REVISIONS				702489
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	C. F. Shelburne	3/31/69
CHECK	D. E. Brown	3/31/69
APPR.	E. K. Shuffstall	3/31/69

SDS
SCIENTIFIC DATA SYSTEMS
SANTA MONICA, CALIFORNIA

TITLE
UNIVERSAL TIME-SHARING SYSTEM (UTS)
FUNCTIONAL SPECIFICATION

MODEL NO.

NEXT ASSY.

SIZE	DWG. NO.	CHANGE LETTER
A	702489	

SCALE | DO NOT SCALE DRAWING | SHEET 1 OF 308

UNIVERSAL TIME-SHARING SYSTEM (UTS)

FUNCTIONAL SPECIFICATION

VOL. 1 - PARTS I - VI

By

E. Bryan
B. Doeppel
J. Smith

31 March 1969

PREFACE TO THE MARCH REVISION

The March issue of the UTS Functional Specification includes updates, revisions, major additions, and minor editorial changes to bring the spec in line with changes suggested by various SDS departments and with the current course of UTS implementation. It is issued in two volumes, the first containing Parts I-VI which provide a general overview of UTS, and the second containing Parts VII-XIII which include details of several of the UTS subsystems.

Any errors or omissions which reviewers find in the current document should be called to the attention of Programming Development through Ed Bryan.

Details of the revisions and additions are:

Part I	-	Minor editorial changes
Part II	-	Major addition. Description of operation and sample outputs for the performance control program.
Part III	-	Minor editorial changes
Part IV	-	Addition of a description of UTS virtual memory layout
Part V	-	Minor editorial changes
Part VI	-	Major revisions including the addition of SET Commands for I/O assignments.
Part VII	-	Replacement of the description of EDIT by reference to the BTM version plus extensions and descriptions
Part VIII	-	Addition of several new DELTA commands and clarification of the operation of others
Part IX	-	Addition of a few new commands; clarification of others
Part X	-	Minor changes only
Part XI	-	Expanded description of the memory management routines; addition of two new CALs
Part XII	-	A new part describing I/O services for the terminal
Part XIII	-	A new part describing METASYMBOL operation

OVERALL TABLE OF CONTENTS

Part I. OVERALL SUMMARY OF UTS

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.

A. Processors and Associated Languages Primarily Related to On-line Users

1. 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 subsystem of UTS. Each subsystem 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.

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

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

4. 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.

5. 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)

B. Distinct bodies of code that regulate and provide information about the activities of UTS and its users. These include routines for: 1) scheduling activities; 2) managing time and storage; 3) measuring and displaying the cumulative and individual behavior of UTS and its users; 4) handling information passing to and from remote terminals on an asynchronous basis; and 5) fixes required to use the memory map.

C. Changes and fixes to BPM and its component processors. These include: 1) modifying compilers and assemblers so that they produce information necessary for on-line debugging; 2) creating versions of processors and run-time packages (and all other public routines) that are reentrant and, therefore, capable of being shared among more than one user; 3) simplifying input-output interfacing with BPM and speeding-up its file-management services; 4) 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; and 5) fixes required to use the memory map.

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 antogonistic. 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, but makes no allowance for service to resident real-time programs -- an effect which can be catastrophic to reasonable service. Delays to typical on-line requests of 30 users should exceed .4 seconds no more than 10% of the time and exceed four 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 .4 seconds are noticeable, particularly to people using processors that maintain intra-line dialogues with their users, when delays cannot be blanketed by the carrier-return times associated with typing requests. Although typical delays will be just less than .4 seconds, variations will occur frequently. However, users will hardly ever have to wait more than three or four 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 .6 seconds still occurring 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 two to three times greater than that required to service the typical demands of 30 on-line users. What is left of main-frame capacity (65% 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.

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. Almost all UTS installations will require a 7212 and a 7232.

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 and results in severe delays in both on-line response and batch turnaround. With the feature, uncomplicated, low overhead management and scheduling disciplines can be used, as can reentrant 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 basic 80K core configuration.

It must be clearly understood that large transient or installation-systematic 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 redesigns 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.

SERVICES AND FACILITIES

UTS provides its users three classes of service:

A. 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.

B. 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. Remote batch operations from the 7670 will be available in UTS as specified for BPM. In particular, compilers and assemblers are compatible across the two classes of service at both source and relocatable levels:

1. processors for SDS FORTRAN IV and Meta-Symbol are available both in batch and on-line;
2. programs compiled or assembled in batch can be linked with those produced on-line, and can be run and debugged on-line;
3. programs compiled or assembled on-line can be linked and run in batch.

C. 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 subsystems 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 subsystem, whichever seems appropriate in context.

1. 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 subsystems 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; and who last talked to him. These are often clear in context so long as the system adheres to some reasonable rules of behavior.

Teletypes -- either SDS 7015 or teletype models 33 and 35 -- are assumed to be the most common on-line terminals used with the UTS system. SDS 7550 and 7555 keyboard displays are compatible terminals for UTS and special software is in progress which will capitalize on the special capabilities of the keyboard display for editing. Later versions will provide for use of IBM 2741 and Model 37 teletypes as on-line terminals.

2. 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) SDS FORTRAN compilations, and Metal-Symbol assemblies; d) linking and loading of relocatable programs; e) controlling the execution of programs; f) saving intermediate core status for later resumption; and g) submitting batch jobs

and monitoring their status. 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 subsystems.

3. Text-Editing Subsystem (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 updating by EDIT and their use by other processors.

4. Peripheral and Information Control Subsystem (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.

5. 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.

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

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

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

(d) Searching programs and data for specific elements and subelements.

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.

6. FORTRAN Debugging (FDP)

The language is easy to use, readable, and more powerful than 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. The complete specification for FDP is given in document #702528.

7. Symbol-Control Subsystem (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.

8. Object-Program Linking (LINK)

All operations that can be performed under the LINK executive command can be performed under the subsystem. The notation and conventions for specifying the retention, deletion, and merging of internal symbols remain

the same. On the surface, the subsystem'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.

9. Subsystem for Basic Programmers (BASIC)

Under this subsystem, Basic programs may be composed, edited, executed, and debugged. All the appropriate commands of the editing and debugging subsystems 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 subsystem's language and its responses are covered in complete functional specifications for the subsystem (#702452)

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.

Part II. PREDICTING, MEASURING, TUNING UTS

TABLE OF CONTENTS

INTRODUCTION

The effectiveness and quality of each class of service (batch, real-time, on-line) depends 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 interactive service to 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 problems, 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 required 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 6/10 seconds for up to sixty users. Delays exceeding 6/10 seconds should be experienced 10% of the time; delays greater than four seconds are expected to 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 are 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; and 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 are 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 neglibile, 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 -- (18K 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". The system operator may vary the level of this preferred treatment for batch and may, if desired, "fix" the batch job into core permanently.

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. UTS Installation Control

This section describes the parameters which the installation manager may use to control the overall operation of the UTS system. Three broad areas are covered:

- System Performance Management

  Includes those parameters which control system operation by limiting the number of on-line users, controlling batch sequencing, adjusting computing quantas, and setting subprocessor use.

- Login Controls

  These controls limit the entry of on-line users into the system and limit their use of files, disc packs, tapes, and system

peripherals. In addition, through this mechanism the on-line user may get direct connection to a particular processor or other standard software programs.

- Use Accounting

  A variety of use parameters are separately measured and charged by reference to a rate schedule. Several rate schedules may be operative for different users at the same time and the schedules may be changed dynamically to provide differential rates by time of day or other factors at installation option.

1. Performance Management

Certain dynamic parameters are stored in core memory and used to control UTS scheduling, accounting, and overall operation. These parameters are initialized at SYSGEN time to default values, but may be set during a SYSGEN within certain ranges, by !IMC (for Installation Management Control) cards. Once the system is in operation, the parameters may be set through a user console via the control program described in Section E. Two error messages may result from the SYSGEN process: "UNKNOWN" for unrecognized parameter names, and "INVALID" for values outside the allowable range. No change of value will be made in cases of error and the default will apply. The SYSGEN command form is:

!IMC name = value

Names and allowable values are listed in Section E below.

2. Login Controls

During user login three items are requested from the user: a) name, b) account number, and c) password. In both batch and on-line environments these items are used to reference a "login" file which controls entry of the job into the system and, if the job is allowed, controls the type of usage and system privileges extended the user. This file is created by a specially authorized program similar to the BTM SUPER program which may be run in the batch stream or from any user console.

The login file exists under the LOGINLBE account and has a name and password known only to the processors dealing directly with the file, i.e., a) the logon processor, and b) the system processor used to enter, delete, and update records within the login file. Records within the file are named by the concatenation of a) account number and b) the name of each valid user. The record contains the user's password (which he may change by using the Password Command) and other information which controls the system facilities granted to that user. Other than password, this account control record may only be changed, deleted, or entered into the file, by running a specially authorized system program under management control. One of the records of the login file is a special system record which is used by a system manager to prevent unauthorized meddling in the file. This record has the name "LOGINLBE" (formed by the concatenation of the account "LOGIN" and the name "LBE". Initially, the password is blank. As long as the password is blank, anyone may invoke the system update program to enter,

change, or delete records in the login file by logging in with account LOGIN and name LBE. Once a password has been entered (using Password which runs under account LOGIN, name LBE), only by supplying this new password at logon time can the login file be altered. The password may be changed or reset to blanks any number of times. This mechanism is supplied to provide security to the list of accounts within the system.

Contents of the login record are:

LR:PW	The user's password (0-8 characters)
LR:PU LR:TU LR:DU LR:FU	Are flags which allow (when set) use of a) peripheral devices (printer, punch, paper tape, card reader) via symbionts, b) magnetic tapes, c) disc packs, and d) RAD file space. These flags are transferred to JIT and control user I/O CALs. Additional flags may be set or reset to control use of the various system processors. They will be defined as the need arises.
LR:CP	Eight-character name of the "automatically connected" processor. Automatic connection to one of the system processors is controlled by this item which, if set, causes an automatic call for the processor
LR:CF	If set, the user is automatically connected to a SAVEd program as if he had given a GET command (eight-character file name).
LR:CS	Connects the user to a charge structure for accounting.

3. Use Accounting

At login time each job (user) is connected to a charge class which in turn connects to a charge rate table in much the same way that an I/O operational label connects to a device. Management may control usage by changing the rate tables used by different charge classes as a function of time of day or by type of usage.

Charges for each activity are accumulated dynamically as they occur in the user's context area so that the rate table may be switched or the charge class changed during a job's execution.

Initially, three rate schedules will be supplied. One each for on-line, batch, and real-time users. Two charge rate tables will be supplied with initial rate parameters as outlined below. New rate tables may be added by SYSGEN and the connection to charge classes established.

Installation charge totals are accumulated in a special record in the system accounting log as each batch job completes and as each on-line user logs off. This record is output at system shut-down or on operator request (WRITELOG keyin) together with the other accounting records in the file.

Charge Rates

For each user, batch, on-line or real-time individual, counts are kept in his context block (JIT) of several activities:

a) CPU use time in 2 ms units
b) Number of file I/O CALs
c) Number of console input CALs (interactions)

d) Console time in 600 ms units (.01 minutes)
e) Number of tape reels or disc packs mounted

These items are printed at the end of the job along with the current summary produced by BPM. A limited subset is output automatically at the time an on-line user logs off. Other details of use accounting are available through the TEL status command. At log-off the automatic report includes CPU time, console time, number of interactions, and total charge as follows:

```
!OFF
CPU=H:MM.MMM, CON=H:MM, INT=NN, CHG=XXXX
```

CPU time is in hours, minutes, and thousandths of minutes, CONsole time in hours and minutes, INTeraction count is an integer, and charges are in units as explained below. In all cases high order zeros are omitted except following the decimal point.

The total charge for a job is computed by accumulating units at the rates shown in the rate table as each chargeable event occurs. Names of the rates for each item and default values are given below. This is the RT:1 rate table

Charge for	Name	r value	unit calculation	approx. charge (10^5 units = \$1)
CPU time	R:CUP	6	r·t	5¢ per sec for a 4K program;
CPU·Core size	R:COR	1	r·t·s	s in 1K blocks; t in 2 ms tics.
On-line transaction	R:OT	10^4	r·n	10¢ per interaction
I/O CALs	R:FIO	10^3	r·n	1¢ per I/O command
Console time	R:CON	200	r·T	12¢ per console hour
Tape & Disc Pack	R:TD	10^5	r·n	\$1 per tape or pack mounted.
File·Time Usage	R:FIL	10^4	r·D·P	10¢ per page per day.

The monetary values are for example only. The value of a charge unit is up to the installation as is the makeup and assignment of charge classes.

Rate tables are generated by SYSGEN and resident in core during system operation. Each is a half word table containing the six rates given above. Without SYSGEN instructions to the contrary three rate schedule header words will be generated RS:1, RS:2, and RS:3 all connected to the standard rate table RT:1 above during prime shift. During non-prime shift the three schedules will be connected to RT:2, a second standard table, which arbitrarily contains half the values of RT:1.

Rate tables, ~~rate schedules~~ (change class), and the initial connections between them may be established for a given installation through SYSGEN.

~~Rate schedules~~ (change class) are connected to rate tables by commands of the form:

```
!RS:3 - RT:1
```

A ~~rate schedule~~ (CC) header is created for each such entry.

A rate table and its contents are defined by commands of the form:

```
!RT:1, R:CPU = 6,  R:COR = 1,  R:OT = 10000;
!R:FIO = 1000,  R:CON = 200,  R:TD = 100000
```

Values must be in the range $0 \rightarrow 2^{17} - 1$ or the message "VALUE OUT OF RANGE" message will be printed and value zero substituted. Zero values are used if rate table entries are not specified.

1) amount of use used by I/O driver prog.
2)

E. Performance Control

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 are 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 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 is adjustable so that operational data can be displayed more often if special tests are to be made.

1. Function in General

Performance measurements are important during the initial debugging stages, and their importance increases as the system is tuned to meet the load of the user's particular environment. In the debugging stages these measures are most relevant to the designers and implementers of the system. Later they are of primary interest to the installation manager and the maintainers of the system.

Some of the functions of performance measurement are:

a) To measure how well the system performs.

b) To indicate weak points in the system.

c) To suggest the causes of such weaknesses.

d) To warn of immediate problems; e.g., permanent storage is filling up, response time is becoming noticeably slower, large numbers of console errors are occurring.

e) To help tune the system for both current and general load conditions.

f) To measure the importance of various parts of the system; e.g., to measure the relative use of various processors in CPU and connect time. This might have implications for whether a particular processor is dropped or whether its use justifies the effort to add new capabilities.

Having performance measures on-line means that tuning of the system can be done in response to the current state of the system. If there are problems in the system (e.g., an unusual number of disc errors), the installation manager will not be the last to know. Knowing now instead of later means that he can cause action to be taken now instead of later.

On-line performance control is the capability to modify the basic system parameters (such as max core size allowed on line users) in response to on-line performance measures and other information.

2. Summary of Performance Monitoring and Control

UTS combines on-line performance measurement and management in the CONTROL program. The user can display selected control parameters and modify their values. He can also cause the display of values that measure the performance of the system. These values may be displayed periodically at a time interval specified by the user. The user may specify one of several canned displays by name or he can build his own display.

The following is an example of how the program can be used to modify control values and display performance measures.

!RUN CONTROL cr	The user calls the CONTROL program.
-SL:OU=40 cr	The user sets SL:OU (i.e., the max no. of on-line users) to 40. - is the prompt character for the program cr stands for a carriage return or line feed
-SET UP DISPLAY 4 cr	The user sets up canned display number 4 which gives a summary of the system performance.
-USE ITEMS DISPLAY cr	The user requests a use items display.
INTERVAL IS--60 cr	The user is prompted for the time interval. He indicates that the interval between displays should be 60 seconds.
FIRST TIME PERIOD HAS BEGUN	60 seconds later the first instance of the display occurs.

	Overall	Sample
Number of Users		57
Tasks per Minute per User		3.7
% of Tasks which are Interactive	87.3	95.8
CPU msecs per Interactive Task	15	5
90% point for Response Time (in seconds)	0.2	0.1
Execution Multiplication Factor	20	5
Number of Users in Core		10
RAD and Tape Reads and Writes per sec	31	25

The display is repeated several times at 60 second intervals.

Then the user hits the break key and is prompted for another command. He decides to exit from the program.

```
-EXIT  cr
```

```
EXIT. CR?--YES cr
BYE.
```

The user is asked if he wants to exit. He indicates that he does.

```
!
```

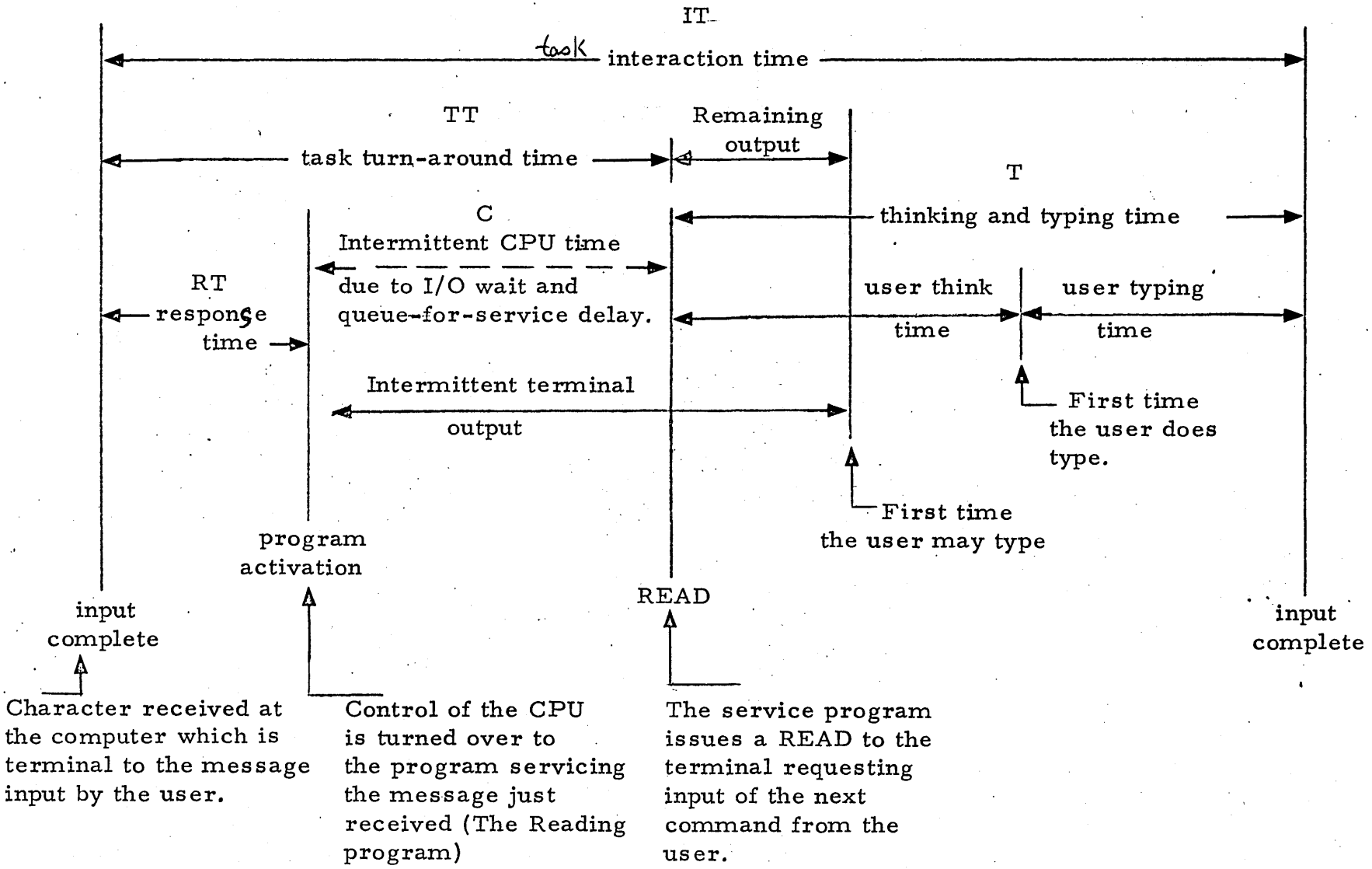
The TEL prompt character appears indicating control has been returned to TEL.

3. Key Concepts

3.1 Definitions for the Breakdown of Terminal Interaction

Interaction time is the time between the completion of one input command and the completion of the next. Response time is the time between the completion of input and the first program activation. Task turnaround time is the time between the completion of input and the following terminal read. Compute time is the time spent in computing in one interaction period. Thinking and typing time is the time between the terminal read by the program and the end of the user response (input complete). Session time is the time between log-on and log-off.

3.2 Diagram Explaining Concepts Relating to Terminal Interaction.



DRAWING NO. 102489
SHEET 37 OF 308

4. Types of Items That can be Displayed

An item is a control parameter or a use item. A control parameter is a parameter of the system which can be modified to tune the system; e.g., the maximum number of on-line users is a control parameter. Changing its value may change average response time, etc.

A use item is one of the following:

a) A use distribution
b) A use group
c) A siding

A use distribution shows what percentage of occurrences of a particular kind of event falls within given ranges on an appropriate scale. An average is also included with each distribution. For example, there is a distribution for the amount of compute time per interaction. The distribution shows the percent of interactions in which compute time is under one millisecond; the percent of interactions in which compute time is between one and two milliseconds; etc.

A use group is a group of related use values plus text in the form of sidings and headers to explain the values.

A siding is text that appears to the left of a value and helps explain the value.

5. Items That can be Displayed

A list of items that can be displayed is given below. The names are the names used in control commands to add or drop items from the list of items to be displayed.

5.1 Control Parameters

Name	Description	Unit	Low	High	Default
SL:OC	Max core size allowed on-line users	K words	1	Core Size	8
SL:OU	Max number of on-line users.	Users	1	128	32
SL:TB	Number of characters at which block terminal output.	Characters	1	256	40
SL:UB	Number of characters at which unblock terminal output.	Characters	1	SL:TB	10
SL:OF	Max file space allowed on-line users.	K words	0	RAD Size	100
SL:OT	Max number of tapes allowed on-line users.	Tapes	0	No. of Drives	1
SL:BB	Batch Bias. The percent of execution time which batch computation receives in its turn.		0	100	50
SL:BP	Batch Priority relative to on-line users. (1 for equal; 0 for low).		0	1	1
SL:BL	Batch Lock. If set (i.e., 1), enforces a partitioned system in which the batch job is never swapped.		0	1	0
SL:QMIN	The amount of uninterrupted computing guaranteed a user after selection.	Milliseconds	0	SL:QUAN	40
SL:QUAN	The time slice by which com-	Milliseconds	SL:QMIN	$2^{31}-1$	300

5.2 Use Groups

Name	Description
SUMMARY	Overview of the System Number of Users Tasks per Minute per User % of Interactive Interactions Milliseconds per Interactive Task 90% Point for Response Time Execution Multiplier Users in Core RAD and Tape Reads and Writes per Second
CPU	Percent CPU Time (since the system came up and during the last time period) for: On-line User Programs On-line Monitor Services Batch User Programs Batch Monitor Services Overhead, i.e., scheduler, COC and symbiont Idle, i.e., no service request from any user Swap WAIT, i.e., the only service request available is not yet in core.
CPU PROC	Percent CPU time by Processor (since the system came up and during the last time period) for: Basic Delta Edit FORTRAN Metasymbol User Programs
USE PROC	Current Number of Active Users by Processors for: Basic Delta Edit FORTRAN Metasymbol User Programs

4 Use Distributions (call per interaction)

Name	Description	Scale	Unit
SYS RESP	Distribution and average of response time for the whole system.	Log	Seconds
SYS INTE	Distribution and average of interaction time for the whole system.	Log	Seconds
SYS THIN	Distribution and average of thinking and typing time.	Log	Seconds
SYS TURN	Distribution and average of task turnaround time for the whole system.	Log	Seconds
SYS COMP	Distribution and average of compute time for the whole system.	Log	Milliseconds
X TYPE	Analogous to SYS TYPE*	Log	Seconds
X TURN	Analogous to SYS TURN*	Log	Seconds
X COMP	Analogous to SYS COMP*	Log	Milliseconds
SYS INPU	Distribution and average of input length	Linear	Characters
SYS OUTP	Distribution and average of output length	Linear	Characters
X INPU	Analogous to SYS INPU*	Linear	Characters
X OUTP	Analogous to SYS OUTP*	Linear	Characters
SYS SWAP	Distribution and average of users to swap out per swap.	Special	Users

*where X = BAS for Basic,
DEL for Delta,
EDI for Editor
FOR for Fortran,
MET for Metasymbol, or
USE for User

Name	Description
I/O	I/O Rates (per second since the system came up and per second in the sample) for: Service Requests, i.e., CALs Terminal Reads and Writes Characters Input from Terminals Characters Output to Terminals RAD and Tape Reads and Writes Symbiont and coop Reads and Writes Out Swaps
CON TIME	Timing Averages (average per interaction since the system came up and in the sample period) for: Interaction Time (in seconds) Think and Typing Time (in seconds) Turnaround Time (in seconds) Response Time (in milliseconds) CPU Time (in milliseconds)
USERS	Current Number of Users for: All Active Users In Core Compute Bound Inputting Batch
INTERACT	Number of Interactions per minute by Processor for: Basic Delta Edit FORTRAN Metasymbol User Programs
OTHER	Miscellaneous Other Values Average Size of Program Core Time Ratio of Single to Multiple Swaps CPU % for Interactive Tasks Executable Users not in Core Executable Users in Core with Priority above the Compute Queue.

5.3 Use Sidings

Name	Expanded Name
LOG SCAL	Log Scale
LIN SCAL	Linear Scale
SLI SCAL	Special Linear Scale

These sidings look as follows when displayed:

LOG SCALE	LINEAR SCALE	SPECIAL LINEAR SCALE
AVG	AVG	AVG
<1	<5	<1
<2	<10	<2
<5	<15	<3
<10	<20	<4
<20	<25	<5
<50	<30	<6
<100	<35	<7
<200	<40	<8
<500	<45	<9
<1K	<50	<10
<2K	<55	<11
<5K	<60	<12
<10K	<65	<13
& UP	& UP	& UP

6. Command Summary and Formats

6.1 Command Summary

6.1.1 Commands for Setting up Displays

The USE ITEMS DISPLAY Command displays those items that have their print flags on. The primary purpose of this group of commands is to allow the user to specify the display he desires by turning on the print flags for just those items he wishes to display.

Command	Function
ADD	Turns on the print flag for the specified items.
DROP	Turns off the print flag for the specified items.
SET UP	Turns on the print flags for a numbered canned display. Turns off all other print flags. Can be used to turn off all print flags.
LIST	Lists the names of all items with their print flags on.

6.1.2 Commands Relating to Control Parameters

Name	Function
CONTROL PARAMETERS DISPLAY	To display all control parameters with their print flags on.
CONTROL PARAMETERS DISPLAY!	To display all control parameters.
SET	To set the value of a specified control parameter. The set command can also be used to lock processors in core or to unlock them.

6.1.3 Commands Relating to Use Items Display

Name	Function
USE ITEMS DISPLAY	Displays those use items which have their print flags on at a specified time interval.
PROCEED	Continues an interrupted use items display.
OFF	Turns off a use items display to allows adds and drops.

6.1.4 Other Commands

Name	Function
EXIT	Exits from the program.

6.2 Standard Command Formats

Command	Format	Comment
ADD	-ADD cr ITEMS TO BE ADDED --<Name> cr --<Name> cr . . -- cr	Where <Name> is one of the names listed in Section 5.
DROP	-DROP cr ITEMS TO BE DROPPED --<Name>cr --<Name>cr . . -- cr	Where <Name> is one of the names listed in Section 5.
SET UP	-SET UP <Number> cr	Where <Number> is 0, 1, 2, 3, or 4.
LIST	-LIST cr	
CONTROL PARAMETERS	-CONTROL DISPLAY cr	The display follows immediately after the command.
CONTROL PARAMETERS DISPLAY!	-CONTROL DISPLAY! cr	The display follows immediately after the command.

Command	Format	Comment
SET	-<Name> = <Number> cr	Where <Name> is one of the names listed in Section 5.1 and <Number> is an unsigned integer in the range for the name.
	or -<Name> = LOCK or -<Name> = UNLOCK	Where <Name> is the four letter name by which some processor is called.
USE ITEMS	-USE DISPLAY cr INTERVAL IS-- <Number> cr	Where <Number> is the number of seconds between displays. The display occurs below at the specified time interval.
PROCEED	-PROCEED cr	
OFF	-OFF cr	
EXIT	-EXIT cr EXIT. OK?--cr BYE.	

NOTATION:

(a) cr stands for carriage return or line feed. The blank before cr in the examples is for readability. The cr should come immediately after the preceding nonblank character.

(b) - & -- are prompts from the CONTROL program.

(c) All lines not ending in cr are messages from the system.

6.3 Remarks about Command Formats

(a) For all the commands except the SET, SET UP, and CONTROL DISPLAY commands only the first letter is relevant in response to the - prompt; e.g., the following are all equivalent:

```
-ADD cr
-A cr
-ADD ITEMS TO PRINT LIST cr
-AXYZ cr
```

(b) No extraneous blanks should be included in a SET command.

(c) The last digit occurring in a SET UP command is used to identify the canned display.

(d) An initial C identifies a command as a CONTROL DISPLAY command. If a ! is present, all control parameters will be displayed. Otherwise, those control parameters with their print flags on will be displayed.

(e) X is an alternate initial character for the EXIT command.

7. Discussion and Examples of Individual Commands

7.1 ADD Command

The ADD Command is used to turn on the print flag of items so that they will be displayed when a use display (or control display) is invoked. Consider the following example:

-ADD cr ITEMS TO BE ADDED --CPU PROC cr --USE PROC cr --cr	The print flags for CPU PROC and USE PROC are turned on.
-USE DISPLAY cr INTERVAL IS--60 cr	A use items display will be made every 60 seconds including all items with their print flags on. The use groups referred to by CPU PROC and USE PROC will be included since the ADD command above turned on their print flags.

The items that may be added (or dropped) are those listed in Section 5.

RESTRICTION:

ADD (and DROP) commands are illegal during an interruption of a USE DISPLAY. However, if the display is not going to be continued, an OFF command may be used to turn off the display. Then adds and drops may be given.

7.2 DROP Command

The DROP Command is used to turn off the print flag of items so that they will not be displayed when a use display (or control display where all is not requested) is invoked. Consider the following example:

-DROP cr ITEMS TO BE DROPPED --CPU PROC cr --USE PROC cr --cr	The print flags for CPU PROC and USE PROC are turned off.
-USE DISPLAY cr INTERVAL IS--120 cr	A use items display will be made every two minutes. The use groups referred to by CPU PROC and USE PROC will not be included since their print flags are off.

The items that may be dropped (or added) are those listed in Section 5.

RESTRICTIONS:

DROP (and ADD) Commands are illegal during an interruption of a USE DISPLAY. However, if the display is not going to be continued, an OFF Command may be used to turn off the display. Then adds and drops may be given.

7.3 SET UP Command

The SET UP Command may be used to turn off all the print flags. A user would want to do this if he were going to specify his own display from scratch. The command for this is:

-SET UP 0 cr

The SET UP Command can also be used to turn on just those flags required for a particular canned display. All other print flags are turned off.

There are five canned displays at present:

0 Turns off all the print flags

1 Turns on the print flags for:

LOG SIDE
SYS RESP
SYS INTE
SYS THIN
SYS TURN
SYS COMP
LIN SIDE
SYS INPU
SYS OUTP

2 Turns on the same print flags as display 1 and then turns on:

X TYPE
X TURN
X COMP
X INPU
X OUTP

Where X is a processor specified by the user in response to a prompt.

3 Turns on the print flags for:

CPU
CPU PROC
USE PROC
I/O
CON TIME
USERS

4 Turns on the print flag for:

SUMMARY

E.g., in the following example the appropriate print flags are set on (and off) for display 3:

-SET UP 3 cr

RESTRICTION:

Commands which turn on and/or off print flags are illegal during an interruption of a use items display. See the OFF Command.

7.4 LIST Command

The LIST Command lists all items with their print flags on. It can be used to verify the items to be included in a canned display, or to make sure that a series of ADDs and DROPs had the desired effect; e.g.,

```
-SET UP 3  cr

-LIST  cr
CPU
CPU PROC
USE PROC
I/O
CON TIME
USERS

-SET UP 0  cr

-LIST  cr
-ADD  cr
ITEMS TO BE ADDED
--SLI SIDE  cr
--SYS SWAP  cr
--cr

-LIST  cr
SLI SIDE
SYS SWAP
```

7.5 CONTROL PARAMETERS DISPLAY Command

The CONTROL PARAMETERS DISPLAY Command is used to display all control parameters with their print flags on. A list of all control parameters is given in Section 5.1; e.g.,

```
-SET UP 0 cr

-ADD
ITEMS TO BE ADDED
--SL:OC cr
--SL:OU cr
--cr

-CONTROL cr
Max K Core On-Line Users =     8   Max Number On-Line Users  = 32
```

7.6 CONTROL PARAMETERS DISPLAY! Command

The CONTROL PARAMETERS DISPLAY! Command is used to display all control parameters whether or not their print flags are on; e.g.,

```
-CONTROL ! cr
Max K Core On-Line Users  =   8   Max Number On-Line Users  = 32
No. Char at which Block   =  40   No. Char at which Unblock = 10
Max File Spc On-Line User = 100   Max Tapes On-Line User    =  1
        % Batch Bias      =  50           Batch Priority    =  1
        Batch Lock        =   0   Msec without Interruption =  40
Millisecs per Time Slice  = 300
```

7.7 SET Command

The SET Command is used to change the value of a control parameter; e.g.,

```
-SET UP 0  cr

-ADD  cr
ITEMS TO BE DISPLAYED
--SL:QUAN  cr
--cr

-CONTROL DISPLAY  cr
Milliseconds per Time Slice  =  300


-SL:QUAN  =  450  cr

-CONTROL DISPLAY  cr
Milliseconds per Time Slice  =  450
```

The SET Command can also be used to lock or unlock a processor; e.g.,

```
-BASIC = LOCK  cr        Basic is locked into core.
    .
    .
    .
-BASIC = UNLOCK  cr      Later Basic is unlocked.
```

Special care should be taken not to lock a processor into core which is already locked or to unlock a processor which is not locked.

7.8 USE ITEMS DISPLAY Command

The USE ITEMS DISPLAY Command is used to display all items with their print flags on (including control parameters). The display takes place at an interval specified by the user; e.g.,

```
-SET UP 1  cr

-USE ITEMS  cr
INTERVAL IS--180  cr
FIRST TIME PERIOD HAS BEGUN
```

Starting 3 minutes later, display number one will be displayed every 3 minutes.

7.9 PROCEED Command

The PROCEED Command may be used to continue a use items display which is in progress; e.g.,

```
-SET UP 0  cr
```

```
-ADD  cr
ITEMS TO BE DISPLAYED
--USERS  cr
--cr
```

```
-USE ITEMS DISPLAY  cr
INTERVAL IS--60  cr
FIRST TIME PERIOD HAS BEGUN
    .
    .
    .
```

60 seconds later the number of active users, users in core, compute-bound users, inputting users, and batch users is displayed. The program is interrupted.

```
-CONTROL!
    .
    .
-PROCEED  cr
```

All control parameters are displayed.

If less than 60 seconds have elapsed since the last time period began, the displays will continue at the next interval.

7.10 OFF Command

ADD, DROP, and SET UP Commands are not allowed while a use items display is in progress. However, if the display is not going to be continued, the display may be turned off; e.g.,

```
-ADD  cr
NO ADDS OR DROPS DURING USE ITEMS DISPLAY
USE OFF COMMAND TO TURN OFF DISPLAY
```

```
-OFF  cr
```

```
-ADD  cr
ITEMS TO BE ADDED
--CPU PROC  cr
--cr
```

7.11 EXIT Command

The EXIT Command is used to exit from the perform program and return control to TEL; e.g.,

```
-EXIT cr
EXIT OK?--YES cr
BYE.                          A lone cr or any string beginning
                              with Y means yes.

!                             ! is the prompt character for TEL.
```

If the user accidentally keys in an exit command or changes his mind after he types it in, he need not exit; e.g.,

```
-EXIT cr
EXIT OK?--NO cr               Anything but a lone cr or string
                              beginning with Y means no.
```

8. Approach to Implementation

There are four different kinds of code in the system which pertain to performance measurement:

(a) A user program which displays the measurements in a format requested by the user and modifies control parameters for the user.

(b) A Monitor page or pages containing the control parameters and use values and distributions on which the displays are based.

(c) Special code at critical points in the system which modify the Monitor page(s) directly (e.g., upping counts) or which report an event to the special code in the Monitor.

(d) Special code in the Monitor which makes more complicated calculations and changes to the values and distributions in the Monitor page(s).

When a use display is being performed, the program is awakened periodically (at a time interval specified by the user) and produces a display. The program is implemented in such a fashion that it is relatively easy to add new measurements and construct new displays.

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. In this case 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 degress 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 (if such is not the case in BPM, then changes will be made). That is, tape and disc transfers are parity-checked. Critical transfers are checksummed and/or address checked. Detected errors are recovered by reread or rewrite 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 retrial 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
(2) Detection
(3) Recovery
(4) Isolation
(5) Recording
(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 interrups)
(10) Checks for controller and device unusual end conditions

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). Certainly this difficulty should not be construed to limit efforts for time-shared diagnostics or exercisors.

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 mode 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, in general, to recover (although in most cases recovery can be accomplished). 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 re-starting the job from the last swap image on 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 overlooked 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 will be made (in a circular buffer) to enable a very detailed analysis when warranted. Time and effort required to make this record is paid for on the first error found, 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
AIO, TIO, TDV status	special system temps
I/O command used	core dump (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.

Machine Modification may be necessary to achieve reliable system operation. Specific areas of concern are:

(1) A register to report directly the address of a memory parity error

(2) Direct connection of the reporting log to avoid dependence on an IOP.

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.

Part III. SYSTEM CAPACITY AND LOADS

TABLE OF CONTENTS

INTRODUCTION

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 turnaround 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 bookkeeping 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 - Reentrant programming and use of the memory map allow users to share commonly used processors such as editors, debuggers, libraries, 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 loads, 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: RAD usage and CPU usage. RAD 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 RADs 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, including the most commonly used processors but not all (e.g., COBOL is not included in this estimate).

CPU utilization for all noncompute-bound and non-batch operations is estimated. Under the assumptions used, 65% 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 interactive service for thirty noncompute-bound terminal users.

A. Number of RAD Transfers

Table 1 below summarizes in seven broad catagories the number of transfers required of a RAD in a UTS system. In each category the underlying assumptions are noted. It is generally assumed that write checking is not done. If this is desirable, additional RAD loads above that shown will occur.

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)	2.0
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	25.5

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 of 100-300 cards per minute computing.

3. File I/O necessary for problem execution naturally depends on the program, and ranges from zero to whatever rate the file device is capable of. Note that symbiont I/O has already been included in 1) and 2) above so this I/O is intermediate such as the files generated between pass 1 and 2 of Meta-Symbol. We guess that a conservative estimate would be represented by a program which processed one logical record each 50 ms of computing. If the records were 100 bytes long, then 20 records would fit in a blocking buffer. Two I/O actions would be required each second (50 ms x 20 records), one for the blocking buffer and one for the associated index buffer. The 50 ms per intermediate I/O action is chosen to be representative of the range shown by SDS processors: BASIC 6 ms, META 65 ms, SDS FORTRAN 90 ms. (Assuming that one intermediate record is read/written per source line translated.)

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 three physical records transferred per terminal interaction. Three physical 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 user's 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 service 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 five 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 RAD 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 ASSIGNs, 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 three minutes. We choose the conservative figure.

B. Number of RAD Transfer Times and Loads

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 three SDS RADs are

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

Table 2 below repeats Table 1 but also lists the percent of RAD capacity, required for data transfer only -- latency is 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	2.0	1/2	2.4	1.1	.18
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	25.5		107%	52%	8%
	Latency @ 17ms/Xfer (25.5 Xfers)			43%	43%	43%
	GRAND TOTAL			150%	95%	51%

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	63	31	5
Latency Load	34	34	34
Total	97%	65%	39%

C. Compute Load on the CPU

Table 3 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~~ 2400 μsec/transfer.	.5
5)	File I/O management and transfer at 7 ms per logical record	17.0
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	36.2

Some notes on the assumptions used in computing the various loads are again 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 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/logical record of 100 bytes. Scheduled improvements will reduce the figure by about 2 ms/logical record and new access methods may add to the improvement. For the disc transfers of item 3) Table 1, the overhead is estimated at 7 ms for each of the 20 records transferred or 140 ms per second. The disc transfers of item 4) Table 1, included the transfer of one record per user console interaction or 21xN/20 ms per second for N users. Thus, the total is 170 ms for 30 users.

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 four 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 surprising.

Thus, about 65% 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 since the resource is overloaded. Scheduling of compute-bound jobs is controlled by installation management through control parameters discussed in a previous section.

D. 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 problems. 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 depicts results for assumption of exponential arrival and service times and a first-in-first-out service discipline. The lower curves assume exponential arrivals and constant service times for the same service discipline. Our service times are neither constant nor exponential, 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.

*A general expression for calculating this load is given later in this section.

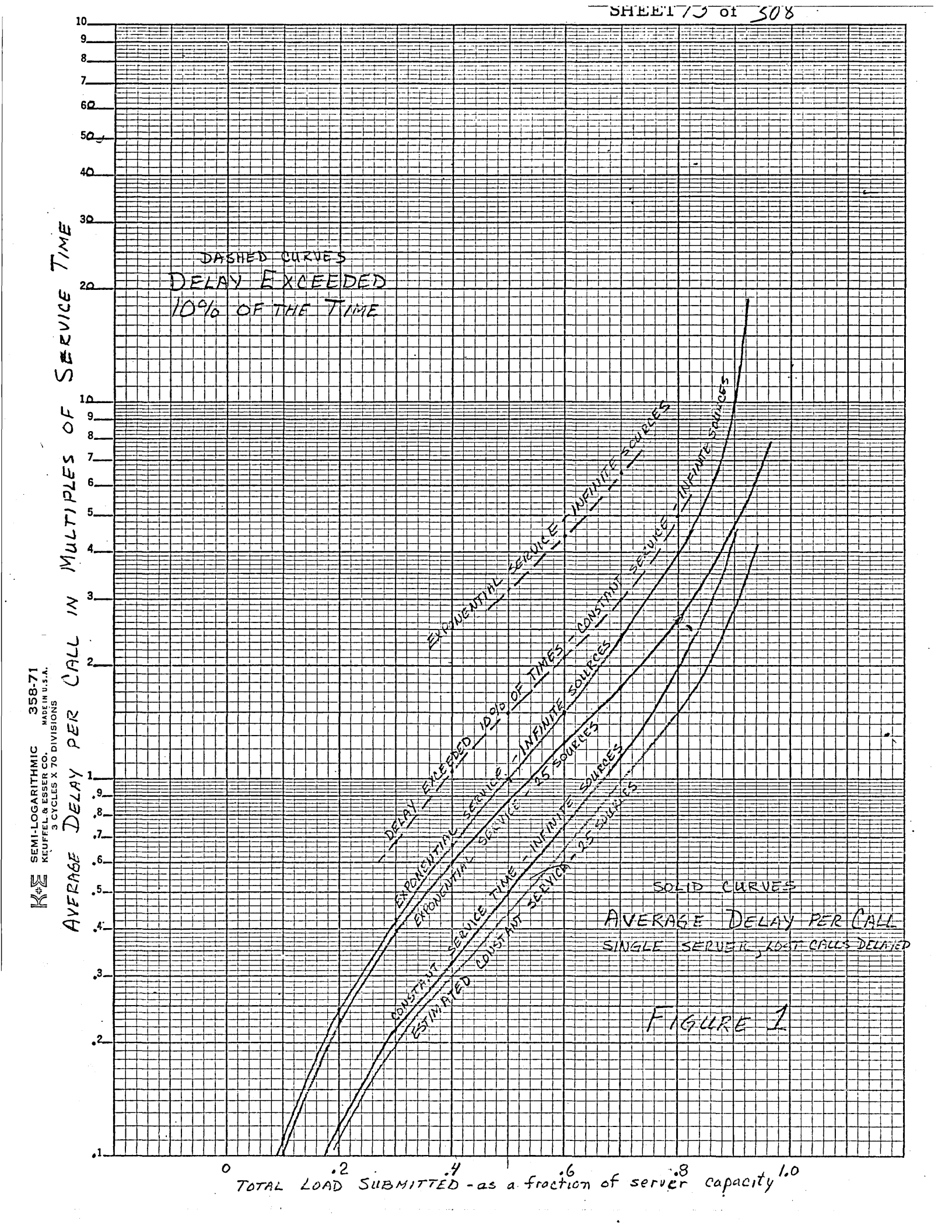


FIGURE 1

SEMI-LOGARITHMIC 358-71
KEUFFEL & ESSER CO. MADE IN U.S.A.
3 CYCLES X 70 DIVISIONS

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, and the expected delay is that required to service all users in the system.

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 expected will often be observed to approach that required to service all users 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 150 ms is sufficient for more than 95% of on-line interactions (those requests requiring less than one quantum). The table below shows the service times for 4000 word programs on the three SDS RADs, including two way data transfer and 17 ms latency for each unit transferred.

Here it is assumed that space on the swapping RAD is allocated in such a way that each RAD transfer for an individual user comes from a set of contiguous sectors. This is achieved by reserving a pool of sectors on the RAD for swapping and assigning pages in such a way that available pages are evenly distributed over the circumference of the RAD.

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 one RAD service time. Thus, with the 7212 RAD, responses to interactive users (those with average compute requests of ≤ 50 ms) 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 delay may be approximated by the number of users multiplied by the service time -- seven 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:

$$L = \frac{d}{10}\left(5.8 + \frac{N}{4} + \frac{2}{Q} + \frac{500}{j}\right) + \frac{r}{10}\left(2 + \frac{N}{10}(3/4 + S) + \frac{2S}{Q} + \frac{250}{j}\right)$$

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) - 1000's of wo
j	=	average batch job time - seconds
Q	=	time slice quanta - seconds.

J. Shemer has made a more detailed study of expected response to an on-line user's interactive requests.* His model includes the effect on response of Monitor CPU overhead and waiting of interactive users for each other as well as the RAD delays examined above. The user obtains his response after waiting and being serviced in three queues:

1. a Monitor CPU overhead queue to receive the interrupt initiating the interaction in conflict with all other users;
2. a RAD swap queue where he waits to enter core memory; and
3. an interactive queue where he waits for first-in-first-out service with all other interactive requests.

The model is used to examine five cases in which several load parameters are varied -- primarily that due to Monitor CPU overhead. His results are given in the table below:

CASE	1	2	3	4	5
users	30	60	30	60	60
Monitor queue load P_1	.25	.39	.50	.78	.78
RAD queue load P_2	.60	.73	.60	.73	.73*
interactive queue load P_3	.07	.15	.07	.15	.15
Average delay (ms.)	162	212	202	450	575

*on each of two 7212 RADs

*Comments on UTS Functional Specification, memo to R. Spinrad from J. Shemer, dated July 25, 1968

The results for Case 1 are substantially the same as the case examined above for the 7212 RAD and essentially the same result is obtained: 162 ms. vs. 150 ms. average delay. RAD load parameter P_2, denotes the same parameter as L, above. The fact that the results are nearly the same is due to canceling of two opposing effects which were neglected in the simpler model. First, delays due to RAD load are over estimated by assuming that RAD service time is equal to that required by a swap when, in reality, the part of the load due to file I/O has a much shorter service time. Secondly, the delay in the interactive queue waiting for other high priority users is not accounted for at all in the first model. For the case of 30 users and the other parameters assumed, the two effects approximately cancel each other. Note that RAD load P_2 in these examples differs from the loads in the first model. See below for a recalculation.

Case 2 increases the number of users to 60 while leaving other parameters the same.

Cases 3 and 4 are modifications of cases 1 & 2 with the Monitor CPU overhead doubled. Since much of the capacity of the CPU is now devoted to overhead (78% of the machine in case 4), substantial delays occur because little is left over for service in the interactive queue. This kind of effect is dangerous and must be avoided. It can come from only three places: 1) resident real-time programs, 2) bad coding in the Monitor (which will be avoided), and, what is more important, 3) inability to keep a sufficient number of computing users in core and thus not being able to use available CPU capacity. The study thus emphasizes the effect of trying to time-share with insufficient core memory.

Since Shemer's study, the estimates of batch file I/O have been refined. The changes are reflected above primarily in a reduction to two from five of the number of I/O actions generated by batch programs. The table below repeats Shemer's cases 1 and 2 with the new data. Two more cases, numbered 6 and 7, are examined using his model. These are modifications of Case 2 but increasing the size of the average program from 4,000 words to 8,000 and then to 12,000 words. (Still assuming contiguous sector mapping on swap RAD.) These cases may also be used to estimate the effect on a 4K program of not being able to achieve a swap mapping in contiguous sectors.

CASE	1'	2'	6	7
users	30	60	60	60
user size	4K	4K	8K	12K
Monitor queue load P_1	.29	.32	.32	.32
RAD queue load P_2	.54	.71	.75	.79
interactive queue load P_3	.07	.15	.15	.15
Average delay (ms)	149	177	193	241

Based on these models and given that the assumptions are reasonable and that sufficient core memory is available we conclude that reasonable service can be obtained in UTS.

Part IV. SCHEDULING AND MEMORY MANAGEMENT

TABLE OF CONTENTS

INTRODUCTION

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 signaling 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 (event signals):

1. Input complete--activation character received

2. Output limit reached--sufficient output for 3-5 seconds

3. Output nearly empty--only 1/2-1 second 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 retrial 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 for file, disc pack, or tape

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 (>4) 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

signaled 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 partial list of the state queues in descending priority order is given below:

NRRT	-	Nonresident real-time interrupt received
INT	-	Interrupt or break received queue
IR	-	Input activation queue
TUB	-	Console Output unblocked queue
IOC	-	File I/O complete queue
COM	-	The interactive compute queue
BAT	-	The batch compute queue
CU	-	Current user in execution
IOIP	-	File I/O in progress queue
TOB	-	Console output blocked queue
TI	-	Waiting for console input queue
W	-	Queue of users to be awakened

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 quantum 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 quanta 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 TI 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 causes a state change to IR. 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 the swap the Scheduler is again called and it now finds a high priority user ready to run. The user's 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 user's state to TI 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 TOB-TUB-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, four to 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. No swaps occur unless a user who is out of core enters a high priority queue.

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 out first, retaining in core the set most likely to require execution. The swap algorithm will operate so that: 1) if there is room in core for three user programs; 2) if two users are computing steadily; and 3) 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 30 concurrent users in prime shift.* That is, 10-20% of the time no on-line user is requesting time. During non-prime time 80% or more of CPU time would be available to batch.

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 80% + on non-prime time. In this scheme, batch can be biased to get a different quantum than on-line users.

*In Part III we estimated that 65% of CPU capacity would be available to batch and on-line compute-bound combined. Here we estimate that on-line users will use 50 or 60% of the total CPU leaving 5-15% for batch.

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 user's 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 user's 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 set up 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 and execute-only. (It may have the user associated data area initialized in first pass only.) When these criteria are met the processor has the following special characteristics:

1. Its name is known to the executive (TEL); it may be called on by name.

2. It will have dedicated residency on swap storage established at ~~SYSGEN~~ SYSMAKE time.

3. Its use will imply a particular virtual map for the user.

4. A single copy will be used by all requesting users.

H. Memory Layout

UTS makes full use of the Sigma 7 mapping hardware, access protection, and write locks in order to allocate arbitrary available physical core pages to users according to the priority of their request without need for program relocation or physical moves. Full protection is provided not only of one user from another but also protection of real-time programs from the Monitor and the Monitor from real-time. All programs including real-time programs and the Monitor itself are divided into procedure and data and the procedure is protected against inadvertent stores by either write-locks or access codes or both.

Central features of the use of write-locks to protect master mode programs are:

1. The Monitor operates with a key of 01. It may store in

   (a) Its own data area (LOCK = 01)

   (b) Any batch, on-line user, or shared processor core (LOCK = 01)

   (c) Resident real-time data area (LOCK = 00)

   It may not store in

   (a) Its own procedure (LOCK = 11)

   (b) Pure procedure of resident real-time (LOCK = 11)

2. Resident real-time operates with a key of 10.

   It may store only in its own data area (LOCK = 00)

   It may not store anywhere else (LOCKS of 01, 11)

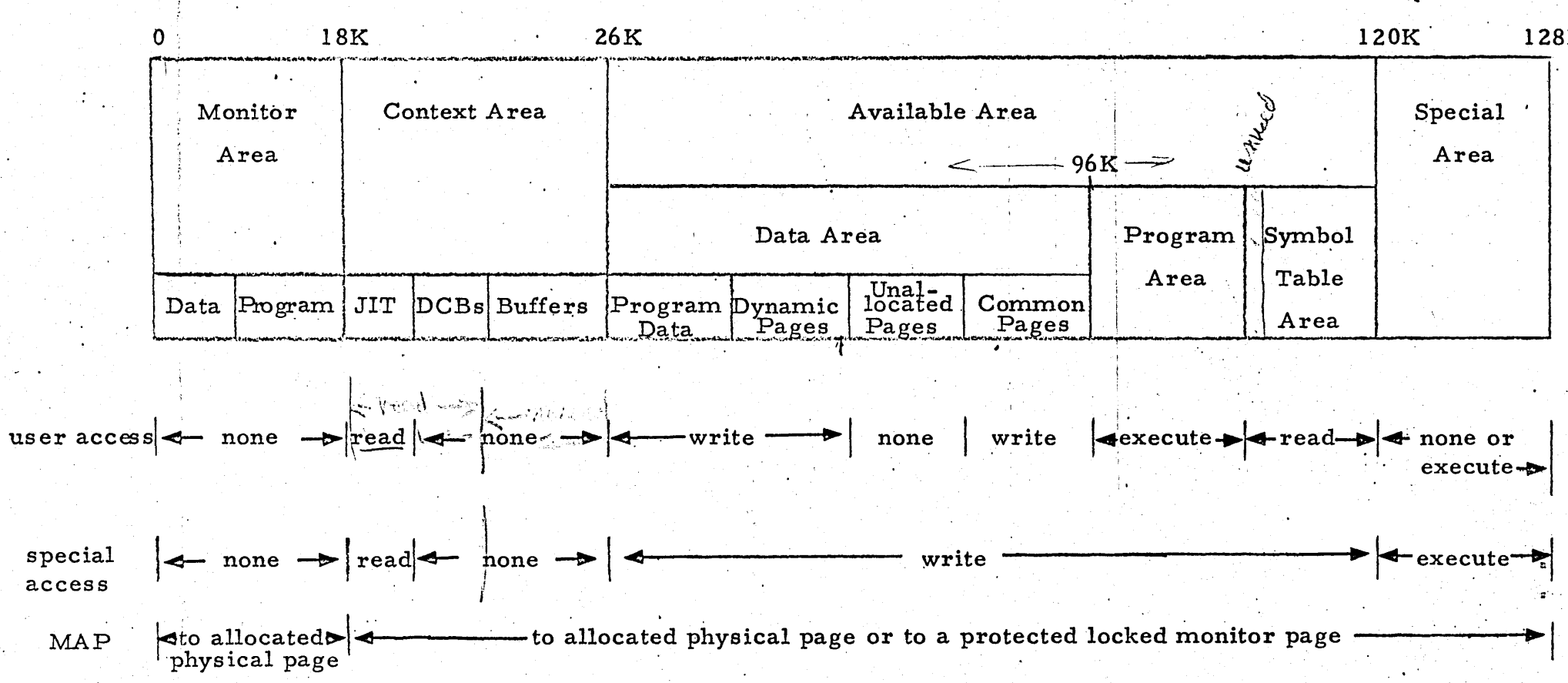
3. Keys of 00 and 11 are never used nor is a lock of 10.

4. Write-locks are initialized only once at system start-up and not changed thereafter except when running under control of EXEC DELTA where they are used to enable data breakpoints.

The access codes on virtual memory pages control references by slave mode programs - user programs and shared processors. Two code images are retained in JIT for each user, the first is loaded when the user is in control and the second when one of the special shared processors gains control. In addition, under TEL and LOGON write access to JIT and other job context areas is given.

The layout of virtual memory which applies to user programs and ordinary shared processors is shown in the figure below. Core addresses shown are those appropriate for a typical system but more (or less) core may be established for the resident Monitor at SYSGEN time depending on installation needs (e.g., requirement for real-time options; desire to retain Monitor overlays in residence for efficiency). More (or less) area may also be desirable for the library area and for the job context area to accommodate more buffers. These bounds may also be adjusted at SYSGEN time. The bound at which the one pass loader LINK places the user's program is adjustable at load time.

Virtual pages not currently allocated to the user are mapped into a resident Monitor page which is write-locked and have their access code set to no access. Thus, slave mode programs are denied access through the access code and attempts to store at these virtual addresses by a master mode program are protected by the write-locks.



CONTENTS

Job information table
DCBs
File blocking buffers
File index buffers
Coop buffers

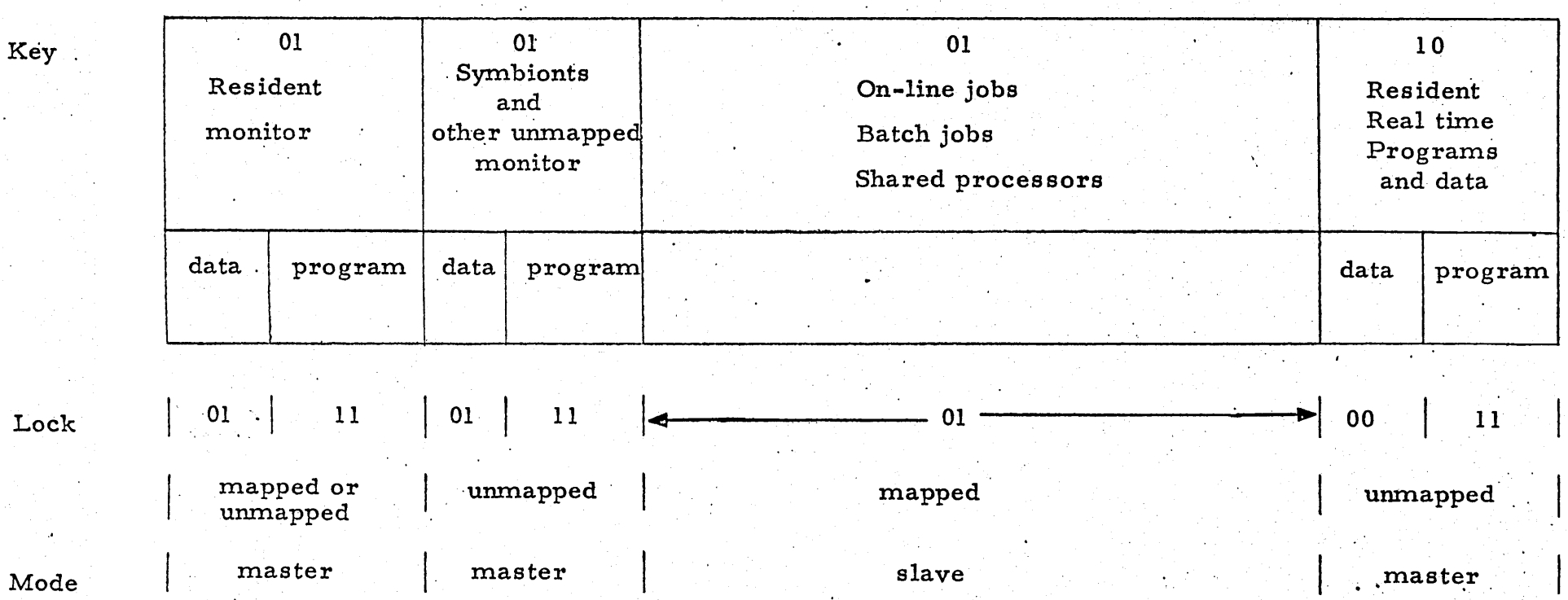
User programs, data, and symbol tables
Ordinary Shared processors including
- Root segment
- Initial data
- one overlay

Special shared Processor and data:
- LINK
- DELTA
- TEL, CCI
- FDP
- Libraries
- LOGON/OFF

---

access codes:
- none - no access of any kind permitted
- read - read access only
- execute - execute or read access
- write - write, execute, and read permitted

Typical User - Program Virtual Memory Layout (not to scale)



unused keys: 00, 11
unused lock: 10

Typical Physical Memory Layout (not to scale)

Part V. SYSTEM REQUIREMENTS and CONFIGURATION

TABLE OF CONTENTS

INTRODUCTION

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 16-18K words.
2. UTS is predicted on and requires a symbiont system.
3. Since real-time requirements are preemptive and installation-dependent, no allowance is given here to these demands on the machine. Users with real-time applications must add core to support the real-time programs and suffer interactive delays and batch throughput loss due to less CPU availability.

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.

BASIC UTS HARDWARE CONFIGURATION

In order to support 32 on-line users (who are generally not compute-bound) and maintain a high rate of compute throughput (about 80% of BPM rate), the following configuration is presented. It is considered the basic UTS configuration to which equipment may be added to satisfy additional requirements of particular installations. UTS design optimization will focus on this system level. The configuration is shown graphically in Figure V-1.

CPU

Model	Description
8401	Sigma 7 CPU
8413	Power Fail-Safe
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, 80K

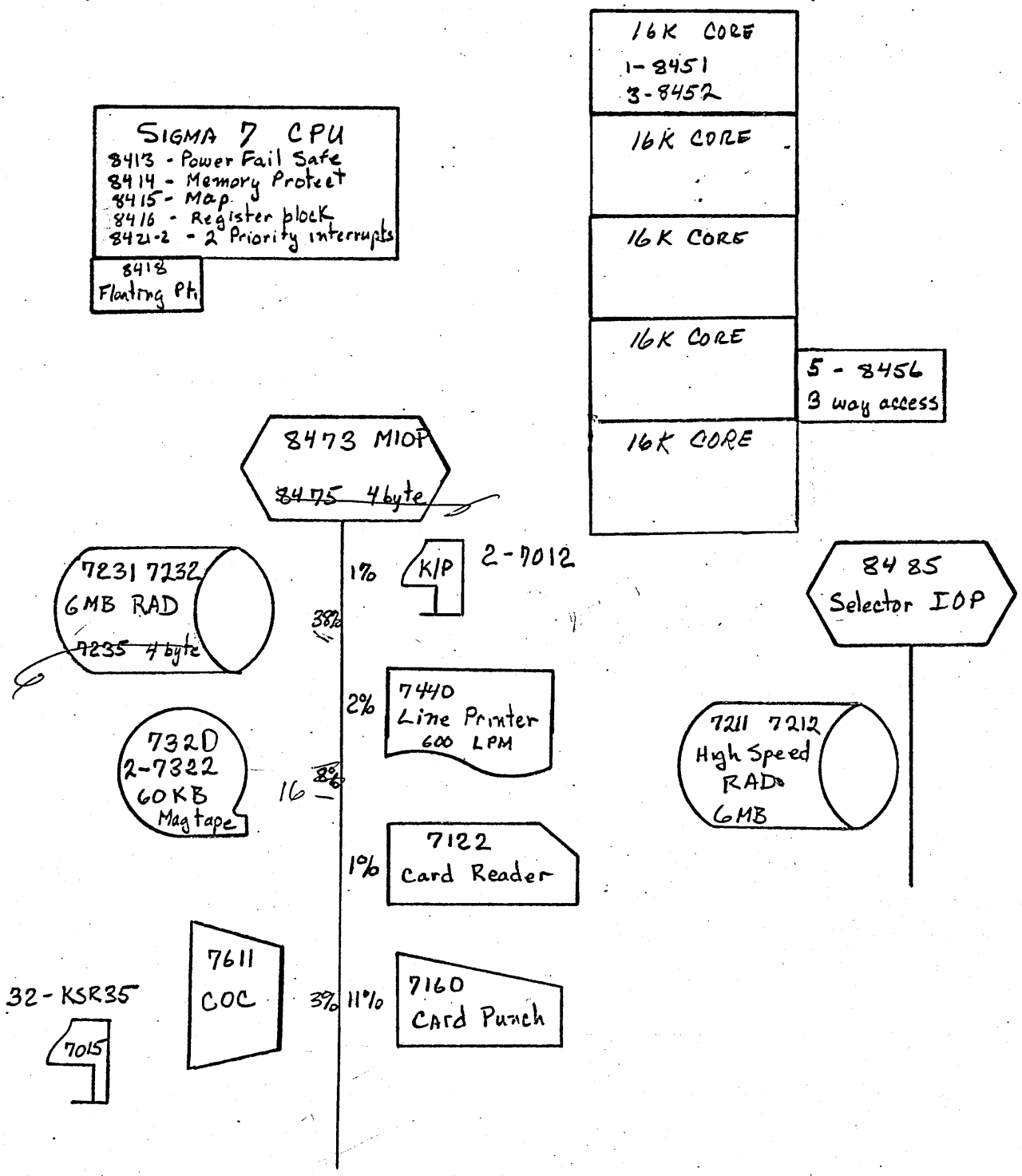
8451	Memory Module, 5 each
8452	Memory Module, 15 each

Basic I/O

8473	MIOP
~~8475~~	~~Four Byte Interface~~ only for disc pack
7012	Keyboard Printer (2)
8485	Selector IOP
7211	Hi-Speed RAD Controller
7212	Hi-Speed RAD
8456	3 Way Access, 5 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

7122	Card Reader
7160	Card Punch
7440	Line Printer
7320	Magnetic Tape Controller
7322	Magnetic Tape Units (2)
7231	RAD Controller ~~(4 byte interface for 7231)~~
7232	RAD Storage (6MB)



BASIC UTS HARDWARE CONFIGURATION

FIGURE V-1

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

64K is a minimum core size, suitable for single language installations for example, but 80K is considered necessary to maintain a high level of performance for this example.

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, might maintain acceptable performance with two 7232 RAD's, for example.

4. Secondary Storage and Peripherals

The minimum 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. CRAM and disc pack units are supported and may be added as the customer requirements dictate. Band width requirements of the IOP's must of course be met. Heavy use of these devices may degrade system performance through CPU use and core buffer space required for I/O transfers. Generally speaking addition of core memory will improve performance in these larger systems.

5. Terminals

Terminals are not restricted to SDS 7015's alone. The SDS 7550, 7555 keyboard/display operates compatibly with UTS. Also supported are teletype models 33, 35, and 37 including paper tape input and output, and IBM model 2741 selectric typewriter terminals. Software flexibility is present so that other terminals may be added with relatively little difficulty.

More than 64 terminals may be added to the system with more 7611 equipment and the addition of two external interrupts for each group of 64 lines. However, except for special situations (say an all BASIC system), more core, RAD, etc. will be required and even then response times may suffer.

Remote batch terminals, SDS 7670, are supported in accordance with specifications in a separate document (Dwg. No. 702514).

6. RAD Storage

Requirements for RAD storage may be divided into two categories: a) swap storage is used for on-line users, for symbiont and co-op buffers, and for absolute core images of the UTS Monitor and the system processors (FORTRAN, LOADER, BASIC, etc.), and b) file storage for all users of the system including those system processors kept in ROM or LM form on file (usually SYSGEN, COBOL, Libraries, etc.). Swap storage will usually require one to three megabytes on the 7212 RAD. File storage is best estimated by the installation in question. For each on-line user, 25-50,000 words of file storage is often estimated. Thus, if 100 users have access to a 32-line system 2.5M-5M words of file storage would be needed requiring two or three additional 7232 RADs.

7. Resident Real-Time

Core and CPU requirements for resident real-time operations must be satisfied over and above the basic configuration of this section. Among the devices supported in the real-time mode is the graphic display, SDS 7580.

CORE MEMORY

Providing enough core memory is particularly crucial in a time-sharing system. Operating with a memory of insufficient capacity for the installation's typical load reduces the systems ability to keep core loaded with ready-to-run programs and increases the frequency and duration of times when I/O is not overlapped with computing. This applies both to on-line systems and multiprogrammed batch operation. With enough core the system is able to keep several programs resident at once and thus obtain a very high probability of completely overlapping compute and I/O tasks. The frequency of swaps is reduced and so is the intendent system overhead thus releasing the CPU to execution of users' tasks. On-line response is doubly affected being doubly reduced of the number of the number of swaps and indirectly affected by the lowering of CPU overhead load.

The following charts indicate what factors should be evaluated in determining core size. They are presented as a guide and require careful interpretation in determining system core requirements.

I. Core Requirements for other than on-line User Memory Requirements

A.	UTM Resident System	18K	Conservative
B.	Resident Foreground	variable	
C.	Remote Batch	1K	When in use
D.	Graphic Scope	6-10K	If used
E.	Other (Cal Comp Plotter, allowance for future devices, etc)	variable	
F.	K/D	1K per terminal	If message mode option is used
G.	Allowance for core I/O buffers, DCB's etc. which are locked in core while I/O is in process (This allowance is for user I/O, symbionts, monitor overlays, and swaps)	10K	Estimated for average load as described within this spec. for about 30 users

II. Core Requirements in order to have a high probability of keeping 4 users in core.

A.	Batch	variable	
B.	User's program	variable	
C.	UTS Processors		

	Shared Copy	Core per User: User Associated Data
1. Fortran IV- pass 1	11K	6-7K
- pass 2	11K	6-7K
2. Meta-Symbol - pass 1	11K	4-6K(no BPM proc
- pass 2	11K	8-10K(with use of BPM proc's)
3. BASIC - Compile	10K	4-6K
Execute	10K	4-6K

	Shared Copy	Core per user: User Associated Data
4. Editor	3K	2K
5. Delta	3K	.5K plus user program plus symbol table
6. FDP	2K	object program plus symbol table plus run-time
7. PCL	2K	2K
8. TEL	4K	2K
9. LINK	4K	2K plus ROM size plus local symbol table plus REF/DEF/ Expression tal Estimated to b 1.75 times RO size.
D. Core Library	variable (3-8K)	

An installation which desires high rate of batch throughput and reasonable service for 32 on-line users doing a variety of work, might make the following core estimate:

1. Resident UTS	18K
2. Allowance for I/O in progress	10K
3. Batch	12K
4. On-line users	
a. A processor, say Fortran IV	18K
b. A debugging run (DELTA)	10K
c. An EDIT process	5K
d. User with core library and FDP	7K
	80K

A 64K system could be expected to often result in a condition where less than 4 users are in core simultaneously, and this in turn can be expected to result in much lower CPU utilization, longer response times, and less batch throughput.

A system which is essentially dedicated for one processor (BASIC, for example), or a system which uses only a few processors, has small programs, and less users -- could run acceptably in 64K:

1. Resident UTS	18K
2. Allowance for I/O in progress	5K
3. Batch	12K
4. On-line users	
a. Edit	5K
b. BASIC shared processor	10K
i. user compiling	4K
ii. user debugging	4K
iii. user executing	4K
	60K

However, systems with heavy I/O requirements and large executing programs will need more core to achieve high CPU utilization, good on-line response, and adequate batch throughput:

1. Resident UTS	18K
2. Allowance for I/O in progress	20K
3. Batch	20K
4. On-line users	
a. A Fortran IV compilation	18K
b. A DELTA debug run	10K
c. An EDIT process	5K
d. A user with core library and FDP	12K
	103K

Generally, since system performance is extremely sensitive to core size, it is strongly recommended that a generous approach to providing core be taken. No system should be configured with less than 64K, and the Programming Division requests that its approval be obtained for systems with less than 80K of core.

Part VI. TERMINAL EXECUTIVE LANGUAGE (TEL)

TABLE OF CONTENTS

See Tuning + Meas for Login record p 26

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

need syntax to set "L" edit in deb.

COMMUNICATION CONVENTIONS

A. Keyboard Control

As previously mentioned, control of each user's keyboard is proprietary: either the user or the system has control. The assumption is made that all terminals in use are attended. Terminal communication conventions are 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 subsystem are first requested by the user, that subsystem will identify itself in plain-talk before turning control over to the user.

3. All subsystems 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. Subsystems for editing use an asterisk (*); subsystems for combining object programs and manipulating their associated symbol tables all use a colon (:); utility subsystems for file management and information transfer use the numerical relation sign (<); all subsystems for working with other programming languages use the sign (>). These identifying marks notify the user that one of a class of

subsystems is awaiting a command from him. Which subsystem it is and which language must be used should be in the head of the user. This is possible, since no subsystem 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 subsystems. 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. One of the terminal services available to aid in this effort is the specification of a prompt character which is issued preceding each read command. See Section XII.

B. Typing Lines

The mechanisms for correcting characters, for erasing messages that may be hopelessly mistyped, for signaling end-of-message, and for line spacing are uniform. These are given below for users with TTY terminals. Details for other terminal types are given in Section XII.

1. The user can erase his last unerased token by depressing the RUBOUT key. UTS responds by typing a slant-line (\) to indicate that it has effectively backspaced and erased. On terminals that can backspace, backspacing will be nonerasive 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. On SDS Keyboard displays the last typed character is erased from the screen.

2. The user can erase an entire message by depressing two keys sumultaneously, CONTROL and X. UTS types a back arrow (←), returns the carrier to the beginning of a fresh line, and returns control to the user without further comment. The user may then retype the correct message.

3. Blank lines are ignored by UTS's executive processor. 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 subsystem 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.

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 characters to the line; 8" platens provide for 72 characters. UTS counts 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) blank lines to page bottom; b) a heading line, containing date, time, user identification, console identification, and page number; c) five more blank lines; and d) the user's heading line, if any. 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 the users to request that tabs be simulated by successive spaces. Tabs are not normally simulated; to turn on tab simulation, depress the ESC key and then depress the T key. To turn off tab simulation, repeat the procedure. The setting and clearing of tab stops is provided via set commands for individual DCBs and via the TABs Command for overall control over all output to the user terminal.

7. Echoing of characters back to the terminal is at the discretion of the user. Normally, UTS will echo; to request no echoing (for a local printing console), the user must depress ESC 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 Part XII.

C. Interrupting UTS

1. Whenever one of UTS's subsystems is in control of the keyboard, the user can interrupt and temporarily suspend operations by simultaneously depressing the CONTROL and E keys. UTS responds by stopping the current operations as soon as it reaches a convenient breakpoint, and then turning the user over to the executive processor, TEL.

2. Whenever UTS or one of its subsystems is in control of the keyboard, the user may interrupt what is being done for him at the moment by depressing the BREAK key which gives control to that part of the system currently in communication with the terminal (e. g., a subsystem). Since some actions can only be stopped at points of convenience and others have so much inertia that they cannot be stopped at all, and since machine or programming errors may have disabled the program's response to BREAK, a succession of BREAK signals in excess of three returns control directly to the UTS executive. 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 subsystems will accompany the functional description of the subsystem; handling of interrupts by users' object programs is covered in the sections which describe the calls that programs can make on UTM services. Baldly speaking, however, interruptions of the system or any of its major subsystems 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 noises 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 backup to the executive processor. Programs being run under control of debuggers or under control of a programming language subsystem 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 is either the new line or 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 nonexistence 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 subsystems that carry on line-by-line dialogue 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 subsystems, and can be listed here:

(a) Garbled, malformed or unintelligible commands:

EH? @ n (where n gives the character position at which the confusion was first encountered)

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

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

(c) References to (deleting, reading, overwriting) a nonexistent file:

NO FILE ...

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

ON FILE ... ?

(e) Errors, abnormalities, storage-limit overruns 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 subsystems. Procedures for doing so are described in the next section.

Three pieces of information are required:

1) the user's personal identification (id) (1-8 char)
2) the user's account identification (account) (1-8 char)
3) a password (password) (1-8 char)

These may be represented by a string of no more than eight contiguous letters and/or decimal digits. Embedded underscores may be used as separators (they count as characters); these print as left-facing arrows (←) on Models 33 and 35 TTYs.

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

---

*For the limitations on the lengths of the character strings for name, account, and password, see PCL, Part IX. Except that file names are limited to 10 characters.

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); and 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 subsystems. When a connection with UTS has first been established, UTS responds by typing:

```
UTS AT YOUR SERVICE
ON-AT (time and date)
LOGON 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 types an exclamation mark (!) at the left margin of the top line of a new page and then awaits the user's first command. If the identification is garbled or otherwise invalid, UTS notifies the user and then repeats the initiation procedure. The messages are

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

The user may change the password in his logon file at anytime by typing PASSWORD xxxx where xxxx is any 1-8 character string. Requirement for a password is reset if an xxxx is not given.

p. 28

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 subsystems. Figure 3 indicates how such activites are carried out from the console; TEL commands are capitalized, and subsystems indicated.

1. FORTRAN and METASYMBOL programs are created, filed away and changed through the EDIT subsystem either by explicitly calling for EDIT or by the EDIT and BUILD directives.

2. Programs are compiled or assembled via the commands FORT4 and META from the files or from the terminal (line-at-a-time) into relocatable modules (ROM).

3. ROMs may be LINKed into load modules (LM).
4. ROMs and LMs may be LINKed and may be modified by SYMCON.
5. LMs can be LINKed into core and execution STARTed.
6. Linking, loading and starting of ROMs 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 or 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.

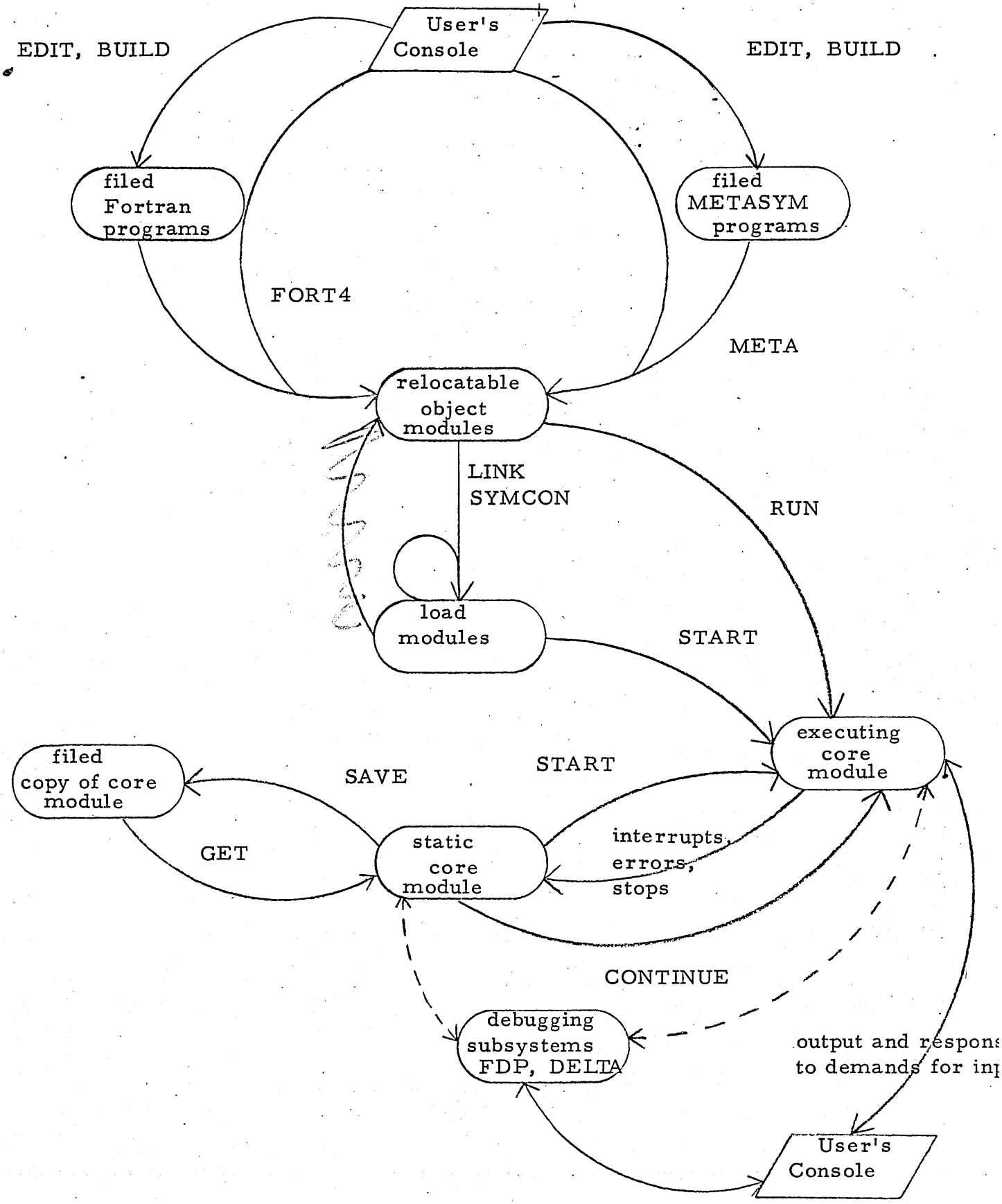


Figure 3. FORTRAN and ASSEMBLY-LANGUAGE PROGRAMMING

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

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. If no input specification is given explicitly, TEL assumes input from the terminal (ME). To signal end of input via the end-of-file signal, the user depresses ESC and then F.

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

FORT4 sp, sp, ..., sp ON rom, list

META sp, sp, ..., sp ON rom, list

Listing, commentary, and ROM output may be specified beforehand.

LIST ON list
(or OVER an existing file)

OUTPUT ON rom

COMMENT ON list

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

FORT4 file1, file2

FORT4 file1, file2 ON romfile3

Listing specification hold over all subsequent operations until changed or until specifications accompany a compile or assemble command. Output files specified via ON and OVER clauses hold from the time given throughout the session or until reset, and output to them is extended from job step to job step (see Section 4 below).

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, or to turn on or off error comments and listing output.

LIST	or	DONT LIST
OUTPUT	or	DONT OUTPUT
COMMENT	or	DONT COMMENT

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. Extension of Output Files

File extension is a convention used when opening certain system output DCBs by which a file (RAD, tape, disc pack, CRAM, etc) connected to a DCB is positioned to a point just following the last record in the file. Thus, when additional output is produced through the DCB it is added to the previous file contents thereby "extending" the file. File extension is necessary to simulate output to physical devices such as line printers, punches, typewriters, etc., when output is actually directed to a file. It takes place on all opens of system-output DCBs and, in particular, it takes effect between job steps (compiles, assemblies, loads, and executions are all steps within the same job) when opening system output DCBs, which were closed at the termination of the preceding job step, and which are normally connected to devices but have been assigned to files. The DCBs which are treated this way are M:LO, LL, DO, PO, BO, SL, SO, CO, AL, EO, and GO.

File extensions do not apply, that is a new file is created, on occurrence of a reassignment (SET) or when the file is opened with an OPEN Command giving an explicit file name. Extension of the GO file is terminated following a LINK or RUN Command.

5. 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, aborts occurring outside of TEL, i.e., within compilers, assemblers, or user programs will result in all previous specifications for listings, commentary, etc. and/or file assignments being restored to that in effect at the beginning of the job step. Source Input (SI) specifications are reset to the default, ME, for each job step. On syntax errors in input messages, the input is erased and an entirely new command must be entered.

6. Entering Programs From the Terminal

Whenever the input designator, ME, is encountered, the carrier is returned to the left margin of a fresh line and a prompt is sent 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 types the end-of-source command for the subsystem in use, normally "END". For purposes of formatting, print columns on the terminal's platen are in one-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, 19, and 37. The general handling and simulation of tab stops is covered completely in Part XII. Briefly, tabs are simulated so that longer fields can be used: spaces are sent to the terminal to bring the carrier position to that indicated by the next tab position set. Tabs given when the carrier is beyond the last set position are simulated by a single space. On input, tab characters accompany the source statements literally, and assemblers and compilers will treat them properly.

7. 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 is not produced by the compiler. Such information increases the size of object programs and slows them down. To turn on the production of this information for a specific compilation, the user follows the verb FORT4 by a parenthesized letter "D" or the parenthesized word "DEBUG".

B. Linking ROMs and LMs to Form LMs

ROMs are representations (of programs and data) that are specifically designed for efficient combination with other ROMs; LMs 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 assigned to them. Many of these symbolic identifiers are used and referred to solely within the module that may be referred to in other modules (DEFs) or are used to refer to objects defined within other modules (REFs). Functionally, these modules are black boxes with labeled 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 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 subsystem 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 subprograms to be linked freely in the face of conflicting naming conventions.

---

*Drawing Number 702477.

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 ROMs and LMs may be linked. Their identification mfl (for "module for linking") may be a fid (file identification) or the dollar sign ($) which refers to the ROM(s) produced on the M:GO file.

The subsystem, LINK, is specifically designed for linking and is described in Section X. However, most commonplace linkages can be carried out directly in TEL.

LINK mfl, mfl, ..., mfl ON lm	(ON A NEW FILE)
LINK mfl, mfl, ..., mfl OVER lm	(OVER an existing file)
LINK mfl, mfl, ..., mfl	(in a special file which may be STARTed)

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

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; and 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 subtables 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 give instructions to merge

several 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. Either all or none of the input modules may be merged on a file. This convention was adopted in favor of, say, choosing a distinct command for the process, to maintain uniformity with the conventions of the LINK subsystem. Multiple uses of internal identifiers are resolved by assigning to them the object that they identify in the first 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, its subtables are merged 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 semicolon. For example:

LINK mfl, mfl, . . . , mfl; 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, "DONE" is typed and then control is returned to the user.

6. Loading LMs Into Core

It is possible to load any stored LM into core by presenting TEL with the LM name as a command verb. A password may be optionally present. If the LM exists under another account, that account may be specified. If no account is specified, the :SYS account is assumed. TEL will scan the remaining portion of the input line in an attempt to create assignments as is done for the META and FORT4 commands. If the line scan is not desired and there is more line content, the user may bracket the "no-scan" portion with parentheses. TEL will ignore all data imbedded within the parentheses but will reject as a syntax error excessive close parens. For example:

TESTOR	loads the LM from the :SYS account
TESTOR.	loads the LM using the Logon account
TESTOR.1234	loads the LM using the account '1234'
TESTOR..SECRET	loads the LM under logon account and the password 'SECRET'
TESTOR FILEA ON FILEB, FILEC	loads the LM and assigns FILEA as the source in (SI DCB). Any output through the GO DCB is placed on FILEB, and FILEC will contain any output through the LO DCB.
TESTOR (ABC(DEF(GHI)JK))	loads the LM and passes the line image directly to the program.

C. Initiating Execution

To start execution of the last LM formed by LINK, the user types

START

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

START lm

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

RUN

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

RUN mfl, mfl, ...

All options of the LINK command may be exercised in the RUN command, in exactly the same manner. Normal options are the same:

RUN (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.

D. Initiating Debugging Operations

Execution of programs can be started under control of either of the two debugging subsystems, DELTA or FDP:

RUN ... START	UNDER	DELTA	(for assembly-language debugging)
		FDP	(for FORTRAN debugging)
		XXX	(under special shared processor XX

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 DELTA debugging subsystem 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:

DELTA

E. 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 subsystem. 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 LP or ME
DELETE fid

Once started, deletions 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.

F. Editing

Line-at-a-time composition and editing of files of sequentially numbered lines is provided by the EDIT subsystem, which can called in two ways:

EDIT fid (an existing file)
BUILD fid (a new file)

In the first case, EDIT has already been apprised of which file is to be edited, and has opened that file for updating. In the second case, EDIT assumes that the user wishes to type in a new file, a line-at-a-time, beginning with line number 1.000 and continuing in steps of 1.

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 carriage return key as the first character of the line, and then type

END

G. Submitting Batch Jobs

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

BATCH fid

UTS responds 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 jid

H. Calling Subsystems

All subsystems are called by typing the subsystems identifications, e.g.,

PCL

All subsystems 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 subsystems are described in separate parts of these specifications. The identifying marks are:

EDIT	(*)
PCL	(<)
FDP	(@)
SYMCOM	(:)
LINK	(:)
BASIC	(>)
DELTA	(bell)
TEL	(!)
PM	(-)
FORTRANs	(>)
META	(>)
user entry	(?)

I. Continuing and Quitting Major Operations

Whenever a major operation, a subsystem 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

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 (SAVE, GET)

During interruptions of exception, 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-out activities -- exactly where he was when he SAVEd.

B. Assigning I/O Units and DCB Parameters (SET)

Most of the parameters carried in DCBs which control I/O in UTS may be set from the UTS terminal by use of the "SET" command. The SETable information includes that which may be given in BPM ASSIGN commands and many of the parameters which are established using OPEN or DEVICE CAL instructions in a program. The complete list of SETable parameters are given below.

UTS retains, for each user, all the information supplied by SET Commands in permanent tables (called assign-merge tables) associated with each user (on RAD). Each time a new program or processor is loaded for the user this stored information is merged into the DCBs associated with the program. SET Commands may not be given during the operation of a program.

Information given by a SET Command is in effect from the time the command is given until revoked by the user, independent of whether one or many job steps are included in the session. A simple example of the assignment of listing output is

Set M:LO 7T #123; TABS = 7, 12, 22, 37

which makes a device assignment to the M:LO DCB indicating I/O to seven-track tape on reel number 123 with processing to simulate tab stops at positions 7, 12, 22, and 37.

As in BPM, assignments are one of two types:

1) device like printer, punch, mag tape, or
2) files on RAD, disc pack, CRAM, or labeled tape.

If a file assignment is given for a DCB already assigned to a device, then the new information replaces the old in the assign-merge tables. The same procedure applies to device assignments for DCBs currently assigned to files. Changes of device parameters are added to DCBs assigned to devices. Change of device parameters given for DCBs assigned to files yield an error message.

DCBs must be named using either M: or F: followed by up to a maximum of five characters.

The number of DCBs which may be assigned (and thus require an assign-merge table entry) is limited to 12, including a different entry for each chained SI specification. The user may delete assign-merge entries by the command SET ~~DCB~~ 0.

Syntax of the SET command is

SET { M:x / F:i } [{ device [#ser. no.] / MT [#ser. no.] /fid / DC/fid }] [; { SPACE=n / VFC / NOVFC }] ...

Spaces may be used arbitrarily between numbers, words, and identifiers but may not appear within words or numerals.

Some examples of its use are

SET M:LO DC/N.A.P	The M:LO DCB is given a file assignment to file N under account A with password P.
SET M:SI MT#403/fid	The M:SI DCB is assigned to files on labeled tape reel number 403.
SET M:LO; TAB=27,38,47,75; VFC; SPACE=2	The TAB, VFC, and space parameters are added to the M:LO DCB. It must have had device assignment previously else an error would have resulted.

Device Options

Device options, like their program counterparts the M:DEVICE CALs, may be given between job steps but not during an interruption of an executing program. They take effect on subsequent I/O carried on through the DCB. The information is stored in the assign-merge entry if they are

of device rather than file type. Thus, the effect carries on over job steps until reset by the user. If the DCB or A-M entry is currently assigned to a file, all device options are illegal and result in an error message.

The device options and the meaning of each is given in the table below. Not provided are commands corresponding to the M:DEVICE CALs: PAGE, FORM, SIZE, and HEADER.

DEVICE OPTIONS

NAME	RANGE OF VALUES
TAB	A list, separated by commas, of up to 16 decimal numbers giving the column position of simulated tab stops. If all 16 stops are not specified, the stops given are assigned to the first stops and the remainder are reset (to zero).
LINES	A single decimal value giving the number of printable lines per page. Maximum is 255.
SPACE	A single decimal value giving the number of lines to space after printing. Values of 0 or 1 result in single spacing. Maximum value is 255.
DRC NODRC	A switch which turns on or off special formatting of records.
VFC NOVFC	A switch which controls formatting by the first character of record.
COUNT	Turns on page counting and specifies column number at which to print the page number.
BCD BIN	Controls the binary-BCD mode for device reads and writes.
FBCD NOFBCD	Controls the automatic conversion between external hollerith code and internal EBCDIC code (so-called "FORTRAN BCD conversion").
PACK UNPACK	Controls the packed or unpacked mode of writing seven track tape.
DATA	A decimal value which controls the beginning column for printing or punching. Maximum is 144.
SEQ	Specifies that sequence numbers are to be punched in columns 77-80. Four characters of nonblank sequenc id may be given for columns 73-76. Fewer than four characters are left-justified and blank-filled.

DCB Assignment

Device assignment is effected whenever a set command contains an expression containing an op. label or device code. An assign-merge table entry is built or an existing one is modified. DCB assignment is specified by giving one of the two-letter codes below.

1.	Op. Label BI, C, CI, EI, SI, uc BO, CO, EO, SO, PO, DO, LO	The DCB may be assigned to one of the system operational labels. The actual device connected will be that specified by the op. lab
	NO	No assignment.
2.	Device	The following codes may be used to obtain symbiont connection to the named device.
	CP	Card Punch
	PP	Paper Tape Punch
	LP	Line Printer
3.	Tapes	These codes may be used to specify nine-track, seven-track, or arbitrary tape drives.
	9T	Nine-track tape
	7T	Seven-track tape
	MT	Any mag. tape
4.	Files	These codes may be used to obtain specific file connections.
	DC	Any data file. This is the default assumption if no code is given.
	DP	Disc pack
	CM	Cram file

File Option

When an assign-merge entry is given for a file (either disc or labeled tape), certain options may be given in addition to the name account password of the file id and the tape reel number (#xxx). These options are listed below. Items from BPM ASSIGN commands which are not handled in UTS set commands are:

Read/Write account numbers (default applies)
Only one tape reel number
Record length (RECL)
"tries" specification (TRIES)
Key max (KEYM)
Volume number (VOL)

They are not included primarily to reduce storage requirements in the assign-merge tables.

UTS SET Command File Options

File Option	Meaning
1. organization	
CONSEC	Consecutive record organization in the file.
KEYED	Keyed record organization in the file.
2. access	
SEQUEN	Records will be accessed sequentially.
DIRECT	Records will be accessed by key.
3. function	
IN	File is read only.
OUT	File is write only.
INOUT	File is to be updated.
OUTIN	File is scratch.
4. disposition	
REL	The file is to be released on closing.
SAVE	The file is to be saved on closing.

Example:

```
SET M:SI DC/Name; KEYED; SEQUEN; IN
```

C. Determining Status of Current User Sessions

The user may have the current accounting records applying to his session displayed for examination by typing the command

STATUS

The information displayed is as in the preceding section on user accounting, namely:

1) CPU time
2) Console time
3) Number of interactions
4) Number of Monitor CALs
5) Number of disc packs or tapes mounted
6) Core size
7) Total charge units
8) Disc reads and writes
9) Tape reads and writes
10) I/O retries

See page 28

D. Listing System Load Parameters

The DISPLAY Command is used to request printing of specific information about current system operation. The format is

DISPLAY option

where the option may be any of the following:

DISC	available disc space (in pages)
TAPES	number of available tape drives
PACKS	number of available pack drives
USERS	number of users currently active
PERFORMANCE	current values of interactive and compute response times.

Other options for the DISPLAY Command will be added as the need arises.

E. Setting Simulated Tab Stops

The positions of simulated tab stops for the user's terminal may be set via the command

TABS a,b,c, ...

where a,b,c, ... are the character positions-on-line where simulated tab stops are to be placed. Up to sixteen may be set and they must ~~occur~~ be given in ascending numerical sequence. They are acted on (enough spaces are sent to the terminal to position the carriage to the next higher position) when a tab character is to be sent to the terminal or received from it and the simulation control switch has been set by the user (ESC T). The setting applies until superseded by another TABS Command or by a program issued M:DEVICE TABS CAL.

F. Changing Terminal Type

The command TERMINAL may be used at any time to inform the system about the type of ASCII code terminal which is in use. The system uses this type code to adjust character translate tables, responses to line delete and character delete operations. The command form and options are

TERMINAL	[KD	for SDS Keyboard display
	33	for Model 33 teletypes
	35	for Model 35 teletypes
	37]	for Model 37 teletypes

2741 7
0

G. Reporting Terminal Platen Width

The user may change the effective width of his terminal platen with the PLATEN Command:

PLATEN n ↵

When more than n characters are written to the terminal without a new-line or carriage return character, a new line character sequence is inserted to break up the output into segments no longer than the specified n.

INDEX OF COMMANDS - TEL

allowed at break

no	BUILD	Calls EDIT and accepts a new file from the terminal
no	META	Assembles specified source program
	BATCH	Enters specified file in batch job stream
	CANCEL	Cancels the designated batch job
	COMMENT	Directs error commentary to specified device
	CONTINUE	Continues processing from point of interruption
	COPY	Copy a file to specified device
	DELETE	Deletes the specified file
	DISPLAY	Lists current values of various system parameters
	EDIT	Calls the EDIT subsystem
	FORT4	Compiles an SDS FORTRAN IV source program
	GET	Restores previously saved core image
	JOB	Requests status of remotely entered jobs
	LINK	Forms load module as specified
	LIST	Directs listing output to desired device
	OFF	Disconnects user from system
	OUTPUT	Directs object output to specified device
	PASSWORD	Assigns a new password to the user's login control reco
	PLATEN	Sets value of terminal platen width
	QUIT	Terminates current operation
	RUN	Loads specified load module and starts execution
	SAVE	Saves current core image on designated file
	SET	Assigns file or device to a DCB; sets DCB parameters
	START	Begins execution of program just loaded
	STATUS	Displays the current values accumulated in the various charge categories
	TABS	Sets override tab stops at the user's terminal
	TERMINAL	Sets the terminal type for proper ~~I/O~~ character translations
	PRINT	~~Stat~~ Start printing of accumulated print file
	MESSAGE	send message to operator

UNIVERSAL TIME-SHARING SYSTEM (UTS)

FUNCTIONAL SPECIFICATION

VOL. 2 - PARTS VII - XIII

By

E. Bryan
B. Doeppel
J. Smith

31 March 1969

Part VII. TEXT EDITING SUBSYSTEM (EDIT)

TABLE OF CONTENTS

INTRODUCTION

The UTS subsystem, EDIT, is a line-at-a-time context 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 keyed file structure of sequence numbered variable length records. This structure permits EDIT to directly access each line or record of data. EDIT functions are controlled through single line commands supplied by the user. The command language provides for the insertion, deletion, reordering, and replacement of lines or groups of lines of text. Selective printing and renumbering commands, and a series of commands to perform context editing operations of matching, moving, and substitution line by line within a specified range of text lines. File maintenance commands are also provided to allow the user to build, copy, and delete whole files of text lines.

A. Calling EDIT

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

1. EDIT an existing file
2. BUILD a new file

In both cases, the EDIT subsystem is brought into play. The first 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 by typing "EDIT HERE", and its identifying mark, the asterisk (*).

The second 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 "BUILD" are included as part of the EDIT command language and are described further below. If the EDIT command is given at executive level without a file identifier, the editor will prompt for further command input with the '*' character.

B. Operation

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.

The editing process is based on a sequence number associated with each line. These numbers may be automatically generated by the editor during insertions, or may be supplied by the user. Unsequenced files of text lines may be sequenced using the copy command.

EDIT files are stored on disc as keyed records with the sequence number used as a key. Thus, the file is always in key order and editing operations modify the records in place as the EDIT commands are executed. Users may find it desirable to make their own back-up copies of EDIT files to protect against their own or machine errors.

DESCRIPTION

The UTS editor is an extension of the editor available in BTM. Thus, the details of the language and its use are contained in the BTM Manual (90 15 77A) as extended and modified in the following two sections.

The first section describes extensions common to both BTM and UTS versions, while the second section describes unique characteristics of the UTS version.

A. Common Extensions to the BTM Editor

1. a. The modified BTM editor will write variable length records. The byte count written will only be large enough to encompass the last nonblank character in the record, followed by the Cr used to terminate the record. This format should double the effective record density in a given number of granules of file storage.

The editor creates source output files with keyed organization. The keys are one word binary representations of the sequence numbers seen at the terminal. The sequence number DDDD.DDD is taken as a seven digit integer and converted to binary, giving a key with a maximum length of three bytes.

For example, the following record created in a BUILD operation would have key value 8000_{10} and record length 20 bytes (including the Cr).

```
8.000  B2  LI,5  0  Cr
```

Should the Cr be preceded by a number of blanks they will not be carried in the output. The record terminator can be either carriage return or line feed (new line). It is carried in the record as X'15'.

b. The maximum record size in the editor has been increased to 140 characters including the Cr. (In BTM, however, a single input line cannot exceed the COC input buffer size which is set at SYSGEN. The default size is 100 characters. Therefore, the maximum record size under BTM is the minimum of 140 and the input buffer size.)

c. When a file created by EDIT is read by a processor, the records transmitted will appear in the format currently obtained when reading the TY device under BPM. That is, the Cr character will be in the record, and the byte count transmitted will be given in the ARS field of the DCB. The processor must avoid producing a syntax error due to the presence of the Cr, and it must not let short records pick up spurious characters from previous longer records.

2. Type commands

a. The TY and TS commands have been extended to allow the user to display contents of records between specific column bounds. The syntax is now

$$\underline{*}\ \text{TY}\ \ N_1\ \left[-N_2\right]\ \left[,c\ \left[,d\right]\right]$$

The editor displays records in the range N_1 to N_2. Only the portions between columns c and d are typed. If N_2 is omitted, only record N_1 is displayed.

Defaults: c = 1
d = 140

Errors: -BAD COL. NO. PAIR
The columns specified are not in the range 1 through 140, with $c \leq d$.

The intrarecord versions of TY and TS do not allow column specification.

b. A new record command, TC, has been added. This command has the same syntax and display format as TY, except that when a portion of a record is displayed with the TC record, all blank fields are compressed to length one. This facilitates the display of assembly language code. For example,

* TC 0-100 1, 36

The TC command does not have an intrarecord counterpart.

Under UTS, where blank fields may be carried internally as tab characters, X'05', the tab will be altered to one blank.

3. Merge command

A command has been added which allows transfer of records between files. The syntax is

* MERGE fid_1 [,N_1[-N_2]] INTO fid_2, N_3[-N_4] [,i]

fid_1 must exist, in keyed format, or the command will be aborted. If no range specification is attached to fid_1, all of its records are subject to the move. If a range specification exists, the editor checks that at least one record is contained in it. For example,

* MERGE fid_1 INTO ... merge all of fid_1

* MERGE fid_1,10 INTO ... merge record 10.000 of fid_1

* MERGE fid_1,10-12.5 INTO...merge range 10.000 through 12.500

After these validity checks on fid_1, the editor checks for the existence of fid_2. The two possible cases are:

a. fid_2 does not exist. In this case, the editor creates a file identified by fid_2. It then moves the appropriate record set from fid_1 into fid_2, resequencing from N_3 and incrementing by i (in its absence, by 1). This is roughly equivalent to a COPY operation, except for the selection of records from fid_1.

b. When fid_2 exists, the editor first deletes from it all records in the range N_3-N_4. Then the appropriate records from fid_1 are inserted into fid_2, starting at sequence N_3 and incrementing by i (in its absnece, by 1).

If an EDIT operation was in progress, it is halted with the message

. . EDIT STOPPED

The start of the MERGE operation is noted by the message

. . MERGE STARTED

When the MERGE has been successfully completed, the editor prints the last sequence number assigned in fid_2.

-- DONE AT N_5

For example,

* MERGE ALPHA. ACCT1 , 100-120 INTO BETA, 400-440

.. EDIT STOPPED

.. MERGE STARTED

-- DONE AT 420.

In the event that, when fid_2 exists, the number of records to be moved at the specified increment causes the editor to equal or exceed the next highest existing sequence number above the destination range N -N , the MERGE is stopped with the message

-- CUTOFF AT N_5 (N_6)

where:
N_5 = last sequence number assigned in fid_2.
N_6 = sequence number of last record moved from fid_1.

The operation is terminated normally, and the user can investigate how to move the remaining records.

Defaults: i = 1

Messages:

-- EOF HIT The range N_3-N_4 passed beyond the end of file in fid_2.

- PL:NO SUCH FILE fid_1 does not exist.

- NOTHING TO MOVE the specified range in fid_1 contains no records.

- MERGE SOURCE NOT KEYED fid_1 must be a keyed file.

- MERGE DESTINATION NOT KEYED fid_2 must be a keyed file.

4. CR command

This command has the form

$$\underline{*}\ \text{CR}\ \begin{Bmatrix}\text{ON}\\ \text{OFF}\end{Bmatrix}$$

and allows the user to suppress inclusion of the X'15' terminator in his output file. Normally, it will always be carried along as this is the standard format for teletype and typewriter records. However, if the user wishes to reproduce the file on cards or tape, for use by other than BPM/BTM/UTS software, he may not want the terminator to be present.

After this command has been given, specifying OFF, the terminal X'15' will not be included on any records written by the editor, and the user can copy the file producing one in the desired format.

$\underline{*}$ CR ON restores inclusion of the terminator and is the default setting.

B. Characteristics of UTS Editor

The unique features of the UTS editor, apart from those included in the extended BTM version, fall into the following categories:

1. Calling sequence
2. File identification (fid) format
3. BREAK key
4. The TA command for tab setting.

1. Under UTS, the EDIT processor may be called at executive level in one of two ways. Following the executive prompt, the user may type:

! EDIT [fid] , or

! BUILD fid [,n [,i]]

These commands are executive level equivalents of the corresponding EDIT commands. When they are given, the editor is entered, and it responds with

EDIT HERE

and the EDIT, or BUILD, command is executed. The editor will then prompt the user for the appropriate input.

2. File Identification

The file identifier, or fid, is constructed by placing the character '.' between the file name, account, and password. The various cases are:

filename	default log-on account
filename . account	specific account
filename . account . password	specific account with password
filename .. password	default account with password

filename is 1-31
account is 1-8
password is 1-8

} characters from among the set
A-Z a-z 0-9 $ * % : # @ -

For example,

! BUILD TESTFIL..PASS, 10, 5 Cr

3. BREAK Key

When the BREAK key is depressed it causes an immediate interrupt in EDIT activity. Any partially completed input is discarded by the COC handler; any waiting output (already in the COC handler) is drained to the terminal.

The editor stops any command in progress and reverts to accepting command input from the user.

If a command was in the process of being executed, and the BREAK caused an interrupt in an I/O operation (READ, WRITE, OPEN, DELETE record), the I/O is completed.

The user will be given the option of continuing the execution of the command to the normal conclusion, or terminating it immediately.

In the case of record or intrarecord commands, the current EDIT file will remain open. All other commands will terminate by closing all files.

If the command in progress was of the display variety, for example TY, it will be obvious where the interruption took place. The display will stop within the next several lines after the interrupt is given.

However, if the command produces no display while operating on a range of records, it will not be so clear-cut. For this reason, the following commands will produce a message similar to the --CUTOFF message of MERGE, denoting the sequence number(s) of the record(s) being processed at the time of the interrupt.

COPY, MERGE,

DE, FD, FT, MD, MK,

SE

4. The TA Command

The UTS version of the editor contains an additional command for setting and resetting terminal tab stops. The command has the following syntax:

$$\underline{*}\ \text{TA}\ \begin{Bmatrix} F \\ M \\ S \end{Bmatrix}\ C_r$$

The tabs are set as follows:

F implies FORTRAN: tab at col. 7

M implies META-SYMBOL; tabs at 10, 19, 37

S implies META-SYMBOL, short form; tabs at 8, 16, 30

These are logical tab settings corresponding to record col. numbers. They will be offset to provide for the line number produced at the left margin of the user terminal. The command may be given while performing an EDIT operation. It may not be used as an intrarecord command.

Error: -NOT F/M/S	The parameter supplied is not from the legal set.

5. The UTS COC handler simulates input TAB stops at the terminal by spacing the carriage (if the user has requested tab simulation). However, the actual TAB character, X'05', is passed to the processor initiating the read, and will thus be included in files created by EDIT. Records will contain embedded tab characters rather than the blank strings they represent.

   Tabs for the terminal can be set with the executive TABS command, or with the TA command in the editor.

   The editor contains a number of very useful commands which allow specification of column bounds during their operation. These column bounds could correspond, for example, to opcode - argument - comment fields in assembly language code, which the user identified with tab characters when building the file. When he is editing the file, the embedded tab characters must be made to represent the appropriate number of blanks. In the current file management system, this tab information is not kept with the file.

Therefore, the editor will expand each record it reads according to the current terminal tab stops contained in the M:UC DCB. (This action will not be performed however, when the records are not subject to the EDIT command; a COPY or MERGE operation requires no such expansion.) In the same way, each record written to the RAD will be checked against the current tab stops in the M:UC DCB and re-compressed, if possible.

Should the M:UC DCB not contain tab settings when the editor is expanding a record and finds a tab character, the user is notified (only once) with the message:

- TAB CHARACTER FOUND. NO TAB STOPS SET.

Part 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 normally 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 the program before passing it control.

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 (optionally using the system library routines) 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 and space must be reserved in the users program area for DELTA's context. 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 input ROM 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 input constants typed by the user.

6. Special symbols used to examine machine flags and to control operating bounds for 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 instruction, data, and transfer breakpoints.
10. Commands for searching memory.
11. Miscellaneous commands.

In outlining the commands, the following conventions are used in depicting the format of the orders 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	EXEC Keystroke
cr	carriage return	carriage return
lf	line feed	EOM
↑	shift and N	&
\	shift and L	¢
tab	control and I	tab
bk	BREAK	Sigma 7 interrupt

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 =. (lf and ↑ are EOM and & in the executive version).

Whenever user DELTA gives control of the terminal to the user for input, it sends its "prompt" character, the bell, to the console. The executive version of DELTA does not prompt nor does the user version when connected to a keyboard display.

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 opened; that is, it may now be changed.
e(f/	Displays the contents of cell e in format f.
e1,e2/ 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. (e ¢ in the executive version)
/	Slash alone following a display displays, but does not open, the cell addressed by the display. (Displays the cell addressed by the last quantity typed (;Q)).
tab	Tab alone following a display displays and opens the cell addressed by the display.

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 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 & in the executive version)
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 word	
I	signed decimal integer	
C	EBCDIC characters	
R	symbolic instructions with symbolic addresses	
A	symbolic instructions with hexadecimal addresses	
S	short floating point number	user version only
L	long floating point number	user version only
(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 introduced by a period (.BAD), 2) EBCDIC characters when surrounded by single quotes ('BAD'), and 3) decimal when the constant consists of just 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 address	
;I	instruction counter	As set by the last entry to DELTA or changed by the user
;C	condition code	
;F	floating controls	
;3	Master/slave, map and arithmetic mask bits	
;4	Write Key	
;M	search mask	
;1	lower search bound	
;2	upper search bound	
;Q	last quantity typed	

7. Symbol Table Control

s;S	Select internal table s.
;S	load global symbol table.
;U	Display undefined symbols.
e(f<s>	The symbol s is assigned value e and format f.
s(f!	The symbol s is assigned the value of the currently open cell ($) and format code f.
s;K	Symbol s is flagged in the symbol table. It will not be used in output expressions. It can still be used in input expressions.
:K	Removes <u>all</u> symbols except instruction mnemonics and special symbols.
;KI	Removes current internal symbol table.
;KG	Removes global symbol table and any symbols defined from the console.

8. Execution Control

e;G	Begins execution at e.
;G	Begins execution at current location counter value (;I).
e;X	Executes the instruction e.
n;P	Proceed with no output the next n times the current instruction breakpoint is encountered.
)	Execute the current instruction and display the next one.

9. Breakpoints

e,n;B	Set the n^{th} instruction breakpoint at location e.
e;B	Set the next available breakpoint at location e.
e,n,loc;B e,,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.
;B	Display all active instruction breakpoints.

e,n,val,m;Dr
(e,n,val,m;DTr)
Causes data break n to occur whenever the contents of cell e (masked by m) are in relation r to val. (T for trace mode.)

e,,val,m;Dr
Same as above but pick the next available data breakpoint.

The relations are:

LS	e < val
EQ	e = val
GR	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.
;D	Display all active data breakpoints.
;P	Proceed from the break.
n;P	Proceed and do not break until the breakpoint has been passed n times (n applies to instruction breakpoints only).
;T	Set 'trace mode' for the current break.
bk	Break at the current execution point (analogous to the machine's idle switch).

Output produced when a breakpoint is reached is n;B loc (or n;D>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

;Y	start transfer breakpoint mode at contents of ;I. Do not display branches specified in SAT.
loc;Y	start transfer breakpoint mode at loc.
;YT loc;YT	Same as above but proceed automatically after printing (trace mode).
o;Y	turn off the transfer breakpoint mode.
loc,1;Y	Same as above but trace only branches mentioned in SAT.
loc,,1;Y	Same as above but also trace BDR and BIR instructions.
1,m,n,o;YS	Set entries in SAT.
1m,n,o;YR	Release entries in SAT.

what the loc is SAT ?

;YR	Release all entries in SAT.
;YD	Display SAT

Output on occurrence of a transfer breakpoint is loc1 -> loc2 where loc1 is the address of the branching branch and loc2 is its target address.

10. Memory Searching and Modification

Memory between the bounds specified in ;1 and ;2 (initially set to the lower and upper limits of assigned user data memory) are searched under the mask in ;M (initially all ones). If field 2 is specified in the search command, the value in that field is stored through mask ;M into each location which meets the specified condition.

e;W	Search for and display words which match e under the mask ;M.
e1, e2;W	Store e2 through mask ;M in locations which match e1 through the mask.
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, e2;L	Set ;1 to e1 and ;2 to e2.
e;M	Set the search mask to e.

11. Miscellaneous Commands

;R	Display memory addresses as symbol plus relative hexadecimal offset.
e;R	Same as above but set the hexadecimal offset to e.
;RK	Display addresses as csect type symbol plus any hex offset. If the value displayed is equal to that of any symbol, then display the symbol. If there is no csect type symbol, display as a hex constant.

;A	Display locations as hexadecimal numbers.
e1,e2,v;Z	Store v in memory from e1 through e2 (memory pages must be assigned).

DESCRIPTION

A. Syntax, Symbols, and Such

The language of DELTA follows the DDT formula of simplified expressions and single (or few) letter commands, which holds the number of keystrokes required of the user to a minimum. Because every keystroke counts, and users can find most errors easily by eye, only a few syntax error conditions are explicitly commented. The most common commands have been assigned to lower case keys in order to simplify typing.

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 line delimiters (end-of-message characters) and cause UTS to deliver the command line to DELTA for interpretation. The characters which are command line 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 (EOM in exec DELTA)
↑	The store and open previous command (& in exec DELTA)
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.

More than one command can be input on a command line. The command delimiter within a command line is the space character. For example:

```
TAG1;B    TAG5,,.1500,.FF00;DEQ    ;B   ;D   Cr
```

2. Fixing Typing Errors

Before giving one of the command delimiters, the user may (in user DELTA only) 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 question mark cancels the command line.

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 code. Symbols longer than seven characters are truncated to include only the first seven. Thus, symbols which were originally longer than seven characters are indistinguishable from each other; only the last received definition is retained.

The symbols used by DELTA follow similar rules to 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 than eight. For symbols supplied with eight or more characters, DELTA retains only the first seven characters.

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) Hexadecimal

The default mode of the display command will be to examine the symbol table for a symbol at or with the next smaller location value than that requested and use the conversion type given. This means that a memory dump via the slash command 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	
$ 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 in memory searches.
;3	Bits 8-11 of word one of user PSD at interrupt.
;4	First byte of word two of user PSD at interrupt.

;3 and ;4 are not used to form the PSD for entry to user in user DELTA (they are used in EXEC DELTA).

;Q	The last quantity typed by DELTA, or the value stored by the user with the commands cr, lf, and tab.
%D	The highest user data location loaded by LINK.
%P	The highest user program location loaded by LINK.

deleted from manual

Except for $, ., %D, %P, 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;I 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 convenient in special situations:

```
ALPHA/AI,5  7   ;Q+2  cr     (Store contents of open cell plus two
                              in the open cell.)
./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 period (.); EBCDIC obtained by enclosing the characters in single quotes (') -- note that a single quote may not be introduced using this conversion type --; and decimal, the conversion used on strings of numerals. EBCDIC character strings follow the same rules as symbols used by DELTA (Section 3, Symbols), except that the maximum length is four characters.

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 and zero filled.

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.

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 (the 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.

```
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
(C;/        X/          A
```

If 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 or using the ¢ key in executive DELTA). 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 them in the remembered mode.

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

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 and 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, minus (+-), and space may be evaluated by u

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 default format or 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 + 5 (I = 10

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/ .68000643 / .78C cr

./ EXU LS

Note in the above that a temporary display format was extablished 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 (e.g., display contents or not) 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.

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 by 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 types 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 period (.). X 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 (within the specified hex offset) location symbol if no symbol occurs at the location in question. If no symbol is within range, default is to symbolic mode.

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 some types of terminals.

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 symbol corresponds to the value, then the next lower symbol plus a relative hexadecimal offset is printed. Values less than 50 decimal are always printed in hexadecimal. If the index field (bits 12-14) is nonzero, it is printed as a decimal integer (1-7) following the address and 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.

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).

The final two conversion types S and L are not available in the executive version of DELTA.

F. Execution Control: The ;G, ;P, ;X, and) 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 (or space if it is in a multi-command line). 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 (interrupt button in exec DELTA),
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

PRIVILEGED INSTR AT  .77B

;I=   .77B
```

Proceeding from a stop condition is directed by typing the ;P or the ;G command without a preceding address expression. The effect is to continue execution from the location specified by the current value of ;I; i.e., where execution left off or a location specified by adr;I input by the user. The user 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 re-execution of the violating instruction and another trap.

```
MEMORY PROTECT FAULT AT   .74B
;G
MEMORY PROTECT FAULT 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.

If the return is to other than 1, 2, or 3, the results are unpredictable.

The) command is the step mode execution command. It executes the instruction in the currently open register and opens and displays the next program step (i.e., if the instruction executed by ')' is a branch, the effective address is the location opened and displayed). By using '/' to open and display a location and repeatedly hitting ')', a user can step through portions of his program.

G. Breakpoints: The ;B, ;D, and ;Y 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
e;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. If n is not specified, DELTA will assign the next available break number. If none are available, an error condition results with the message "NONE". The user may then release one of the 8 breakpoints he has set and try again. The breakpoint 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+3,1,R5;B  A;G

1;B>A+.3 R5/  .54
```

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 this break occurs again.

```
PH+8,2,R2;B  PH;G
2;B>PH+8   R2/  .4   ;P or ;G
2;B>PH+8   R2/  .5   ;P or ;G
2;B>PH+8   R2/  .6  5;P
2,B>PH+8   R2/ .12
```

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

All breakpoints can be removed by the command 0;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 sets the trace mode at the current breakpoint.

The currently operative instruction breakpoints may be listed for inspection with the command ;B. The list has the form:

n {T} loc display

for each established breakpoint where n is the breakpoint number, a T is printed if the trace mode is set for that breakpoint, loc is the break location, and display is the address to be displayed when the break occurs.

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:

LS	$(e)_c <$val
EQ	$(e)_c =$val
GR	$(e)_c >$val
GE	$(e)_c \geq$val
NE	$(e)_c \neq$val
LE	$(e)_c \geq$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 specific variants of data break insertion commands are:

e, n;D	Set data break n
e;D	Set next available data break Terminal control returns to the user immediately after each change in the contents of e and printing of the data break message.
e;DT	Set next available dat break in trace mode Each time the contents of e change, the data break message is printed and execution continues.
e, , val;Dr	Set next available data break with value val and relation r. Terminal control returns to the user when the contents of e stand in relation r to the value val.

e,,val;DTr	Print data break message and continue execution when the condition holds.
e,n,val,m;Dr	Set data break n with value val masked by m and relation r.

The currently operative data breakpoints may be listed for inspection with the command ;D. The list has the form:

$$n\left\{\begin{matrix}T\\ \ \end{matrix}\right\} \text{ loc cond value mask}$$

for each established breakpoint where n is the breakpoint number, a T is printed if the trace mode is set, loc is the break location, cond is the break condition relation, value is the break value, and mask is the mask under which the data is tested.

Any data breakpoint may be removed by the command n;D. All data breakpoints may be removed with the command o;D. The output resulting from a data break has the form n;D>loc e/cont 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 "cont" is the new value as just modified. Some sample data breakpoint settings are given below:

```
A,1,3;DGR
A+5,2,.FF,.FF;DEQ
AB,3;D
SDS,4,CSC;DGE   ;P
4;D>PH   SDS/   CSC+2
```

The data breakpoint will not detect changes caused by direct hardware I/O transfers into the user's area.

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 giving the

location of the break. If the user hits the BREAK key while his program is in execution, the message is

BRK AT loc

The ;P command will continue execution after such a break. If the break occurs while DELTA is executing, the message is

BRK IN DELTA

4. Transfer Breakpoints and Interpretive Execution

Transfer breakpoints allow the user to halt (or trace) execution when a branch instruction is encountered which branches when executed. The command has the form

loc,do/don't,Bdr/Bir;Y

This command differs from the two other breakpoint commands in that it initiates execution as soon as the command is decoded and processed.

Set transfer break mode (TBM)

;Y — Starts interpretive execution at the current value of the location counter (;I). If no options are specified, execution will halt as soon as a branch instruction is encountered which branches. Output is

loc1→loc2 where loc1 is the address of the branch and loc2 is the location branched to. To continue execution in Transfer Break mode, issue a ;P. The proceed count is not meaningful for transfer breaks (see Section F).

o;Y — Turns off Transfer Break mode.

loc;Y — Start TBM execution at loc.

;YT or loc;YT — Instead of halting at each branching branch, continue execution after outputting the Transfer Break message.

loc, DO/DON'T, BDR/BIR;Y

Special action options.

DO/DON'T = 0 Do trace all branches except those specified in the special action table (SAT). If the address of this branching branch appears in the special action table (see below), do not output message and continue execution. Honor all other branches. (Nominal setting)

= 1 Do trace only those branches specified in SAT. If the address of this branching branch appears in the special action table, do output the TB message and continue or not as specified by the trace mode flag. If the address does not appear, continue execution.

BDR/BIR = 0 Do not trace BDR or BIR branches. (Nominal setting)

= 1 Trace BDR and BIR branches.

Special Action Table (SAT) set up

loc1, loc2, loc3, loc4;YS

Enter loc 1-4 in SAT if space is available (maximum of eight). These locations are meaningful only if their contents are branch type instructions. The action to be taken depends on the setting of the DO/DON'T option.

;YR Release all SAT entries.

loc1, loc2, loc3, loc4;YR

Release specified locations in SAT.

;YD Display SAT.

The) command interpretively executes the instruction at the current location counter (;I) and displays the instruction at the new value of ;I.

```
ALPHA    /   LW,3     .A000    )   executes instruction at $
ALPHA+.1/    STW,3    .A201    )   displays the next instruction
ALPHA+.2/    B        TAG1     )   in program flow
TAG1     /   LI,0     0
```

H. Memory Searching and Modification: 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. If fields 1 and 2, and no others, are specified, the value in field 2 will be stored through the mask; M in all locations which meet the specified condition (match or mismatch). Display occurs after the substitution. 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 the lowest and of ;2 the highest current user data area address. Before the test for a match is made, the word from memory is masked with a work 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 (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                     A, BB;L  is equivalent
BB;2
2;M                     Mask bit 30 of the word.  Search for all words
2;W                     between A and BB which have a 1 in bit 30.
```

or

```
A+.2/     2
A+.3/     3
A+.6/     6
A+.7/     7
A+.A/     .A
BB/       .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.

```
ERR,.OUT;W
```

Substitute OUT for ERR

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

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 (although some may be combined by LINK), 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 a load file, then the internal symbol tables are merged with the last instance of conflicting symbols being retained.

During debugging, the user always has the global symbols of the load and he may select one of ther 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. The ;S command alone will reload the global symbol table (this implies that it was released via ;K or ;KG command - see below).

```
.B73/    LW,4   IOP+.A7    lf
IOP+.CB/    BAL,6IO+.17F    IOPF;S ↑
IOPT2+.6/    LW,4   K34
```

The user may wish to release back to the system the pages used for the symbol tables. The command ;K releases the pages containing the global and internal symbol table; ;KG only pages containing the global symbol table; ;KI only pages containing the internal symbol table.

s;K	Disables use of the symbol s in constructing output. They are still evaluated when typed in. Symbols return to use if the user reloads the symbol table.
;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. Global symbols are restored by ;S.

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.

NOTE: if LOC(X/ .3250A000 / .43

format X holds till Cr

thus, LOC(X/ .3250A000 SYM!
is the same as: SYM(X!

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. If the final angle bracket is followed by a K, the symbol is flagged as a csect type symbol in the symbol table. If K was specified before the first angle bracket, an error is reported.

Single Line Macros

Since the symbol table definition capability gives a 32-bit value to constant symbols, it may be as a macro definition facility for single word values. For example, using EXEC DELTA the interrupt inhibits may be reset via the sequence

WD,0 .25(K<RI>	defines RI to have value of the Write direct instruction which resets all inhibits.
RI;X	executes the reset instruction
RI+.10;X	executes a set inhibit instruction

or, alternately

```
RI+.10(K<SI>
SI;X
```

J. Miscellaneous Commands: The ;A, ;R, and ;Z 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 (or by spaces in a single line command).

;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 the 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
```

;R may be preceded by a value -- n;R -- which sets the maximum offset to be used in address output. If no symbol lies within 'offset' of the value, the address is printed as absolute hex. Thus, 10;R will cause DELTA to display symbol plus relative offset only when a symbol lies within ten locations less than the display address.

;RK This command sets relative address output mode with the restriction that only location symbols flagged as control section type symbols (see Section I, ;S command for setting csect type) will be output unless there is an exact match between the symbol value and output value. If there are no csect type symbols, the output will be a hexadecimal value. Thus, output will be 'csect symbol plus any hex offset', or 'symbol' or 'hex constant'.

;Z The command for zeroing memory takes the form a,b;Z, where a is 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
?  7
```

A third field may be added to the ;Z command. If so, it is a value to be stored in the range a,b.

If the memory is not assigned, a memory protect fault results.

Zeroing, or otherwise modifying, the user's area may be used to erase the user's program and/or 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.

K. Printer Output: The ;O and ;J Commands

Two commands are provided which use the line printer for output (via symbionts in UTS). They are ;O to produce hexadecimal dumps on the line printer, and ;J to direct DELTA output to the line printer (particularly useful in the cases of formatted displays and output from tracing breakpoints).

The printer and tape I/O routines are completely self-contained in the executive version with no dependence on system I/O routines. The executive version of DELTA operates with all interrupts disabled.

a,b;O header — 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. If any input follows the O, it is printed as a header. Each dump starts printing at the top of a fresh page.

;J — Toggles the output location switch alternating between the terminal and the line printer on each instance of the command. Output from the equal command, from non-tracing breaks, from trap, abort and error returns, and from syntax and other error conditions in Delta are always directed to the typewriter.

```
A,1;B
X,2,3;DTE     ;J     B;G

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

1;B>A
```

L. Commands for the Executive Version: The ;V ;H ;E Commands; Interrupts.

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. If u is not specified, 32K words (an assembly parameter of DELTA) are dumped. Before dumping, DELTA asks for confirmation of a correctly mounted tape with the message:

MOUNT A80, TYPE CONFIRMING PERIOD

;E	This command causes DELTA to display the current contents of all cells (exclusive of the general registers) into which the user has stored during his session. DELTA keeps a table of stored addresses (maximum of 64) and displays them plus their contents when ;E is given.
;H	The ;H command has two options if Delta was given trap control at boot time.
TM;H	Sends traps directly to the Monitor trap routine.
TD;H	Control remains in Delta. If NL is input, control will go to the Monitor trap routine; any other input leaves control in DELTA.

Control of Console Interrupts and Traps

At System boot time, Delta types out 'TRAPS'. If the response is the letter 'U', EXEC DELTA will take control of trap location X'40'. Any other response leaves the trap location unaffected. When a trap occurs and Delta puts out the appropriate message, control is directed to the Monitor trap routine by typing new line (NL), or to EXEC DELTA by typing anything else. Delta control of the console interrupt location is obtained in the same way when DELTA types 'CONSOLE'.

Interrupts

Control may be given to the executive version of DELTA at any time. The system programmer may get control at the operator's console by pressing the Sigma 7 panel interrupt button. Typing a new line character following DELTA's response message sends control to the BPM Monitor interrupt routine. Typing anything else leaves control in DELTA and the programmer may examine or change memory registers or set breakpoints in the system. Return to the point of interrupt is via a ;P or ;G command.

Limitations: CAL's, XPSD's, or LPSD's which depend on following calling sequences will not operate properly if they have an instruction break on them. BAL's are interpreted and OK.

M. Errors and Error Messages

Errors which result in machine traps are reported explicitly to the user and console control is returned to him to await further commands. Each message is accompanied by the location, symbolically if possible, of the offending instruction. The messages are

NONEXIT INSTR AT _____

NONEXIST MEMORY REF AT _____

PRIVILEGED INSTR AT _____

MEMORY PROTECT FAULT AT _____

STACK LIMIT FAULT AT _____

UNIMP INSTR AT _____

FIXED OVFLW AT _____

FLOAT FAULT AT _____

DECIMAL FAULT AT _____

Syntax errors are reported by the message ? n where n is the number of the character in the command line that DELTA was processing when the error occurred. 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/ ? 8	too many commas
'ABCDE'= ? 6	constant value larger than one word
ABC;K ? 5	symbol not in symbol table
FF;M 100, XY;L .6B;W ? 13	symbol value not found remainder of command string ignored
A, 5;E ? 5	command unknown
LW*5 ALPHA= ? 3	asterisk in a funny place
.3ACR/ ? 5	illegal character in hex number
(B;/ ? 2	illegal relation
LOC, , 3;DNQ ? 10	illegal relation
;T ? 2	no break in effect to set trace mode on

N. Program Exits

When called, DELTA takes control of program exits via the M:SXC CAL. DELTA reports execution of exit CALs with a message of the form

EXIT n AT loc

where n is the exit code as defined in the table below and loc is the address of the CAL or instruction causing exit.

Code	Type of Exit	Example
0	Normal	M:EXIT
1	Trap Error	decimal or floating trap
2	I/O Error	no error address
4	Limits	max time; max pages output
10		operator aborted job
20	Termination	operator errored job
40	Abnormal	M:XXX
80	Job Errored	M:ERR

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
(	introduce format code
)	execute current location and display next
;1	set lower limit
;2	set upper limit
;3	bits 8-11 of word one of entry PSD
;4	first byte of word two of entry PSD
;/	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; display break table
;C	set condition code
;D	set data breakpoint; DT set trace mode; display break table
;E	display patch table (executive version only)

;F	set floating controls
;G	being execution
;H	Trap control (execution version)
;I	set instruction counter
;J	divert output to line printer
;K	remove (kill) symbol table entry
;L	set upper and lower limits for search
;M	set the search mask
;N	search for word mis-match
;O	hexadecimal dump to line printer
;P	set proceed and proceed from breakpoint
;Q	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
;Y	set up for and begin execution in transfer break or transfer trace mode
;Z	zero memory

Part IX. PERIPHERAL CONVERSION LANGUAGE (PCL)

TABLE OF CONTENTS

Need sig to set "2" bit in dcb.

SUMMARY

This part of the UTS Functional Specification describes a peripheral utility subsystem designed for both on-line and batch operation. The Subsystem, 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 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 printer, 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. Actual input-output operations are carried out using standard BPM CALs; the restrictions and advantages of this I/O system therefore apply throughout.

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 by typing its name to the Terminal Executive (TEL). PCL responds by typing "PCL HERE" and then typing its 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 stautus 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.

* Wherever "TO" is specified "ON" may be substituted.

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 throughout 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. (not available on-line)

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 -- standard BPM format paper tape

PR paper tape - files are separated by two successive !EOD codes

RB remote batch terminal

2. File Identification (fid). Files are identified by name, account, and password in that order separated by periods (.). 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 identification 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

X hexadecimal-dump

5. Modes. These codes dictate control modes for the files or devices

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

7T, 9T — magnetic tape

PK, UPK — seven track binary tape packing or unpacking

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

FA, NFA — Controls whether or not the attributes of the source file are to be carried over to the destination file. If the file name remains the same from source to destination the attributes are copied if neither FA or NFA is specified e. g.

COPY DC/A TO LT#4/A

causes a copy of file A to Labeled tape with exactly the same attributes it had on disc. When the name on source and destination is different than the normal case is not to copy over attributes. But information specified in ASSIGN or SET commands takes effect.

DEOD — In this mode multiple files from the source are copied into a single output file. Thus while COPY FT copies upto and including the first file mark, COPY FT (DEOD) copies files to a double end-of-file without copying the single end-of-file to the output.

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

CS (id, n, k) — Options include: 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 EDIT style. file starting at n in sequential steps of k. Line numbers must be between 1 and 99, 999.

NLN — no line numbers.

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).

9. Extensions Using ASSIGNs Not all of the facilities available in the BPM I/O system are made available through PCL. The user interested in more complicated data transfers may specify them by using ASSIGN cards in the batch mode or SET commands if working on-line. PCL reads through M:EI and writes through M:EO so special information (e. g. lists of read account numbers and write account numbers for the output file) may be pre-specified by ASSIGNing either the input or output DCB.

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	-		-	-	-	-	-	-	-
	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

10. 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 (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.

```
< 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 TO FT(BIN,7T,H)Cr
<
```

This example points out the use of a scratch tape. Line images are read from disc file A, converted from EBCDID 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 /SOURCE TO ME Cr
10010 START LW, R1 ALPHA
10020       AI, R1  5
10030       CW, R1 BETA
  :    :     :
<
```

This command requested a Meta-Symbol source file on disc be dumped at the user console. Note that the line numbers occupy the first seven 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 /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.

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.

The command

```
≤ COPYALL FT#3 TO FT #4
≤
```

makes an exact copy of tape number 3 onto tape number 4 through and including the first double end-of-file encountered on tape 3. The copy is unrestricted as to format except that record size must fit in the allowable installation-set allocation of core to a single job.

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.

DELETEALL acct

Deletes all files in the indicated account. A confirmation similar to that for DELETE is required.

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) under the indicated account (defaults to the current user's account), 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

COPY	copies device(s) and/or file(s) TO* / OVER device or file
COPYALL	copies file catalogs between disc and labeled tape
COPYCAT	same as COPYALL
DELETE	deletes a file
DELETEALL	deletes all files in an account
LIST	lists file names from account directory or labeled tape
REMOVE	removes reels from tape units
REW	rewinds tape reels
SPE	spaces to end of last file on labeled tape
SPF	spaces device forward or backward n files
TABS	set tab stops for output
WEOF	write end-of-file on device

*The word "ON" may be substituted for "TO".

Part 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), starting execution of the loaded LM (START), or combining the above steps (RUN). 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 (LM) which may later be combined with other load modules or ROM's, the load module must be of one protection type. A LM program of mixed protections type may not be combined, however, ROM's of arbitrary protection type mixture and LM's of single type may be combined. The resulting memory layout is in two areas with protection for data and pure procedure.

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)

Only two protection types result in the final program for execution: data and pure procedure. Static data and no access information, if specified, are loaded with pure procedure.

LOAD MODULE STRUCTURE

A load module (LM) formed by LINK may be thought 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 and is generated by the compilers and assemblers with protection type 01. All sections with non-data protection type are also included here.

2. Data or program context - This section of code has read, write, and access protection and is generated by the compilers and assemblers with protection type 00.

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 the first size declared. All subsequent F4:COM declarations must be less than or equal to that size.

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 is defined in a later section 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 are 48 words long consisting of:

22 word standard initial segment containing standard default op labels if the DCB is one of the Monitor DCB's (see Section XI).

Five of the variable length items including a control word for each and space for:

3 word file name
2 word account number
2 word password
3 words for 3 input serial numbers
3 words for 3 output serial numbers
8 word key buffer

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 guarantees that each DCB is contained within a page. This allows the operating system to access DCB's in either mapped or unmapped mode.

5. Public (core) libraries - Each installation has the ability to define a set of reentrant subroutines which together constitute a public core-library. Via SYSGEN, the installation may specify several different core libraries containing collections of routines useful in different environments. REF/DEF stacks for these libraries stored under special names

in the system account are used to LINK programs. Only one core library may be associated with an executing program. The reentrant portion of each 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, each 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 reentrancy can be obtained.

6. System library - The system library, much the same as the public core-libraries, 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 a 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 are 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 object (DEF's) within a module which may be referred to (REF'ed) in other modules. This table is available to DELTA, for use in debugging.

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 file and identified by its 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 original 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 such multiply defined symbols is the first 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, halfword, word, doubleword, and constant. The following are 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 are detailed below.

Location Symbol - code = 01

01	CT	S_1	S_2	S_3
S_4		S_5	S_6	S_7
t	res		value	

where

CT is a six-bit field containing the character count of the original symbol.

S_i are the first seven (7) characters of the symbol. Symbols with fewer than seven characters are zero filled.

t is a five-bit field where the values are:

00000 - instruction
00001 - integer
00111 - EBCDIC text (also for unpacked decimal)
00010 - short floating point
00011 - long floating point
00110 - hexadecimal (also for packed decimal)
01001 - integer array
01010 - short floating point array
01011 - long floating complex array
01000 - logical array
10000 - undefined symbol

res is a three-bit field representing the internal resolution. The values are:

000 - byte
001 - halfword
010 - word
011 - doubleword

value Location symbols are always represented as a 19-bit byte resolution value.

Constants - Code = 10

10	CT	S_1	S_2	S_3
S_4		S_5	S_6	S_7
value				

where

CT and S_i have the same meaning as above.

value is the 32-bit value of the constant.

Object Language Extensions

The following new object code control bytes are added in order to supply the information necessary in the formation of symbol tables.

Object code control byte (HEX)	Type of Load Item
11	Provides type information for external (global) symbol.
12	Provides type and EBCDIC for internal symbol.
13	Provides EBCDIC, and forward reference number for undefined symbol.

The details of these object code items are listed below.

Control byte 11 -- Type information for external symbol

The load item is as follows:

11	Control byte
t	One byte which is a five-bit type field and a three-bit internal resolution field. The five-bit type field contains a code which is the same as the type information specified above. The three-bit internal resolution field is the same as the res field information specified above.

DN	Declaration number - One or two bytes (depending on the current declaration count) which specifies the declaration number of the external (global) definition.

Control byte 12 -- Type and EBCDIC for Internal Symbol.

This control item supplies type and EBCDIC for an internal symbol. The load item is as follows:

12	Control byte
t	The type and resolution as above
n	a byte specifying the length of the EBCDIC name in characters
.	name in EBCDIC
.	
.	
EXPR	Expression defining the internal symbol

Control byte 13 -- EBCDIC for an undefined symbol

This control item is used to associate a symbol with a forward reference.

The load item is as follows:

13	Control byte
n	length of name in bytes as above
.	name in EBCDIC
.	
.	
FRN	Two bytes specifying the forward reference number with which the above symbol is to be associated.

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 subsystem. 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 periods. 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. Its length must be 10 or fewer characters.

lm — specifies the name (file identification) of the load module to be created by LINK. Its length must be 10 or fewer characters.

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, public library, P1, and system library are to be searched, in that order, 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 system library is to be searched to satisfy external references which have not been satisfied by the program.

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

(Pi) specifies that the i^{th} public (core) library is to be associated with the program. Default is to P1 if not specified. Only one public library may be associated with a given program.

(NP) specifies that a public (core) library search or association 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 specificiations for the linking process are D, C, NM, and L.

Any specifications stated or implied hold over subsequent commands to LINK so long as LINK is not recalled by TEL.

COMMANDS WHICH INITIATE THE LINK SUBSYSTEM

The LINK subsystem 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 subsystem 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 subsystem (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.

* : is LINK's prompt character

B. RUN

The LINK subsystem 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 subsystem normally has no interaction with the user.

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 subsystem (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; e.g.,

```
RUN (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.

C. LINK called as a processor

The subsystem is called directly by using the command LINK without parameters. 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 subsystem 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:

```
                                    LP (printer)
LIST (loading map, et al)  ON       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:

```
!LINK

LINK HERE

:LINK (ND) (NC) (NL) A, B ON JED
:LINK C
:SEARCH (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 The LINK session is concluded by the command END and control is returned to the executive.

BREAK KEY

Depression 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 simply behave as it usually does after finishing a command. If called as a subsystem, LINK will return control to the user after typing its identifying mark; if called as a subroutine, 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

Default values for the options are underlined.

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 identifiers
	M/NM		Display or don't display loading map
	L/NL		Search or don't search the public and system libraries
	I/NI		Construct or don't construct internal symbol tables
	Pi/NP	P1	Associate or don't include the public library
OUTPUT			Specify the LM file.
	none		
LIST			Specify display options to be listed on printer, file, or terminal.
	D/ND C/NC M/MN		Same as LINK.
SEARCH			Specify which libraries are to be used in satisfying unsatisfied references
	L/NL Pi/NP	 P1	Same as LINK

END — Specifies the end of a linking process

None

RUN — Specifies to link, load, and start execution.

Pi/NP	P1	Same as LINK
D/ND		
C/NC		
M/NM		
L/NL		
I/NI		

get status
get error log

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 just 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

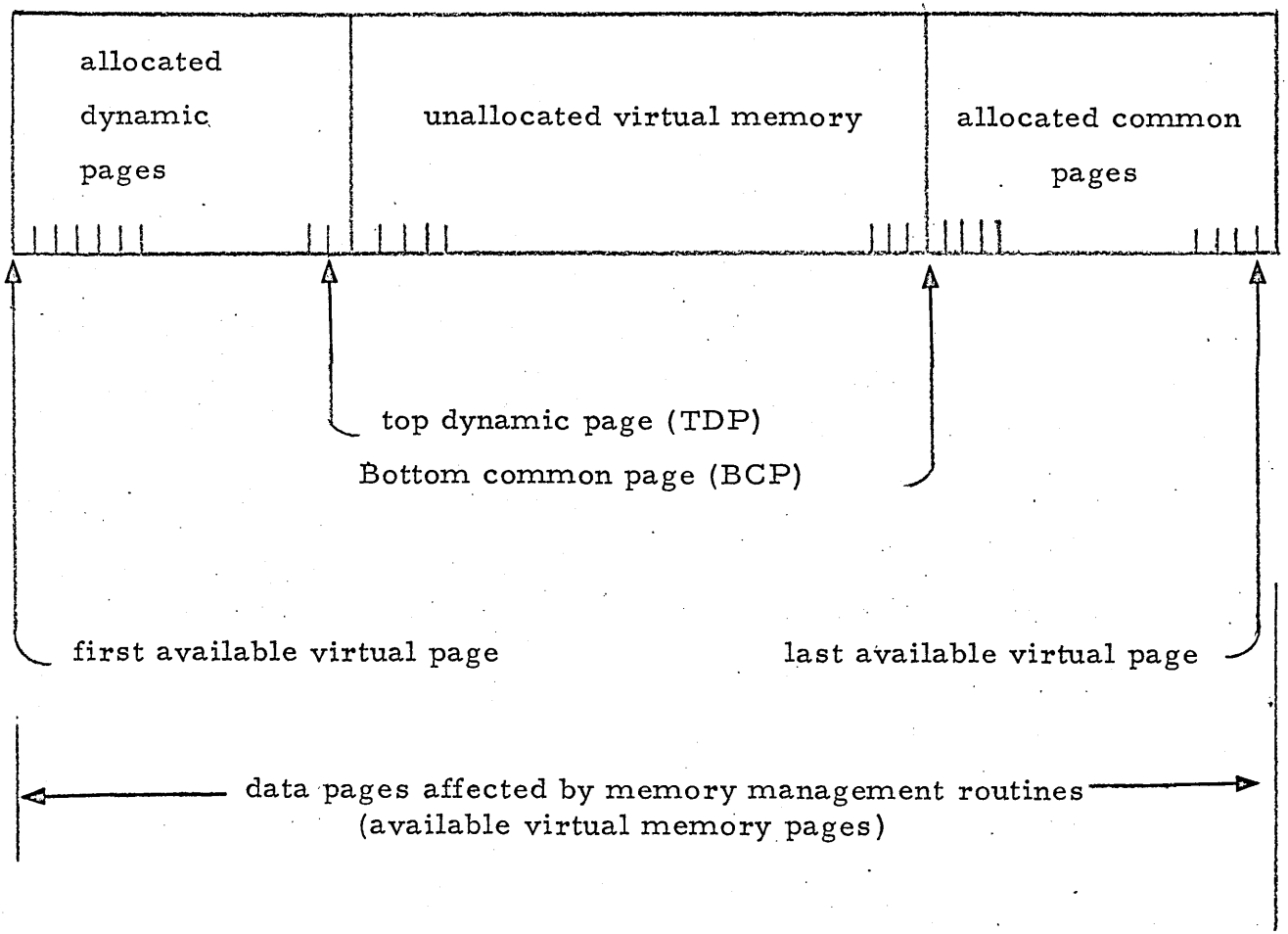
UTS provides the user with two general ways of getting and releasing pages of core memory in his data area: relative allocation and specific allocation.

Relative allocation is exactly compatible with the BPM CALs for getting and releasing pages: Pages may be obtained (allocated to the user program) from the lowest unallocated page address toward higher addressed pages. Pages obtained in this upward direction are called dynamic pages. Memory pages may also be obtained beginning with the highest addressed unallocated page toward lower addressed pages. Pages obtained in this downward direction are called common dynamic pages or simply common pages. Pages obtained in the dynamic and common area may not overlap and, if attempted, an error indication results. Neither may the relative allocation CALs be used to allocate a page already allocated by a specific allocation routine.

Specific allocation allows the user to get or release any page in the user's data area by reference to the word address of the first word of the page. Errors result on attempts to get already allocated pages or to release already released pages. Pages allocated via the relative allocation routines may not be released by specific allocation routine reference.

A limit applies to the number of physical pages which may be allocated to a user for all purposes. This limit is initially set by SYSGEN and may be modified dynamically by the performance control program.

Virtual memory available for allocation may be pictured as shown below.


allocated
dynamic
pages
unallocated virtual memory
allocated common
pages
top dynamic page (TDP)
Bottom common page (BCP)
first available virtual page
last available virtual page
data pages affected by memory management routines
(available virtual memory pages)

A. Get Common Limits

M:GL The GL routine returns the lowest and highest word addresses within the data area presently allocated to the user by get common page CALs. The lowest address is returned in SR1 and the highest address is returned in SR2.

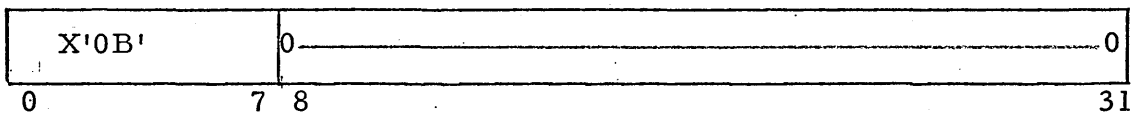
The M:GL procedure call is of the form

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 allocates a specified number of pages at succeedingly lower address from the current lower limit of common storage (BCP) and extends that limit downward.

Pages are obtained and allocated to the user at succeedingly lower addresses beginning just below BCP until:

1) the required number of pages are obtained,

2) the installation set limit on the number of physical core pages is reached, or

3) a page already allocated via M:GP or M:GVP is encountered.

Returned information for the three cases is

CASE	CC1	SR1	SR2
1	0	No. of pages allocated	New BCP
2	1	No. of pages allocated	New BCP
3	1	No. of pages allocated	New BCP

Access codes for the allocated pages are set to 00 (Read, Write, or execute).

The M:GCP procedure call is of the form

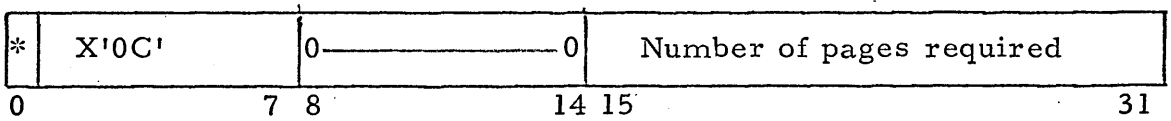
M:GCP pages

Pages specifies the number of memory pages by which common storage is to be extended.

Calls generated by the M:GCP 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 releases a specified number of pages at succeedingly higher addresses beginning at the current BCP and moves that limit upward.

Pages are released beginning at BCP toward succeedingly higher addresses until:

1) the requested number of pages have been released

2) the last available virtual page is released

In case 2), CC1 is set to one. In case 1), CC1 is set to zero.

Pages released by FCP have access codes set to 11 (no access) and any subsequent reference to these pages will result in a trap.

The M:FCP procedure call is of the form

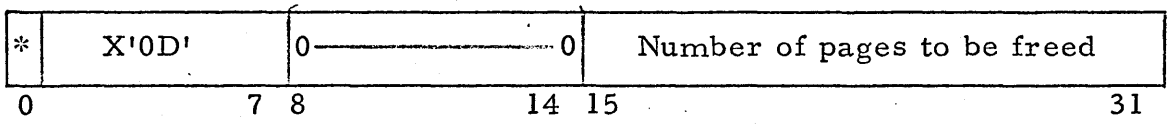
M:FCP pages

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 Dynamic Pages

M:GP The GP routine allocates a specified number of pages beginning with the page just higher addressed than TDP and extending that limit upward until:

1) the required number of pages are allocated,

2) the installation set limit on the number of physical core pages is reached, or

3) a page already allocated via M:GCP or M:GVP is encountered.

Returned information for the three cases is

CASE	CC1	SR1	SR2
1	0	Number of pages allocated	New TDP
2	1	Number of pages allocated	New TDP
3	1	Number of pages allocated	New TDP

Access codes for all allocated pages are set to 00 (Read, Write, or execute).

The M:GP procedure call is of the form

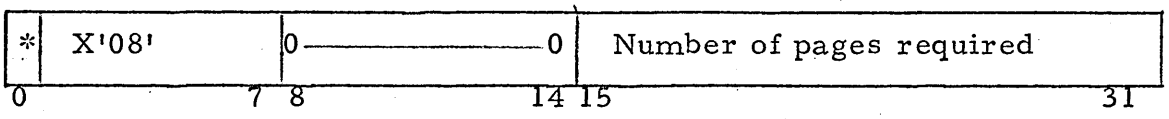
M:GP pages

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 Dynamic Pages

M:FP The FP routine releases a specified number of pages at succeedingly lower addresses beginning with TDP and moves that limit downward until:

1) the requested number of pages have been released, 0

2) the first available virtual page is released. 1

In case 2), CC1 is set to one; in case 1) CC1 is set to zero.

Pages released have their access codes set to 11 (no access) and any subsequent reference to these pages will result in a trap.

The M:FP procedure call is of the form

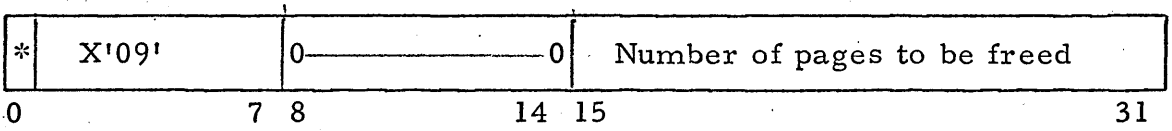
M:FP pages

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 allocates a specific page of virtual memory to the user. If the request is allowed, access for the page is set to 00 (Read, Write, or execute) and CC1 is set to zero. The request is disallowed if

1) the installation limit on number of pages allowed would be exceeded,

2) the page has already been allocated,

3) the page requested is outside the limits of unallocated virtual memory.

In these cases CC1 is set to one, and no page is allocated.

The M:GVP procedure call is of the form

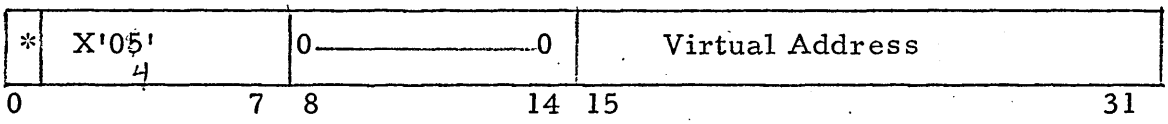
M:GVP virtual address

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 release a specific page of virtual memory. The indicated page is released and CC1 set to zero unless the request is for a page less than or equal to TDP or greater than or equal to BCP, in which case CC1 is set to one and no page is released.

M:FVP procedure call is of the form

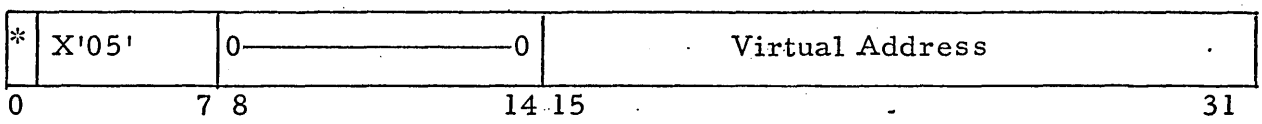
M:FVP virtual address

Virtual address specifies the address of the first word on the virtual page to be released.

Calls generated by the M:FVP procedure have the form

CAL1,8 FPT

FPT is the address of a word as shown below.



NEW UTS SERVICE CALs

Two new calls have been added to UTS in order to provide setup of communication with the terminal I/O handler. They may be issued only by on-line user's program and are ignored if issued by a batch program.

A. 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 (TEL)	!
EDIT	*
PCL	<
LINK	:
BASIC	>
META	>
FORTRAN	>
DELTA	bell
SYMCON	:
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). If given in batch mode, no operation results.

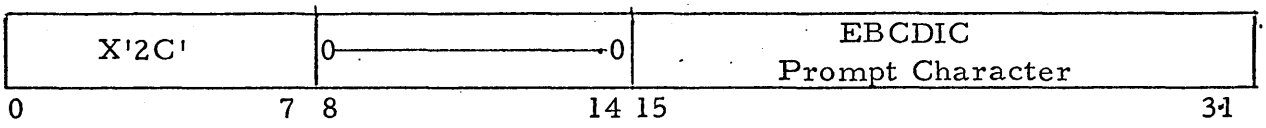
The procedure call is of the form

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.



B. 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 characters 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.

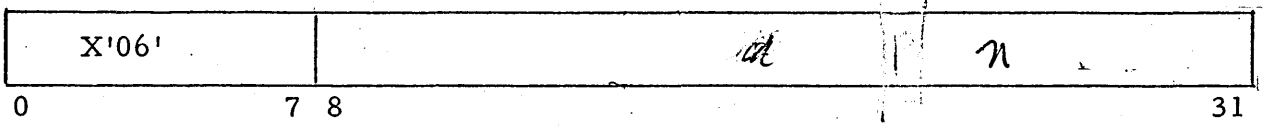
The procedure call is of the form

M:CT n

n specifies the number of the desired table $1 \leq n \leq 5$. The procedure generates a

CAL1,8 FPT

FPT is the address of a word as shown below.



The current tables translate for Models 33 and 35 Teletypes, and SDS Keyboard Displays. Additional tables are contemplated for Model 37 Teletypes, and Frieden 7100's. Since translation tables are assigned to lines at SYSGEN time, it is unnecessary for users of fixed location consoles to use this command. Dial-up lines are another matter.

The current assignments for the n parameter are

n	Meaning
0	33
1	Use standard Mod, ~~33~~ 35 TTY table (cr lf and ESC)
~~2~~ 3	Use the standard K/D table (all cursor movements, hard copy signals, mode changes, and roll commands activate)
~~3~~ 2	Reserved for Model 37 TTY
4	Reserved for IBM 2741
5	Reserved for Frieden 7100

C. Suspend Program for N Seconds

This CAL causes N seconds of real-time to elapse before the next instruction in sequence is executed. It may be used by programs which wish to operate time periodically. They are said to "sleep" during the suspend period. When they are awakened they begin execution at a priority just higher than the computer queue.

The procedure has the form

M:WAIT N

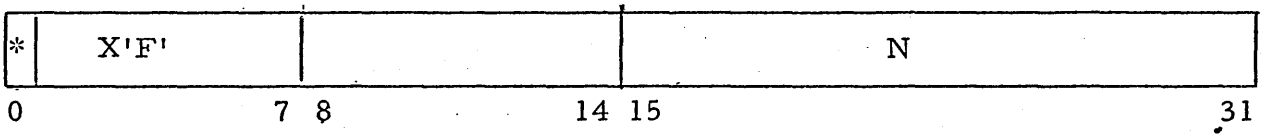
Where N specifies the number of seconds to wait.

Wait is restricted to on-line use.

Code generated by this procedure is

CAL1,8 FPT

FPT has the form



D. Change Virtual Map

This CAL is provided to allow special system processors and other specially privileged programs (e.g., those of the system programmer) to see into and display or change portions of the resident Monitor.

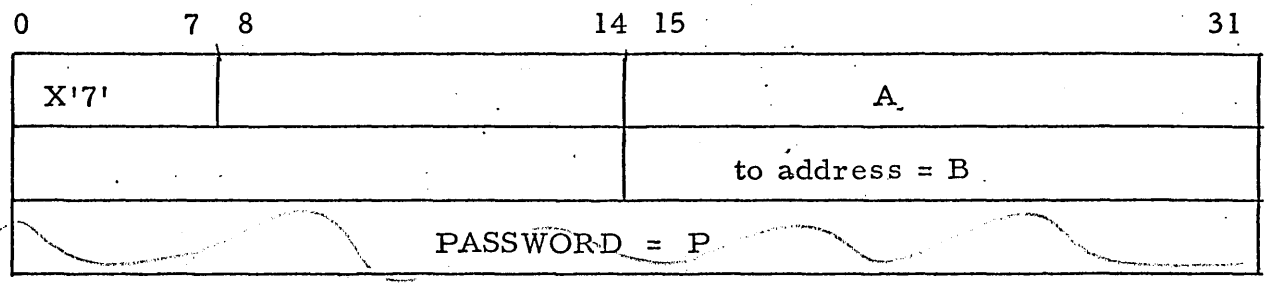
The CAL must have procedure has the form:

M.SAD A,B,P

which generates the following code:

CAL1,8 FPT

where FPT is the address of:



The real core page address A is placed in his map at the "to" address. Access for the page is set to data, but write locks for the physical page are not changed.

Restrictions:

1) The password supplied must check with a system password known to the Monitor.

2) The virtual "to" address must not be already assigned to the user.

In either of the above cases, no map change is made and CC1 is set.

E. Read and Write Assign Merge Record

Throughout a job or on-line session, I/O unit and file assignment information is retained for merging into user or processor DCB's at each job step. This information is maintained in an Assign Merge record on RAD, with one record location assigned to each user by the log-on procedure which places the AM record disc address in JIT.

Special CAL's are used to read (RAMR) and write (WAMR) this record. They must reference a closed DCB at least eight words long.

RAMR and WAMR are both CAL1,1 FPT instructions. The FPT code for RAMR is X'2D' and for WAMR is X'2E'. Otherwise, the FPT formats are similar and as follows:

*	Code					DCB Address
P_1	P_2	P_3	P_4		P_6	present bits
P_1	*					error address
P_2	*					abnormal address
P_3	*					buffer address
P_4	*					byte size
P_6	*					byte displacement

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. (See Part VIII for messages when DELTA is in control).

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). Postmortem dumps are performed, and the C device is read ignoring everything up to the next control command.

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 different. 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 specified 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 pushed into a 19-word block of user's memory (on a doubleword boundary) and a pointer to the stack pointer doubleword 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 controlled 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 procedure 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.

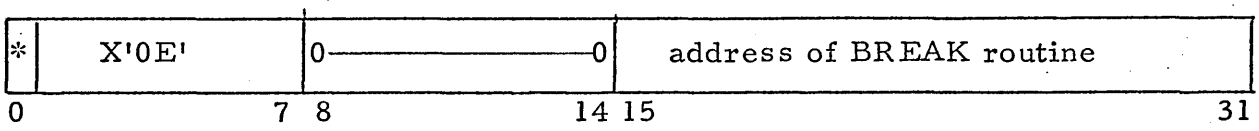
The M:INT procedure call is of the form:

M:INT address

Address specifies the location of the entry to the program's 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.



A zero address resets break control. If the address specified is in the range of virtual addresses assigned to the Monitor, then zero is substituted (break control is reset).

ERROR AND ABNORMAL MESSAGES

All error or abnormal conditions which normally results in the batch Monitor skipping 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 messages reside in the system file (:MESS). The keys to the error text records are the codes established by the Monitor for the error or abnormal conditions.

The second message has the form:

EXECUTION STOPPED 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 CAL instructions (CAL1, CAL2, CAL3, and CAL4) provided by the Sigma 5/7 hardware. CAL instructions are used for requesting Monitor services. Execution of a CAL instruction causes the executing program to trap to the Monitor where a validity check is made, and then the CAL is decoded to determine the service requested and the requestor. The requestor may be a user, processor, real-time task, or the Monitor. If valid, the requested service is performed. If invalid in either type of CAL or type of service requested, the request is not honored and the user is informed by a console message.

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 catagories:

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
CAL1,1	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'07'	M:ADFILE
		X'08'	M:CAT
		X'09'	M:UNCAT
		X'0A'	M:FEXT
		X'0B'	M:DEFICE (DRC/NODRC)
		X'0C'	M:RELEC
		X'0D'	M:DELREC
		X'0E'	M:UFILE
		X'0F'	M:TFILE
		X'10'	M:READ
		X'11'	M:WRITE
		X'13'	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:DEFICE (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 (INLINES)
		X'2B'	M:DEVICE (CORRES)
		X'2D'	M:RAMR
		X'2E'	M:WAMR
		X'2F'	M:JOB

CAL	address	FPT CODE	FUNCTION
		0	Type to operator
CAL1,2	FPT	X'01' MSLL	M:PRINT
		X'02' M;Eqyuc	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'07'	M:SAD
		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'0F'	M:WAIT
		X'10'	M:TIME
		X'11'	M:STIMER
		X'12' ~ X'13'	M:TTIMER ← DISPLAY
		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	FPT	X'06'	M:CT

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
		X'0D'	M:SXC
		X'0E'	M:DED
		X'0F'	M:UNDED
		X'10'	M:IOSTOP
		X'11'	M:IOSTART
		X'12'	M:IOEX SIO
		X'13'	M:IOEX TIO
		X'14'	M:IOEX TDV
		X'15'	M:IOEX HIO
		X'16'	M:ABSLOAD

CAL	address	FPT CODE	FUNCTION
CAL1,9	7	X'07'	M:CLEAR
	8	X'08'	M:TERM
	9		Reserved for real-time extensions
	A		
	B		
CAL1,A	FPT	X'00'	Save Monitor's interrupted environment
		X'01'	Restore Monitor's interrupted environment

E. Monitor Only

CAL1,1		X'16'	Direct Disc Read
		X'17'	Direct Disc Write
CAL1,9	6	----	Close Cooperative File
CAL1,B		----	Event Mark — Reserved for generalized event measurements
CAL1,C		----	Event Count — Reserved for generalized event measurements
CAL1,D		----	Event Time — Reserved for generalized event measurements
CAL1,E		----	Event Auto-Display Control — Reserved for generalized event measurements
CAL2,0		----	Branch to overlay segment (0B) — Used for internal Monitor overlays
CAL2,1		----	Branch and save segment number (OBAL) — Used for internal Monitor overlays
CAL2,2		----	Restore segment and B*SR4 (OBSR4) — Used for internal Monitor overlays
CAL2,3	code*	----	System Recovery
CAL2,4	code*	----	Reserved for internal debug routine

*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's 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
s	use restricted by password or other

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
		0			
-	CAL1,1 FPT	1	REW		
-		2	WEOF		
-		3	CVOL		
-		4	DEVICE (PAGE)		
-		5	DEVICE (VFC)		
-		6	SETDCB		
-		7	M:ADFILE		
-		8	M:CAT		
-		9	M:UNCAT		
-		A	M:FEXT		
-		B	DEVICE (DRC)		
-		C	RELEC		
-		D	DELREC		
		E	UFILE		
-		F	TFILE		
-		10	READ		
-		11	WRITE		
-		12	TRUNC		
-		14	OPEN		
-		15	CLOSE		
m		16	----		Direct disc read
m		17	----		Direct disc write
		18			~~Read error stay granule~~
-		1C	PFIL		
-		1D	PRECORD		CA 1,6
-		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
m		2D	RAMR	*	Read assign merge record
m		2E	WAMR	*	Write assign merge record
s		2F	JOB	*	Add input symbol table

30-3F reserved for internal functions

USE CODE	CAL	FPT hex code	M: NAME	UTS	Description
-	CAL1,2 FPT	1	PRINT	*	
-		2	TYPE	*	Type message to operator (or user)
-		4	KEYIN	*	Type message and await response
-		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		
o		C	RXC		
o		D	SXC		
		E	DED		
		F	UNDED		
		10	IOSTOR		
		11	IOSTART		
		12	IOEX SIO		
		13	IOEX TIO		
		14	IOEX TDV		
		15	IOEX HIO		
		16	ABSLOAD		
	CAL1,6	unused			
	CAL1,7	unused			

USE CODE	address	FPT hex code	M: NAME	UTS	Description
		0			
-	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
s		7	SAD	*	See and display
-		8	GP	*	Get dynamic core page
-		9	FP	*	Free dynamic core page
-		A	SMPRT		Set memory protect
-		B	GL	*	Get available core limits
-		C	GCP	*	Get core page in common
-		D	FCP	*	Free core page in commo
-		E	INT	*	Connect to BREAK key
o		F	WAIT	*	Suspend program n secon
-		10	TIME		
-		11	STIMER		
-		12	TTIMER		
-		14 (13)	TRAP ← Display	*	Return system buf parameter
-	CAL1,9 1	----	EXIT	*	Normal program terminat
-	2	----	ERR	*	Error termination of job step
-	3	----	XXX	*	Error termination of job
-	4	----	STRAP		
-	5	----	TRTN		
m	6	----	----- Super close		Close all cooperative files
r	7	7	M:CLEAR		
r	8	8	M:TERM		
r	9	----			Reserved for real-time
r	A	----			
r	B	----			
	C		DISPLAY		Return system buf parameter
r	CAL1,A FPT	0	-----		Save Monitor environment
r		1	-----		Restore Monitor environment
	B				
m	CAL1,B code	----	Event marker	*	
m	CAL1,C code	----	Event counter	*	
m	CAL1,D code	----	Event timer	*	Monitor performance
m	CAL1,E code	----	Display	*	measurement
m	CAL2,0 --		OB		Branch to overlay segmer
m	CAL2,1 --		OBAL		Branch to overlay and sav return
m	CAL2,2 --		OBSR4		Restore segment and retu *SR4

CODE	address	FPT hex code	M: NAME	UTS	description
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 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 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 - "file" no default	Read number of bytes specified
C	Control input (same as UC) not assignable	Terminal UC	Read number of bytes specified or to message complete
CI	Compressed input	Disc no default	Read number of bytes specified
EI	Element input	Disc no default	Read number of bytes specified
SI	Source input	Terminal	Read number of bytes specified
BO	Binary output	Symbiont punch output- CP	Write number of bytes specified
CO	Compressed output	Disc Symbiont punch output - CP	Write number of bytes specified
EO	Element output	Disc no default	Write number of bytes specified
SO	Source output	Disc Symbiont punch output - CP	Write number of bytes specified
PO	Punch output	Disc Symbiont punch output - CP	Write number of bytes specified
UC	Users Terminal (console) not assignable	Terminal	Read or write number of bytes specified
DO	Diagnostic	Terminal	Break into carriage-size records, insert carriage returns, and type up to 132 characters.
LO	Listing output	Line Printer LP	Write number of bytes specified up to one line
GO	Binary output for execution	Disc default $ROM	Write number of bytes specified

Part XII. TERMINAL OPERATIONS and SERVICES

TABLE OF CONTENTS

INTRODUCTION

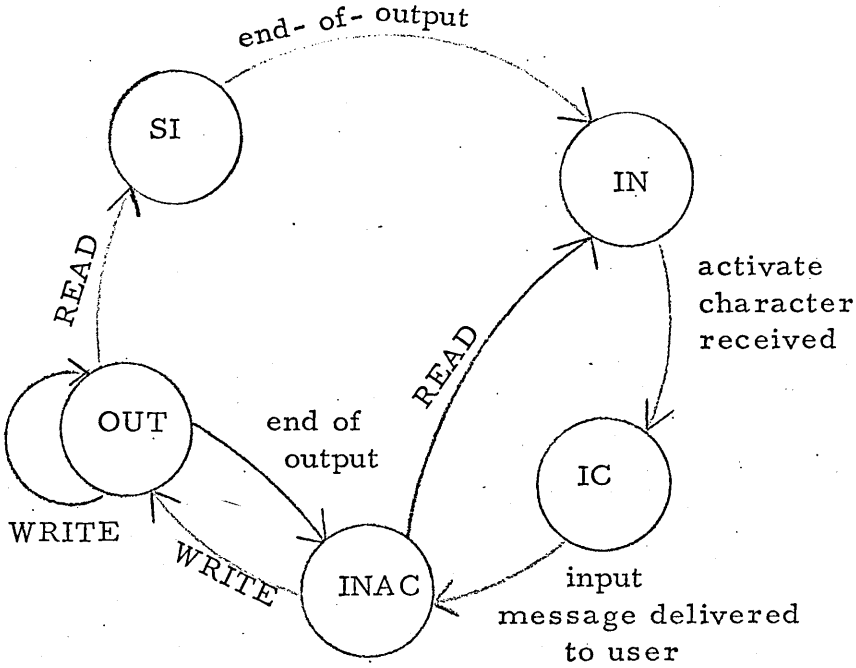
This section provides a detailed description of the UTS COC routines which handle input and output of messages to on-line users at typewriter-like terminals. It is intended that it provide source material for users of the routines (both processors and user programs writing and reading from the terminal), for the implementor of the routines as a functional sub-specification.

The functions performed by the COC routines are primarily the following:

1. Device handling for the COC hardware
2. Character translation to and from internal EBCDIC codes and the external codes of the various terminals which may be attached to the COC. (TTY, 7015, 7550, 7555, 2741, and perhaps others)
3. Parity generation and detection by character for those terminals requiring it.
4. Division of input character strings into messages as defined by receipt of activation characters. (Usually Cr, and Lf, and FF but other sets are specially available)
5. Communications with the UTS scheduler on break, read, read complete, output blocked, output unblocked, and other events which effect swap and execution scheduling.
6. Special interpretation of certain characters for intra-line editing and COC control functions.

The COC routines enforce proprietary use of the console. That is, either the user is typing an input message (and no output is being delivered to the terminal by the computer) or the computer is outputting and the user may not type input. The communication line is used in full duplex mode however and the user may always regain control of the terminal through use of the BREAK key or the executive escape key (E^c on teletypes).

The proprietary use is best illustrated by the state diagram below. Each terminal is in one and only one state at a time. The events which cause a line's state to change are given on the arrows connecting the states:



The states are:

INAC - no current activity on the terminal line; all input characters except BREAK, E^c, and COC control character sequences are discarded.

OUT - The current output message from the user's program is being typed.

More output may be presented by the program and if so it is queued and buffered for output. When enough output to sustain the terminal for some time (say 4 seconds) is queued the COC routines report to the scheduler which suspends execution temporarily and may swap the program to secondary storage. When the number of characters remaining to be typed falls below some threshold (say 1 second of typing) the COC again reports to the scheduler which then requeues the program for execution.

SI - The state which "remembers" that a READ command was given while output was still in progress. On completion of output the COC routines will begin to accept user terminal input. On any READ command COC reports to the scheduler which suspends the program until input is complete.

IN - The state during which input characters are received by the COC routines and packed into buffers. These buffers are resident in Monitor memory and none of the users program or data is required in core during input typing.

IC - When an input end-of-message (activation) character is received the line is placed in this state and the scheduler is signalled to re-queue the user for program. No further characters received are accumulated until the program reads again.

Input and output is carried in four word blocks each containing 14 characters plus a halfword link to the next related block. After a read is complete the input message is moved from these buffers directly to the user's area; actual number of characters received is reported in ARS of the DCB. On WRITE the users output message is moved to COC buffers to await transmission. Unused buffers are held in an available pool. The program is blocked appropriately when needed buffers are not available and restarted when they become available.

USER PROGRAM AND TERMINAL USER OPERATIONS

1. Writing Records to the Terminal

Records are written to the user's terminal using the write CAL (CAL1, 1 FPT). The number of bytes specified are moved from the user specified buffer to the COC's buffers. The operation is always effectively "wait" -- that is, the text has been removed from the users area before return from the CAL even though the transmission to the terminal is not yet complete. Keys, if specified, are ignored.

The error return is taken in the following cases:

Bad DCB address (CAL error return)
Bad buffer address (DCB error return)

If no error return is specified control returns to TEL and an error message is printed at the users console.

Output in excess of 140 bytes from a single WRITE CAL is ignored; the first 140 bytes are transmitted, and the CAL abnormal exit is taken if one exists. If not, return is to TEL and an error message is printed.

If the specified record size is zero no action is taken and no characters are transmitted.

If the write is through the UC dcb the characters are transmitted exactly as supplied except that the pair (Cr, lf) is supplied for both Cr and lf (NL) characters. The user may therefore make up single lines through a series of writes (without Cr characters) or may produce several lines at the terminal with a single write (by inserting several Cr's in the buffer).

If the write is through a dcb other than UC (say LO or DO) then the COC routines supply a (Cr, lf) pair at the end of the specified character string (but see VFC for special format control in section 5). That is the number of bytes specified in the FPT is moved from the users area to COC buffers and the pair (Cr, lf) is appended in COC buffers.

Trailing blanks are suppressed from output lines for writes through all dcbs except UC but the programmer should set his record size to avoid this overhead if at all possible.

If lower case letters are sent to a single case terminal they are translated to the upper case cognates.

For all writes to the user's terminal a count of characters on each line (between carriage returns) is kept and if the line if too long, as determined at login, for the physical terminal in question then additional (Cr, lf) pairs are inserted to break the line. Line length is a terminal specific parameter supplied via a logon dialog with the user and retained in JIT. (See console commands below.) A count is also maintained of lines on the page, and a page heading line is supplied to the terminal as outlined below.

2. Reading Records from the Terminal

The read command M:READ (CALl,1 FPT) causes the COC routines to accept input characters from the terminal. (If a prompt character has been specified by the program it is sent to the terminal first, see below). The operation is always "wait". That is, the input message is complete in the users area before control passes to the next instruction following the M:READ. Messages are terminated (completed) on receipt of the number of characters requested or one of the characters cr, lf, form feed, or ESC ESC which will

be the last character in the buffer. (Additional special termination (activation) characters are supplied in the case of a DELTA issued read. They are tab, ↑ , =, and /.) The actual number of characters, including the activation character, in the message received is returned in ARS, word 4 of the dcb. No more characters than specified in the M:READ FPT are transferred to the user. If there were more characters in the input message than specified, and an abnormal exit is specified, then it is taken.

On receipt of either cr or lf the appropriate characters are sent to the terminal to insure carrier return; however, only the actual character received is placed in the buffer. When form feed is received, FF (EBCDIC OC) is placed in the buffer, the pair (cr, lf) is sent to the terminal, and the next issued read or write is preceded by page heading output. When the pair ESC ESC is received the carrier is not moved. When bk (the BREAK key character) is received cr lf is sent to the terminal the message is deleted and the break entry of the program (if any) is taken. If the character pair ESCF is received the end of file exit from the READ CAL is taken.

Characters received with parity error are indicated by placing the PE code (EBCDIC 2F) in the buffer, and the character # is sent to the terminal. Lower case letters, if received are translated to their proper EBCDIC form.

If the user types more than 140 characters before giving an activation character the COC routines simulate a line cancel -- that is the current line is deleted, ← cr and lf are sent to the terminal and the read continues. In addition to the line cancel which the user may initiate by typing X^c (control shift and X) the user may delete individual characters by typing RUBOUT in

which case the last character typed is removed from the COC buffer and the character back slash (\) is sent to the terminal. He may rubout n characters by typing n RUBOUT's and n\'s will be sent to the terminal. On the keyboard display X^c deletes the current text line on the scope face and rubout backspaces the cursor and erases the last character typed. As in the write CAL bad information (character parity errors) is reported via lost data (07) code to the abnormal CAL exit if it exists. If no abnormal exit is specified then the bad information is not reported.

Other activation sets may be provided via the CAL described in the Part X. The number of different activation sets may be increased at SYSGEN time to accommodate special terminals. They may also be associated initially with fixed line numbers.

Error returns are also taken in the following cases:

Bad dcb address	(CAL error return)
Bad buffer address	(DCB error return)

Abnormal returns are taken for:

Parity errors in received message	(CAL abnormal return)
Lost data -- message longer than read request	(CAL abnormal return)
End of file -- ESC F character pair received	(CAL abnormal return)

Prompt Characters

The user program or processor may set up a "prompt" character to be delivered to the console just prior to each read. Any valid EBCDIC character may be specified. A null character (EBCDIC 0) turns off the prompt action. The character is set by using a CAL1, 1 FPT where the one word FPT contains X'2C' in the high order byte and the prompt character in the low order byte.

Since the prompt character is carried in COC resident tables for each line, the TEL and DELTA processors do not prompt via this mechanism but rather by writing single character records before issuing a read.

3. BREAK (bk) Character Action

Action on receipt of the break character depends on whether the console is inputting or not. If inputting, cr and lf are sent to the console, the message, if any, is deleted and the current read is terminated.

Whether inputting or not, control goes to an alternate address associated with the users program, with the users environment (PSD and registers) as of the point of interrupt placed in the users temp stack (pointed to by his TCB.) The program may be continued from the point of interrupt by giving a trap return (M:TRTN or CAL1, 9 5). The actual alternate address used depends on the user program and associated processors in the following order:

(a) If DELTA is associated with the program then control goes to DELTA.

(b) If the user has given a M:INT CAL the address specified by that CAL is used. (CAL1, 8 FPT where FPT is *X'OE' break address.) A zero or invalid address resets break control.

(c) Finally if neither 1) nor 2) obtains then control goes to TEL, a message is printed for the user, and TEL issues a terminal read for commands from the user.

In any of the above cases all current output is drained to the terminal -- none is lost. Because of the blocking action of the COC this output is not usually longer than 4 seconds or 4 seconds plus one line.

In order to provide fail safe operation against program errors in the user break handling routine, to at the same time allow special sub-processor action on multiple break signals (primarily FDP), and to provide compatable operation with future communication gear which does not have full duplex lines (CIOP), BREAK signals are counted by the COC handler. If four BREAK signals are received without intervening characters from the user terminal, then control is given to the TEL executive as if an E^C character had been received. See below.

4. Monitor Escape

The console user may always put himself in comminication with the UTS executive, TEL by typing E^C (control shift and E keys pressed together). No current output is lost, but if the console is in read status (IC or IN states) the current input line is canceled -- ← cr lf are sent to the console. If the users program is restarted from the point of escape to the executive -- via the CONTINUE command -- and the console was previously reading the read is re-issued.

5. Device and Set DCB CALs

The M:SETDCB CAL may be used to set error and abnormal addresses in one of the dcbs associated with the users console. The abnormal return is only taken if an OPEN is attempted on an open file. The error exit is taken when a read or write specifies an invalid buffer address. The error code and other information communicated to the user program is as specified in Appendix H of the BPM manual. If no error address is specified control is transferred to TEL and an appropriate message is transmitted to the user.

The COC routines acknowledge the following CALs with action as listed. All other CALs if given for a DCB assigned to the users console are ignored without comment. In general CALs which set the DCB may be given and will result in the indicated modifications to the dcb but the COC routines only make use of certain of the parameters as defined below. On three CALs the COC routines gain control to carry out the requested action:

M:DEVICE CAL	COC Action
PAGE	Page heading is typed at the user console. See section 6.
LINES	The number of print lines per page is saved in the COC line tables.
NLINES	The number of lines remaining on the current page is returned in SR1. The user may at any time examine the current number of lines on the console page by looking at the JIT byte JB:LC.
FORM	The message indicated by the FPT is typed at the users console. COC suspends operation until a break character is received from the terminal. This is taken to mean that the paper has been changed appropriately and the program is enqueued for resumption.

Parameters in the dcb which are recognized and acted on by the COC routines are as follows:

Parameter set by M;DEVICE CAL	COC Action
SIZE	This record size in bytes is used by reads and writes for which no size is specified in the CAL. If neither is specified no characters are transmitted and return is immediate.
SPACE	If this parameter is set and VFC is not on, the number of spaces indicated minus one are inserted before each write. Counts of 0 and 1 result in single spacing, that is, no spaces are inserted before each write. Counts of 0 and 1 result in single spacing, that is, no spaces are inserted before the test line.
VFC	If this flag is set the COC routines simulate the printers vertical format control as specified in the first character of the text lines written. The simulation is limited to the following cases.

hex code	action
C1 - CF	COC insert 1-15 spaces before the print line. (Page check on each insert.)
F1	COC skips to "top of page" by skipping lines to page top and printing the heading information followed by the print line.
60, E0	COC does not insert (cr, lf) after print line.

In all cases except the latter the print line is followed by (cr lf) with an appropriate check for page overflow.

DRC/NODRC	Used to inhibit automatic page heading
COUNT	See page heading section
HEADER	See page heading section
TABS	See tab section

6. Page Control and Page Headings

The COC routines count lines transmitted to, and received from, the users terminal in a line associated cell. Whenever a read or a write operation is given this line count is compared to the maximum lines per page carried in the dcb through which the read or write was given, and if this maximum has been exceeded, a new page heading set is produced. (The maximum may have been exceeded by several lines if several input lines were cancelled via X^C at the exact bottom of the page before the next read or write was issued. If so an appropriate adjustment is made in the heading.) Also page headings are produced whenever the device CAL PAGE is issued by the user program, and whenever the terminal user types in the character FF (control shift and L keys). The latter case is similar to page overflow in that the heading information is not produced until the associated user program or processor issues its next read or write command.

Two kinds of page heading are produced:

(a) the standard, automatic, page heading, and
(b) a user heading as specified by HEADER and COUNT device CALs.

The automatic heading may be suppressed, if desired, by giving a NODRC CAL. Heading information is taken from the dcb through which the read or write was given, thus if writes are given to the terminal through several dcbs the heading printed will depend on the dcb through which the top line of the page was written.

The automatic page heading includes current time, date, user id and account number, user line number, page number, and possibly an administrative message. It is intended for output on the top line of the form just under the fold (if any). The heading information is preceded by 6 (cr, lf) pairs (fewer if excess lines were printed on the preceding page) and followed by 5 (cr, lf) pairs. (The terminal is not assumed to have a form eject mechanism.) This spacing plus a standard 54 printed lines per page produces 11 inch pages with one inch margins at top and bottom. The standard heading line may be sliced off these pages to produce clean copy if desired.

Sample heading form is shown below

12:01	12/12/67	1A-03	NAME	ACCT	[36]	Administrative message
a	b	c	d	e	f	g

a) Twenty four hour clock time that the page heading was issued.

b) Current date

c) Line number of COC line and user number (the schedulers job identification)

d) The first four characters of the login id

e) The first four characters of the login account number

f) The page number, enclosed in brackets is centered for a 72 character wide terminal

g) The administrative message is supplied by the system operator via this mechanism to all users. It is limited to 32 characters.

If NODRC is specified in the dcb then the text of the heading is not produced but the (cr, lf) pairs for spacing are retained.

Headings specified in the dcb of the read or write are produced following the automatic heading with position, text, and page number as specified in the BPM manual. The page count in this heading is that carried in the dcb, and is reset with each COUNT device CAL while page count for the automatic heading is carried in JIT and never reset.

7. Tabs

Tab stops as set in output dcbs by the device CAL TAB or by SET commands result in spaces inserted in the output stream to bring the current count on the typed line to the character position indicated by the next higher tab stop given in the dcb. Note that this is like typewriter tabbing action and is different from the BPM tabbing action in some cases -- it is not possible to overlay information using tabs. The platen width test is still in effect and (cr, lf) pairs will be inserted if the count-on-line exceeds the carriage width. If tab stops are not set the tab character is sent directly to the terminal.

Tabs for input (READs) are handled somewhat differently in that they are simiulated on input at the discression of the user. Tab locations specified by the most recent tab setting CAL for a dcb connected to the console are packed into one of the COC input buffers and associated with the COC line tables. Use of this table for simulating tabs is controlled by a software flag turned on and off by the user via the character pair ESC T. These characters are never placed in the buffer. Each use of the pair toggles the tab simulation flag. When the flag is on and a HT (ASCII 09) is received, enough blanks are sent to the terminal to move the carrier to the next higher tab position. The HT character is placed in the input buffer for the reading program. Carriage returns are not inserted to split extra long input lines.

In addition to setting tabs via the CAL instruction the user may set them by the TEL command TABS a, b, c, ... where a, b, c, ... are the column numbers at which tabs should be simulated. The column numbers must be in ascending sequence. Tabs may also be turned off independent of the COC flag by TABS 0. Setting tabs via this mechanism affects all output through all DCBs connected to the terminal even if subsequent device CALs are given, and applies throughout the user session until reset.

8. Paper Tape I/O

Paper tape is input and output by the COC routines via a special mode entered when the TAPE (DC2) character is received and exited, when TAPE (DC4) is received.

In this mode the usual functions of RUBOUT, X^c, and tab are suppressed, cr is echoed but not followed by lf, and nothing is echoed for lf. Tapes must be produced with both the cr and lf characters punched (as they would be if they were produced directly from computer output by simply turning on the tape punch during a listing, data output, etc.)

9. User - COC Communication Keystrokes (TTY)

Certain user keystrokes are interpreted specially by the COC routines. Keystrokes and the actions which they produce are dependent on terminal type and are controlled out of the input character translation table. For model 33 and 35 Teletypes and SDS 7015, 7550, 7555 terminals the following conventions apply:

CAN (X^c) or ESC X	When the character CAN (generated by pressing both control shift and X keys) or the character pair ESC X is received the current partial input message is erased. The charcter triplet (←, Cr, lf) is sent to the terminal. On the keyboard display cursor return plus erase are sent to remove the current line from the screen.
RUBOUT or ESC DEL	When RUBOUT or the pair ESC DEL is received the last received character in the buffer is deleted and the character "\" is sent to the terminal. If the entire message is deleted by repeated used of RUBOUT then the pair (Cr, lf) is sent to the terminal. On the keyboard display cursor left plus erase are sent for each RUBOUT recieved until the entire message is erased.
Cr or Lf	When either of these characters are received the character pair (Cr, lf) is sent to the terminal. The message activation condition is set (the COC line state goes to IC) causing the program which issued the read to be restarted. On the keyboard display NL is echoed for either.
ESC ESC	The message activation condition is set. ESC character is placed in the input buffer.
ESC E	When the user presses the ESC key followed by the E key the COC toggles the flag which controls echoplex output. Normally the terminal is assumed to be local printing and echoplex is off.
ESC T	When the user presses the ESC key followed by the T key the COC routines toggle the tab simulation switch. Normally tab simulation is off and no tab stops are set.
FF (L^c) or ESC L	When the user presses control shift and L keys together or the pair ESC L the COC routines place a FF (EBCDIC OC) character in the input buffer, echos cr lf to the console, and, on the next read or write to the terminal, produce page heading information at the terminal. FF terminates the input line.
ESC F	When the character pair ESC F is received from the terminal the end-of-file abnormal exit is taken from the current READ command.
ESC U	When the character pair ESC U is received from the terminal a mode is set which causes all received lower case letters to be treated as upper case. translated to and

Tabs
ESC I

10. Terminal Users Logon Procedures & Information

For login purposes terminals are divided into broad classes depending on the character codes used for transmission. Terminals of each class may only enter the system via line numbers (telephone numbers) assigned to that class at SYSGEN time. Otherwise the characters received would be indecipherable and the initial dialog could not take place at all. Operator keyins will be provided for changing terminal class after the system is in operation.

Most terminals will be of the ASCII class initially and during login the following information is established via TEL commands:

a) Type:
   a) TTY 33, 35, SDS 7015
   b) TTY 37
   c) SDS 7550, 7555

b) Carriage width:
   a) 72 for 33, 35
   b) 86 for 7015, 7550, 7555
   c) variable for 37

Default terminal type is 33 with carriage width 72. Carriage width may be changed by a TEL command any time during the session.

Parity checking on input is not done until the 7550 or 7555 is established as the type.

Part XIII. MACHINE LANGUAGE ASSEMBLER (METASYMBOL)

TABLE OF CONTENTS

ON-LINE METASYMBOL OPERATIONS

The Meta-Symbol assembler is invoked by the UTS terminal user with the command

!META sp [, sp , ..., sp] ON [rom] [, list]

As explained in the document on the terminal executive language, sp represents a source program, and rom and list designate the destination of binary and listing output. Output specifications may be dispensed with if they have been pre-set with LIST ON or OUTPUT ON executive commands.

The sp list is entirely composed of file identifiers (fid) or the user terminal name, ME; rom must be a file identifier. If unspecified, the binary output will be placed in a scratch file which the user may later reference with the symbol $.

List may represent the user's terminal (ME), a disc file (fid), or the line printer (LP). However, should the listing output be directed to a file or line printer any errors encountered in the assembly will be displayed also on the user's terminal. For example:

! META ALPHA ON BIN, ME

Assembles the source program from file ALPHA, putting binary in file BIN, and producing a listing on the terminal.

! META ALPHA, BETA, GAMMA ON BIN, LP

Assembles the source programs contained in the files indicated, putting binary in file BIN, producing listings on the line printer.

The executive command META effectively replaces the following control cards needed to perform the equivalent operations through batch processing:

```
! JOB - etc.
! ASSIGN M:SI - etc.
! ASSIGN M:LO - etc.
! ASSIGN M:BO - etc.
! METASYM SI, LO, BO
```

Meta-Symbol has a number of optional assembly features which are not easily specified on the META executive command. Therefore, immediately after it is in control, the assembler will prompt the user for specialized options, in the following manner:

```
! META  ALPHA  ON  , ME   (Assemble ALPHA, binary on file $,
                          listing on terminal.)
```

WITH:

Typically, the casual user would terminate this request with a carriage return (or new-line), and the assembly would take place. Alternatively, any or all of the options described under OPTIONS can be specified, and when the carriage return terminates the option request, they will be performed during the assembly.

INPUTS TO METASYMBOL

Each file mentioned in the input list will typically contain one source program, created and maintained through use of the EDIT subsystem. If more than one file is mentioned, the assembler will proceed through them in order from left to right, performing the assemblies with the options indicated by the user.

Should an input file contain more than one source program (more than one END directive), all will be assembled. However, a file need not contain an END

directive since the Monitor automatically changes from file to file. It may contain only a piece of code which has been selected for assembly into a larger program.

For example:

```
! META ALPHA, BETA, ME  ON  BIN
≥     END START
≥     ESC F                    (the terminal end-of-file code)
```

Files ALPHA and BETA may not have contained an END directive. The user has chosen to assemble them together, supplying the necessary END directive directly from the terminal.

When input is from a keyed EDIT file, a decimalized representation of the sequence number for each record will be placed in the assembly listing in the position normally held by col. 73-80 of an input card.

It is possible, in addition, to make use of Meta-Symbol's internal editor in conjunction with compressed source files while running on-line. The editor and source compression facility are oriented toward card image batch processing needs, but could be useful where backup files must be kept on cards -- or where work must be done in strictly BPM compatible fashion. These features of Meta-Symbol are described in the reference manual (90 09 52), chapter 12.

If a program in compressed format exists on RAD, either as the output of the assembler or as the result of a FMGE operation, the user can assemble it with on-line Meta-Symbol simply by mentioning it as input. For example, if the name of such a file were CI-FILE, then it can be assembled with the following command:

```
! META  CI-FILE  ON  BO-FILE, LP
WITH:     Cr
```

Meta-Symbol distinguishes between the keyed source format of the EDIT files and the sequential binary format of compressed files. Files of both types can be given as input to the assembler in the same input command.

The on-line user can maintain a file of edit records which can be provided as input to the assembler to modify a compressed file. For example:

```
! BUILD   UPDATE-FILE
    1.000  +4,6
    2.000                  BANZ  EXIT
    3.000  +  10, 10
    4.000  +  END
    5.000     Cr

* END
! META  UPDATE-FILE, CI-FILE  ON  BIN, ME
WITH:        Cr
```

Meta-Symbol edit commands

The assembler, in processing UPDATE-FILE, will recognize the editing notation and will apply the update records to the next file in the list, which must be in compressed format.

OPTIONS

The complete set of options available in the batch version of Meta-Symbol is described in Chapter 12 of the reference manual (90 09 52). The executive command for calling the assembler covers options concerned with source or compressed input and listing and binary output. The following options may also be given the assembler when it prompts for them. Should more be given, only the standard listing and binary will be produced.

Option	
AC (ac_1, ..., ac_n)	This option is used when the assembler must access system files which are not logged under the system account (:SYS). The ac are the alternate accounts.
CN	This option requests that a symbolic cross-reference listing be included with the assembly listing. When this option is given, the assembler will access the user terminal for the concordance control records which indicate special concordance options. The assembler will prompt with the character '>', and the user may respond with the control record. For example: ! META SOURCE ON BO, LP WITH: CN ≥.SS X1,X2 (suppress X1 and X2) ≥.IO CAL3 (include op-code CAL3) ≥.END (terminate concordance control
CO	This option causes the assembler to produce a compressed version of the input program on the file specified in the M:CO DCB. This DCB must have been previously assigned with a SET Command.
LU	This option requests that the assembler include a listing of Meta-Symbol update records with the listing of the program.

NS

This option requests that no assembly summaries be included with the listing.

SD

This option causes the assembler to produce symbolic debugging object code for use with the DELTA debugging program. This object code is included with the standard binary output ROM.

SO

This option causes the assembler to create a source output file corresponding to the input program. The input program may be EDIT - source, compressed, or compressed with updates. The M:SO DCB must have been previousl assigned.

When the assembler is creating source output during an on-line assembly, the file will be written in the keyed, short-record format used by the editor. The new source file will be resequenced according to the line numbers of the assembly listing produced. For example, line number 5 in the listing will correspond to the record with sequence number 5.000.

This feature will allow the user to copy and resequence a source file, obtain the listing and binary (and perhaps even a compressed version for external backup), in one operation with the assembler. For example:

```
! SET M:SO   DC, SOURCEOUT
! SET M:CO   CP
! META SOURCE ON BIN, LP
WITH: SO, CO, CN, SD
≥. END
```

VII. DESIGN AND IMPLEMENTATION CONSIDERATIONS

Many systems like UTS have been designed so that most of the major operations must be carried out within sub-systems; only minor ones and some limited file management and message processing being allowed at the executive level.

A. Organizational Questions

The precise responsibilities of the sub-systems and their hierarchical relationships depend strongly on the programming languages and facilities offered by the system. There are two general approaches:

1. Delineate sub-systems functionally, on the basis of the more common programming activities: editing, compiling, assembling, linking, debugging, file-managing, interactive calculating.

2. Delineate them "linguistically", on the basis of the programming languages to be provided: FORTRAN, XSYMBOL, BASIC, ...; provide within each sub-system the facilities for carrying out the activities associated with the programming language the sub-system handles.

A common attack on the problem consists of: a) constructing the system on a functional basis; b) making the sub-systems explicitly available to the user; c) designing things so that sub-systems can call on the services of other sub-systems. Such systems can be linguistically colored by building the language-specific sub-systems so that they make use of the functional ones. Unfortunately, the architecture of current computers is a bit antithetical to such system designs in terms of "production" efficiency; the systems turn out to be conceptually simple and amenable to growth, but at a serious cost in terms of space and time. An initial design of UTS attempted to strike a compromise by basing itself around what might be called a "portmanteau" sub-system: a shared sub-system for editing, simple file-management, compilation, controlled execution and debugging; one that would be used by all the linguistic sub-systems. Essentially, this sub-system would have been EDIT with sufficient advantages to allow it to become: a) the BASIC sub-system; b) a sub-system for FORTRAN programming (a-la TEL) and debugging (a-la FDP); c) ditto for assembly-language programming; d) the EDIT sub-system.

The idea was attractive on many grounds, but it raised problems of implementation (particularly in the areas of storage management) that could not be resolved in the time allotted. It was therefore discarded by the invocation of Rubric 3 (see Part I, Introduction). An alternative to a single, portmanteau sub-system lies in making separate versions of it, each appropriate for activities associated with a different programming language. This system would still be colored linguistically, in that the casual programmer would simply announce his programming language to UTS and then have at his disposal the facilities appropriate to that language. This alternative is not ruled out by the proposed design, since all major operations will, in fact, be carried out by sub-systems or by sub-processors that behave like sub-systems (see Section C. below). Indeed, the array of facilities provided at the executive level for commonplace FORTRAN and assembly-language programming (ASSEMBLE, COMPILE, LINK, and so on) can be viewed as being incorporated within a hidden sub-system for FORTRAN and assembly-language programming.

B. Sub-System Prompting

Sub-systems designed from scratch for on-line use generally have reasonable rules for interfacing with their users. They usually do so on a "transaction" or "command-response" basis: the user issues a directive, and the system does what has to be done, responds appropriately and then waits for the next directive. Interfacing traditional batch facilities -- and processors designed for batch operations -- with on-line users turns out to be a drag. Almost everything is wrong and hardly anything fits. In batch operations things run either to abortion or to normal termination, so that batch "commands" must be preceded or accompanied by a cloud of pre-planned specifications: to process in this mode or that; to save or not to save; to give up or not to give up, and in what cases; to list or not to list, and where; to list all or part; and so on. Although much of this can be avoided by assuming some reasonable specifications in default of any, provisions must be around for the on-line user to specify and respecify as required. Two tacks can be taken.

1. Specifications can be made on a "transaction" basis. The user states his specifications and choices at any stage of the game. If anything vital is found to be missing during subsequent execution of a command, the system notifies the user so that he may take corrective actions or any other actions. This is really the essential difference between on-line and batch operations: the batcher must pre-plan everything, since he won't be around to take corrective actions or change things; the on-liner need not plan anything, since he can usually recover quickly and at little cost, and can even abort operations that are really messed-up but which would go to completion in batch.

2. Specifications can be made on a "prompting" basis. Before any major operation is started, the system notifies the user that specifications must be given, and then waits for the user to give them (and nothing else). Once given, the process is started and carried out as in batch (to within interruptions): it either aborts or runs to termination.

In addition to being a real mess to implement properly (with all the chit-chat and counseling that may be required), prompting does not allow a user to change his mind or respecify in the middle of an operation to take care of the unexpected. Accordingly, neither TEL nor any of its sub-systems do any prompting except during the log-on procedure and in a few other isolated cases, where prompting takes the form of an error message (as a reminder to take corrective action) or of a request for a negative or affirmative answer to a question.

C. Sub-System Design

To the casual user, it appears that the major operations associated with FORTRAN and assembly-language programming are being carried out directly by TEL. Actually, all operations are being carried out, under the rug, by sub-systems. This activity may be clear for linking and editing, where the TEL and sub-system commands overtly overlap; it is covert in the case of assemblies and compilations, operations that are carried out by processors that are built as sub-systems. The only difference between these and other

sub-systems is that, in the initial version of UTS, the former never interact directly with the user. However, they and most other sub-systems of UTS will be designed so that they recognize two different modes of invokations: one as a sub-system, the other as a sub-processor (usually called by TEL). In the former case, the sub-systems must identify themselves to the user before turning control over to him. In the second case, the sub-systems simply go to work on the TEL command. Several cases arise during processing.

1. The command is garbled or in some manner invalid, or an error -- from which the sub-system allows the user to recover -- occurs while carrying out the command.

2. The command is carried out to completion, although minor error messages may have been spit out by the sub-system during processing.

3. A disastrous error, from which there is no recovery, occurs while carrying out the command.

4. The user interrupts processing by depressing the BREAK key. The actions taken by the sub-system depend on how it was invoked. The responses in each of the four cases follow.

1a) If called as a sub-system, the error is reported and control returned to the user.

1b) Otherwise, the sub-system reports the error, changes its invocation status, and then identifies itself to the user before returning control to him. This behavior may be changed to that of 3a below in some exceptional cases; e.g., if EDIT, say, is used to carry out COPY and DELETE commands.

2a) If called as a sub-system, control is returned to the user after "DONE" has been typed.

2b) Otherwise, control is returned to TEL accompanied by a report of "success".

3a) Whether called as a sub-system or as a sub-processor, the error is reported, things are cleaned-up, and control is returned to TEL accompanied by a report of "failure".

4a) If called as a sub-system, the sub-system will wait for a convenient stopping point, and then type "REVOKED BY INTERRUPT" or some other appropriate message before returning control to the user.

4b) Otherwise, the sub-system preserves context, and returns to TEL, reporting "interrupted".

D. TEL's Implementation

Much of TEL's work, as a processor, is carried out by sub-systems and sub-processors -- all shared; little is permanently resident. Included among TEL's processors are:

a) an ON sub-system (and UTM state) that is invoked at session initiation to handle the log-on dialogue;

b) an OFF sub-system (and UTM state) that is invoked when the user turns off (or re-identifies himself) to clean things up and print accounting information;

c) an ASSIGN processor to handle ASSIGN commands;

d) a RUN processor to supervise the business of linking, loading and initiating execution;

e) an EXEC processor for determining the form of commands, for controlling the execution of commands, for coordinating the uses of sub-systems and sub-processors, and for fielding returns from them.

The organization and residencies of TEL's sub-components will be covered in its implementation specifications. Topics and sub-processors critical to TEL and all of UTS, that will be covered in separate documents are: a) virtual memory layouts and storage management; b) users' general context areas; c) interfaces with the monitor (UTM); d) COC handler for communicating with remote consoles; e) sub-system conventions. The design of TEL's executive and control processor and its interfaces with sub-systems proceeds in tandem with the first three topics, both depending on them and influencing them.