbols

The symbols used by DELTA for reference to memory locations, computing values, and formatted displays are those supplied from the assembly or compilation of the program plus any added from the terminal by the user. They are carried in DELTA's symbol table as seven characters plus count. Symbols longer than seven characters are truncated to include only the first seven, although the count of characters is retained. Thus, symbols which were originally longer than seven characters and have both the same length and the same initial seven characters are indistinguishable from each other and only the last received definition is retained.

The symbols used by DELTA follow the same rules as those for Symbol and Meta-Symbol -- they are made up of the alphabetic characters A-Z, the numerics 0-9, and the specials $, @, #, :, _, ←; at least one must be non-numeric; and the number of characters must be less that 64. DELTA however only retains the first seven characters and the total count.

Symbols have an associated type code which allows DELTA to use a conversion for display that matches the symbols original use. The types are at least the following and perhaps others as the need arises. Symbols have either a) constant value or b) are associated with a memory location. If the latter is the case, then the type code describes the contents of the location.

a) Instruction
b) Integer
c) EBCDIC Text
d) Short floating point number
e) Long floating point number
f) Decimal number
g) Packed Decimal number
h) Hexadecimal

The default mode of the display command will be to examine the symbol table for a symbol at or with next smaller location value that that requested and use the conversion type given. This means that a memory dump of a machine language program would resemble closely the original source symbolic.

4. Special Symbols

The initial contents of the symbol table include the mnemonic names of Sigma 7 machine instructions and a list of special symbols associated with program debugging. The special symbols may be used in expressions for values. The special symbols and values associated are given below.

Symbol	Value
$ or .	Memory location of the last opened cell.
;I	Instruction counter contents at program interrupt.
;C	Condition code contents at program interrupt.
;F	Floating control contents at program interrupt.
;M	The mask used in memory searches.
;1	The lower bound used in memory searches.
;2	The upper bound used on memory searches.
;Q	The last quantity type by DELTA. Or the value stored by the user with the commands cr, lf, and tab.

Except for $, ., and ;Q the value of these symbol table entries can be set using a special command form in which a defining expression is given followed by the semi-symbol to be set and a carriage return:

"46B;I cr	Set ;I to hex 46B
"FFF;M cr	Set ;M to hex FFF
100;1 cr	Set ;1 to decimal 100

The value of all special symbols may be displayed using the = command.

```
;C = 4
;I = "3BD
;F = 2
```

The symbols $ and . always carry the location of the last opened cell as their common value. The shorthand is convenient in the same way as in symbolic assembly code.

```
A/ LW,4 K45 $(X="105 $/ LW,4 K45
```

The ;Q shorthand for the last thing typed is similarly conveniently in special situations:

```
B/AI,5  7    ;Q+2  cr
./AI,5  9
```

5. Input of explicit constants

When the user wishes to type in numbers he must specify the conversion that he wishes made on his input. Three conversion types are provided by DELTA: hexadecimal obtained by introducing the constant with a double quote mark ("), EBCDIC obtained by enclosing the characters in single quotes ('), and decimal the conversion used on strings of numerals. Within EBCDIC text strings the characters /, ↑, cr, lf, tab ' are not allowed.

Some examples of input constants in various formats are:

```
"ACE    100  "100  14  "A
'EBCD'    'A'
```

Note that the single quote (') is required to terminate the EBCDIC text string, and that it must consist of four or fewer characters. If fewer than four they are right-justified.

```
"ACE  -34
"100  +100
"3FF + 'wxyz'
```

6. Expressions

Expressions are typed by the user for location value, parameter value, and to be assembled into an instruction. Expressions are composed of a) symbols, b) explicit constants, and c) the operators plus, minus and space. Multiplication, division and other operations are not allowed and in fact the characters usually used to indicate them are used for other things -- the asterisk to indicate indirect addressing in instructions and the slash as the command for display.

The user will have little trouble constructing legitimate and correct expressions for the values he wishes as can be seen from the examples below:

```
A
A+3
A+3-B
AI,1 2
STW,7 *LOC
LW,7 TAB,5
CAL1,3 LIST
```

The space character, in addition to its use to introduce the address field in expressions to be assembled into instructions, is also used to mean plus (+). This convention is convenient for a lazy typist as space does not require the case shift that plus does. Thus some equivalent expressions and commands are:

```
A 3 and A+3
LW,5 ALPHA+3 and LW,5 ALPHA 3
A+3,A+9;L and A 3,A 9;L
```

Just exactly how DELTA accomplishes its expression evaluation is described in the next section.

7. Expression Evaluation Algorithm

Expression evaluation accumulates values into four cells 1, 2, 3, and 4 from four possible expression fields. Receipt of a comma advances the accumulation to the next field; when a space is received the accumulation goes to field 3 (the address field if the expression is to be an instruction) unless it is already there or beyond. Plus and minus cause accumulation in the current field. At the end of the expression the four fields hold values which are combined and used in ways dependent on the command supplied.

The flow chart of Figure 1 gives a rough idea of expression evaluation.

If the command following the expression is a store type (cr, lf, , or tab) then "assembly" of the instruction is accomplished as follows: Field 2 is masked and shifted to the register position (bits 8-11); field 4 is masked and shifted to the index position (bits 12-14); and fields 1 and 3 are masked as operations and an address, respectively. Then all four fields are added to form the value for storage in the open location.

For location values, say preceding a slash command, fields 1 and 3 are added and masked to address size to form the first display address. If the sum of fields 2 and 4 are nonzero then they form the upper limit of a display sequence. Thus equivalent commands are:

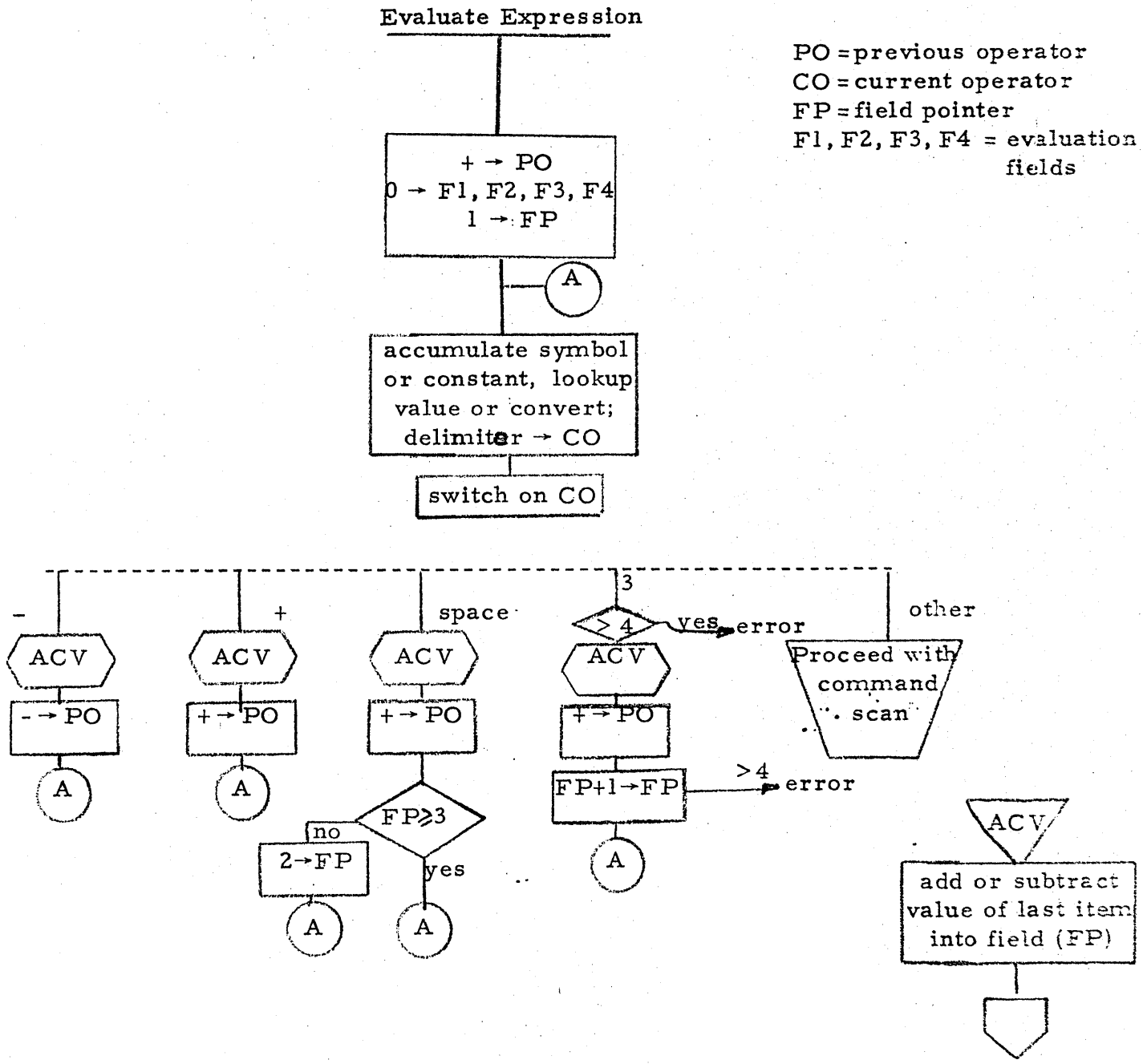
A, A+3/	and	A, A 3/
B+2, B+9/	and	B 2, B 9/
LW, 4 LOC+3, 7/	and	LW+LOC+3, 4+7/

The reader may wish to confirm the correctness of the last rather bizarre example by following the scan on the flow chart. Commands such as those for breakpoint setting use the fields separately, and the use of space for plus may not have the desired result as can be seen in the following:

A 3, 1, LOC;B is not equivalent to
A+3, 1, LOC;B

B. Memory Location Display: The / command

The / character is a command to DELTA to open a memory cell and display its contents. The cell is indicated by an expression preceding the / character. The expression is evaluated and the word address portion is used as a memory address. If no format is given and the default is F (normal case) then the symbol table is searched to find a symbol at or next smaller than the indicated address and the data type associated with the symbol found is used to control output formatting.



EXPRESSION EVALUATION ALGORITHM

Figure 1

More than one cell may be displayed using a single / command. Two expressions separated by a comma define the limits of display. They are the word address of the lower limit followed by that of the upper limit. Following display of the upper limit cell it is open for change.

```
ALPHA, ALPHA + 2/    BAL, 4  SUB
ALPHA + 1/   STW, 5     DCT2
ALPHA + 2/   AI, 6      "100
```

Format codes may be specified with (as in the basic / command.

```
100, 101 (X/    "58000100
101/            "68000200
```

If the user wishes to interrupt a too-long display he presses the break key and any remaining output is discarded. The last displayed cell is opened.

C. Expression Evaluation: The = command

Expressions consisting of program symbols, explicit constants, special symbols (see Section A4), and the operators plus and minus (+-) may be evaluated by use of the = command. The expression may be that just typed by the user or the last one typed by DELTA.

```
2 + 2 = "4
5 + 5 = "A
ALPHA/  BAL, 5  SUB = "6A5006B3
```

The format used for output is either the fault formator an explicitly requested one. The expression for evaluation is followed by a left paren, one of the format codes given in Section E, and the equal sign.

```
5 + 5 = "A
  + 5 (I = 10
  + 5 (O = 12
```

The default format type may be set by the user using the command f; =, where f is the desired format type. The initial default format is X for hexadecimal.

```
5 + 5 = "A
I; = 5 + 5 = 10
```

D. Memory Modification: The cr, lf, ↑, and tab commands

Four commands allow the user to store a typed expression for word value into a memory location -- the one opened by a /, \, or one of the modification commands lf, ↑, or tab. If no expression precedes the command character the action taken is as described except that nothing is stored in the open cell.

e cr The expression e is assembled and stored in the open memory cell. Carriage return and new line are sent to the terminal. Temporary display modes are reset to default values.

```
A/   BAL,4 JWS   BAL,4 GEB  cr
A/   BAL,4 GEB
JED/ EXU LS (X/ "6800643 / "78C cr
./ \ EXU LS
```

Note in the above that a temporary display format was established by the (X/ which carried over until the cr command reset it.

e lf When the user terminates an expression with the lf command the value of the expression is stored in the currently open cell, that cell is closed, a new line is produced at the terminal, and the cell with the next highest location value is opened. The mode of initial cell opening is preserved and carried forward on succeeding openings as is the display format.

```
A (I/    435   436 lf
A + 1/   763   lf
A + 2/   7689  cr

EM\      STM,4   ERS  lf
EM + 1\  BAL,6   LP   lf
EM + 2\  BGE     BB   cr
```

For the executive version the EOM key replaces lf.

```
100 /      34
A1 /       BAL,6  ALPHA
A+1 /      STW,5  BETA
BETA /     ABCD
```

The user may either temporarily or permanently override this output format control by the symbol table code. Temporary change is accomplished by indicating the desired format in the command. The expression for the location is followed by a left paren character, then by one of the format codes (see section E for a complete list), and finally by the command /.

```
X(X/       C1        hexadecimal conversion
X(C/       A         EBCDIC character conversion
X(I/       193       decimal integer conversion
```

Permanent change in output format is achieved by the command f;/ where f is the desired format code. See section E.

```
X/         C1        X
C;/        X/        A
```

If a slash is given without preceding typing by the user the cell addressed by the last thing typed by the computer is examined but not opened. This allows the user to look at the indirect contents of a cell. In the example below ALPHA remains the open cell even though the contents of cell DCT8 are displayed.

```
ALPHA/     LW,5   DCT8 /     "32
```

A cell may be opened without displaying its contents by the use of the \ command. (\ is produced by pressing shift and L keys together). This mode is convenient when the user wishes to insert new contents in memory and is not interested in the current contents. DELTA remembers the mode of opening for cells and on lf and ↑ commands opens in the remembered mode.

```
ALPHA \    BAL,4  SUB     lf
ALPHA+1 \  STW,5  DCT2    lf
ALPHA+2 \  AI,6   "100    cr
```

e↑ Action is exactly the same as lf except that the cell within the next lower location value is opened. For the executive version & is used for ↑.

```
EM + 4/   0    B   JH↑
EM + 3/   0    AI,3  1    cr
```

e tab The tab command causes the typed expression to be stored in a currently open cell. Following output of a carriage return, the cell addressed in the just closed cell is opened and displayed. The effect is like that of a cr command followed by a ;Q/. The tab command is useful for patches:

```
A/   BAL,5  SUB  lf
A + 1/   STW,6  BETA   B   PATCH   tab
PATCH/   0    AI,6   1   lf
PATCH + 1/   0    STW,6   BETA   lf
PATCH + 2/   0    B   A + 2   cr
```

E. Output Format Control

Displays of the contents of memory locations via the / command and expression evaluation via the = command have the output format controlled by codes given with the / or = command or by the default format as set using the (f;/ and (f;= commands. The original default setting of the output conversion format is hexadecimal (X) for = commands and under control of the nearest symbol table type (F) for / commands. Temporary conversion type settings set by using e (f/ or e (f= are retained until the next cr command is given. In particular the temporary conversion type is retained over successive lf, ↑, /, =, and tab commands.

```
(I;/
A(X/     "C   lf
A+1/     "D   lf
A+2/     "E   cr
A+3/     15
```

The codes provided for directing output formatting and conversion are given below. In all conversions leading zeros in the printout are suppressed.

X The word -- contents of memory or expression is typed out as a hexadecimal number. Hexadecimal numbers are always typed with a leading ". This is the original default code for = command.

F Conversion is according to the format code given in the symbol table for the location displayed or that for the next lower valued location symbol if no symbol occurs at the location in question. For = commands F conversion is equivalent to X conversion. F conversion is the default code for / commands.

I The word is converted as a signed decimal integer.

C The word is converted to EBCDIC characters; that is, it is sent to the terminal directly. Non-printing characters may be output in this way, including the EOT (04) character, which will turn off many types of terminals.

O Conversion is to an octal number. Each three bits from the word to be converted starting on the right are converted to a number in the range 0-7. The final two bits on the left are converted to the range 0-3.

R The word is converted to a symbolic instruction: output has the form OP, R ADDR, X similar to assembler symbolic machine instruction format. OP is the symbol table value of the op code part of the word (bits Ø-7) -- %XX is printed if the value XX of the field is not an instruction. R is the value of the register field (bits 8-11) and is printed as a decimal integer, except if zero when it is suppressed along with the preceding comma. ADDR the address field is printed with a leading * if bit 0 is a 1 and followed by the symbol obtained from lookup of value in bits 15-31 -- if no no symbol corresponds to the value, then the next lower symbol plus a relative hexadecimal offset is printed. Values

less than 64 decimal are always printed in hexadecimal. If the index field (bits 12-14) is nonzero, it is printed as an integer (1-7) following a comma.

A The word is converted in exactly the same way as R format except that the address field is always given as a hexadecimal number.

H Halfword. Each 16 bit half of the word is treated as an address quantity and printed as symbol plus hexadecimal addend.

T Base Two. Thirty-two ones or zeros are printed depending on the bits in the word. Leading zeros are not suppressed, and the printout is separated into 4 groups of eight by spaces.

S Short floating point number. The word is converted from internal floating point format to the form. XXXXX E±YY.

L Long floating point number. Same as above except the current word plus the next highest addressed word are converted (same as S for = command).

D Double word decimal integer. The current word plus the next word are converted as a 64-bit decimal integer with sign. (Same as I for = command.)

The final three conversion types S, L, and D will not be available on the executive version of DELTA.

F. Execution Control: The ;G, ;P, and ;X commands

The three commands described in this section allow the user to begin and continue execution of his program. Each of the commands is terminated by carrier return. Execution is started by typing e;G where e is an expression for the starting or GO location. (The value of the expression is masked to form the word address of the starting location.)

BEGIN;G

Execution can be stopped in three ways:

1. encountering a breakpoint (see Section G),
2. a user interruption via the BREAK key,
3. an error causing a machine trap (illegal instruction, memory protect violation, etc.)

In each case the cause of the stop is reported by an appropriate message, the values of ;I, ;C, ;F, are set, and terminal control returns to the user.

```
BREAK AT "5C3
ILLEGAL INSTRUCTION AT "77B
;I=  "77B
```

Proceeding from a stop condition is directed by the ;P command. The use of ;P for instruction breakpoints is covered in Section G. For user interruptions via the BREAK key, execution continues as if the interruption had not occurred.

```
BREAK AT "68C
;P
```

Proceeding from a machine trap will in general cause reexecution of the violating instruction and another trap.

```
MEMORY VIOLATION AT "74B
;P
MEMORY VIOLATION AT "74B
```

In either of the above cases any expression typed before the ;P is ignored.

The ;X command assembles and executes the expression just preceding the ;X.

```
LH, 3    TABLE 4;X
STB, 6   *LOC;X
```

If the expression does not result in a legitimate instruction, the illegal instruction message results and other error messages correspond to other illegal constructs just as if the error had been an executing program. If the expression is a branch instruction control goes to the user's program (or causes a memory violation). Thus the commands B GO;X and GO;G are equivalent. If the expression is a subroutine jump, the subroutine is entered and if it returns normally (to the calling location plus 1, 2, or 3) control returns to Delta and terminal control to the user.

G. Breakpoints: The ;B and ;D commands

Delta provides the user with multiple breakpoints of two kinds: 1) on instruction execution, and 2) on a change in data value. Eight breakpoints of each kind are available to each user. As each breakpoint is reached, a small amount of information is printed out giving the break location and an associated value. A special mode allows execution to continue automatically after the breakpoint report to provide a limited kind of trace of both the flow of execution control and of the variation of data values.

1. Instruction Breakpoints

e,n;B — The n^{th} breakpoint (there are eight numbered 1-8) is set to stop execution and return control of the terminal to the user when the instruction at location e is reached. The stop occurs before execution of the instruction at e. When the breakpoint is reached, DELTA prints the number and type of breakpoint and its location.

```
A+3,1;B  A;G
1;B> A+3
```

A third field of the breakpoint command may be used to specify a location to be displayed when the breakpoint is reached. Registers as well as core locations can be displayed in this way. A fourth field specifies the format of the display.

```
A+3,1,R5;B  A;G
1;B>A+3  5/  "54
A+3,1,R5,I:B  A;G
1;B>A+3  5/  84
```

When stopped at a breakpoint, the user may examine and modify his program as appropriate and then continue from the point of interruption by giving the command ;P. A count may be given with the ;P command. If the count is n then the breakpoint will be passed n times before the break occurs.

```
B+8, 2, R2;B   B;G
1;B>B+8   R2/   4    ;P
1;B>B+8   R2/   5    ;P
1;B>B+8   R2/   6  5 ;P
1;B>B+8   R2/  11
```

The n^{th} breakpoint may be removed by the command 0, n;B.

If the user wishes to trace a particular instruction, he may give either of the forms above (display or no display) and specify the T mode; e, n, loc;BT. In this mode, when the instruction at e is reached the breakpoint reporting information is printed and execution continues.

```
A+3, 4, 5;BT    A;G
4;B>A+3   5/   54
4;B>A+3   5/   -1
4;B>A+3   5/   -175
```

The trace mode may be set after a break occurs by specifying ;T which is equivalent to ;P plus setting the trace mode at the current breakpoint.

2. Data Breakpoints

Data breakpoints allow the user to halt execution when any memory location (not register) changes value in a specified way. The command has the form:

e, n, val, m;Dr

It causes the n^{th} data break to be set in such a way that execution halts and terminal control returns to the user whenever the contents of memory at location e when masked by the mask m is in relation r to val. The mask for each data breakpoint is initially all ones. A T or trace parameter applies to data breakpoints in the same way and with the same effects as described above for instruction breakpoints. The letters used for r and the corresponding condition causing a break to occur are the following:

L	e<val
E	e=val
G	e>val
GE	e≥val
NE	e≠val
LE	e≤val

If no r specification is given a break occurs for all changes in the data. The mask, if specified, is ignored in this case.

Some sample data breakpoint settings are given below. Any data breakpoint may be removed by the command 0, n;D. The output resulting from a data break has the form n;D>loc e/ contents where n is the number of the breakpoint, loc is the location of the data modifying instruction, e is the data address in question, and "contents" is the new value as just modified.

```
A, 1, 3;DG
A+5, 2, "FF, "FF;DE
B, 3;DA
SDS, 4, CSC;DGE  ;P
4;D>PH   SDS/ CSC+2
```

3. BREAK Key Breakpoints

At any time during execution the user may cause the execution of his program to halt by pressing the BREAK key. A message is printed for the user given the location of the break. The ;P command will continue execution after such a break.

```
BREAK>MP+34
```

H. Memory Searching: The ;W and ;N commands

The two active search commands e;W and e;N search memory for a match or no match with the expression e. Display of all matching cells (bit for bit identical) occurs in the case of ;W and of all non-matching cells in the case of ;N. The search is carried out between limits determined by the symbol table values of ;1 and ;2; it runs between the lower limit ;1 and the upper limit ;2 inclusive. The initial value of ;1 is zero and of ;2 the highest current user core address. Before the test for a match is made, the word from memory is masked with a word which is the symbol table value of :M. The initial value of ;M is all ones.

The values of ;1, ;2, and ;M are set by the commands e:1, e:2, and e;M (each followed by Cr). In addition, the limits may be set with the single command e1, e2;L which sets ;1 to e1 and e;2 to e2.

```
A;1
B;2                    A, B;L is equivalent
2;M
2:W                    Mask bit 30 of the word. Search for all
                       words between A and B which have a 1
                       in bit 30.
```

```
A+2/     2
A+3/     3
A+6/     6
A+7/     7
A+A/     "A
B/       "B
"1FFFF;M   L, L+"100;L   ERR;W

L+3/     BAL,4 ERR
L+A/     BAL,4 ERR
L+D/     BAL,4 ERR
L+6A/    AWM,1 ERR
```

All words between L and L+"100 with addresses equal to ERR.

The user may interrupt an in progress search by pressing the BREAK key. DELTA halts the search and returns terminal control to the user (rings the bell).

I. Symbol Table Control: The ;U, ;K, ;S, !, < > commands

The symbol table available to DELTA after a load is completed consists of the global symbols (those defined by DEF directives) and a set of internal symbol tables (one for each ROM loaded) which are filed under the name of the file from which the ROM was loaded. Each internal symbol table is a keyed record in the file created for DELTA by the loader. If more than one ROM is contained in the load file, then only the last external symbol table remains at the end of the load since earlier ones are overwritten.

During debugging the user always has the global symbols of the load and he may select one of the internal symbol tables by using the s:S commands, which causes DELTA to load the symbol table from record s (the internal symbols from the program loaded from file s). They replace, for reference purposes, any previously selected internal symbol set.

```
"B73/   LW,4 IOP+ "A7  1f
IOP+ "CB/   BAL,6IOP+ "17F   IOPF; S ↑
IOPT2+6/   LW,4  K34
```

Each of the loaded programs may have contained undefined symbols. DELTA will print all undefined symbols when the ;u is given. Symbols which are undefined and within the range of an assembler LOCAL directive are lost. They are given value zero in the loaded code and do not appear when ;u is given.

Symbols may be defined by the user at any time during his debugging session. Symbols so defined are added to the set of global symbols associated with the program load.

s(f! Adds the symbol s to the global symbol table with the location value of the currently open cell ($ or .) and format type f. If f is omitted, symbolic instruction (R) type is assumed.

e(f<s> Adds the symbol s to the global symbol table with value defined by the expression e and format code f. In addition to the codes of Section E the letter K may be used to indicate constant value. If f is omitted R is assumed.

s:K Removes the symbol s from the symbol table. The removal is permanent if s is in the global table and temporary if s is in an internal symbol table. (It will return if the user switches to another internal symbol table and back again.)

;K Is used to remove all symbols from the symbol table. Symbols defining instruction codes are not erased. Individual internal symbol tables are recoverable using s;S command.

J. Miscellaneous Commands: The ;G, ;P, ;X commands

The commands covered in this section cause DELTA to change its normal or default modes for display and to zero areas of memory. All commands in this section are terminated by carriage return.

;R and ;A

This pair of commands is complementary to one another; they control how DELTA displays location values when typing the contents of cells. The mode of display is either relative (;R) or absolute (;A). When in relative mode, DELTA looks up the location value in the symbol table and displays the symbol if one corresponds to the value; if not it displays the symbol with next smaller value and a word offset in hexadecimal. If the mode is absolute (;A) then all location values are displayed as hexadecimal numbers. Note that these commands control the display of location values and not the display of the address parts of instructions contained in those locations.

```
;R
A,A+5/  LI,1  "10
A+1/  CW,1  K45
A+2/  BGE  ZZZ
A+3/  AI,1  1
A+4/  B  A17
ZZZ/  STW,2  BR13

;A
A,A+5/  LI,1  "10
"5CD/  CW,1  K45
"5CE/  BGE  ZZZ
"5CF/  AI,1  1
"5D0/  B  A17
"5D1/  STW,2  BR13

;Z
```

The command for zeroing memory takes two forms: the first, a,b;Z, names the limits -- a the lower limit and b the upper limit -- of memory to be zeroed. Expressions may be used for a and b. An error results if the value of b is less than that of a.

```
A,A+5;Z
100,1;Z
?
```

The second form of ;Z is without arguments or has zero arguments and is a request that DELTA zero all of user's core. The action requires a confirming period after a query about the users intentions before the command is carried out.

```
;Z
OK.
```

If any other response is made to the OK, the zeroing is not done.

Zeroing the user's area erases all the user's program and data, but not the monitor's context area about the user or the user's I/O buffers. If I/O is in progress directly to or from the user's area the results of the I/O

transfer are unpredictable. Because the user's physical core pages are released to the monitor, any asynchronous references to the area will result in traps which will cause the UTS executive to receive control and report the cause of the trap (illegal instruction, illegal memory reference, etc.) to the user.

K. Additional Commands for the Executive Version:
The ;O, ;J, and ;V commands

Three commands are provided for the executive version of DELTA and are disallowed the on-line terminal user. They are ;O to produce hexadecimal dumps on the line printer; ;J to direct DELTA output to the line printer; particularly in the cases of formatted displays and output from tracing breakpoints; and ;V which allows saving core on a self-loading tape for later restoration.

The commands are designed primarily to aid in debugging of the UTS system itself but may also be used to form the basis of a stand-alone debugging package. The printer and tape I/O routines are completely self-contained with no dependence on system I/O routines. The executive version of DELTA operates with all interrupts disabled.

a,b;O	Contents of memory from location a through location b are printed on the line printer single-spaced, eight hexadecimal words with initial hex location value per line. All zero lines are suppressed.
;J	Toggles the output location switch alternating between the operators console and the line printer on each instance of the command. Output from a non-tracing break is always directed to the operators console.

A,1;B

X,2,3;DTE ;J B;G

(output here from data break #2 goes to the line printer)

1;B>A

u;V — This command saves a core image on tape with a self-loader to enable restoring at a later time. The parameter u gives the highest core location to be saved.

Programmed breaks

Programmed breaks may be inserted into code by assembly or patching in a CAL2,4 XXX,Y instruction. When this instruction is executed control is sent to DELTA by UTM,XXX BREAK>loc is printed at the operators console where xxx is the address part of the CAL2,4 instruction, and DELTA waits for commands from the operator. If Y, the index field, is 1 the break is treated as a tracing break.

Interrupts

Control may be given to the executive version of DELTA at any time via a monitor KEYIN. The system programmer may get control at the operators console by pressing the Sigma 7 panel interrupt button and responding to the "!KEYIN" message with "DELTA". DELTA responds with cr, lf, and the programmer may examine, change, or set breakpoints in the system. Return to the point of interrupt is via the ;P command.

L. Errors

In the cause of brevity DELTA has a single error message: --? . This message is sent to the user whenever DELTA cannot understand the user's command syntax. It is usually simpler for the user to identify the error than for DELTA to be verbosely specific about it. 'Some errors and the reasons for them are shown below:

```
X,Y,Z,2,7/  ?          too many commas
'ABCDE'=    ?          constant value larger than one word
ABC;K                  symbol not in symbol table
?
FF:M 100,XY;L 6B;W     symbol value not found
;L 6B;W ?              remainder of command string ignored
A,5:M                  command unknown
?
LW*5 ALPHA=  ?         asterisk in a funny place
```

INDEX TO DELTA COMMANDS

/	open cell, print contents
\	open cell, no print
cr	store in currently open cell
lf	store in currently open cell, open next cell
↑	store in currently open cell, open previous
tab	store in currently open cell, open cell last named
=	evaluate and print expression
<...>	define symbol
!	define symbol
;l	set lower signal limit
;/	set default display conversion mode
;=	set default display conversion mode
;A	display location values as hexadecimal
;B	set (or clear) instruction breakpoint; BT set trace mode
;C	set condition code
;D	set data breakpoint; DT set trace mode
;F	set floating controls
;G	being execution
;I	set instruction counter
;J	divert output to line printer (executive version only)
;K	remove (kill) symbol table entry
;L	set upper and lower limits for search
;M	set the search mask
;N	search for word mismatch
;O	hexadecimal dump (executive version only)
;P	proceed from breakpoint
;Q	set last quantity typed
;R	display location values as symbol plus hex offset
;S	select internal symbol table
;T	set trace mode and proceed
;U	display undefined symbols

;V	saves core on tape with a self-loader (executive version only)
;W	search for word match
;X	execute instruction
;Z	zero memory

IX. PERIPHERAL CONVERSION LANGUAGE (PCL)

TABLE OF CONTENTS

SUMMARY

This part of the UTS Functional Specifications describes a peripheral utility sub-system designed for both on-line and batch operation. The Sub-system, PCL (Peripheral Conversion Language), provides for information movement between card and paper tape devices, line printers, teletype devices, magnetic tapes, disc files, and labeled magnetic tape files. The command language allows for single or multiple file transfer with options for selection, sequencing, formatting, and conversion of data records. File maintenance and manipulation functions are also available to assist the user.

INTRODUCTION

PCL is a peripheral utility sub-system designed for operation in a batch environment under BPM, or on-line under UTS. It provides for information movement among card and paper tape devices, line printers, teletype devices, magnetic tape, disc files, and labeled magnetic tape files. PCL is controlled by single line commands supplied from a user console in UTS, or by command cards in the BPM job stream. The command language provides for single or multiple file transfer with options for selection, sequencing, formatting, and conversion of the data records. File deletion and positioning commands, and a command to copy complete file catalogs between disc and labeled tape are included. Additional file maintenance and utility commands are also provided to assist the user.

A. Batch Operation

PCL is activated under BPM through an !PCL control command card in the BPM job stream. Once active, PCL reads subsequent command cards directly through the same control (C) device until terminated by an END command (see below) or by encountering another batch control card (!type). All user input and output is done through the M:EI and M:EO DCB's respectively. PCL diagnostic output is transmitted to the device currently assigned to the DO operational label.

B. On-Line Operation

As a UTS sub-system, PCL is called using the CALL command of the Terminal Executive Language (TEL). PCL responds by typing "PCL HERE" and then typing it's identifying mark (<) at the left margin of the next line indicating that it is ready to accept the first command. When accepting or processing a command, PCL is said to be in the command state. Entry to this state is always indicated by the display of the < as described above. Once a valid command begins execution, PCL exits the command state and enters the active state. This status remains in effect until execution of the command terminates, at which time the command state is re-entered and the user may enter his next command. As in batch operation, user input and output is done through M:EI and M:EO DCB's, diagnostics go to DO, and commands are received through C.

C. Summary of Commands

The following is a list of available functions in PCL described in terms of the actual command verbs.

COPY device(s) and/or file(s) TO device or new file

COPY device(s) and/or file(s) OVER device or existing file

COPYALL files on disc TO labeled tape(s)

COPYALL files on labeled tape(s) TO disc

DELETE an existing file

LIST a file directory

SPF (Space file) ± n files on designated device

WEOF (Write end-of-file) on designated device

REW (Rewind) designated tapes

SPE (Space to end) of last file on labeled tape

REMOVE designated tapes

TABS (Set tabs) for output device.

DESCRIPTION

The following description of PCL is oriented toward the on-line user. Nevertheless, only one explanation should be necessary to include both on-line and most batch features. For the batch user, communication is established with input through the BPM job stream and output through the DO device with no user interaction. Thus, all user prompting (* etc.) and terminal operations (Cr, Br, X^c ...) given here do not apply.

A. Conventions and Terminal Operations

For purposes of clarification, certain conventions and terminal operations have been assumed throuout the balance of this document. They include:

1. Underlined copy in examples is that generated by the computer. Copy not underlined represents that typed by the user.

2. Optional parameters within a given command are identified as such by enclosure within brackets, e.g. [OPTION.]

3. Control characters are represented in this document by an alphabetic character and the superscript c, e.g., E^c. The user simultaneously depresses the alphabetic key and the Control key (CTRL) to obtain this function.

4. Carriage Return. The Cr notation following each line in the examples represents a carriage return. Depression of this key informs the computer that an input line is terminated. A carriage return (Cr) will automatically cause the computer to give a line feed. The line feed key operates identically to the Cr within the PCL processor.

5. Escape (E^C). This key enables the user to temporarily escape to the executive command level. Escape may be applied at any time when the user has control of the keyboard. The current status of PCL is retained and may be re-activated using the executive "CONTINUE" command.

6. RUBOUT. The last input character may be deleted with this key. A \ is echoed to the user. N RUBOUTS echo N \'s and delete the previous N characters.

7. Cancel (X^C). This key cancels the current input line. A ← is echoed to the user followed by a Cr.

8. BREAK+ This key, indicated by Bk, causes an interrupt in current PCL activities. When applied during the command state, the current command is ignored as if X^C had occurred. Application during the active state causes PCL to terminate what it is doing (like printing or copying), pass control to the user, and revert to the command state. A Cr response is given if used during input. Effects of the interruption or the termination vary with the command being executed and are discussed in detail with the particular commands. If no mention is made, Bk is assumed to have no effect on the execution of that command.

B. File Copy Command

This command permits single or multiple file transfer between peripheral devices and/or file storage. Options are included for selection, sequencing, formatting, and conversion of the data records. The format is of the general form:

COPY d(s)/fid(s), fid(s), . . . ;d(s)/fid(s), fid(s), . . . ;. . . TO OVER d(s)/fid(s)

or,

TO
OVER d(s)/fid(s)
COPY d(s)/fid(s), fid(s), . . . ;d(s)/fid(s), fid(s), . . . ;. . .
COPY d(s)/fid(s), fid(s), . . . ;d(s)/fid(s), fid(s), . . . ;. . .

where,

/	separates a device from the files on that device
,	separates files on the same device
;	separates devices
COPY	introduces a device or file identification for input
TO	introduces a device or file identification for output
OVER	introduces a file identification of an existing file to be overwritten
d	represents device identification, has the form: device code [#reel no.][#reel no.][#reel no.] Reel numbers apply only for magnetic tapes. Absence of a reel number for a tape device implies scratch tape. Valid device codes are listed below.
fid	represents file identification, has the form: name[-account[-password]]
s	represents specifications for data encodings, formats, modes, etc., has the form: [option][, option]. . .[, option] Options may include any data encodings, data formats, device modes, record sequencing, and record selection listed below. Specifications given at the device level (d(s)) apply for all files on that device. Those given at the file level (fid(s)) apply only for that file and have precedence if a conflict occurs between the two levels.

When given a command of this type, PCL first checks for a destination device or file introduced by the TO or OVER command verbs. If found, the current destination device or file (if any) is closed and the new one opened for output. Files, of course are matched against the user's directory to insure OVER was used to introduce an existing file. The device(s) and/or file(s) introduced by the COPY command verb are then opened for input one at the time in the order given and copied to the destination. The destination device or file remains open until respecified (by TO or OVER) or PCL is terminated (by END) so that more inputs (by COPY) are added to it.

If Bk is applied during execution of a COPY, PCL responds with identification of the last file completely copied.

1. Device Identification Codes (d). These codes are used to indicate the "to" and "from" devices. They include:

CR	card reader - files separated by two successive !EOD cards.
CP	card punch
LP	line printer
ME	interactive users console - input terminated by Bk from teletypes
DC	disc file storage
LT	labeled tape file storage
FT	free form tape - files separated by EOF mark
PP	paper tape punch
PR	paper tape - files are separated by two successive !EOD codes

2. File Identification (fid). Files are identified by name, account, and password in that order separated by hyphens (-). The name (1-31 characters) is required whereas the account (1-8 characters) and the password (1-8 characters) are optional. Thus, four forms of file identificatio may be specified: name, name-account, name-account-password, and name--password. Absence of the account implies the current user's account.

3. Data Encoding. These codes describe the source or destination data encodings to be expected or produced,

E	EBCDIC
H	Hollerith
A	ASCII

4. Data Formats. These codes describe source or destination record formatting to be expected or produced.

C Metasymbol compressed

K Compact file structure (required by EDIT)

X hexadecimal-dump

5. Modes. These codes dictate device control modes for the devices indicated.

BCD, BIN — BCD or binary mode - valid for card, paper tape, and magnetic tape devices.

7T, 9T — seven or nine track magnetic tape

PK, UPK — seven track binary tape packing or unpacking

SSP, DSP, VFC — single, double, or variable format controlled spacing on line printer.

6. Record Sequencing Insertion or deletion of sequence identification for output data records (error if on input side) is accomplished using this specification. Option include:

CS (id, n, k) — card sequencing in columns 73-80 where id is the identification (1-4 characters), n is the initial value, and k is the increment. The id is left-justified in the field (73-80) followed by the sequence number right-justified in the same field. Precedence is given to the sequence number if overlapping occurs.

NCS — no card sequencing - strips columns 73-80 from each output data record.

LN (n, k) — number lines within a Compact (K type) file starting at n in sequential steps of k. Line numbers must be between 1 and 99,999.

NLN — no line numbers - deletes line numbers within a Compact (K type) file.

7. Record Selection This specification permits selection of the logical records to be copied by giving their sequential position within the file.

X-Y — select all records whose position n in the file satisfies the following condition $X \leq n \leq Y$. Multiple selections may be specified, e.g. X-Y, U-V, W-Z. Selections do not have to be in sequential order. Maximum number of selections is 10 for each input file.

8. Valid Option Combinations Not all combinations of from and to devices, data encodings, modes, etc. are valid. Table I shows the valid options, the disallowed combinations, and the default provisions for the possible combinations. If a disallowed combination is found, an appropriate error diagnostic is given to the user. Execution of the command may or may not continue depending on the severity of the error encountered (see Language Syntax).

TABLE I

		FROM DEVICE						TO DEVICE						
		C R	P R	D C	L T	F T	M E	D C	L T	F T	M E	L P	C P	P P
CODES	E	D	X	D	D	D	D	D	D	D	D	D	D	X
	H	X	-	X	X	X	-	X	X	X	-	-	X	-
	A	-	D	X	X	X	*	X	X	X	*	-	-	D
FORMATS	C	X	X	X	X	X	-	-	-	-	-	-	-	-
	K	-	-	X	X	-	-	X	X	-	-	-	-	-
	X	-	-	-	-	-	-	-	-	-	X	X	-	-
MODES	None	-	D	D	-	-	D	D	-	-	D	-	-	D
	BCD	D	-	-	-	X	-	-	-	X	-	-	D	-
	BIN	X	-	-	D	D	-	-	D	D	-	-	X	-
	7T	-	-	-	X	X	-	-	X	X	-	-	-	-
	9T	-	-	-	D	D	-	-	D	D	-	-	-	-
	PK	-	-	-	X	X	-	-	X	X	-	-	-	-
	UPK	-	-	-	-	X	-	-	-	X	-	-	-	-
	SSP	-	-	-	-	-	-	-	-	-	-	D	-	-
	DSP	-	-	-	-	-	-	-	-	-	-	X	-	-
	VFC	-	-	-	-	-	-	-	-	-	-	X	-	-
SEQUENCING	None	-	-	-	-	-	-	D	D	D	D	D	D	D
	CS	-	-	-	-	-	-	X	X	X	-	X	X	X
	NCS	-	-	-	-	-	-	X	X	X	-	X	X	X
	LN	-	-	-	-	-	-	X	X	X	-	X	X	X
	NLN	-	-	-	-	-	-	X	X	X	-	X	X	X

where

D default
X optional
- error, not available, ridiculous
* EBCDIC to ASCII conversion for teletypes is done by COC routines

9. Examples

```
≤ COPY CR TO DC/A-0986-PLEASE Cr
≤
```

After receiving this command PCL opens a new disc file with name (A), account (0986), and password (PLEASE). Successive cards are then copied to this file from the card reader until a double !EOD is encountered.

```
≤ COPY LT#57/B, C TO DC/B--PASS Cr
≤
```

This example demonstrates a multiple file copy. Files B and C from labeled tape with reel number 57 are copied in that order to a new disc file B with password PASS. Note that all files must be under the user's account (as specified at log on or on the !JOB card).

```
≤ COPY DC/A(C) TO LP(DSP) Cr
≤
```

The disc file A under the user's account is uncompressed and listed on the line printer with double spacing.

```
≤ COPY FT#73 TO DC/A (K, LN(5, 5))Cr
≤
```

PCL reads successive records from free form tape #73, assigns line numbers starting at 5 in steps of 5, and writes them to file A on disc in K format.

```
≤ COPY LT#205/SOURCE TO CP (CS (SRCE, 1, 1)) Cr
≤
```

The label tape file named SOURCE on reel number 205 is sequenced and punched. The logical records were given sequence identification (SRCE0001, SRCE0002, ...etc.) in columns 73-80

```
≤COPY PR;PR;PR OVER DC/ALPHA Cr
≤
```

Three consecutive files on the paper tape reader are copied over an existing file ALPHA under the user's account. Each file on paper tape terminated by a double !EOD.

```
≤COPY FT#6(BCD,7T,H) TO LP(X) Cr
≤
```

In this case, free form tape #6 is a 7 track tape in BCD containing Hollerith coded data. Each record is read, converted to EBCDIC, and dumped to the line printer in hexadecimal.

```
≤COPY DC/A(K) TO FT(BIN,7T,H)Cr
≤
```

This example points out the use of a scratch tape. Line images stored on disc in K format are read sequentially, converted from EBCDIC to Hollerith, and written on a 7 track scratch tape in BIN mode.

```
≤TO DC/N3 Cr
≤COPY DC/N1(20-30, 40-100), N2-1234-PASS(50-75) Cr
≤
```

Sections of two files (N1 and N2) are combined to form a third file N3. Records 20-30 and 40-100 of N1 followed by records 50-75 of N2 are copied in that order to N3. The user's account is assumed for files N1 and N3, and N2 is from account 1234 with password PASS. Note that the destination file was defined on a separate line.

```
< COPY DC(K)/SOURCE TO ME Cr
10020 START LW,R1 ALPHA
10020       AI,R1  5
10030       CW,R1 BETA
  .   .   .
  .   .   .
  .   .   .
<
```

This command requested a Meta-Symbol source file on disc in K format be dumped at the user console. Note that the line numbers occupy the first six characters of each line.

```
< COPY FT#7236 TO PP Cr
< COPY FT#7236 Cr
< COPY FT#7236 Cr
<
```

Three successive files from free form tape #7236 are punched as one long file on paper tape. An end of file mark (two !EOD's) will not be written on the paper tape until the device is closed.

```
< COPY LT#5/A,B,C; DC(K)/D,E; FT#8 TO LT#6#7/A Cr
<
```

This example demonstrates the multi-file multi-device capabilities of the file copy command. Files A, B, and C from labeled tape #5, files D and E from disc, and the next file on free form tape #8 are copied respectively to file A on labeled tapes #6 and #7. Tape #7 is used only if #6 overflows. Note the format specification K holds for all files up to the next device id code (files D and E).

C. Catalog Copy Command

This command enables the user to copy his complete file catalog between disc and labeled tape. The command is of the form:

```
\COPYALL  DC  TO  LT [#reel no.][#reel no.][#reel no.]
or,
COPYALL  LT [#reel no.][#reel no.][#reel no.]TO  DC
```

PCL copies all files under the user's account from the input device (LT or DC) to the output device (LT or DC). Files protected by passwords cannot be copied with this command. The Bk key will terminate execution of this command and cause PCL to respond by typing the identification of the last file copied. Consider the example:

```
≤ COPYALL DC TO LT#3#4 Cr
≤
```

All of the files given in the user's catalog are copied to labeled tapes #3 and #4. Tape #4 is used only if #3 overflows. The disc space previously occupied by this catalog of files can now be released for other use.

To restore his file catalog, the user may enter the following:

```
≤ COPYALL LT#3#4 TO DC Cr
≤
```

This causes PCL to copy all the files from labeled tapes #3 and #4 to disc under the user's account.

D. Other Commands

This group of commands provides for file deletion, directory listing, file positioning, and other manipulation and maintenance functions.

1. DELETE

Files may be erased using this command, which is of the form:

DELETE fid

where fid represents name-account-password of an existing file. Following the entry of this command, a confirmation message of the form "DELETE fid?" is typed. The user may respond by typing "YES" to confirm the operation or with anything else to cancel it. If YES is typed, the file is deleted and the disc space released. For example:

```
≤ DELETE SOURCE--PLEASE Cr
DELETE SOURCE--PLEASE?YES Cr
≤
```

Upon receiving this command, EDIT locates the file in user's directory and responds with the confirmation message. After the YES reply, the file SOURCE is deleted.

2. LIST

To list the account directory or labeled tape file names for a designated account, the user enters a command of the form:

LIST LT [#reel no.][#reel no.][#reel no.] , account

or,

LIST DC, account

PCL scans the directory (DC) or tape reels (LT), listing the names of files encountered. Output is to the user's terminal in UTS or the line printer in BPM. Printing may be interrupted and the LIST command terminated with the Bk key. Consider the example:

< LIST LT#3#4, 0986 Cr
ALPHA
SOURCE
A
B
<

Labeled tapes #3 and 4 under account 0986 are scanned for existing files. Four such files are located and their corresponding names printed at the user's console.

3. SPF

This command allows the user to position input peripheral devices forward or backward a designated number of files. The command is of the form:

SPF device id [#reel no.] ,±n

where device id represents one of the device identification codes LT, CR, FT, or PR, ± implies direction and n is the number of files to be skipped. If direction (±) is not given, forward (+) direction is assumed.

For example:

```
< SPF FT#2076, +2 Cr
<
```

Free form tape #2076 is positioned forward 2 files. If an end-of-reel is encountered prior to completion, an appropriate diagnostic is given to the user.

4. SPE

The user may skip to just following the last file on labeled tape through the following command:

SPE LT [#reel no.]

For example:

```
< SPE LT#5 Cr
<
```

PCL positions labeled tape #5 to just following the last file. The user may now add additional files to the tape.

5. WEOF

This command enables the user to write an end-of-file mark on output peripheral devices. The command has form:

WEOF device id [#reel no.]

where device id is any output device code excluding LT and DC. PCL writes an EOF on magnetic tape and double !EOD records on card and paper tape. For example:

```
< WEOF CP Cr
<
```

This example causes PCL to punch two successive !EOD cards.

6. REW

A user may request that designated magnetic tapes be rewound using the following

command:

REW #reel no. [#reel no.] . . . [#reel no.]

PCL rewinds each tape in the order specified. For example:

```
≤ REW #205#206  Cr
≤
```

Tape units currently identified with reels 205 and 206 are rewound.

7. REMOVE

This command permits the user to request removal of tapes no longer needed and thus, release the tape unit for other purposes. The format is as follows:

REMOVE #reel no. [#reel no.] . . . [#reel no.]

Each tape specified is rewound and, upon completion, a dismount message is given to the operator. For example:

```
≤ REMOVE #2075#2076  Cr
≤
```

Tape units associated with reels 2075 and 2076 are rewound. Messages are given to the operator to dismount these tape reels.

8. TABS

This command sets listing tabs for the current output device as defined by the latest TO or OVER command. It is of the form:

TABS $c_1 [, c_2] \ldots [, c_n]$

where c_i represents column numbers of desired tab settings. PCL merges the settings into the current output dcb. For the ME device, settings are transmitted to the COC routines which performs the actual tab simulation in this case. Consider the example:

```
< TABS      10, 19, 37 Cr
<
```

Assuming Meta-Symbol source is being copied to a listing device, this command sets the appropriate tabs for this language.

E. Termination of PCL

In order to close the current output file, it is necessary for the on-line user to indicate when he has finished with PCL functions. The command END fulfills this requirement and also returns control to the UTS executive. Prior to exiting, a termination message is given to the user. For example:

```
< END  Cr
PCL PROCESSING TERMINATED
!
```

This command closes the current output file (if any) and causes PCL to return to the executive command level. The Executive responds with its identifying mark (!) indicating the command state.

F. Language Syntax

The PCL control language is designed to be free form with a few restrictions imposed for simplicity in implementation and use. These include:

1. All commands must comply to the general format given in the definition.
2. Blanks are allowed preceding or following an argument field. Imbedded blanks are not permitted.
3. At least one blank must follow each command verb and must precede an imbedded command verb (TO, OVER).
4. Continuation between input records is not allowed. (One command per line.)
5. End of command is indicated by a period (.) or by end of the input record (column 80 for card input, Cr or Lf for TTY's)
6. An output device or file (TO, OVER) must be defined prior to or on the same line with COPY command. COPYALL, END, TO, or OVER commands terminate the current output specification.

Each command is edited for compliance to the above rules and is checked against Table I. The user is notified of all errors (including I/O errors) through appropriate diagnostics. A severity level of 1, 2 or 3 is attached to each error and has the following effect on the execution of the command in question.

1 - warning, require "GO" confirmation from on-line user, continues execution for batch user.

2 - invalid syntax or I/O error, terminate execution of command, but continue syntax edit for both on-line and batch users.

3 - format error, terminate command, revert to command state for on-line user, read next command card for batch user.

The maximum severity encountered for a command is displayed following diagnostic output. For example:

```
< COPY CC TO DC/A Cr
INVALID DEVICE
SEVERITY 2
<
```

INDEX TO PCL COMMANDS

X. Loading of Programs (LINK)

TABLE OF CONTENTS

INTRODUCTION

LINK is designed to construct a single entity called a load module (LM). A load module is an executable program formed from relocatable object modules (ROM's) and/or library load modules (LM's). ROM's are representations (of program and data) that are generated by a processor such as Meta-Symbol or FORTRAN. The on-line user has, at the executive level his choice of constructing a load module (LINK), loading into core a previously constructed load module (LOAD), starting execution of the loaded LM (START), or combining the above steps (RUN). RUN is used either to load and start execution (LOAD-START) or it is used to link, load and start execution, (LINK-LOAD-START) see PART VI Terminal Executive Language (TEL). A library load module is a single entity formed from relocatable object modules which is constructed in such a manner that it may be combined with other ROM's or library load modules. LINK is a one-pass linking loader (reads each input module once) making full use of the mapping hardware.

LINK is not an overlay loader. If the need exists for overlays the user must call on the overlay loader by entering a job in the batch stream. At a later time a simple chaining feature will be added to LINK to provide a simple form of overlay.

In order to form a load module which may later be combined with other load modules or ROM's the load module must be of one protection type. A program of mixed protections type may not be combined.

Object programs consist of one or more program sections. Sectioning is the arbitrary grouping of areas of a program into logical divisions, such as specifying one section for the main program, one for data, one for literals, etc. Furthermore with memory map and/or write locks, program sectioning enables the programmer to designate the mode of protection he wishes to have for the program divisions.

The access protection features are:

- read, write, and access permitted (data)
- read and access permitted (pure procedure)

- read only permitted (static data)

- no access, read, or write permitted (no access)

LOAD MODULE STRUCTURE

A load module (LM) formed by LINK may be throught of as being comprised of three parts: A. program, B. global symbols, and C. internal symbols.

A. Program

The program may be sectioned into the following parts:

1. Pure procedure - This section of code has read and access protection is generated by the compilers and assemblers as control section 1.

2. Data or program context - This section of code has read, write and access protection and is generated by the compilers and assemblers as control section 0.

3. Common - This is blank COMMON and is generated by compilers and assemblers as a dummy section with the name F4:COM. The size of blank COMMON is determined by largest size declared.

4. DCB's - A Data Control Block is a table containing the information used by the Monitor in the performance of an I/O operation. LINK will construct a DCB corresponding to each external reference with names beginning with F: or M:, or it will satisfy these references from a standard set, allocated automatically for each on-line user.

   The standard set of DCB's will be defined in a later document along with the information contained in the job information table (JIT), fixed context areas for the public library and standard processors, and other UTS standards. DCB's constructed by LINK will be

twenty-two (22) words and will contain default assignments. Additional words are generated allowing space for a file name, account number, password, input serial numbers and/or output serial numbers. The exact number of additional words and what defaults will be assumed is to be specified in the same document. In those cases where the DCB's constructed by the loader do not fit the user's needs the user may define his own. While allocating, constructing and combining DCB's, LINK will always guarantee that each DCB will be contained within a page. This allows the operating system to access DCB's in either mapped or unmapped mode.

5. Public (core) library - Each installation will have the ability to define a set of reentrant subroutines which together constitute the public core-library. The reentrant portion of the core library is shared among users (on-line, batch, and real-time), thus saving physical core memory and allowing for more efficient system operation. The user dependent data for each core library routine is allocated by LINK at a fixed virtual address. Thus, the public library is constructed in two parts: reentrant procedure and direct access context data (i.e. in fixed virtual memory). By forming the library in this manner a speed advantage of from 5 to 20 percent over push-down storage re-entrancy can be obtained.

6. System library - The system library, much the same as the public core-library, is constructed in two parts: reentrant

procedure and direct access context data. Routines which are obtained from the system library become part of the user's program and are not shared. The speed advantage is still maintained by providing a library which accesses a data area in fixed virtual memory.

The difference between the Public Library and the System Library is that every individual user pays core for each System Library routine used while only one instance of the Public Library is required no matter how many are using it. In the Public Library, however, use of just one routine requires core for the whole package. The Public Library contents will be specified and built at SYSGEN time.

B. Global Symbols

While performing the link process, a global symbol table is constructed. This table is a list of correspondences between symbolic identifiers (labels) used in the original source program and the values or virtual core addresses which have been assigned to them by LINK. The global symbols identify objects (DEF's) within a module which may be referred to (REFed) in other modules. This table is available to DELTA, for use in debugging, and to SYMCON.

C. Internal Symbols

An internal symbol table is a list of correspondences similar to the global but which applies solely within the module. Each internal symbol table constructed by LINK is associated with a specific input module and identified by the module's file name. The internal as well as the global symbol tables are created for use by the debug processors, such as DELTA. The user has the ability under DELTA to define which set of internal symbols are to be used for specific debugging activities.

SYMBOL TABLE FORMAT

As has been mentioned above, the main usage of symbol tables are by DELTA. DELTA allows the user to reference both internal as well as global symbols in the debugging of programs. The user operates on his object programs as formed by the loaders, together with the tables of internal and global symbols accompanying them in what appears to be assembly language symbolic.

Both global and internal symbol tables, as formed by LINK and used by DELTA, consist of three word entries. Symbolic identifiers (labels) are limited to seven (7) characters plus count. Symbols originally longer than seven are truncated leaving the initial characters, although the character count is retained. Symbols which are identical in their first seven characters and are of equal length occupy one position in the symbol table. The value or definition for multiply defined symbols is the last one encountered during the linking process. Each symbol entered into the table has a type and internal resolution classification. The internal resolution types are; byte, half-word, word, double word, and constant. The following is a list of the symbol types which are supplied by the object language and maintained in the symbol table: instruction, integer, EBCDIC text, short floating point, long floating point, decimal, packed decimal, and hexadecimal.

In order to provide internal symbols definition together with internal resolution and type classification, the relocatable object language will be augmented. This means that the compilers and assemblers must be changed in order to provide this facility. In addition, existing loaders must be modified in order to process the changes in the object language. The required additions to the object language and the exact symbol table format will be detailed in a separate document.

THE LINK COMMAND

The LINK command may appear both as an executive command (in TEL) or it may appear as a direct command to the LINK processor. All operations that can be performed under the LINK executive command can be performed under the sub-system. The notation and conventions for specifying the retention, deletion, and merging of internal symbols are the same.

The most commonplace LINK commands are of the form:

LINK mfl, mfl, ... ON lm (on a new file)
LINK mfl, mfl, ... OVER lm (over an existing file)

LINK mfl, mfl, . . . (on a temporary file for subsequent loading)

where

mfl specifies the load module or relocatable object module name and is represented by file name, account and password (in this order) separated by hyphens. In the absence of account and/or password, the log-on accounting identification is used. A dollar sign '$' may be used to designate linking of the most recent compilation or assembly.

lm specifies the name (file identification) of the load module to be created by LINK.

Optional specifications on the LINK command control:

A. Load module symbol tables

(I) /(NI) The parenthesized letters "NI" preceding an input module's file identification specifies that no internal symbol table is to be constructed; the parenthesized letter "I" specifies that an internal symbol table is to be constructed. The "I" or "NI" option holds for all subsequent modules mentioned in the command until the occurrence of a new specification. In the absence of any specification "I" is assumed.

Example:

LINK A, (NI) B, C, (I) D ON E

This command specifies that a load module E is to be created for execution from files A, B, C, and D. (By implication, the public library and system library are to be searched to satisfy any external references.) Internal symbol tables are to be created for file A and D but not for files B and C. The global symbol table is always retained.

B. Merging internal symbol tables

(mfl, ...) LINK may be instructed to merge the internal symbols of several files by enclosing the files in parentheses. Only one level of parenthesized nesting is allowed.

Example:

LINK (D,A) (NI) B, C, ON E

This command specifies that no symbol table is to be constructed for files B and C and the internal symbols for files D and A are to be merged. The internal symbol table will be identified by A. The identification given to the internal symbol table will be that of the last input module specified in the merge.

When a load module containing separate internal symbol tables is itself linked, LINK will merge all the tables under that module's name.

C. Library search

;lid,lid... specifies the libraries which are to be searched for program references which have not yet been satisfied. Libraries are identified by account. The list of library accounts separated by commas is appended to the LINK command following a semicolon. In the absence of any other specifications the public library will be searched followed by the UTS system library, any user specification eliminates these searches unless requested by the user.

(L) specifies that the public and system libraries are to be searched to satisfy external references which have not been satisfied by the program.

(NL) specifies that a library search is not requested.

D. Displays

(D) specifies that at the completion of the linking process (including searching libraries, if specified), all unsatisfied internal and external symbols are to be displayed. The unsatisfied symbols are identified as to whether they are internal or external and to which module they belong.

(ND) specifies that the unsatisfied internal and external symbols are not to be displayed.

(C) specifies that all conflicting internal and external symbols are to be displayed. The symbols are displayed with their source (module name) and type (internal or external).

(NC) specifies that the conflicting symbols are not to be displayed.

(M) specifies that the loading map is to be displayed upon completion of the linking process. The symbols are displayed by source with type resolution, and value.

(NM) specifies that a load map is not to be displayed.

The default specifications for the linking process are D, C, NM, and L. Any specifications stated or implied hold over subsequent LINK commands.

COMMANDS WHICH INITIATE THE LINK SUB-SYSTEM

The LINK sub-system may be called as a subroutine or it may be called directly as a processor.

A. LINK

LINK is called as a subroutine when TEL receives a LINK command. In this mode the information and specifications supplied on the LINK command are assumed complete. Therefore, the sub-system will have little or no interaction with the user.

The specified input modules are linked with or without library modules as specified and, if specified, a map is displayed. The user is notified when the operation is complete by the executive system (TEL). The sub-system (LINK) returns control and TEL requests further commands from the user.

Example:

```
! LINK (ND) (NC) A,B,C
DONE
!
```

If, when called as a subroutine, LINK has any need to request information from the user, it will identify itself, identify the problem, and then prompt for input as follows:

```
LINK HERE
(problem identified)
:*
```

In all subsequent requests from LINK only the problem and prompt character are displayed.

B. RUN

The LINK sub-system is called as a subroutine when TEL receives a RUN command. In this mode information and specifications supplied on the RUN command are assumed complete. The sub-system normally has no interaction with the user.

* : is LINK's prompt character

The two forms of the RUN command that may be presented to the executive system (TEL) are:

RUN
RUN mlf, mlf, ...

The first form is used to link, load, and start the result of the last major operation (assembly, compilation or linkage).

If the last major operation was a linkage, the sub-system (LINK) is not needed and will not be called. However, if it was an assembly or compilation, LINK is called as a subroutine. The second form is used to link, load, and start execution of a set of modules. All options of the LINK command may be exercised in the RUN command.

Two options may appear on the LOAD and RUN commands which do not appear on the LINK command. The options are "NG", and "S". The "S" option allows the user to specify when copies of the internal symbol tables associated with a LM are to be carried along with the loaded LM. Normally, the internal symbol tables are not "loaded", and global symbols are "loaded" unless turned off by the parenthesized letters "NG".

e.g.,

RUN (S) (I) A, B, (NI)C

This example requests that files A, B and C are to be linked, loaded, and started. Internal symbols for the first two only are to be kept with the resulting load module; all internal symbols kept with the load module are to be "loaded" with it.

C. LINK called as a processor

The sub-system is called directly by using the command CALL LINK. The notation and conventions for input files and retention, deletion, and merging of internal symbol tables remain the same. The main advantage as a processor is that of interaction. It allows the user to link more modules, search more libraries and in general the user has more control over the linking process. In addition to the LINK command the sub-system recognizes the commands: OUTPUT, SEARCH, LIST, QUIT and END.

Specifications governing the displays and library searches are given immediately following the LINK command verb in the form of a parenthesized code or list of codes.

(D) specifies display unsatisfied internal and external references

(ND) signifies don't display internal and external references

(C) signifies display all conflicting identifiers

(NC) signifies don't display identifiers

(M) signifies display the loading map on completion of the linking process

(NM) signifies don't display loading map on completion of the linking process

(L) signifies search the public and system libraries for unsatisfied program references

(NL) signifies don't search the public and system library for unsatisfied program references

The default specifications are D, C, NM, L. Specifications hold over subsequent LINK commands until changed.

1. OUTPUT

Specification of an output file instructs LINK to complete any previous link process and initiate a new one. The previous output module, if any, is closed and saved for future loading.

As a processor LINK will not initiate any linking until an output file has been identified. The user may specify an output file by the LINK command.

LINK mfl ON lm (new file)

LINK mfl OVER lm (old file)

All commands are analyzed for validity. If any discrepancies appear, the user is informed and LINK requests that corrective action be taken.

2. LIST

At any time prior to completing the linking process the user may request optional displays to be listed on the printer, file, or terminal. The format of the LIST command is:

LIST (loading map, et al) ON LP (printer) / file / ME (terminal)

The default specifications are D, C and M.

3. SEARCH

At any time prior to the completion of the linking process the user may request LINK to search his own and/or system and public libraries to resolve unsatisfied external program references. The format of the SEARCH command is:

SEARCH (L) / (NL) lid, lid, lid...

4. QUIT

At any time prior to the completion of the linking process, the user may request LINK to terminate. Termination results in the release of all core and disc space allocated as the results of the linking process.

5. END

The linking process is terminated with the END command. This command instructs LINK to close and save the current output file, if any, for future loading.

Example:

```
! CALL LINK

LINK HERE

: LINK (ND) (NC) (NL) A, B ON JED
: LINK C
: SEARCH CH (L)
: LIST (M) (C) (D) ON JOE
: END
DONE
!
```

In this example the output file is JED and input modules A and B are linked. No display has been requested. Input module C is then combined with A and B and the system library is searched. Then, the user requests that the map, conflicting and unsatisfied symbols be listed on file JOE. The LINK session is concluded by the command END and control is returned to the executive.

BREAK KEY

Despression of this key causes LINK to terminate whatever it has been doing as soon as it can; however, this signal is ignored if given by the user while he is typing in a command. Usually, LINK will type

REVOKED

as soon as it honors the break. However, if engaged in linking a module or searching a library, LINK will finish that operation and then tell the user how far it has gotten. For example, if working on the command

LINK A,B,C,D

and interrupted while working on file B, LINK will finish linking B and then type

DONE THRU B

If actually finished with a command before honoring the break, LINK will simple behave as it usually does after finishing a command. If called as a sub-system, LINK will return control to the user after typing its identifying mark; if called as a sub-routine, LINK will notify the exec that it has been interrupted, without typing anything at all (TEL will tell the user that the command has been revoked).

INDEX TO LINK COMMANDS AND OPTIONS

COMMAND	OPTIONS	MEANING
LINK		Specify the linking process
	D/ND	Display or don't display unsatisfied internal and external references
	C/NC	Display or don't display conflicting identifier
	M/NM	Display or don't display loading map
	L/NL	Search or don't search the public and system library
	I/NI	Construct or don't construct internal sumbol tables
OUTPUT		Specify the LM file.
	none	
LIST		Specify display options to be listed on printer, file or terminal
	D/ND D/NC M/MN	Same as link.

SEARCH		Specify which libraries are to be used in satisfying unsatisfied references
	L/NL	Same as LINK
END		Specifies the end of a linking process
	None	
RUN		Specifies to link, load and start execution
	D/ND C/NC M/NM L/NL I/NI	Same as LINK
	NG	Do not load the global symbol table with the load module
	S	Load internal symbol tables with the load module

TABLE OF CONTENTS

INTRODUCTION

This document describes the calls for service by user programs, their operation and restrictions in the UTS environment. All facilities and processors now available as BPM services remain available to the batch user in UTS. Some UTS facilities are provided solely for on-line use, while others are available only in batch.

New and modified services allow the user to get and free core storage -- both in the old ways from must above his program area and from common storage and in a new way by specifying the virtual address of the desired core.

New services are provided to a) allow communication and memory protection changes when transferring between a user program and system processors, b) set up a "prompt" character with the terminal I/O routines which will be typed whenever input is requested, and c) control the character translation and end-of-message indication tables in the terminal I/O routines.

Some of the current CAL's behave differently when called by an on-line user. These differences are outlined.

The Monitor service CAL's are listed by the restriction in usage -- on-line, batch, or real-time, and for convenience in numerical order.

The standard assignments to devices of the system operational labels are listed in the final section.

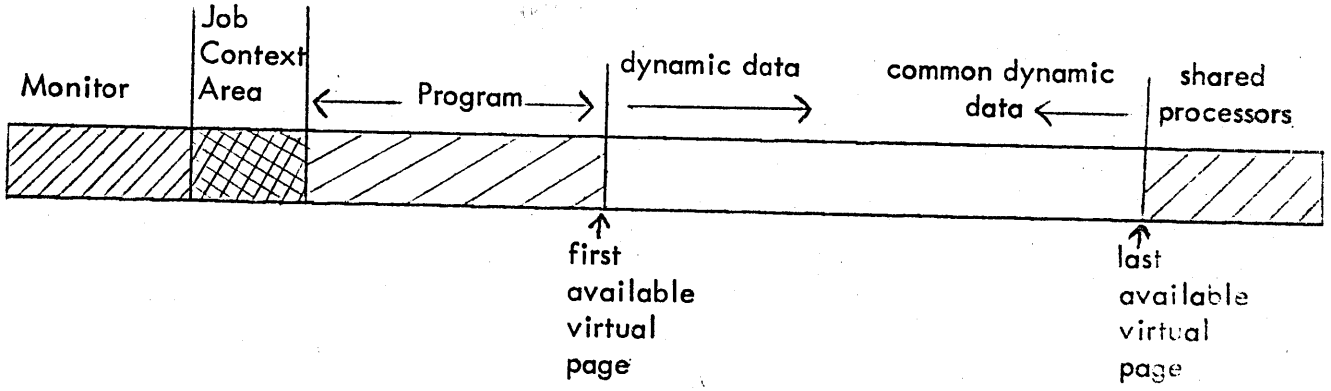
DATA MEMORY MANAGEMENT

The Monitor provides two forms of memory management which allow the user to manipulate his core memory area assigned to data. They are: Relative allocation and Specific allocation.

Relative allocation allows user data to be extended from the top or bottom of core. Memory may be extended from the highest available virtual page above the user's current allocation to the lower limit of the user's common dynamic area. The pages obtained in this manner are called dynamic pages. Memory may also be allocated from the highest available virtual page down to the upper limit of the dynamic area. The pages obtained in this manner are called common dynamic data. The memory management routines do not permit overlapping of dynamic and common dynamic memory.

Installation parameters set at SYSGEN time and modified by operator KEYIN'S separately regulate the amount of core storage available to on-line or batch users.

Specific allocation allows user programs to request or release any virtual memory page between the first available virtual page and the last available virtual page, but the two forms of memory management may not be used within the same program (at the same time). A sample virtual memory layout is shown below.



Seven monitor service calls described below allow user manipulation of memory. They are divided into two groups: 1) CAL's analygous to BPM memory requests for getting and freeing dynamic and common memory, and 2) New UTS CAL's which request and release specific virtual memory pages. The user must confine his memory request to either group 1 or group 2 with an error resulting from mixed usage. A single program may use both groups of allocation commands so long as all memory is released to the monitor between command groups.

A. Get Limits

M:GL The GL routine is used to obtain the absolute hexadecimal addresses of common dynamic core storage. The lower limit is returned in SR1 and the upper limit in SR2.

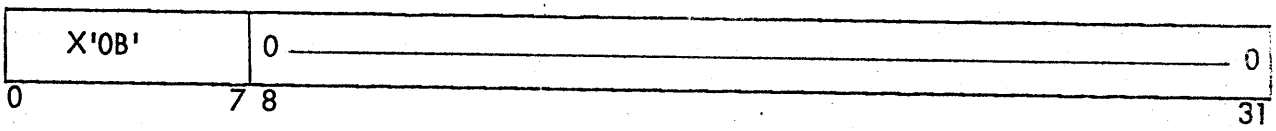
The M:GL procedure call is of the form

```
M:GL
```

Calls generated by the M:GL procedure have the form

```
CAL1,8  FPT
```

FPT is the address of a word as shown below.



B. Get Common Pages

M:GCP The GCP routine is called to extend the lower limit of common dynamic storage by a specified number of pages. If the required pages are available, condition code (i.e., bit 1 of CC) is set to 0. If the required number of pages are not available, condition code 1 is set to 1 and the number of pages actually available is returned in SR1. In either case, SR2 contains the address of the first available common page (lowest address value). If specific allocation is in effect, SR1 and SR2 will be set to zero and CC1 is set to 1, that is the request is denied. The M:GCP procedure call is of the form

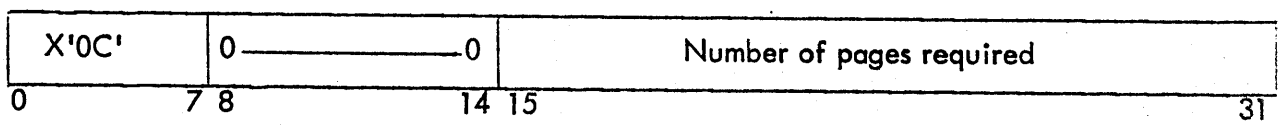
```
MGCP  pages
```

Pages specifies the number of memory pages by which common dynamic storage is to be extended.

Calls generated by the MGCP procedure have the form

```
CAL1,8  FPT
```

FPT is the address of a word as shown below.



C. Free Common Pages

M:FCP The FCP routine is called to free a specified number of pages from the lower li of the current dynamic common storage area. The freed pages are not available for use by the user's program (access protection is set to 11) and any attempt to use freed pages will result in a trap.

If the specified pages are not part of the user's dynamic storage area, or if in specific allocation is in effect, no pages are affected and condition code 1 is set to 1; otherwise it is set to 0.

The M:FCP procedure call is of the form

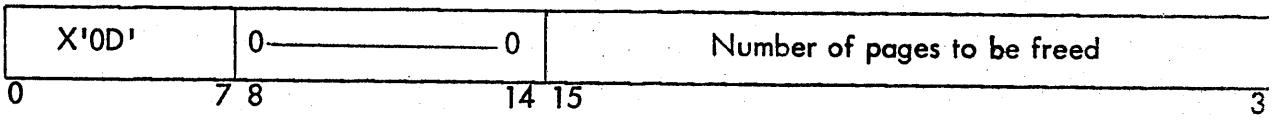
M:FCP pages

Pages specifies the number of pages to be freed.

Calls generated by the M:FCP procedure have the form

CAL1,8 FPT

FPT is the address of a word as shown below.



D. Get N Pages

M:GP The GP routine extends the dynamic area of core storage that may be used by the user's program. If the specified number of additional pages of memory are available, CC1 (i.e., bit 1 of the CPU's condition code register) is set to 0 and the access protection on the allocated pages is set to 00; otherwise, CC1 is set to a 1 and the number of availabl pages is returned in SR1 with the access protection set to 00 on those pages allocated. In any case, SR2 contains the address of the first available page. If specific allocation is in effect no allocation is made: SR1 and SR2 are set to zero and CC1 is set to 1.

The M:GP procedure call is of the form

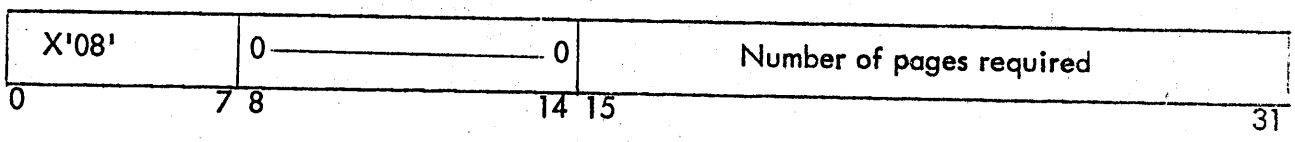
M:GP pages

Pages specifies the number of additional pages requested.

Calls generated by the M:GP procedure have the form

CAL1,8 FPT

FPT is the address of a word as shown below.



E. Free N Pages

M:FP The FP routine frees a specified number of dynamic pages from the high end of the area of core storage that may be used by the user's program. The pages freed are no longer available for use by the user's program, and an attempt by the user's program to access any of the freed pages will result in a trap. If the specified pages are not part of the user's dynamic storage area or if specific allocation is in effect, no pages are affected and condition code 1 is set to 1; otherwise, it is set to 0.

The M:FP procedure call is of the form

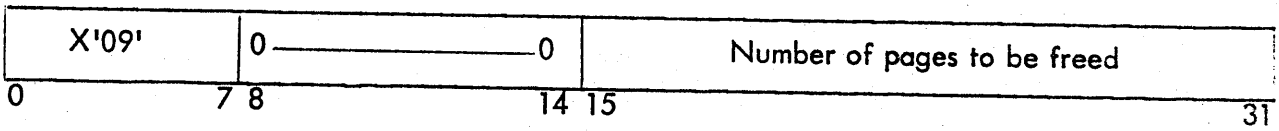
M:FP pages

Pages specifies the number of pages to be freed from use by the user's program.

Calls generated by the M:FP procedure have the form

CAL1,8 FPT

FPT is the address of a word as shown below.



F. Get Virtual Page

M:GVP The GVP routine is called to make a virtual page of memory available to the operating program. If the requested page is in use, or if physical memory limits have been exceeded or if relative allocation has been used, no pages are affected and condition code 1 is set to 1; otherwise, it is set to zero and the monitor sets the access projection to 00 on the requested virtual page.

The M:GVP procedure call is of the form

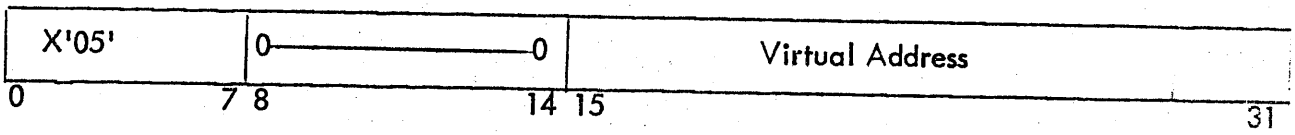
M:GVP virtual address

Virtual address specifies the address of the first word in the virtual page desired.

Calls generated by the M:GVP procedure have the form

CAL1,8 FPT

FPT is the address of a word as shown below.



G. Free Virtual Page

M:FVP The FVP routine is called to free a specified virtual page of memory. If the page to be released is not a data page, or if relative allocation is in effect, no pages are affected and condition code 1 is set to 1; otherwise, it is set to zero and the Monitor sets the access protection to 11 (no access) on the released virtual page.

M:FVP procedure call is of the form

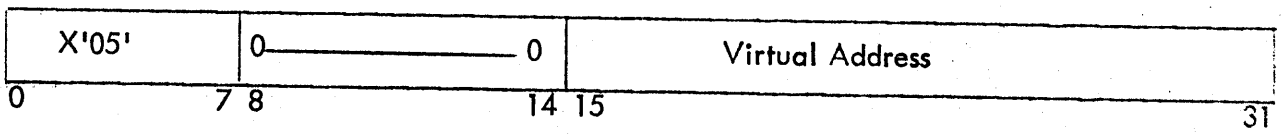
M:FVP virtual address

Virtual address specifies the address of the first word on the virtual page to be released.

Calls generated by the MFVP procedure have the form

CAL1,8 FPT

FPT is the address of a word as shown below.



NEW UTS SERVICE CALs

Three new calls have been added to UTS in order to provide setup of communication with the DELTA debugger, and the terminal I/O handler. They may only be issued by on-line user's program and are ignored if issued by a batch program.

A. Set DELTA Breakpoint

Communication between the user's program and the DELTA debugger is via the M:DELTA routine. Primary use is by DELTA in planting calls to itself in the user's program in response to instruction breakpoint requests. On execution the monitor stores the PSD and general registers in a 19 word block of user's memory (on a doubleword boundary) and places a pointer to that block in register 1. The location given in the FPT is then entered. Return to the user's program is via the TRTN routine.

The M:DELTA procedure is of the form

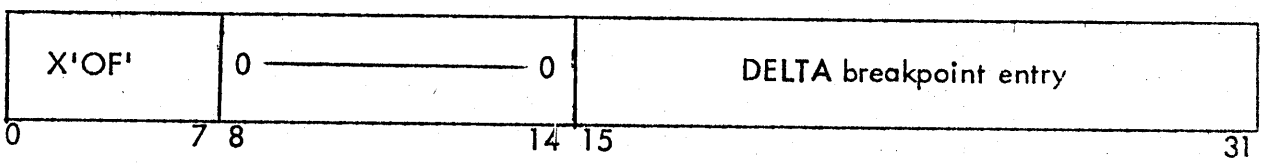
M:DELTA address

Address is DELTA's breakpoint entry location.

The calls generated have the form

CAL1,8 FPT

FPT is the address of a word as shown below.



B. Set Prompt Character

The on-line user's keyboard is proprietary: either he has control for purposes of input or UTS has control for carrying out requests and for purposes of output. Who or what is controlling the keyboard must be made clear at all times. On-line processors are assigned a prompt character which is issued to the user whenever control of the terminal is returned to him for input. This allows the user to know at all times to whom he is talking; who talked to him last and when he can type. A user program may set the prompt character to key his input requests if he wishes. Ordinarily when the control is turned over to the user a null prompt is assigned.

Current assignment of prompt characters is:

Monitor	!
Edit	*
PCL	<
LINK	:
BASIC	>
Assembler	+
FORTRAN	$
DELTA	bell
SYMCOM	:
FDP	/
user	null

M:PC The Set prompt call allows the user's program to set the terminal prompt character (identification mark). This prompt character if non null, will be output (usually at the left margin) whenever input is requested from the user's terminal (UC device.)

The procedure call is of the form

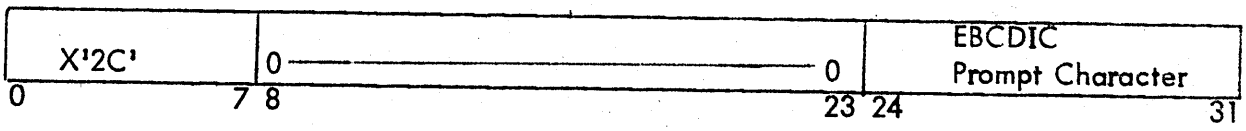
M:PC character

Character specifies the EBCDIC prompt character (identification mark) which is to be associated with the user. An EBCDIC 00 (Null) means no prompt character is desired.

Calls generated by the M:PC procedure call have the form

CAL1,1 FPT

FPT is the address of a word as shown below.



C. Change Terminal Activation and Translation Table

Translation of characters appearing on the user terminal input lines to the EBCDIC internal Sigma 7 standard, translation of EBCDIC to the proper output form for the terminal, and the determination of which characters are to be considered end-of-message or activation charact when received are all controlled by tables resident in the COC I/O handling routines. A Monitor CAL allows the user to switch among the tables available in the system.

The procedure is of the form:

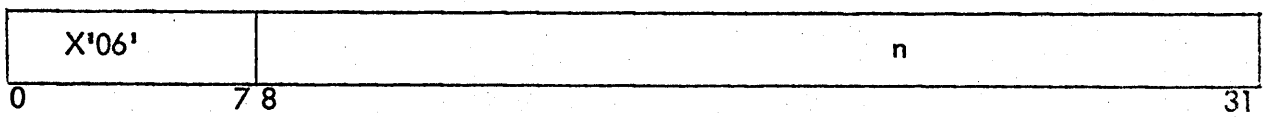
M:CT n

n specifies the number of the desired table $0 \leq n \leq 5$.

The procedure generates a

CAL1,8 FPT

FPT is the address of a word as shown below.



The current tables translate for Model 33 and 35 Teletypes, and SDS Keyboard Displays. Additional tables are contemplated for Model 37 teletypes, IBM 2741's, and Frieden 7100's. Since translation tables are assigned to lines at SYSGEN time it is unnecessary for users of fi: location consoles to use this command. Dial-up lines are another matter.

The current assignments for the n parameter are:

n	Meaning
0	Use DELTA activation set (/ = t cr lf tab)
1	Use standard Mod, 33, 35 TTY table (cr lf and ESC)
2	Use the standard K/D table (all cursor movements, hard copy signals, mode changes, and roll commands activate)
3	Reserved for Model 37 TTY
4	Reserved for IBM 2741
5	Reserved for Frieden 7100

ON-LINE BATCH DIFFERENCES

The monitor has different actions to certain CAL's depending on which they were issued by an on-line or a batch program. The CAL's which depend on the calling environment are described below.

A. Exit Return (M:EXIT)

Batch — The monitor performs any PMDI dumps that have been specified for the program and then reads the C device ignoring everything up to the next control card.

On-line — The monitor returns control to the on-line executive program, which prompts with an '!' at the terminal (UC device) for the input message.

B. Error Return (M:ERR)

Batch — The monitor outputs the message

!!JOB id ERRORED BY USER AT xxxxx

where xxxxx is the address of the last instruction executed in the program. The message plus the contents of the current register block and program status doubleword (PSD) are listed on the LL and DO devices. The monitor also lists the message

!!JOB id ERRORED

on the operator's console (OC device). Post-mortem dumps are performed, and the C device is read ignoring everything up to the next control card.

On-line — The monitor outputs the message M:ERR AT xxxxx where xxxxx is the address of the last instruction executed in the program on the UC and DO devices, if differe The monitor then returns control to the on-line executive, which prompts for the next user message with an '!'.

C. Abort Return (M:XXX)

Batch — The monitor outputs the message !!JOB id ABORTED BY USER AT xxxxx where xxxxx is the address of the last instruction executed. This message plus the contents of the current register block and program status doubleword (PSD) are listed on the LL and DO devices, if different.

The monitor also lists the message

!JOB id ABORTED

on the operator's console (OC device). The M:XXX procedure call is of the form:

M:XXX

when a job is aborted, any specified postmortem dumps are performed, but no further control commands are honored until a JOB or FIN control command is encountered.

On-line — The monitor outputs the message M:XXX AT xxxxx where xxxxx is the address of the last instruction executed in the program. This message is listed on the UC and DO devices, if different. The monitor then returns control to the on-line executive which prompts for the next user action with an '!'.

D. Type a Message (M:TYPE)

Batch — The monitor outputs the specified message on the OC device.

On-line — The monitor outputs the specified message on the UC device.

E. Request a Key-in (M:KEYIN)

Batch — The monitor outputs the specified message on the OC device and enables the operator's reply to be returned to the user's program.

On-line — The monitor outputs the specified message on the UC device and enables the user's reply to be returned to the program. A prompt character is sent if one was specifi by a M:PC.

F. Connect to Interrupt or BREAK key (M:INT)

The purpose of this procedure is to allow execution of the program to be controlled from the terminal or console. When control is given to the INT routine, the PSD and general registers are stored in a 19-word block of user's memory (on a doubleword boundary) and a pointer to word 0 of that block is placed in current general register 1. The TRTN routine may be used to restore control from a console or terminal interrupt.

Batch — The monitor enables the user's program to be connected to a console interrupt (key-in addressing the program). This enables the user's program to be controlle from the operator's console.

On-line — The monitor enables the user's program to be connected to a teletype interrupt (Break key). This enables the user's program to be controlled from the terminal.

The monitor INT routine is called by an on-line program to set the address of a routine to be entered when the user presses the BREAK key on his terminal. The execution of this procedur causes the monitor to store the PSD and general registers into a 19-word block of user's memory (on a doubleword boundary) and a pointer to word 0 of that block is placed in current register 1. The TRTN routine (see M:TRTN) may be used to restore control to the user's program.

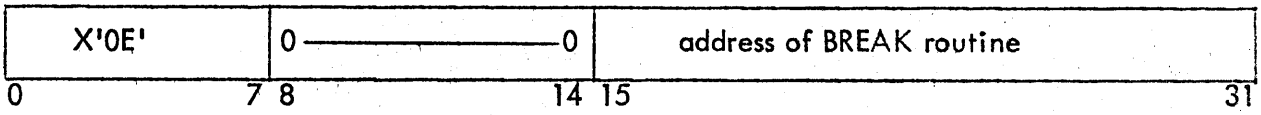
The M:INT procedure call is of the form:

M:INT address

Address specifies the location of the entry to the programs BREAK response routine. Calls generated by the M:INT procedure have the form

CAL1,8 FPT

FPT is the address of a word as shown below:



ERROR AND ABNORMAL MESSAGES

All error or abnormal conditions which normally result in the batch monitor continuing to the next job step will be processed for on-line users as follows:

The monitor outputs two messages. The first message has the form

mmmm ...

where mmmm ... is the specific message identifying the error or abnormal conditions. The message reside in the system file (:MESS). The keys to the error text records are the codes established by th monitor for the error or abnormal conditions.

The second message has the form

EXECUTION STOPPPED AT xxxxx

where xxxxx is the location of the last instruction executed.
These messages are listed on the UC and DO devices, if different. The monitor then returns control to the On-line Executive, which prompts for the next user action with an '!'.

SUMMARY OF CAL's

There are four CALL instructions (CAL1, CAL2, CAL3, and CAL4) provided by the Sigma 5/7 hardware. Execution of a CAL instruction causes the executing program to trap to the monitor where a check is made for the validity. CALL instructions are used for requesting monitor services. The requester may be a user, processor, real-time task, or the monitor. Each CAL trap is decoded to determine the service requested (if any) and the requestor. If valid, the requested service is performed. If invalid in either type of CALL or type of service requested, the request is not honored and the user is informed.

Of the four CAL's provided by the Sigma 5/7, CAL3 and CAL4 are reserved for the installations or users; CAL2 is reserved for monitor use and CAL1 is divided into user, real-time and monitor services.

The CAL's currently assigned are listed below in five categories: 1) On-line, Batch, and Real-Time; 2) Batch only; 3) On-line only; 4) Real-time only; and 5) monitor only.

A. On-line, Batch, Real-Time

CAL	address	FPT CODE	FUNCTION
CAL,1	FPT	X'01'	M:REW
		X'02'	M:WEOF
		X'03'	M:CVOL
		X'04'	M:DEVICE (PAGE)
		X'05'	M:DEVICE (VFC/NOVFC)
		X'06'	M:SETDCB
		X'0B'	M:DEVICE (DRC/NODRC)
		X'0C'	M:RELEC
		X'0D'	M:DELREC
		X'0F'	M:TFILE
		X'10'	M:READ
		X'11'	M:WRITE
		X'12'	M:TRUNC
		X'14'	M:OPEN
		X'15'	M:CLOSE
		X'1C'	M:PFIL
		X'1D'	M:PRECORD
		X'20'	M:DEVICE (LINES)
		X'21'	M:DEVICE (FORM)
		X'22'	M:DEVICE (SIZE)
		X'23'	M:DEVICE (DATA)
		X'24'	M:DEVICE (COUNT)
		X'25'	M:DEVICE (SPACE)
		X'26'	M:DEVICE (HEADER)
		X'27'	M:DEVICE (SEQ)
		X'28'	M:DEVICE (TAB)
		X'29	M:CHECK
		X'2A'	M:DEVICE (NLINES)
		X'2B'	M:DEVICE (CORRES)

CAL	address	FPT CODE	FUNCTION
CAL1,2	FPT	X'01'	M:PRINT
		X'02'	M:TYPE
		X'04'	M:KEYIN
		X'10'	M:MERC
CAL1,3	FPT	X'00'	M:SNAP
		X'01'	M:SNAPC
		X'02'	M:IF
		X'03'	M:AND
		X'04'	M:OR
		X'05'	M:COUNT
CAL1,8	FPT	X'01'	M:SEGLD
		X'04'	M:GVP
		X'05'	M:FVP
		X'08'	M:GP
		X'09'	M:FP
		X'0A'	M:SMPRT
		X'0B'	M:GL
		X'0C'	M:GCP
		X'0D'	M:FCP
		X'0E'	M:INT
		X'10'	M:TIME
		X'11'	M:STIMER
		X'12'	M:TTIMER
		X'14'	M:TRAP
CAL1,9	1		M:EXIT
	2		M:ERR
	3		M:XXX
	4		M:STRAP
	5		M:TRTN

B. Batch Only

CAL	address	FPT CODE	FUNCTION
CAL1,4	FPT	X'00'	M:CHKPT
		X'01'	M:RESTART
CAL1,8	FPT	X'02'	M:LINK
		X'03	M:LDTRC

C. On-Line Only

CAL1,1	FPT	X'2C'	M:PC
CAL1,8	FDT	X'06'	M:CT
CAL1,8	FPT	X'0F'	M:DELTA

D. Real-Time Only

CAL1,5	FPT	X'00'	M:TRIGGER
		X'01'	M:DISABLE
		X'02'	M:ENABLE
		X'03'	M:DISARM
		X'04'	M:ARM
		X'05'	M:DCAL
		X'06'	M:CAL
		X'07'	M:SLAVE
		X'08'	M:MASTER
		X'09'	M:SBACK
		X'0A'	M:RBACK
		X'0B'	M:TERM
		X'0C'	M:RXC
CAL1,9	7		} Reserved for real-time extensions
	8		
	9		
	A		
	B		
CAL1,A	FPT	X'00'	Save Monitor's interrupted environment
		X'01'	Restore Monitor's interrupted environment

E. Monitor Only

CAL	address	FPT CODE	FUNCTION
CAL1,1		X'16' X'17'	Direct Disc Read Direct Disc Write
CAL1,9	6	----	Close Cooperative File
CAL1,B		----	Event Mark Event Count Event Time Event Auto-Display Control } Reserved for generalized ev measurements
CAL2,0		----	Branch to overlay segment (0B)
CAL2,0		----	Branch and save segment number (OBAL)
CAL2,2		----	Restore segment and B*SR4 (OBSR4) } Used fc interno Monito overlay
CAL2,3	code[1]	----	System Recovery
CAL2,4	code[1]	----	Reserved for internal debug routine

[1] The code appears in the address fields of the CAL instruction and is internally assigned.

NUMERICAL LIST OF CAL's

The following list gives all UTS CAL in numerical order with the M: proc name for invoking the routine and a brief description of the function performed. Restrictions on usage to on-line, batch, and real-time are given in the use code column on the left.

m	restricted to monitor use
o	restricted to on-line use
r	restricted to real-time use
b	restricted to batch use
-	usable in all environments

If a CAL is given which is illegal for the current user it is treated in the same way as an illegal instruction.

CAL's marked with an asterisk (*) are new to UTS or have different or extended functions relative to BPM.

Numerical List Of Monitor CAL's

USE CODE	CAL	FPT hex code	M: NAME	UTS	Description
-	CAL1,1 FPT	1	REW		
-		2	WEOF		
-		3	CVOL		
-		4	DEVICE (PAGE)		
-		5	DEVICE (VFC)		
-		6	SETDCB		
-		B	DEVICE (DRC)		
-		C	RELEC		
-		D	DELREC		
-		F	TFILE		
-		10	READ		
-		11	WRITE		
-		12	TRUNC		
-		14	OPEN		
-		15	CLOSE		
m		16	----		Direct disc read
m		17	----		Direct disc write
-		1C	PFIL		
-		1D	PRECORD		
-		20	DEVICE (LINES)		
-		21	DEVICE (FORM)		
-		22	DEVICE (SIZE)		
-		23	DEVICE (DATA)		
-		24	DEVICE (COUNT)		
-		25	DEVICE (SPACE)		
-		26	DEVICE (HEADER)		
-		27	DEVICE (SEQ)		
-		28	DEVICE (TAB)		
-		29	CHECK		
-		2A	DEVICE (N LINES)		
-		2B	DEVICE (CORRES)		
o		2C	PC	*	Set prompt character
-	CAL1,2 FPT	1	PRINT		
—		2	TYPE	*	Type message to operator (or user)
-		4	KEYIN	*	Type message and await response

USE CODE	CAL	FPT hex code	M: NAME	UTS	Description
-	CAL1,2 FPT	10	MERC		
-	CAL1,3 FPT	0	SNAP		
-		1	SNAPC		
-		2	IF		
-		3	AND		
-		4	OR		
-		5	COUNT		
b	CAL1,4 FPT	0	CHKPT		
b		1	RESTART		
r	CAL1,5 FPT	0	TRIGGER		
r		1	DISABLE		
r		2	ENABLE		
r		3	DISARM		
r		4	ARM		
r		5	DCAL		
r		6	CAL		
r		7	SLAVE		
r		8	MASTER		
r		9	SBACK		
r		A	RBACK		
r		B	TERM		
r		C	RXC		
	CAL1,6	unused			
	CAL1,7	unused			
-	CAL1,8 FPT	1	SEGLD		Load overlay segment
b		2	LINK		
b		3	LDTRC		
-		4	GVP	*	Get virtual page
-		5	FVP	*	Free virtual page
o		6	CT	*	Change COC Table
-		8	GP	*	Get core page
-		9	FP	*	Free core page
-		A	SMPRT		
-		B	GL	*	Get available core limit
-		C	GCP	*	Get core page in commo
-		D	FCP	*	Free core page in comm
-		E	INT	*	Connect to interrupt of BREAK key
o		F	DELTA	*	Return to DELTA Debugg
-		10	TIME		
-		11	STIMER		
-		12	TTIMER		
-		14	TRAP		

USE CODE	CAL	FPT hex code	M: NAME	UTS	Description
-	CAL1,9 1	--	EXIT	*	Normal program terminator
-	2	--	ERR	*	Error termination of job ste
-	3	--	XXX	*	Error termination of job
-	4	--	STRAP		
-	5	--	TRTN		
m	6	--	--		Close cooperative file
r	7	--			
r	8	--			
r	9	--			Reserved for real-time
r	A	--			
r	B	--			
r	CAL1,A FPT	0	-----		Save monitor environment
r		1	-----		Restore monitor environmer
m	CAL1,B code	--	Event marker	*	
m	CAL1,C code	--	Event counter	*	
m	CAL1,D code	--	Event timer	*	Monitor performance measurement
m	CAL1,E code	--	Display	*	
m	CAL2,0 --		OB		Branch to overlay segment
m	CAL2,1 --		OBAL		Branch to overlay and save return
m	CAL2,2 --		OBSR4		Restore segment and return *SR4
m	CAL2,3 code		Reserved for error recovery and diagnosis		
m	CAL2,4 code		Entry to executive DELTA		

All remaining CAL2,x instructions are reserved to monitor use.

All CAL3,x and CAL4,x instructions are available for installation assignment.

OPERATIONAL LABELS FOR ON-LINE USE

An operational label is a name (and a set of monitor records) used to identify a logical input/output function. All I/O activity (Reads and Writes) take place through the information in a DCB. One piece of information there is the device address or, alternately, an operational label which in turn is connected to the device. The connection of devices to DCB's through operational labels allows the installation the capability of changing the device assignment of a particular I/O class. The batch user may change the assignments for the duration of the job by using !STDLB cards or the operation may make permanent changes using !SYST key-ins.

For on-line operation the operational label assignments are kept separately from batch and are not changeable by the user. Change by the operator is a possibility and is left as an open question. Table 1 below lists the assignments of op labels for on-line.

TABLE 1. Monitor Operational Labels for On-Line Users

Label	Standard Use	Assigned Device	I/O Function
BI	Binary input	Disc	Read number of bytes specified
C	Control input (same as UC)	Terminal	Read number of bytes specified or to message complete
CI	Compressed input	Disc	Read number of bytes specified
EI	Element input	Disc	Read number of bytes specified
SI	Source input	Disc	Read number of bytes specified
BO	Binary output	Disc	Write number of bytes specified
CO	Compressed output	Disc	Write number of bytes specified
EO	Element output	Disc	Write number of bytes specified
SO	Source output	Disc	Write number of bytes specified
PO	Punch output	Disc	Write number of bytes specified
UC	Users Terminal	Terminal	Read or write number of bytes specified
DO	Diagnostic Output	Terminal	Break into carriage-size records, insert carriage returns, and type up to 132 characters.
LO	Listing output	Line Printer	Write number of bytes specified up to one line