MAC-TR-16

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

CTSS TECHNICAL NOTES

by

J.H. Saltzer

MAC-TR-16

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

CTSS TECHNICAL NOTES

by

J.H. Saltzer

ABSTRACT

This report is a technical description of the 7094 Compatible Time Sharing System in use at Project MAC and the M.I.T. Computation Center. It is designed to acquaint a system programmer with the techniques of construction which were used in this particular time-sharing system. Separate chapters discuss the overall supervisor program flow; console message input and output; the scheduling and storage algorithms; and a thumbnail sketch is given of each of the subroutines which make up the supervisor program.

This report was prepared with the aid of the compatible time-sharing system and the TYPSET and RUNOFF commands.

"Work reported herein was supported (in part) by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government."

Preface

The writer intends these notes to provide a technical introduction to the operation of the 7094 Compatible Time Sharing System for a user who wishes to participate in programming and related development of the time-sharing system. In their present rough form, the notes attempt to fill as quickly as possible a need for tutorial documentation of the time-sharing system.

The reader should have considerable experience in computer programming, including a knowledge of the machine language (FAP) of the 7094 computer. He should also be at least a casual user of the time-sharing system and thus familiar with the operating characteristics of the system, and he should be familiar with the system description provided in the CTSS users' manual (1). However, when highly technical aspects of the 7094 operation or special features of the time-sharing system are discussed, the notes will provide enough background material for a reader familiar with these subjects.

Note: In the interest of getting into distribution a maximum amount of information in a minimum amount of time, sections 5,6, and 7 of the Technical Notes consist mostly of tables and charts, with a minimum of verbal description. They should provide a useful reference source, though they are not ideal for tutorial purposes.

---

(1) F.J. Corbató, et al: The Compatible Time-Sharing System: A Programmer's Guide, M.I.T. Press, Cambridge, Mass., 1963.

CTSS Technical Notes

Table of Contents

1. Introduction to the Technical Aspects of CTSS.

In this section we will review several ideas which are covered in the CTSS Programmer's Guide, but from the point of view of a system programmer rather than that of a user of the system. We will discuss, in turn, the computer on which the system operates; the overall design principles; the place of the disk and drum memories in the system; the relation between the user and the supervisor; the types of commands; and finally, the modular construction of the system supervisor.

The Computer.

While many of the ideas involved in the time-sharing system are to some extent independent of the computer which the system uses, a technical discussion presently requires a specific reference to details of the particular computer. The computer in these notes is the IBM 7094 with several special features. The most important are;

1. Core storage interval timer clock. This includes an "alarm clock" which can cause a program interruption similar to a data channel trap.

2. Memory protection and relocation registers. These permit a section of the computer memory and certain instructions to be declared "off limits" to a program; a trap will occur whenever a program attempts to tread "off limits."

3. At least five data channels, connected to the following equipment:

   Channel A: Printer, punch, reader, tapes, and Chronolog clock.

   Channel B: Tapes.

   Channel C: Disk and Drum Storage.

   Channel D: Direct data input and output for special experiments.

   Channel E: 7750 Communications Channel (which communicates with typewriter consoles).

   The Project MAC Computer has two additional channels:

   Channel F: Disk and Drum Storage.

Channel G: High Speed Drum.

4. Two 32K core storage modules.

These special features are combined with an extensive supervisor program, known as the "time-sharing system supervisor" to provide a complete time-sharing system.

Design Principles.

It will be somewhat easier to understand the general import of the ideas to be presented in these notes if some of the principles of design of the time-sharing system supervisor are stated clearly at this point.

1. Although subsidiary computers (the IBM 7750) are an integral part of the system, as many functions as possible are carried out by the 7094. This centralization of supervisor control is primarily to simplify the job of a person trying to learn how the system operates.

2. The system is designed for the maximum possible interaction rate with the user: the 7094 accepts each character as it is typed by each user. It is not necessary for the user to communicate on a line-by-line or paragraph-by-paragraph basis with his program, although many programs do not attempt to take advantage of the maximum interaction rate.

3. Input and output from user programs are provided by supervisor subroutines which allow the user to specify by name the function he desires without irrelevant detail and bookkeeping on his part. For example, if he wishes to store some information on the disk he gives the supervisor the block of information and a name by which he knows the block; he can retrieve it later by asking for it by name. He need never worry about disk track numbers or complicated data channel programs. Similarly, the mechanics of communication with a typewriter console via telephone lines would deter even the most experienced programmer; but a subroutine call is provided which accepts a message and the information that it is to be typed on a console; the supervisor subroutine takes care of the rest of the details. This principle is carried to the point that the user is required to do all his I/Ø via supervisor subroutine calls.

4. The supervisor is designed to be context-free. That is, although it accepts commands from a user, it has no direct interest in what the commands do; it considers a command name to be merely the name of a program to be found in a directory; when the program has been found it is loaded and started just like a program written by the user.

Use of Disk and Drum Storage.

As mentioned above, input and output are handled by supervisor subroutines for the user. This comment applies especially to the disk file and the magnetic drum memory. A special supervisor module, the disk control, handles all input and output for these units. The disk control module has been designed so that the user cannot distinguish between the disk and drum memories. There are four primary uses of the disk files:

1. User files. These are files of information which a user wishes to store away for future reference. They may consist of programs, data for programs, or any other information the user desires. They are kept on the disk indefinitely and allow a user to retrieve a program several weeks after he wrote it. Thus, the disk replaces the decks of cards and reels of magnetic tape usually associated with a large computer installation.

2. Working programs. When a program actually works, it shares the computer with several other users. Since not all of these users will fit into core memory at once, the excess are stored temporarily on the disk or drum to be brought into the computer when their turn to use the central processor comes up again.

3. Supervisor commands. Whenever a user types a supervisor command, he implicitly requests execution of a program. In most cases, this command program is kept on the disk.

4. Scratch Pad memory. Many programs, such as translators, require large blocks of temporary storage which do not fit into core memory. The disk is also used to fulfill this need.

All files kept on the disk (and drum) are known to the user only by name: the supervisor disk control module keeps for each user a directory of names and corresponding track locations on the disk. A simple chaining procedure is used to locate any given file.

A master file directory, which contains the identification of all potential users of the system, starts at a fixed place on the disk and contains all pertinent information about the user, including the location of his personal file directory. His file directory lists names and starting locations on the disk of each of his personal files. Every user of the disk control module, including even the supervisor, must appear in the master file directory. As will be seen later, the supervisor modules are also stored in relocatable binary form as files on the

disk under the user number of a special BSS loader which starts up the system.

Relation Between User, User Program, and Supervisor.

For every possible user of the system, there is an entry in the Master File Directory of pertinent information about his identification and use privileges. There is no information, however, about the nature of the console he might use; this information is not of interest to the supervisor and only concerns the user (and possibly his program).

When a user logs in to the system, he is assigned a user number and thereby becomes subject to the attention of the scheduling algorithm. A logged-in user is assigned a "state" according to the demands he is placing on the system; this "state" will usually change several times while he is using the system. Each state has an associated code number:

0. "Dead". This state corresponds to a user without a program. He may be in this state because he has just logged in and has not yet loaded a program, or because his program has just completed execution and returned him to the dead state. In all states but this one (and sometimes state 3) there exists a "core image" of a program for this user.

1. "Dormant". A user is in the dormant state when he has a program which is potentially runnable, but which for some reason the user does not want to execute at the moment. His program is probably not being kept in core memory but temporarily on the disk file, until he gives the word for it to begin execution. A user will be in this state if he has just loaded a program but has not yet started it; or if he has just finished program execution and returned to the dormant state. This latter possibility differs from the similar situation described under "dead" in that the core image of the program remains available either for rerunning or for postmortems.

2. "Working". A user is in this state whenever his program is scheduled for execution. Working users actively share the use of the computer, but only one of them is actually in execution at any given instant, while the others may be in core or stored as core images on the disk just like the dormant users. All of those trying to execute are considered to be in the working state.

3. "Waiting Command". If a user is in either the Dead or Dormant state, anything he types is considered a command to the supervisor. When he finishes typing a command, the supervisor places him in state 3 to indicate that he should be actively considered by the scheduling algorithm. When his turn comes to use the computer, the corresponding command program is located, loaded if necessary, and his state is then changed to "working". Note that a command program is considered to be the user's program; once it is loaded it is indistinguishable from one of the user's own programs.

4. "Input Wait". When a working user program attempts to read a line of input from the console typewriter, there is a good chance that the user (typist) has not yet finished typing the line. In this case, the user is assigned state 4, and he is temporarily ignored by the scheduling algorithm until such time as the needed line has arrived. Although in principle a request for input from any device could result in the input wait state, this state does not apply during the reading of a file of information from the disk. The rate of information transfer in this case is so high that more time would be lost switching users than waiting for the first user's input to arrive.

5. "Output Wait". When a user's program attempts to type out a series of messages on the typewriter console, messages may be produced at a higher rate than the console can type. After a few such messages have been absorbed by intermediate buffers, the user is placed in state 5 until the messages clear sufficiently to permit the program to proceed. The output wait state could also apply to any output device requested by the user, but considerations similar to those described under input again apply.

One further noteworthy detail of the present implementation makes clear future remarks. The two memory banks of the computer will be referred to as "memory A" and "memory B". The supervisor is in memory A, and user programs are in memory B. It is not essential that such a division of equipment take place, but a formalized division greatly simplifies the programming within the supervisor.

Supervisor Commands.

A command program is nothing more or less than a complete program previously written by someone, which is stored in the form of a core image ready to load and run. A number of those command programs are considered "public commands" and are stored in the supervisor's disk files.

Other, private command programs may be stored in a user's private file. Examples of public command programs are the MAD and FAP translator programs. An important aspect of the way in which the supervisor handles commands (requests for the execution of a command program) is that the supervisor remains aloof, as it were, from the operation of the command program.

Suppose a user types the command MAD with appropriate arguments. The supervisor accepts this input line after checking that it is in fact a command. When it is the user's turn to run, the supervisor looks up the command name in a command directory which contains BCD command program names and command program starting locations. If the command named MAD is found in the directory, the file named "MAD TSSDC." is read from the disk into core B and started at the location given in the directory. This command program is now the user's program and runs exactly as if he had written and loaded it himself. Note, however, that the supervisor itself has no information about the command or what it does, except its name and starting location. It is in this sense that the supervisor is context-free.

There are, in fact, three kinds of commands; we have just described the operation of "disk-loaded" commands. A second type of command, also context-free, is the B-core transfer command. If the user types a B-core transfer command and is in a dormant state (that is, has a B-core image), his core image is loaded and the transfer is made to a special place in his program (again given in the supervisor's command directory). Examples of B-core transfer commands are PM and FAPDBG. These functions are carried out by subprograms which are loaded as part of the user's program. The third command type is the A-core transfer command, which requests an action intimately connected with the supervisor and is thus not strictly context-free. Examples of this type are LØGØUT, LØGIN, and SAVE. Here no loading is needed, since the command programs are built into the A-core supervisor.

The Modular Time Sharing System.

The supervisor program has been written in the form of modules, for ease in understanding and modification. Each module of the supervisor takes charge of a specific function, such as typewriter coordination, disk file input and output, or scheduling of user programs. The modules are written in languages which produce binary programs in BSS relocatable form; thus the modules may communicate with each other only through the two standard communication procedures of BSS programs, namely the transfer vector and program common. Thus the number of interconnections between separate parts of the supervisor is minimized and such

interconnections can only exist on a formal, advertised basis. Most importantly, the reader can study and understand the operation of a specific module while he still has but a vague idea of what happens inside other modules. Figure 1.1 is a schematic illustration of a layout in core memory of the modules of the supervisor program, and the user programs. The diagram shows only a few of the more important modules.

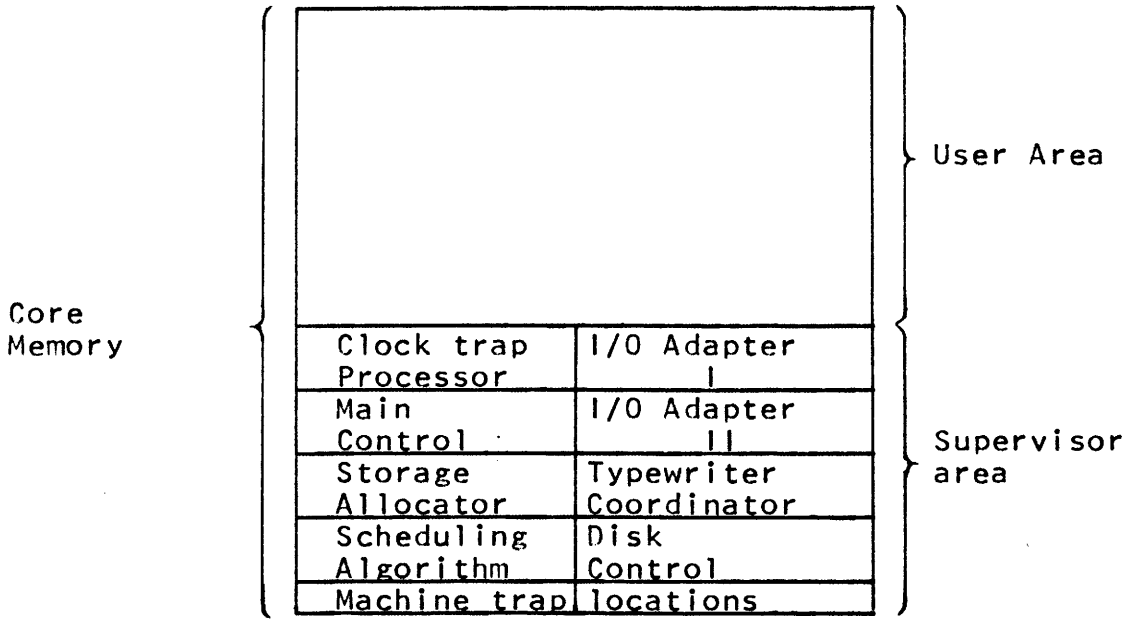


Figure 1.1 -- Possible core Memory Layout.

While the system is operating, the modules of the supervisor are in the form of a complete program loaded and linked together in core memory. However, copies of each module in BSS form are stored in the disk file. In addition to the modules normally used in the system there may be newer modules being debugged, or older modules kept as backup in case the presently used version suddenly develops an unforeseen bug. The system is started up by placing a modified BSS loader into core memory and giving this loader a special control card which specifies a list of modules to be loaded and linked together. The steps involved in starting the system are as follows: first, a 7094 program loads the disk from a tape copy made of the disk following the latest previous use of the time-sharing system; then, another 7094 program loads and starts operating the 7750 computer, which handles console typewriter input and output. Now, a special BSS loader which contains a copy of the disk

control module is loaded, from a card reader or tape, into the 7094. The loader is followed by a "supervisor name card." The BSS loader has a user number, and thus has access to the disk files with the aid of its disk control module. The "supervisor name card" specifies the name of a disk file accessible to the BSS loader which contains a list of names of the disk files to be loaded to form a supervisor program.

There are two interesting aspects of this procedure for loading the time-sharing system supervisor. First, maximum use is made of existing programs, such as the disk control module. Second, if a module being checked out develops a bug, it can be very easily removed from the system (and an older, reliable version substituted) with a minimum of fuss and bother, simply by a reloading of the special BSS loader with a supervisor name card which specifies a different list of disk files to be loaded to produce the supervisor. This feature is vital in a system which is in use while still in a state of development.

Epilogue.

In conclusion, a picture of the magnitude of this undertaking, in terms of relative size of the programs involved, may be of interest. The supervisor program consists of about 12,000 (decimal) instructions plus tables. This figure compares with 11,000 for the MAD compiler, 16,000 for the FAP assembler, and some 60,000 instructions in the older IBM Fortran II compiler. Thus in the proper perspective, the time-sharing system supervisor is an undertaking comparable to the development of a completely new compiler system. In addition to the FAP and MAD translators, command programs totaling another 6,000 instructions are necessary to frame out a "usable" time sharing system.

2. Construction of the Supervisor Program

Introduction.

In this section we examine the general flow of control within the supervisor program and consider when and how it obtains control, and what happens when it does. In doing so, we will get a slightly closer, but not detailed view of several of the important supervisor modules.

Supervisor Program Flow.

Suppose a user's program is operating. The program is located in core B, and has control of the computer; the supervisor is located in core A but is not presently in operation. There are three events which can cause the control of the computer to transfer to the supervisor, as indicated in figure 2.1.

First, if any user at any typewriter types a character, he causes a data channel trap at the 7094. Control passes to one of several special supervisor modules called Input-Output Adapters. The appropriate Adapter accepts the character, performs any necessary code conversions on it, and places it into a common input pool buffer along with the typist's "user number". Control then returns to the interrupted program which continues as if nothing had happened. A data channel trap is a true "interruption" as it may occur at any point in the user's program.

A second event which gives control to the supervisor is the following: after a period of time known as a "clock burst," and typically of value 0.2 seconds, a clock trap occurs, which passes control of the computer to the supervisor clock trap processor. At this time the supervisor does most of its "housekeeping" work. The typewriter coordinator processes input and output between user programs and typewriter consoles. The supervisor examines commands typed by other users and makes notes. Finally, the supervisor consults the scheduling algorithm module to learn whether or not this user should be permitted another "clock burst" of running time. If he is allowed to continue running, his program is restarted at the point at which it was interrupted by the clock trap; if not, another user is allowed to run, and a "swap" may have to take place.

We have thus far looked at two ways in which control may pass from the user's program to the supervisor. Both of these traps, data channel and clock, have the property that they may occur at any point in the user's program. The third event which causes control to pass to the supervisor however, is completely under the user's control. This event

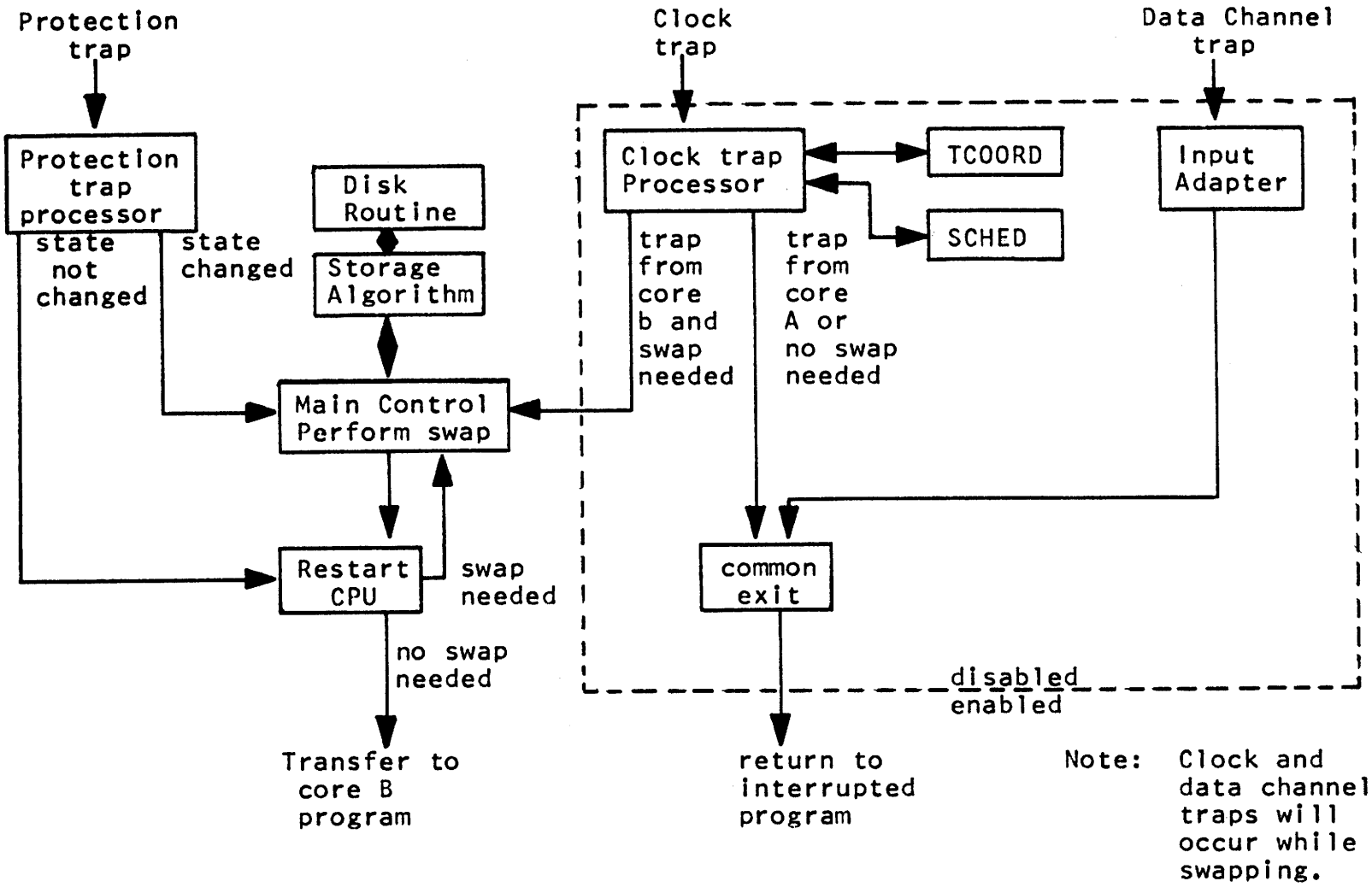


Figure 2.1 -- CTSS supervisor, overall flow.

is the (presumably) intentional protection mode violation; the signal that the user is calling on the supervisor to perform some special subroutine function. When such a subroutine call occurs, of course, the operation of the supervisor depends on exactly which subroutine has been requested. In general, however, if the user's state has not changed as a result of the subroutine call, control returns to him directly as soon as the subroutine operation has finished, unless he used up his "clock burst" during the operation of the subroutine. If the user's state has changed as a result of a subroutine call, for example, a call which requests a change from working to dormant status, control passes instead to that portion of the supervisor concerned with locating and running another user program.

Data Channel and Clock Traps.

For the interpretation of the flow diagram in figure 2.1, the operation of the data channel and clock traps must be understood. These two traps may be either enabled or disabled. In the supervisor, they are almost always enabled and disabled together. If a trap is enabled, the program in operation, whether user or supervisor, may be interrupted at any time; the program has no control over interruptions except to disable the traps. On the other hand if the traps are disabled, when a trapping condition occurs the program is not interrupted; instead the trap is remembered until such time as the traps are re-enabled (restored). The ability to disable traps, yet remember them, is necessary in order that the supervisor may handle all traps in an orderly manner.

The dotted boundary in figure 2.1 is the disable-enable boundary; all programs inside the boundary run with data channel and clock traps disabled, those modules outside the boundary run with data channel and clock traps enabled. Thus a data channel trap cannot occur while in the scheduling module but may occur while in the disk control module.

When a clock or data channel trap occurs, further clock and data channel traps are immediately disabled. The supervisor continues to run with traps disabled until it either returns to the interrupted program or it goes to the sway (main control) section.

Note that care must be taken to insure that an enabled supervisor subprogram is never entered following a trap from the very same program. To make sure that this does not happen, the clock trap processor never goes to the swap section if a trap has come from core A. Instead, if a swap is needed, only a switch is set, and return is made directly to the point of interruption of the core-A subroutine. When

the subroutine has finished, it returns to the user program via the common user return RSTCPU. Since the user may have run out of time during the subroutine operation, RSTCPU checks the swap switch and if a swap is needed, performs the transfer to the swap section which the clock trap processor was afraid to do before.

System Modules.

After the brief consideration of a general picture of the supervisor operation, it may be useful here to list all of the modules with a brief sketch of their diverse purposes. The block diagram, figure 2.2, shows the general relationships between the various modules, although flow of control between modules is not unambiguously indicated.

Each module is a subroutine, or group of related subroutines, filed by a six-character primary name and a secondary name corresponding to the language in which the module is written. The first four characters of the primary name are mnemonically related to the function of the module and the last two characters are a number indicating the version of the module. Some modules, because of their size, are split between two or more files. Such a split may or may not imply separateness of functions of the parts of the module. The names of the modules and their functions in the version "1A1" system are:

Primary File Name	Function
MAIN	Initialize time-sharing supervisor.
CLØC	Clock trap processor.
CTRL	Main Control Section.
STØR	Storage algorithm.
SCDA, SCDB, SCDC, SCDD, SCDE, SCDF, SCDG	Scheduling algorithm.*
TCØR	Typewriter coordinator.
PMTA	Protection mode violation processor.

RTRN	Common exit routines to return to trapped programs.
SAVR	Save and restore user machine conditions.
UTRP	Process STR, floating point, and data channel traps for user programs.
RFLX	Processes user input lines.
CØMC	Miscellaneous subroutines.
CØMD	Command directory (no instructions).
CØNV	BCD conversion routines.
ØNLN	Do on-line I/Ø for supervisor.
EDBG	Post mortem and trace routines for debugging the supervisor.
LØGA LØGB LØGC	LØGIN, LØGØUT commands.*
SAVC ØCTC	Start, save, restor, resume commands. ØCTLK, ØCTPAT, and ØCTTRA commands.
DSKI	Disk control.
ADPI	7750 I/O adapter.
AP75	7750 Write subroutines.
TSTØ	Assign 7750 storage.*
CHNE	Channel E hardware subroutines.
HIGH	High speed line adapter.
UNIT	Assign and look up logical user numbers.
DCER	Handles channel E errors.
PLØTS or KLUD	Channel D I/O adapter.

* Indicates a module written in the MAD language. The other modules are written in FAP.

NOTE: More detail about the entry points of each module is contained in Chapter 7.

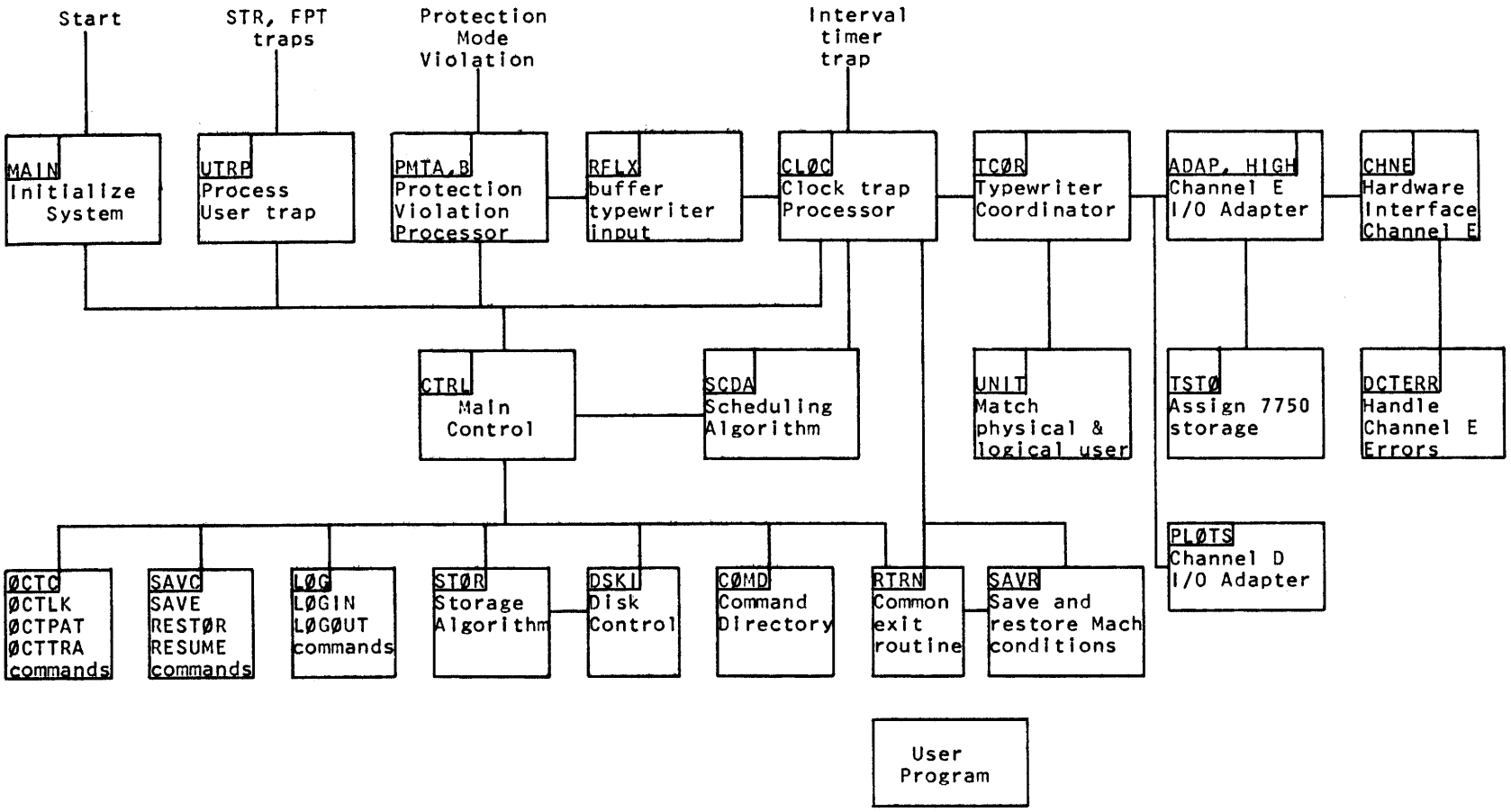


Figure 2.2 -- Block diagram of supervisor showing important inter-module links.

3. Input and Output

Introduction.

In this section we will study the communication between the user and his program in the time-sharing system. We will discuss specifically how a typewriter communicates with the system, although the ideas can easily be extended to more exotic forms of input and output devices

To allow a continuous flow of input or output between the typewriter and the user's program, which may not be in core memory at all times, the supervisor provides buffers for the data being transmitted. Input messages are buffered in core A, within the supervisor, while output messages are buffered in the 7750 computer.

The General Logic of Input Flow.

Figure 3.1 is a flow diagram of the handling of input from a typewriter by the time-sharing system. We may begin with a user typing a character on his typewriter. This character travels via telephone lines to the 7750 computer. The 7750 accepts the character, and turns on the 7909 data channel. The 7909 data channel places the input character in a buffer in the 7094 memory and causes a 7094 data channel trap to the appropriate 7094 input adapter. The input adapter program moves the character into a character pool buffer, in the form of a word containing the character in the address and logical user number in the decrement. This format, used for all character-oriented devices, is known as "Interface I". The character pool buffer is capable of holding about 600 such characters (for 30 users). The 7094 then returns to whatever business it was about when the trap occured.

The characters in the character pool buffer are thus left for a later section of the supervisor to examine and eventually route to the proper destination. It is fundamental that the 7094 computer responds to input character by character so that if a user program desires, it can communicate back and forth on a character by character basis with the user.

Further processing of input resumes when a clock trap occurs, giving the supervisor program intentional, complete control of the computer. At this time the input characters are processed in two stages. The first stage, handled by the typewriter coordinator module, collects characters into messages. Each user has a separate secondary read buffer, and all characters which he types are moved to his secondary read buffer by the typewriter coordinator. No further

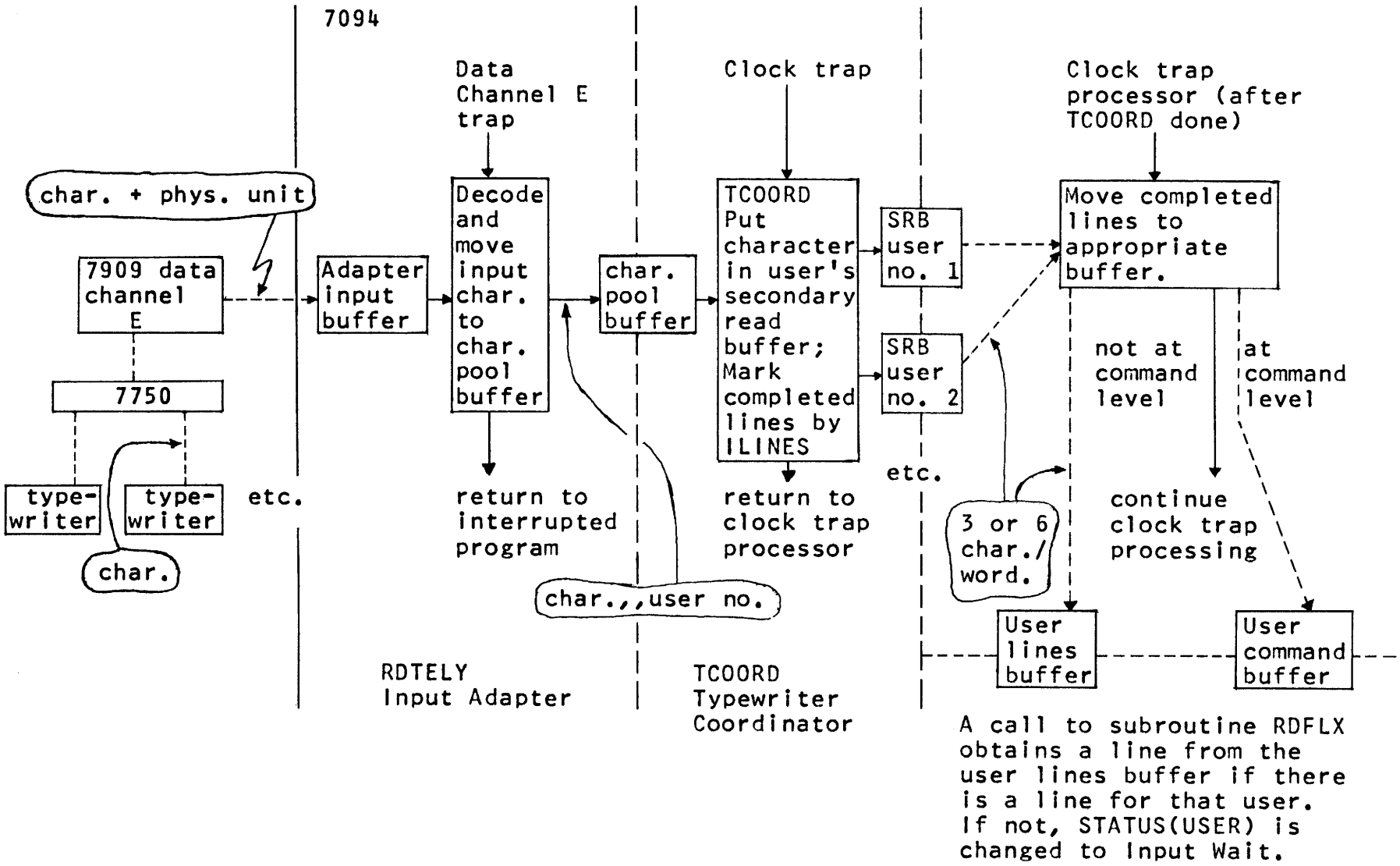


Figure 3.1 -- Simplified Input Flow.

action is taken unless one of the characters typed by the user is found to be a "break" character such as a carriage return or other special character designated by his program as a break character. When a break character is found among the input characters his message is considered to be complete, and no more characters are placed in his secondary read buffer. If more characters remain in the character pool buffer for this user, they are left there until his secondary read buffer is free. A program switch is set to indicate that this user has completed typing a message. The typewriter coordinator attempts to thus dispose of all of the characters in the character pool buffer by distributing them to user secondary read buffers.

When the typewriter coordinator is finished moving characters into the secondary read buffers, the second stage of input processing begins. At this stage, handled by the clock trap processor module, completed messages are delivered to their final recipients' "mailboxes". If the user is at command level (dead or dormant) the message is a command and is placed in the user's command buffer. If the user is not at command level, the message is for his program, and so it is moved to the "user lines" buffer. There the message remains until his program calls upon the supervisor subprogram RDFLX for final delivery.

Note that the second stage, which removes the message from the secondary read buffer, is not strictly necessary; the user's secondary read buffer could be considered as his mailbox. The second stage could consist only of noting completed messages from users at command level. A floating buffer scheme to do this simplification could easily be implemented.

An important reason for the modular construction of the system supervisor is well illustrated by the input scheme. There may be any number of input adapter modules communicating with character-oriented devices. Each of these modules, upon obtaining control from a data channel trap caused by its hardware input device, can place characters in the character pool buffer in the standard Interface I format. The processing done by the typewriter coordinator is completely independent of the source of the character in the character pool buffer. The adapters can be considered as "matching devices" between the specialized hardware requirements of different input devices and the standardized characteristics of the input interface of the typewriter coordinator.

The General Logic of Output Flow.

Having looked briefly at the input side of communication between user and user program, we will postpone detailed discussion of the input modules until we

have surveyed the related output scheme. Output is handled in a way quite similar to input but with some necessary differences. The major difference is that a program can produce output at a very high rate, while a typist can produce input only much more slowly. On the other hand, the output processor can easily turn a program off if too much output comes out in too short a time; the input processor cannot turn off a typist without annoying him and perhaps losing some of his typed characters.

Thus one can make sure during input that the system keeps up with the typists by making the character pool buffer large enough to accept the maximum number of characters that all the typists could produce in, say, two seconds. Then, if the typewriter coordinator attempts to empty the character pool buffer at least once per second, only very rarely will the buffer overflow and the typists told to desist.

In contrast, the output sections of the supervisor must constantly expect to be overburdened with output lines produced by programs which run faster than the typewriters. The supervisor handles the problem very simply by refusing to accept output from a program unless there is room for the data in the output buffers. If the buffers are full, the user's program is placed in "output wait" status. The user program does not return to working status until buffer space is available for the program's output.

With these considerations in mind, we can now look at figure 3.2, a flow diagram of output processing. Output originates when the user's program calls for the supervisor sub-program WRFLX or WRFLXA. The user's message is converted to 12-bit form if it is not already in that form (that is, if the user is in the normal, 6-bit mode) and moved into the primary write buffer. This buffer has room for 29 words; since three 12-bit characters are stored in a word, there is room for a "line" of 84 characters plus a carriage return.

Subroutine WRFLX now calls the output adapter module at the entry point WRTELY to write the information out on the 7750 computer. The output adapter may perform an error return, however, to indicate that the 7750 does not have room for the message. If this is the case, WRFLX itself performs an error return which places the user in output wait status. When space becomes available, the 7750 will send a completion signal back to the 7094, and this user will come back to working status. As soon as the user begins executing again, subroutine WRFLX starts over from the beginning, moving the message into the primary write buffer (since the buffer may have been used by someone else while the first user was in output wait).

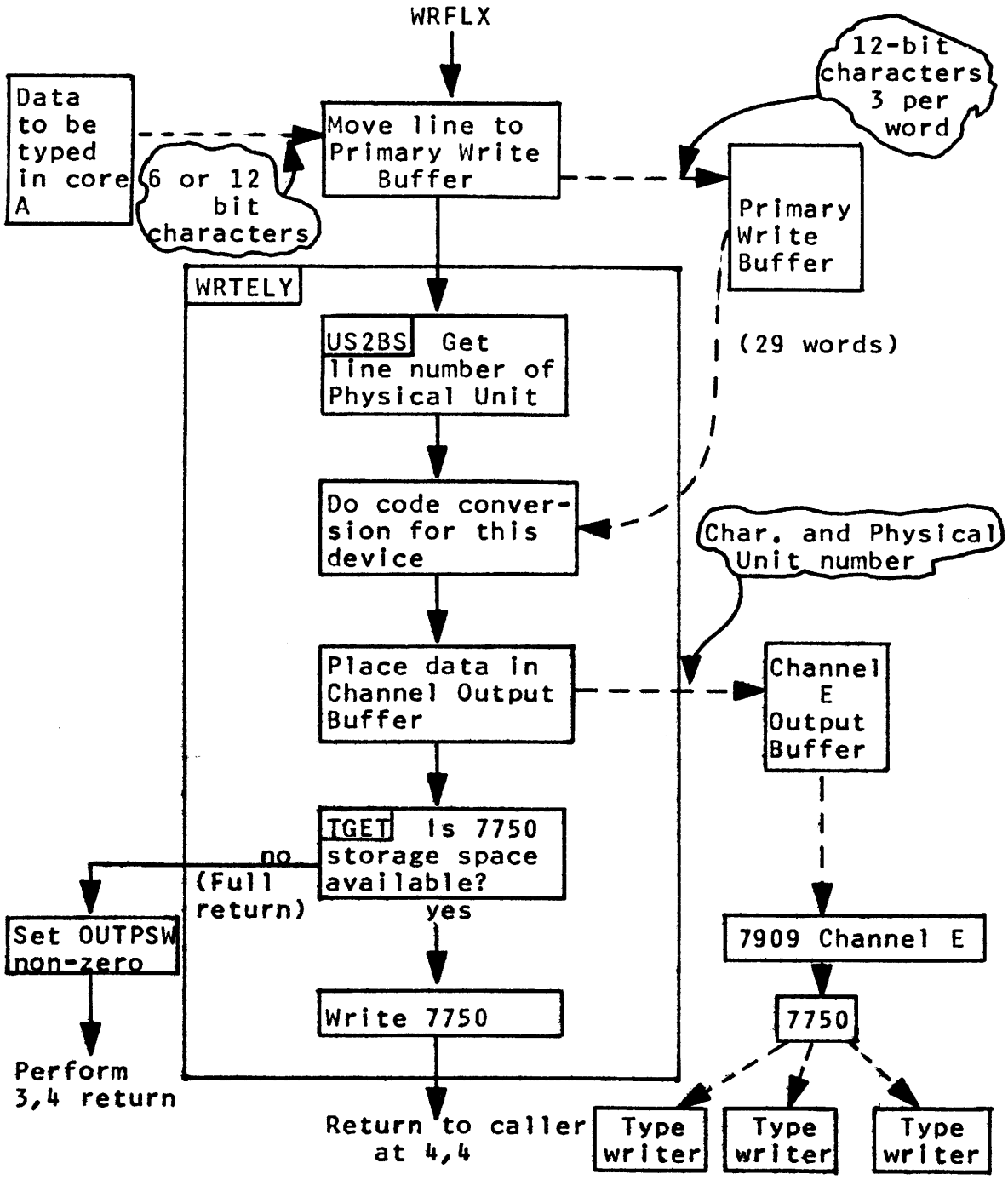


Figure 3.2 -- Output flow. (TCOORD and WRTELY)

Assuming, however, that the 7750 has room for the message, the output adapter encodes the message and delivers it to the 7750. A detailed description of the output adapter, and of the criterion used to determine whether or not the 7750 has "room" for a message is the subject of the next section.

WRFLX also does some processing of the output line. It locates the last non-blank character in the line and inserts a carriage return character after it, while it deletes the trailing blanks. For those applications where this processing is not desired, an alternate entry, WRFLXA is provided which puts out the line exactly as given.

Since calls to WRFLX and WRFLXA always must specify an integral number of words, they also always specify a multiple of 3 or 6 characters, depending on the status of FULSW. In those cases where a different number of characters is desired, the null character, 57 (octal) may be inserted to fill out the last word in the block. The typewriter coordinator will ignore null characters found in the secondary write buffer.

Note that the only communication between the input-output adapter (the two functions are really handled by a single module) and the rest of the supervisor is via the primary read buffer (Interface 1) and the subroutine-type call to the output adapter. The resulting independence makes it very easy to remove one I/O adapter program and insert another for a different class of input-output devices.

An I/O Adapter Module.

We are now familiar enough with the general logic of input and output to study in detail the modules which perform it. We start with an I/O adapter module, but remember that this is only a description of a typical adapter module and that any other program with similar characteristics with respect to the primary input and output buffers can, and occasionally does, replace the particular one we are studying.

The I/O adapter module is of course split into two quite independent parts, one handling input and the other output. Let us consider the output section first, as illustrated in figure 3.2. The output adapter performs the necessary code conversion for the user's particular device (teletype, 1050, flexowriter, etc.) and places the data in the proper format for the 7750, one character per word, with the user's telephone line number in the decrement. The supervisor module UNIT maintains a table of correspondence between actual user, as identified by telephone line numbers, and internal logical user numbers. Each user, as he dials into the system, is assigned a logical user number for easy identification. The adapter must then establish whether or not there is room for the message in the 7750 buffer area. A separate module, TSTØ, keeps track of how much space is available in the 7750, and this module also decides the policy of who should be allowed how much space

there. There is room for 10,000 characters in the 7750 buffer, and the amount which any user may have is known as his allotment, ALØT. If N characters are actually in use at the time a user asks for output space, his allotment is calculated as

$$ALØT = (10000 - N)/4$$

If the total number of his characters in the 7750 will not exceed ALØT, he is allowed to perform his output. If he will exceed ALØT, an error return is given to indicate that he should go into output wait.

Consider now the input adapter module, figure 3.3. In this case, control comes to the input half of the module via a data channel trap; there is at least one character in the adapter's input buffer. The input adapter picks up the character, converts it from the 7750 format to Interface I format for the character pool buffer and replaces the user telephone line number with his internal logical user number. It then checks to see if this character is really a completion signal from the 7750 saying that a 31 character buffer has been typed out on this user's typewriter. If it is a completion signal, the adapter calls TSTØ (at entry point TGIVE) to tell it to release a block of 31 characters assigned to this user. All other characters are placed into the character pool buffer for later processing by calling entry point TØPØØL in the typewriter coordinator. The input adapter restarts the 7909 data channel program, and returns via the common exit module to the program that was interrupted by the data channel trap.

The Typewriter Coordinator Module.

Figure 3.4 is a flow diagram of the typewriter coordinator program. As indicated, the coordinator only handles input processing. Actually, WRFLX and WRFLXA, described previously, are written as part of the typewriter coordinator module.

The typewriter coordinator is called as a subroutine once every time a clock trap occurs, by the clock trap processor. Its purpose, you may recall, is to collect characters from the character pool buffer into messages in individual secondary read buffers. The coordinator begins by examining the characters in the character pool buffer at one time. Let us follow the path of processing of a single character. First, the character is checked to see if it is one of the characters in the break character list. If it is one of the three special "quit-class" characters, (quit, interrupt, or data-phone hang-up) this character by itself is considered to be a complete message to the supervisor, and the ILINES table is set to indicate that there is a

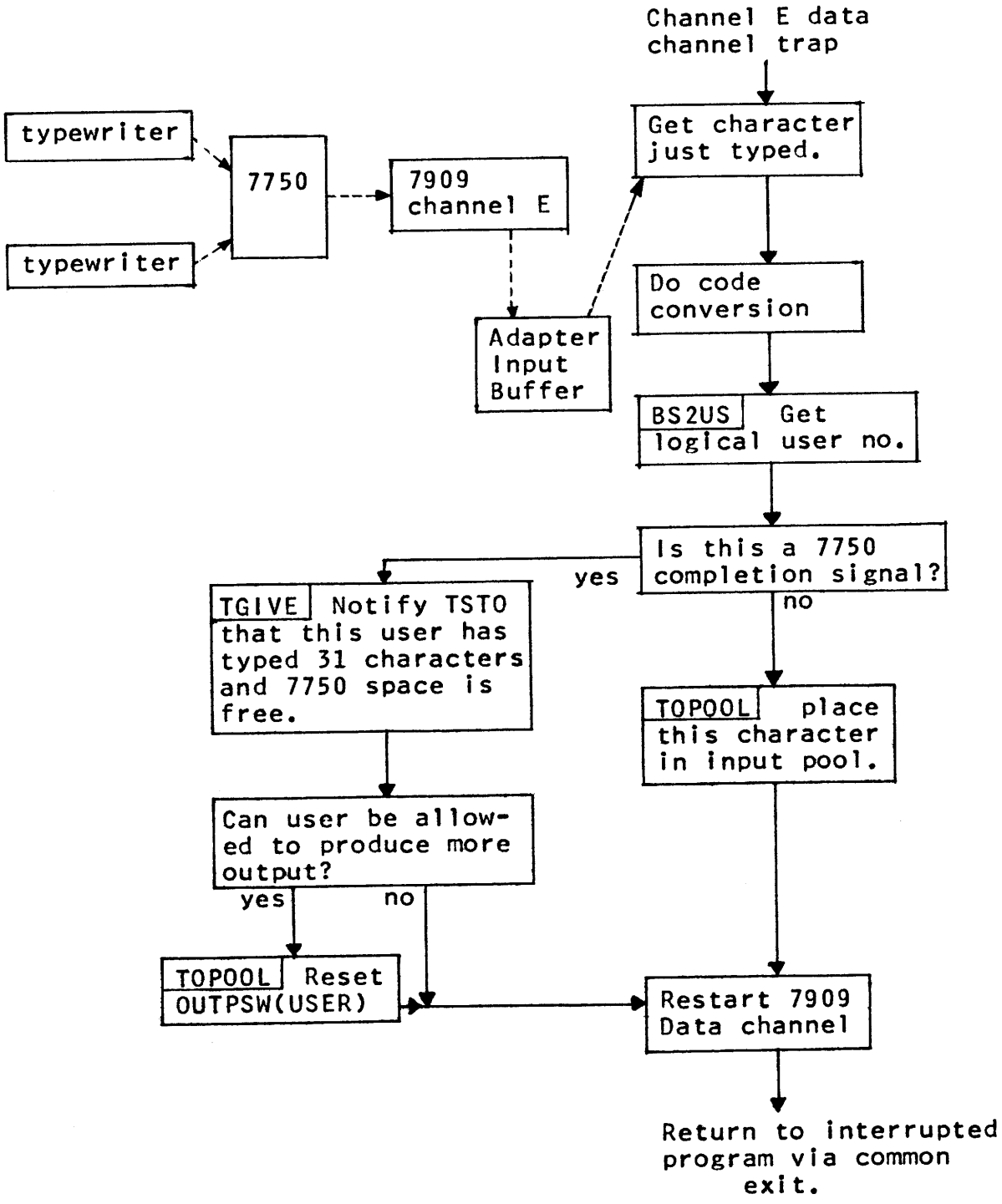


Figure 3.3 -- Input Adapter Flow Diagram.

waiting message from this user. To alert the supervisor that this is a special message, the prefix of ILINES(USER) is set to MZE and the quit-class character is placed in the address of ILINES(USER). If the character is not a break

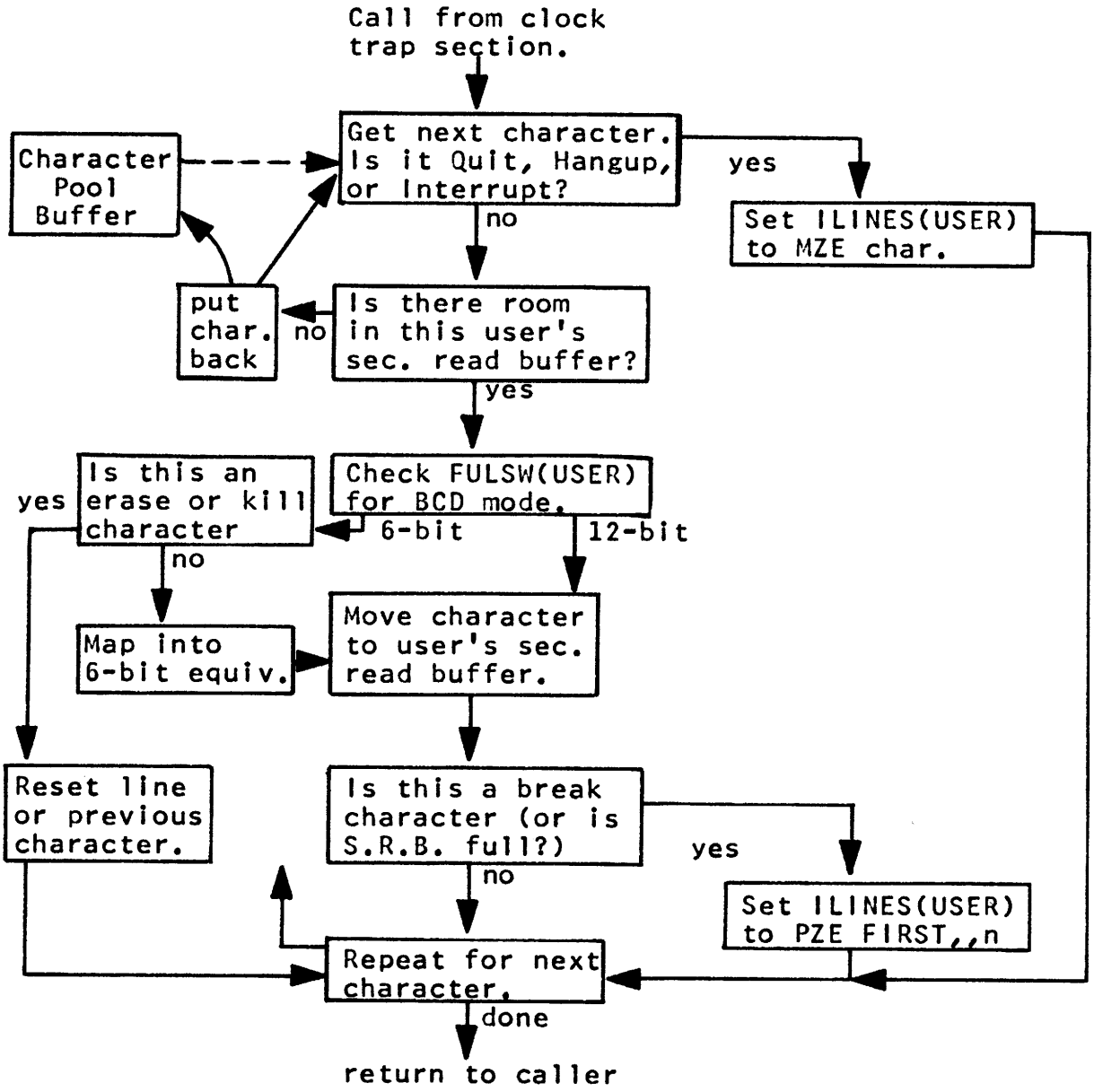


Figure 3.4 -- Flow diagram of Typewriter Coordinator.

---

character it will have to be moved into the user's secondary read buffer, so the program then checks to see if there is any room left in the secondary read buffer. If the buffer is full, the variable ILINES(USER) will contain some non-zero buffer address; this is an indication to the coordinator program not to attempt to use the secondary read buffer. Instead, this character is put back into the character pool buffer.

Assuming that all these tests are passed, the program then checks the variable FULSW(USER) to determine whether or

not the user's program is using a full (12-bit BCD) mode. If the user is using the ordinary 6-bit mode, the 12-bit character coming from the typewriter will have to be translated. This translation includes two important features. First, if the character is either the "delete line" or "delete preceding character," the line, or last character, in the user's secondary read buffer is discarded. Secondly, on all other characters a mapping is performed, when possible, from the 12-bit character to one of the allowed 6-bit BCD characters. For example, a small letter "a" and a capital letter "A" can both be mapped into the BCD letter "A" with octal code 21; however, certain special characters such as the semicolon have no possible mapping into 6-bit codes. If these non-mappable characters are encountered, they are discarded at this point.

Having performed a 12-to-6 bit conversion when necessary, all characters other than quit-class break characters are stored in the user's secondary read buffer, packed either 3 or 6 characters per word, depending on whether the user is using 12-bit or 6-bit mode. The final check is to see either if the character is an ordinary break (end-of-message) character or if it filled up the secondary read buffer. If either case is true, the variable ILINES(USER) is set to contain "PZE FIRST,,n" where FIRST is the address of the secondary read buffer, and "n" is the number of words in the buffer. This is the indication to the supervisor that this user has a complete, waiting input line.

This entire processing operation is repeated once for each character found in the character pool buffer. We have not discussed here the inter-console communication facilities provided by the ADØPT feature of the typewriter coordinator.

Typical buffer sizes used by the typewriter coordinator are:

Primary read (Character Pool) buffer:	600 characters (for 30 users).
Secondary read buffer; 2 per user:	14 words.
Primary write buffer; 1 Only:	29 words (1 line).

The typewriter coordinator consists of about 500 instructions and about 2000 words of buffer space.

Other I/O Devices: Interface II.

So far, the discussion has been restricted to character-oriented input/output devices, including the typewriter. All such devices have worked through the character interface of the time-sharing system, known as Interface I. Any character-type device can easily be attached to the system by providing an I/O adapter program which converts the raw hardware interface into the standard format of Interface I, which consists of one character/word in the character pool buffer.

There is also another broad class of devices, such as magnetic tape, which work in terms of words, and blocks of words. A second interface is provided for these devices. The details of Interface II can be found in M.I.T. Computation Center memorandum CC-226. For any input or output device for which Interface II appears to be appropriate, an I/O adapter module may be written to perform the function of matching the hardware characteristics to Interface II.

4. The Scheduling Algorithm and the Storage Algorithm

Introduction.

In this section we examine the operation of two important modules of the time-sharing supervisor: the module which decides who should run next and for how long, and the module which allocates the user memory area among various user programs. These two functions, scheduling and allocation, are in fact closely related, and have been separated in the supervisor because the particular algorithms used permit the separate consideration of the two problems. A more complex algorithm might consider both of these functions simultaneously and therefore encompass both modules. A third function, time accounting, has crept into both of these modules, although it is nominally handled by a separate module. The proper arrangement of these functions is still open to debate, and the modules described here may not represent the best possible organization.

The Scheduling Algorithm

All scheduling policy is contained in the scheduling module; in fact no mechanics of the time-sharing system are performed by this module. The scheduling module is for the rest of the supervisor a sage who is occasionally asked for an opinion, but not asked to do anything else. Since mechanics are absent, the scheduling module is well suited to the MAD language, in which it is written. Since it is hardly necessary to write an involved description of a well-organized MAD program, only the policy involved will be stated. A reader interested in exploring how the policy is carried out can easily understand the program itself. The explanatory comments at the beginning of the program serve to provide sufficient documentation.

A Typical Scheduling Policy

The particular scheduling policy described here is not the only such policy, and may not be the best policy, particularly in the choice of the parameters given in the program listing. However, it is a typical policy, and the parameters given are typical parameters. If one concludes that a different parameter, or a completely different policy, will produce better results, then a different version of the scheduling algorithm may be easily inserted into the system instead. Here, then, are the important aspects of this algorithm. Figure 4.1 is a flow diagram of the scheduling algorithm which may be easier to follow than

the program itself on the first reading of the policy rules. These rules apply to system version 1A3.

1. All users in state 2 or 3, "working" or "waiting command," are kept in a set of queues, which may be considered to be one long queue of users to run, in order.

2. The queues may be re-ordered at the end of each clock period or more frequently, depending on events occurring during that clock period. A clock period is a short period of time, typically 200 ms., during which some user program runs uninterrupted except for data channel traps.

3. Each queue has an integer valued priority level. In general, all the users in the queue with level "j" are run before any users in the queue with level "j+1". There are MAXLVL+1 levels, numbered from 0. (Typically, there may be 9 levels.)

4. A "quantum" is the shortest period of time the algorithm ever attempts to run a user. A user may run more than one quantum, depending on his level. A user at level "j" is normally allowed to run 2.P.j quanta, although he may be preempted by the arrival of a user with higher priority.

5. A user at level "j" is moved to level "j+1" after he has run 2.P.j quanta at level "j". Usually, he then stops running in favor of other users at level "j".

6. A user at level "j" is moved to level "j-1" after he has waited "QNTWAT" 60ths seconds without running at all. Typically, this waiting time may be 60 seconds.

7. A user starts at a level depending on his program length such that the time required to load his program is a fixed proportion of the running time permitted at that level. This proportion fixes roughly the maximum efficiency of the system. (The efficiency may be lower, because of pre-emption. See 8, below.) The "level of entry" function can easily be changed to reflect a different policy.

8. A user at level "j" may be pre-empted at the end of the next clock burst by a user entering the queues with a higher priority. The pre-emption will take place if the user now running has run longer than would the pre-emptor.

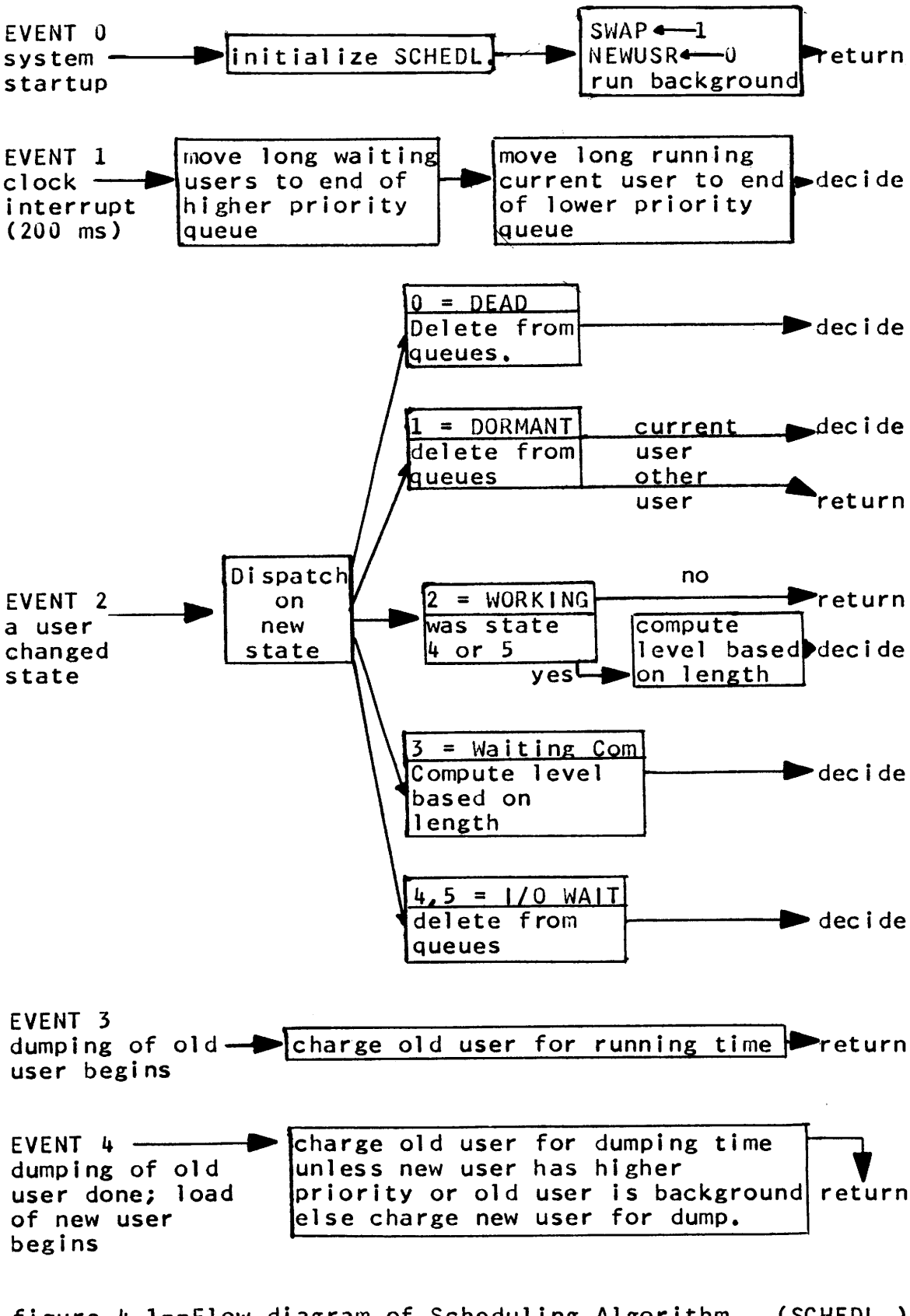
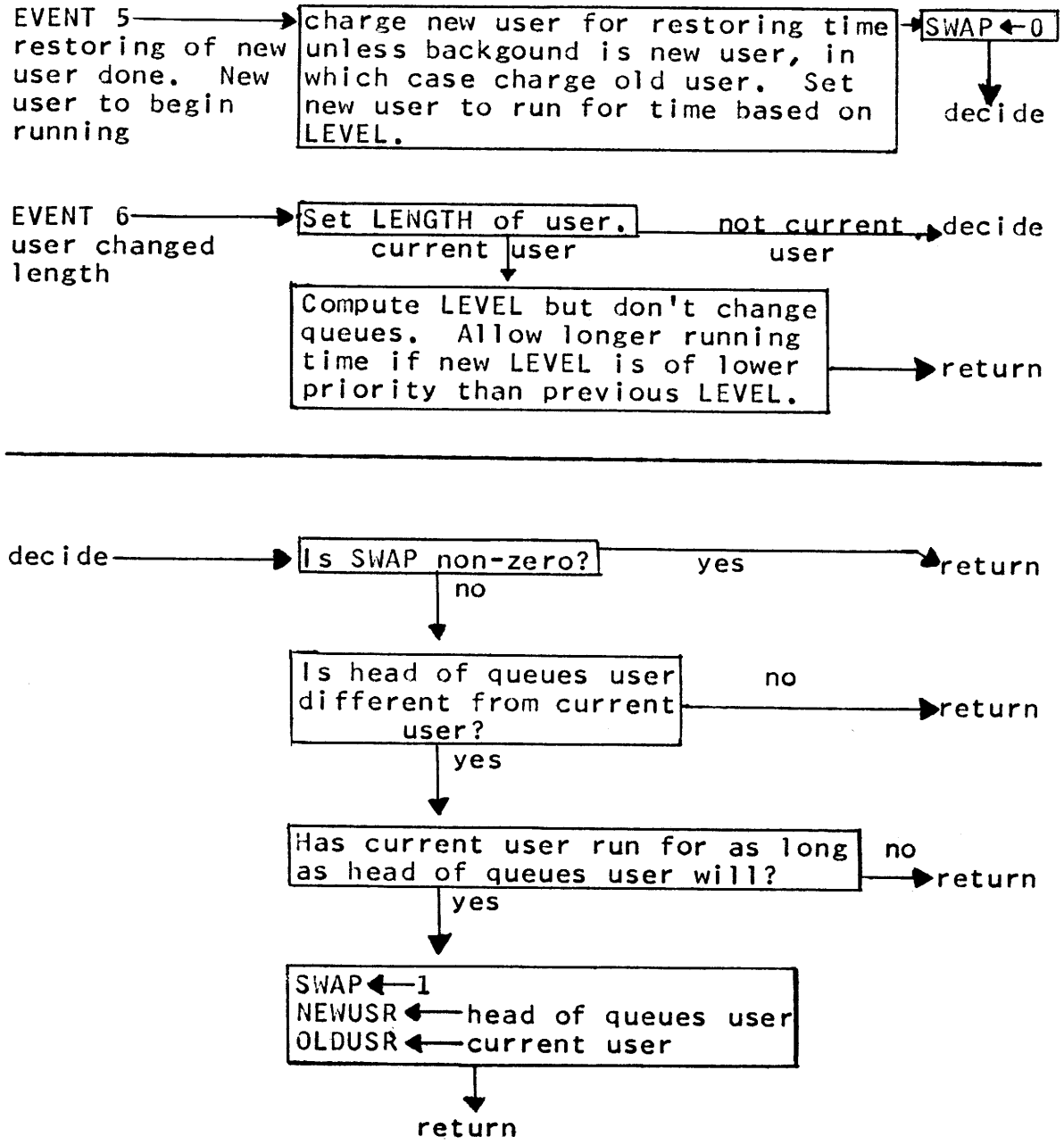


figure 4.1--Flow diagram of Scheduling Algorithm. (SCHEDL.)

Flow diagram of Scheduling Algorithm (cont.)



(When the SWAP switch is set non-zero, the supervisor will call EVENT's 3, 4, and 5 in succession as soon as it can. NEWUSR will be the next user to run.

figure 4.1--Flow diagram of Scheduling Algorithm. (SCHEDL.)

9. When a user leaves states 2 or 3 he is removed from the queues. When he returns to state 2 or 3 he is assigned a new priority level; his previous running time is not taken into consideration. Thus the algorithm concerns itself primarily with running times within interactions. A program calling for a new command is not considered a new interaction, but its level may be changed if the new command is longer.

10. If a user returns to state 2 or 3 he is placed at the end of the queue at his priority level based on program length.

The Background System.

It will be seen from the program coding that one user, user number zero, may be accorded special treatment at various points in the algorithm. User zero represents the background batch-processing system which is maintained by the supervisor both for compatibility with older systems and to provide a guaranteed backlog of work for the computer in case no regular ("foreground") users should need service for a time. The following additional policy rules describe the position of the background system with respect to the other users:

11. The background system is always at the end of the queue of users to be run. If the queue should become empty, the background system will then run until some other user enters the queue.

12. The background operator may "force" the background system to be run by depressing certain keys on his console. The background system will be brought in at the next clock trap, and run exclusively until the operator signals that normal, time-shared operation should continue.

13. It is possible to guarantee the background system a certain percentage of the facility of the computer, by setting the variable "PB" to the desired percentage in the initialization of the algorithm. When "PB" is non-zero, the background system will pre-empt whenever it falls behind its guaranteed percentage.

Policy on Charges

The above enumeration of policy rules does not describe all of the coding in the scheduling algorithm module. As mentioned in the introduction, a certain amount of time-accounting policy is maintained in this module. This policy is handled by the coding following "events" 3, 4, and

5. (See program listing for definition of an "event.") A brief description of this charging policy can be stated in the following five rules:

1. Each user pays for central processor time used.

2. A user may pay for the time it takes to load him and dump him depending on whom he follows or precedes. Note that with the present storage algorithm, loading and dumping of a user are not overlapped with computation.

3. The background system is specially privileged; it never pays for loading or dumping itself.

4. With the exception of background, all users pay for their own load time. The previous user pays for background loading time.

5. A user pays for his dump unless he is being pre-empted by a higher-priority user. In the latter case, the higher priority user pays for the dump. The next user always pays for a background dump.

The "Onion-Skin" Storage Algorithm.

The storage allocation algorithm presently used by the time-sharing supervisor has as its main virtue simplicity. There is no question that a more sophisticated procedure can be devised, and will be when time permits. However, even the present simple allocation algorithm illustrates some of the important features which must be possessed by any storage algorithm.

The simplicity of the storage allocation algorithm results from two basic features: dumping and loading of user programs are not overlapped with computation, and relocation of user programs is not attempted. Thus all users are loaded starting at absolute location zero.

The simplest possible storage algorithm would operate as follows: when a user must be dumped on to the drum, his entire program is dumped; assuming only one user in memory at a time, all of memory is now available for the next program, when necessary. The current storage algorithm attempts to improve this performance by dumping only enough of a user to fit in the next user; the earlier user is therefore split into two parts, the one part on the drum memory, and the other part in core memory, where its integrity is insured by the memory protection feature. Thus if the first user should be allowed to return to core memory his next loading can be done more quickly than in the case where he was completely dumped from core memory.

Since a third user, following the second, may be larger than the second, it may be necessary at a later time to dump more of the first user.

To illustrate the splitting, consider figures 4.2-4.6, in which several successive states of the user memory area are shown. In 4.2, user A is occupying most of core memory, and is about to be dumped in favor of user B. Following the dump, core memory appears in fig. 4.3, with part of user A in core, part on the drum. Now, user C is to be brought into memory, so user B is split in the same fashion, as shown in figure 4.4. If the next user, D, requires a larger space, as indicated in fig. 4.4, C must be completely dumped, the dump of B must be finished, and the dump of A continued a little farther. The result is shown in 4.5. If now, following user D, user A is to run again, the dump of user D will have to be complete but only the part of A which was dumped will have to be restored; thus saving some time. The result is shown in 4.6.

In this example, only two users, A and B, were split between memory and the drum at one time. As many as 16 users may be split between core and the drum. All dumps onto the drum are made in blocks of 2048 words.

Although in the illustration only a small amount of time was saved ultimately by not dumping all of A at first, in a different situation the technique may be much more effective. Consider for example, the situation when only two or three console users, each with small programs, are using the system, with background running a large share of the time. Since background is always a 32K program, most of it remains in core at all times; only enough is dumped to make room for the smaller foreground users when they need time. In this case, time saving can be large.

The User Dump.

When a user is dumped from core memory, a file is created on the drum memory. This file includes two sections:

1. The User machine conditions status table, and the User disk status table (including the current section of his user file directory)

2. The user's core image.

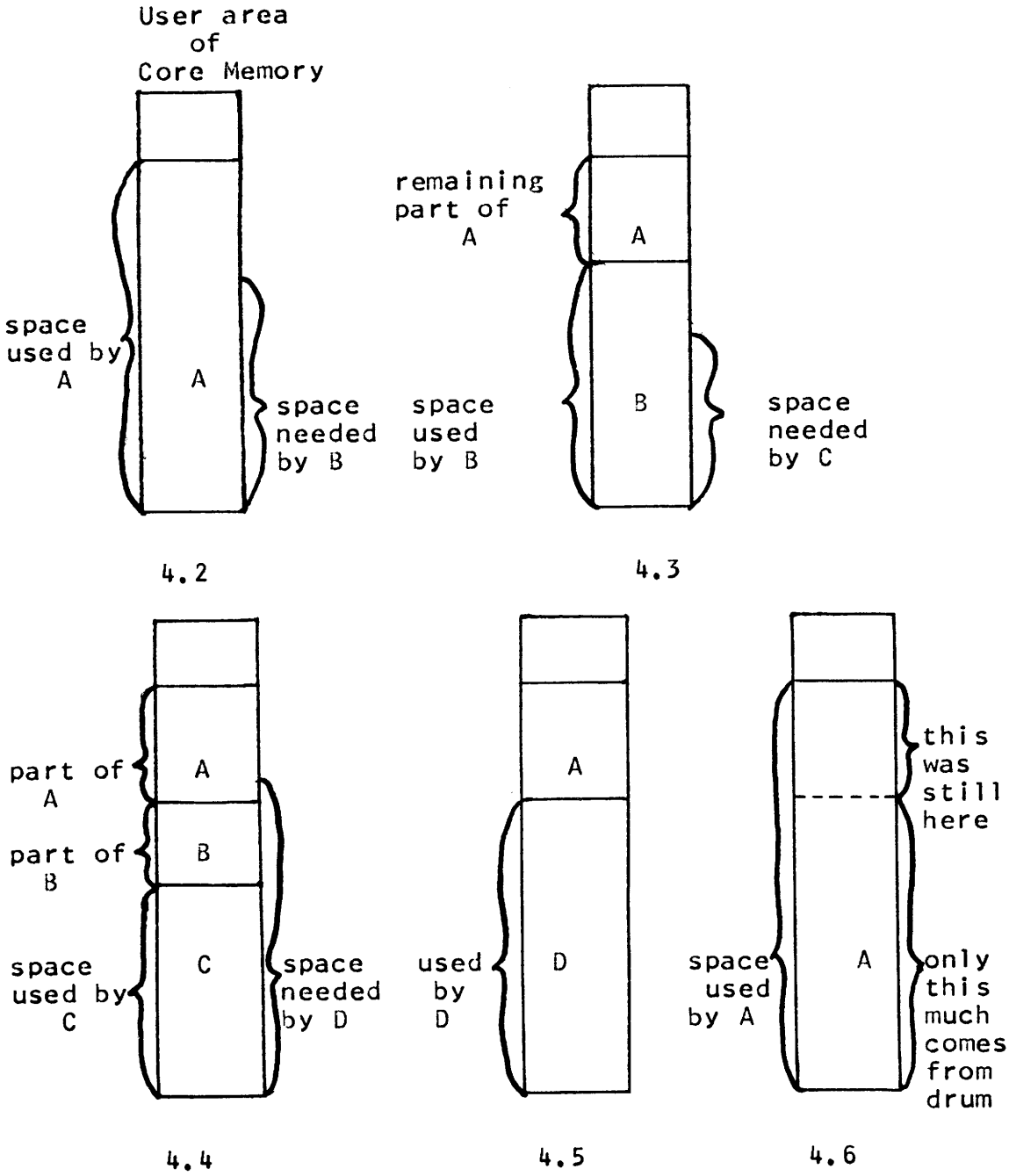
The second section is not written for a user going to the "dead" state. The file created is of temporary mode, and named "000021 UDUMP." for user 21, etc. This file is considered one of the supervisor's personal files and does not appear in the user's file directory.

In figures 4.8, 4.9 and 4.10, are illustrated the three subroutines of the storage algorithm, DUMP, UNDUMP, and FREEUP. DUMP and UNDUMP are subroutines called by the supervisor to perform the functions indicated by their name. The following notes may help in interpretation of these flow diagrams.

1. If DUMP is called to dump a working status user, it actually only dumps his machine and disc status tables and leaves the core memory dump to the routine which tries to make room for the next user. This is done because at the time the order is given to dump a user it is not known how much of him will have to go.

2. UNDUMP calls on subroutine FREEUP to dump out enough space for the new user to fit into core. Only then can it restore the new user core image and his status tables.

3. Subroutine FREEUP is called with one parameter ("NWORDS"), the number of words of core memory needed for the next user.



figures 4.2-4.6--The Onion Skin Storage Algorithm.

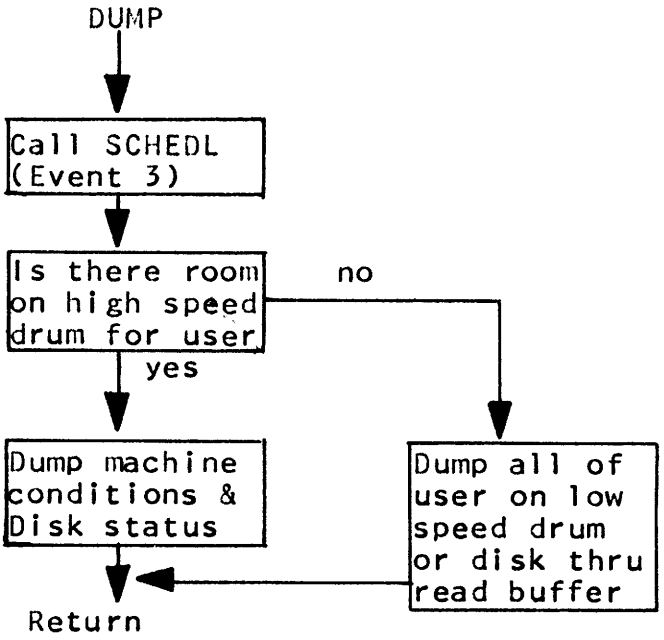


figure 4.7--Flow diagram of subroutine DUMP.

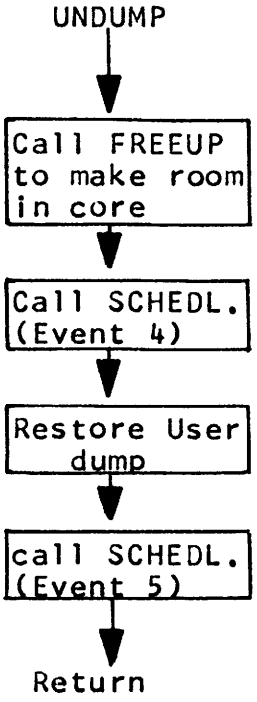


figure 4.8--Flow diagram of subroutine UNDUMP.

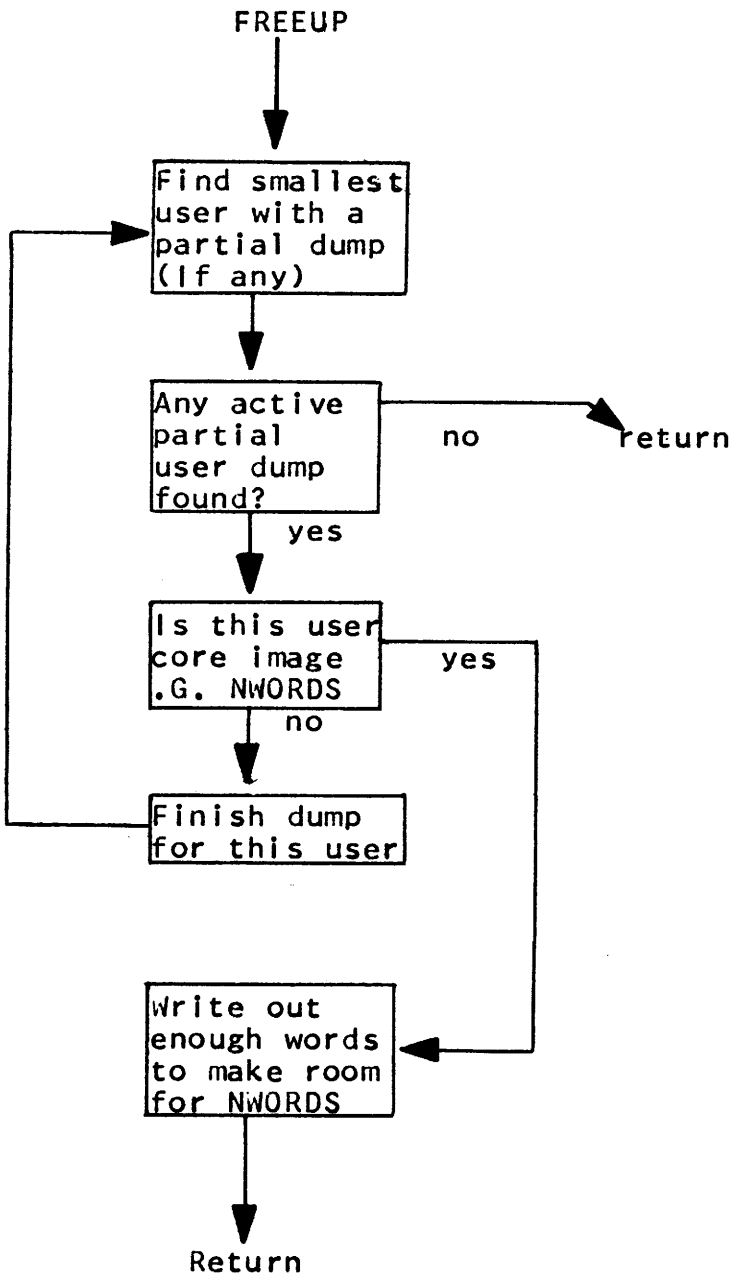


Figure 4.9 -- Flow diagram of subroutine FREEUP.

Appendix 4.A

States of a User

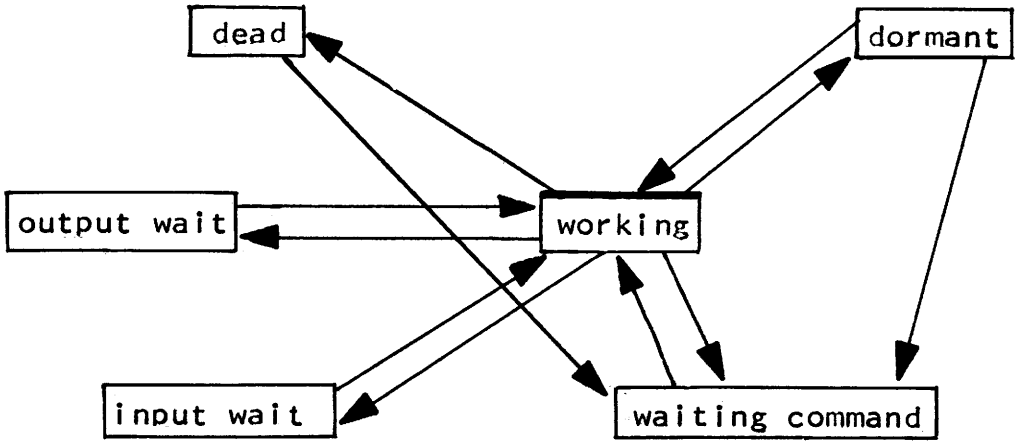


Figure 4.10

Additional transitions not shown:

1. Quit signal - any state to "dormant"
2. Forced LOGOUT - any state to "waiting command"

Description of states:

0 dead - not waiting to run and no core image; command level.
1 dormant - not waiting to run but has core image; command level.
2 working - waiting in queues to run or running.
3 waiting command - waiting in queues for first run of a command.
4 input wait - program waiting for input from console.
5 output wait - typewriter output buffers filled.

All input from the console is interpreted as commands when a user is dead or dormant. He is said to be at command level when in either of these two states.

Description of Transitions

dead to waiting command - the user typed in a defined command and is placed in the queues to wait his turn to run.

waiting command to working - the user begins to run the command for the first time. From now on the command program is treated exactly like a user program.

working to input wait - the user's program needs input from the console and the user has not yet typed in an input line. (RDFLX)

input wait to working - the user typed in an input line terminated with a break character.

working to output wait - the user's program has generated enough output to fill the output buffers. (WRFLX)

output wait to working - the output buffers are empty.

working to waiting command - the user's program issued a command. (CHNCØM or NEXCØM).

working to dormant - the user's program finished its computation but the core image is still useful. (DØRMNT)

dormant to waiting command - the user typed in a defined command which may operate on his core image or may destroy it and start fresh.

working to dead - the user's program finished its computation and the core image is to be destroyed (DEAD).

dormant to working - supervisor is restarting the user's program without a console interaction. (SLEEP)

Appendix 4.B

```
SCDA    R********** TIME SHARING SCHEDULING ALGORITHM ****************
        R     T. HASTINGS AND R. DALEY
        R
        R  THE SCHE ULING ALGORITHM PERFORMS THE FOLLOWING FUNCTIONS
        R
        R   1. DETERMINES WHICH USER IS TO RUN NEXT
        R   2. DETERMINES WHEN NEXT USER IS TO RUN
        R   3. DETERMINES HOW LONG NEXT USER IS TO RUN
        R   4. CHARGES USERS FOR SWAPPING AND RUNNING TIME
        R   5. KEEPS TRACK OF THE STATUS OF EACH USER
        R
        R  THE   HEDULING ALGORITHM IS CALLED FROM THE SUPERVISOR BY
        R   EXECUTE SCHED.(EVENT, USER, ARG)
        R     AFTER ALL TRAPS HAVE BEEN DISABLED
        R  'USER' IS BETWEEN 0 AND THE MAX. NO. OF USERS, 'MXUSR'
        R  THE SIGNIFICANCE OF 'USER' AND 'ARG' DEPEND ON 'EVENT'
        R     OR ARE MEANINGLESS AS DESCRIBED BELOW
        R     'EVENT'   DESCRIPTION
        R     0         INITIALIZATION OF SCHED.
        R     1         CLOCK INTERRUPT
        R     2         'USER' HAS CHANGED TO STATE 'ARG'
        R     3         BEGINNING OF SAVING 'USER' CORE IMAGE
        R     4         BEGINNING OF RESTORING 'USER' CORE IMAGE
        R     5         'USER' BEGINS RUNNING, AFTER SWAP
        R     6         'USER' CORE IMAGE NOW HAS LENGTH 'ARG'
        R     7         OPERATOR SET BACKGROUND KEYS TO 'ARG'
        R     8         'USER' LOGGED IN, 'ARG' IS LINE MULTIPLIER
        R     9         'USER' LOGGED OUT
        R    10         IS 'NEWUSR' STILL RUNABLE
        R    11         'USER' DIALED UP COMPUTER
        R
        R  ALL TIME IS KEPT IN SIXTIETHS OF A SECOND AND VARIABLES
        R     ENDING WITH 'TIM' ARE TIMES SINCE SYSTEM WAS
        R     LOADED WITH THE EXCEPTION OF 'SYSTIM'
        R  SCHED. HAS SOLE RESPONSIBILITY FOR SETTING AND CHANGING
        R     THE FOLLOWING COMMON ARRAYS AND VARIABLES
        R
        R  THE FOLLOWING COMMON ARRAYS ARE USED
        R   'STATUS' - THE STATUS OF EACH USER
        R   WHERE STATUS(J) MAY BE
        R         0 DEAD - NOT WAITING TO RUN AND NO CORE IMAGE
        R         1 DORMNT - NOT WAITING TO RUN
        R         2 WORKING - WAITING IN QUEUES OR RUNNING
        R         3 WAITING COMMAND - WAITING IN QUEUES FOR COM.
        R         4 INPUT WAIT - PROGRAM WAITING FOR INPUT
        R         5 OUTPUT WAIT - OUTPUT BUFFERS FILLED
        R   'LENGTH' - LENGTH OF USER CORE IMAGE IN WORDS
        R   'LEVEL' - USER'S PRIORITY LEVEL(0, ... , 'MAXLVL')
        R   'TIMLEV' - ELAPSED TIME RUN AT CURRENT LEVEL
```

```
R   'WATTIM' - THE LAST TIME THAT A USER BEGAN TO WAIT
R   'LINMUL' - USER LINE MULTIPLIER
R   'PLIST' - THE POSITION LIST SPECIFIES THE POSITIONS
R           OF THE USERS WHICH ARE IN THE WORKING QUEUE
R   'ULIST' - THE USER LIST INDICATES THE USER NUMBERS
R           WHICH CORRESPOND TO THESE QUEUE POSITIONS
R   'ENDPTR' - ENDPTR(J) IS END OF QUEUE J IN PLIST
R   'NOTIME' - NOTIME(J) IS SET TO 2 IF USER INACTIVE
R     AND USER J WILL SUBSEQUENTLY BE LOGGED OUT
R
R     THE FOLLOWING COMMON VARIABLES ARE USED
R   'MXUSRS' - MAX. NO. OF FOREGROUND USERS
R   'CURUSR' - CURRENT USER, RUNNING OR SWAPPING
R   'OLDUSR' - LAST USER TO BE RUN, WHEN 'SWAP' .NE. 0
R   'NEWUSR' - NEXT USER TO BE RUN, WHEN 'SWAP' .NE. 0
R   'PAYUSR' - THE USER CURRENTLY PAYING FOR TIME
R   'SYSTIM' - TIME SYSTEM WAS INITIALIZED
R   'BEGTIM' - THE LAST TIME 'CURUSR' BEGAN TO RUN
R   'QUANTM' - MAXIMUM RUNNING TIME AT LEVEL 0
R   'MAXTIM' - USER RUNS AT SAME LEVEL UNTIL 'MAXTIM'
R   'TBASE' - BASE TIME FOR COMPUTING 'MAXTIM'
R   'PAYTIM' - LAST TIME A USER WAS CHARGED FOR TIME
R   'LEVTIM' - LAST TIME 'CURUSR' WAS RUNNING AT CURRENT LEVEL
R   'SWAP' - NON-ZERO REQUESTS SUPERVISOR TO RUN
R             'NEWUSR' AS SOON AS IT CAN
R   'MAXLVL' - THE MAXIMUM PRIORITY LEVEL(0 ... 'MAXLVL')
R   'MINLVL' - THE MINIMUM PRIORITY LEVEL ALLOWED
R   'FULLVL' - INIT. LEVEL FOR 'FULLEN' TO FULL CORE USER
R   'EMPLVL' - INITIAL LEVEL FOR EMPTY CORE USER
R   'FULLEN' - LENGTH FOR ENTRY AT LEVEL 'FULLVL'
R   'PB' - GUARANTEED PERCENTAGE FOR BACKGROUND
R   'QNTWAT' - QUANTM WAITING TIME BEFORE LEVEL CHANGE
R             TO NEXT HIGHEST PRIORITY LEVEL
R   'LEVINC' - AMOUNT PRIORITY LEVEL IS INCREASED WHEN
R     USER RETURNS TO WORKING FROM INPUT OR OUTPUT WAIT
R   'INACTV' - MAX. TIME INACTIVE BEFORE LOGOUT
R   'HANGUP' - MAX. TIME BEFORE INACTIVE LINE IS HUNGUP
R
R     COMMON VARIABLES REFERRED TO BY SCHED. BUT
R       NOT SET OR CHANGED BY SCHED.
R   'BKGTIM' - TOTAL TIME BACKGROUND HAS RUN
R   'SWPSW' - NON-ZERO WHEN SUPERVISOR IS SWAPPING AND
R      COMMAND LOADING
R   'PROBN(J)' - NON-ZERO WHEN USER J IS LOGGED IN
R   'ADOPT(J)' - PROBN(J) .AND. ADOPT(J) .E. 1B, THEN
R     USER J IS ADOPTED
R
R     SCHED. CALLS THE FOLLOWING SUBROUTINES
R   INITQ. - INITIALIZES QUEUES
R   HEDUSR. - RETURNS THE HEAD OF QUEUE USER
R        AT HIGHEST NON-EMPTY PRIORITY LEVEL OR 0
R   DELQUE.(J) - DELETES USER J FROM QUEUES
R   ENDQUE.(J) - PLACES USER J AT END OF QUEUE LEVEL(J)
```

```
R   BEGQUE.(J) - PLACES USER J AT BEG OF QUEUE LEVEL(J)
R   ILOG2.(N) - RETURNS INTEGER PART OF LOG TO BASE 2 N
R   I.(J) - CONVERTS FORWARD INDEX 'J' TO BACKWARD
R           INDEX FOR REFERRING TO MAD ARRAYS
R   INITIM. - INITIALIZE TIME ACCOUNTING
R   INTIM. - USER 'U' LOGGED IN
R   OUTIM. - USER 'U' LOGGED OUT
R   CHARGE.(U,T) - CHARGE USER 'U' FOR TIME 'T'
R   GETOTL. - RETURNS THE TOTAL TIME SYSTEM HAS RUN
R   DELTIM.(T) - RETURNS DELTA 'T' - THE DIFFERENCE
R        BETWEEN 'GETOTL.()' AND TIME 'T'
R        TIME 'T' IS ALSO SET TO GETOTL.(0)
R   CURTIM.(0) - RETURNS THE CURRENT TIME SINCE MIDNIGHT
R       OF DAY SYSTEM WAS INITIALIZED
R   MONSC1.(EVENT, USER, ARG) MONITORS SCHED.
R   MONSC2. IS CALLED WHEN SCHED. CHANGES COMMON
R   PLOT1.(EVENT, USER, ARG) PLOTS SYSTEM ON ESL SCOPE
R   PLOT2. IS CALLED WHEN SCHED. CHANGED COMMON
R
 EXTERNAL FUNCTION(A, B, C)
 ENTRY TO SCHED.
 NORMAL MODE IS INTEGER
R
R.. SHORTEN LINKAGE, SETUP USER INDEX, CALL MONITORING SUB.,
R..   CALL PLOTTING ROUTINE
R..   ASSUME COMMON WILL BE CHANGED, AND DISPATCH ON 'EVENT'
R
 EVENT = A
 USR = B
 IUSER = I.(USR)
 ARG = C
 EXECUTE MONSC1.(EVENT, USR, ARG)
 EXECUTE PLOT1.(EVENT, USR, ARG)
 MONITR = CHANGE
 STATEMENT LABEL MONITR, RETURN, CHANGE
 TRANSFER TO EVNT(EVENT)
```

```
          R.. 'EVENT' .E. 0, INITIALIZE SCHEDULING ALGORITHM FOR N USERS
          R..   INITIALIZE INDEPENDENT COMMON VARIABLES
EVNT(0)    MXUSRS = 31
           MAXLVL = 8
           MINLVL = 0
           FULLVL = 3
           EMPLVL = 2
           FULLEN = 4096
           PB = 0
           QNTWAT = 3600
           LEVINC = 0
           QUANTM = 30
           TBASE = 0
           INACTV = 216000
           HANGUP = 7200
          R
          R..   INITIALIZE QUEUES AND TIME ACCOUNTING
           EXECUTE INITQ.
           EXECUTE INITIM.
          R
          R..   INITIALIZE TABLES
           THROUGH JLOOP, FOR J = 0, 1, J .G. UMAX
             JUSER = 1.(J)
    JLOOP   LINMUL(JUSER) = 1
          R
          R..   SET BACKGROUND(USER 0) TO RUN
          R..   USER 0 IS ALWAYS IMPLICITLY AT END OF QUEUES
           SYSTIM = CURTIM.(0)
           STATUS(1.(0)) = 2
           SWAP = 1B
           FIRST3 = 1B
           BGMAX = 180
           TRANSFER TO CHANGE
```

```
          R.. 'EVENT' .E. 1, CLOCK INTERRUPT
          R..   ASSUME COMMON WILL NOT BE CHANGED
EVNT(1)    MONITR = RETURN
           ICUR = I.(CURUSR)
           T = GETOTL.(0)
          R.. DO THE FOLLOWING CHECKING EVERY 10 SECONDS
          R..   CHARGE PAYING USER FOR TIME
          R..   MOVE LONG WAITING USERS UP IN PRIORITY
          R..   LOGOUT INACTIVE USERS, HANG UP INACTIVE LINES
           WHENEVER T .G. CHECKT
             CHECKT = T + 600
             EXECUTE CHARGE.(PAYUSR, DELTIM.(PAYTIM))
             THROUGH KLOOP, FOR K = 1, 1, K .G. UMAX
               WHENEVER K .E. CURUSR, TRANSFER TO KLOOP
               KUSER = I.(K)
               DELT = T - WATTIM(KUSER)
               WHENEVER STATUS(KUSER) .E. 3 .OR. STATUS(KUSER) .E. 2
                 WHENEVER DELT .G. QNTWAT .AND. LEVEL(KUSER) .G. MINLVL
                   MONITR = CHANGE
                   EXECUTE DELQUE.(K)
                   LEVEL(KUSER) = LEVEL(KUSER) - 1
                   EXECUTE ENDQUE.(K)
                   WATTIM(KUSER) = T
                   TIMLEV(KUSER) = 0
                 END OF CONDITIONAL
               OR WHENEVER PROBN(KUSER) .NE. 0
                 WHENEVER DELT .G. INACTV .AND. LINENO(KUSER) .E. 0
                   MONITR = CHANGE
                   NOTIME(KUSER) = 2
                   WATTIM(KUSER) = T
                 END OF CONDITIONAL
               OTHERWISE
                 WHENEVER DELT .G. HANGUP .AND. ADOPT(KUSER) .E. 0
                   MONITR = CHANGE
                   NOTIME(KUSER) = 4
                   WATTIM(KUSER) = T
                 END OF CONDITIONAL
               END OF CONDITIONAL
    KLOOP      CONTINUE
           END OF CONDITIONAL
          R..   MOVE LONG RUNNING 'CURUSR' DOWN IN PRIORITY
           WHENEVER CURUSR .NE. 0 .AND. T .G. MAXTIM
          1     .AND. .NOT. SWAP
             MONITR = CHANGE
             EXECUTE DELQUE.(CURUSR)
             WHENEVER LEVEL(ICUR) .L. MAXLVL,
          1       LEVEL(ICUR) = LEVEL(ICUR) + 1
             EXECUTE ENDQUE.(CURUSR)
             LEVTIM = T
             TIMLEV(ICUR) = 0
             MAXTIM = T + TRUN.(CURUSR, LEVEL(ICUR))
           END OF CONDITIONAL
           TRANSFER TO DECIDE
```

```
          R.. 'EVENT' .E. 2, 'USR'('IUSER') CHANGED STATE
          R..   DISPATCH ON NEW STATE, IGNORE REDUNDANT TRANSITIONS
EVNT(2)    WHENEVER USR .NE. 0, TRANSFER TO STAT(ARG)
           TRANSFER TO RETURN
          R
          R..   'USR'('IUSER') WENT DEAD, EVENT 6 WILL NOT OCCUR
   STAT(0) EXECUTE DELQUE.(USR)
           STATUS(IUSER) = 0
           TRANSFER TO DECIDE
          R
          R..   'USR'('IUSER') WENT DORMANT WHILE RUNNING
          R..     OR PUSHED QUIT BUTTON
   STAT(1) EXECUTE DELQUE.(USR)
           STATUS(IUSER) = 1
           WHENEVER USR .E. CURUSR, TRANSFER TO DECIDE
           TRANSFER TO CHANGE
          R
          R..   'USR'('IUSER') TO BEGIN WORKING AFTER I/O WAITING
          R..     OR ALARM CLOCK RETURN FROM DORMANT TO WORKING
   STAT(2) WHENEVER STATUS(IUSER) .GE. 4 .OR. STATUS(IUSER) .E. 1
             WHENEVER STATUS(IUSER) .NE. 1
               WHENEVER LEVEL(IUSER) - LEVINC .GE. MINLVL,
          1         LEVEL(IUSER) = LEVEL(IUSER) - LEVINC
               LEV = LEVELF.(LENGTH(IUSER))
               WHENEVER LEV .L. LEVEL(IUSER), LEVEL(IUSER) = LEV
               TIMLEV(IUSER) = 0
             END OF CONDITIONAL
             EXECUTE ENDQUE.(USR)
             WATTIM(IUSER) = GETOTL.(0)
             STATUS(IUSER) = 2
             TRANSFER TO DECIDE
           END OF CONDITIONAL
           TRANSFER TO RETURN
          R
          R..   'USR'('IUSER') BEGAN WAITING FOR A COMMAND
   STAT(3) LEV = LEVELF.(LENGTH(IUSER))
           WHENEVER STATUS(IUSER) .E. 2 .OR. STATUS(IUSER) .E. 3
             WHENEVER LEV .G. LEVEL(IUSER)
               EXECUTE DELQUE.(USR)
               TRANSFER TO COMAND
             END OF CONDITIONAL
           OTHERWISE
    COMAND   LEVEL(IUSER) = LEV
             EXECUTE ENDQUE.(USR)
             TIMLEV(IUSER) = 0
             WATTIM(IUSER) = GETOTL.(0)
           END OF CONDITIONAL
           STATUS(IUSER) = 3
           TRANSFER TO DECIDE
```

```
        R..   'USR'('IUSER') ENTERED INPUT WAIT
   STAT(4) WHENEVER STATUS(IUSER) .E. 2
             EXECUTE DELQUE.(USR)
             STATUS(IUSER) = 4
             TRANSFER TO DECIDE
            END OF CONDITIONAL
            TRANSFER TO RETURN
        R
        R..   'USR'('IUSER') ENTERED OUTPUT WAIT
   STAT(5) WHENEVER STATUS(IUSER) .E. 2
             EXECUTE DELQUE.(USR)
             STATUS(IUSER) = 5
             TRANSFER TO DECIDE
            END OF CONDITIONAL
            TRANSFER TO RETURN
        R
        R.. THE NEXT THREE EVENTS ALWAYS OCCUR IN SEQUENCE
        R..   WHEN CONTROL IS TRANSFERRED FROM 'OLDUSR' TO 'NEWUSR'
        R..   AS A RESULT OF 'SWAP' BEING SET NON-ZERO.
        R.. 'OLDUSR' DOES NOT PAY FOR HIS DUMP, UNLESS
        R..   'NEWUSR' IS OF EQUAL OR LOWER PRIORITY.
        R.. 'NEWUSR' ALWAYS PAYS FOR BEING RESTORED EXCEPT
        R..   BACKGROUND NEVER PAYS FOR DUMP OR RESTORE.
        R
        R.. 'EVENT' .E. 3, SAVING OF 'USR'('IUSER') IS BEGINNING
        R..   EVENT 3 MAY BE CALLED FOR ANY OF THE FOLLOWING:
        R     1. FREEING UP CORE B BECAUSE 'CURUSR' EXTENDED SIZE
        R     2. FREEING UP CORE A DRUM BUFFERS FOR SWAPPING
        R     3. DUMPING 'OLDUSR'
        R     4. DUMPING OTHER USERS TO MAKE ROOM FOR 'NEWUSR'
            BOOLEAN SWPSW, FIRST3, DMPOLD, SWAP
EVNT(3)     WHENEVER SWPSW
              WHENEVER FIRST3
                FIRST3 = 0B
                EXECUTE CHARGE.(PAYUSR, DELTIM.(PAYTIM))
                WHENEVER LEVEL(I.(NEWUSR)) .GE. LEVEL(I.(OLDUSR))
        1          .AND. OLDUSR .NE. 0 .OR. NEWUSR .E. 0
                  PAYUSR = OLDUSR
                OTHERWISE
                  PAYUSR = NEWUSR
                END OF CONDITIONAL
                TIMLEV(I.(OLDUSR)) = TIMLEV(I.(OLDUSR)) + DELTIM.(LEVTIM)
              OTHERWISE
                EXECUTE CHRGSW.
                WHENEVER USR .E. OLDUSR
                  DMPOLD = 1B
                OR WHENEVER DMPOLD .AND. USR .NE. OLDUSR
        1           .AND. NEWUSR .NE. 0
                  PAYUSR = NEWUSR
                END OF CONDITIONAL
              END OF CONDITIONAL
            END OF CONDITIONAL
            TRANSFER TO CHANGE
```

```
          R.. 'EVENT' .E. 4, RESTORING OF 'NEWUSR' IS BEGINNING
EVNT(4)    EXECUTE CHRGSW.
           WHENEVER NEWUSR .E. 0
             PAYUSR = OLDUSR
           OTHERWISE
             PAYUSR = NEWUSR
           END OF CONDITIONAL
           WHENEVER STATUS(I.(OLDUSR)) .E. 2,
          1    WATTIM(I.(OLDUSR)) = GETOTL.(0)
           CURUSR = NEWUSR
           TRANSFER TO CHANGE
          R
          R.. 'EVENT' .E. 5, 'NEWUSR' BEGINS RUNNING AFTER RESTORE
EVNT(5)    EXECUTE CHARGE.(PAYUSR, DELTIM.(PAYTIM))
           PAYUSR = NEWUSR
           WHENEVER STATUS(I.(NEWUSR)) .E. 3, STATUS(I.(NEWUSR)) = 2
           BEGTIM = GETOTL.(0)
           LEVTIM = BEGTIM
           MAXTIM = BEGTIM + TRUN.(NEWUSR, LEVEL(I.(NEWUSR)))
          1    -TIMLEV(I.(NEWUSR))
           SWAP = 0B
           FIRST3 = 1B
           DMPOLD = 0B
           TRANSFER TO DECIDE
```

```
          R.. 'EVENT' .E. 6, 'USR'('IUSER') CORE IS OF LENGTH 'ARG'
          R..   JUST BEFORE ENTERING WAITING COMMAND
          R..   OR LENGTH CHANGED WHILE RUNNING
EVNT(6)    LENGTH(IUSER) = ARG
           WHENEVER USR .E. CURUSR
             LEV = LEVELF.(LENGTH(IUSER))
             WHENEVER LEV .G. LEVEL(IUSER),
          1      MAXTIM = BEGTIM + TRUN.(CURUSR, LEV) - TIMLEV(IUSER)
           END OF CONDITIONAL
           TRANSFER TO CHANGE
          R
          R.. 'EVENT' .E. 7, OPERATOR SET KEYS TO 'ARG'
EVNT(7)    KEYS = ARG
           BACKGR = ARG
           TRANSFER TO DECIDE
          R
          R.. 'EVENT' .E. 8, 'USR'('IUSER') LOGGED IN PROPERLY
EVNT(8)    LINMUL(IUSER) = ARG
           EXECUTE INTIM.(USR)
           TRANSFER TO CHANGE
          R
          R.. 'EVENT' .E. 9, 'USR'('IUSER') LOGGED OUT
EVNT(9)    EXECUTE OUTTIM.(USR)
           TRANSFER TO CHANGE
          R
          R.. 'EVENT' .E. 10, IS 'NEWUSR' STILL RUNABLE
EVNT(10)   WHENEVER STATUS(I.(NEWUSR)) .E. 2
          1    .OR. STATUS(I.(NEWUSR)) .E. 3, TRANSFER TO RETURN
           SWAP = 0B
           TRANSFER TO DECIDE
          R
          R.. 'EVENT' .E. 11, 'USR'('IUSER') DIALED UP COMPUTER
EVNT(11)   WATTIM(IUSER) = GETOTL.(0)
           NOTIME(IUSER) = 0
           TRANSFER TO CHANGE
          R
```

```
          R.. COMMON EXIT FROM SCHED.
          R.. DECIDE IF IT IS TIME TO RUN A NEW USER
          R
          R..   NO DECISION WHILE SWAPPING
DECIDE     WHENEVER SWAP, TRANSFER TO MONITR
          R
          R..   CHECK IF BACKGROUND NOT MEETING GUARANTEED PERCENTAGE
           WHENEVER BKGTIM .L. (PB/100.) * GETOTL.(0)
          1    .AND. CURUSR .NE. 0, BACKGR = 1
           U = HEDUSR.(0)
           WHENEVER BACKGR .NE. 0 .OR. KEYS .NE. 0 , U = 0
          R
          R..   RUN USER 'U' IF 'CURUSR' HAS RUN AS LONG AS 'U' WOULD
           WHENEVER U .NE. CURUSR .AND.
          1     (PREMPT.(TRUN.(U, LEVEL(I.(U)))) .OR. CURUSR .E. 0)
          2     .OR. STATUS(I.(CURUSR)) .NE. 2 .OR. BACKGR .NE. 0
              MONITR = CHANGE
              SWAP = 1B
              NEWUSR = U
              OLDUSR = CURUSR
              BACKGR = 0
           END OF CONDITIONAL
          R
          R.. CALL MONSC2. IF COMMON CHANGED, ELSE JUST RETURN
           TRANSFER TO MONITR
CHANGE     EXECUTE MONSC2.
           EXECUTE PLOT2.
RETURN     FUNCTION RETURN
```

```
       R.. INTERNAL FUNCTIONS
TRUN   R..   'TRUN' - COMPUTES RUN TIME FOR USER 'DU' AT LEVEL 'DL'
        INTERNAL FUNCTION TRUN.(DU, DL) =
       1    TBASE + LINMUL(1.(DU)) * QUANTM * 2 .P. DL
       R
LEVELF R..   'LEVELF' - COMPUTE PRIORITY LEVEL BASED ON LENGTH 'LEN'
        INTERNAL FUNCTION(LEN)
        ENTRY TO LEVELF.
        WHENEVER LEN .GE. FULLEN
          L = FULLVL
        OTHERWISE
          L = EMPLVL + ILOG2.(LEN/(FULLEN/(2 .P. (FULLVL-EMPLVL))))
        END OF CONDITIONAL
        FUNCTION RETURN L
        END OF FUNCTION
       R
PREMPT R..   'PREMPT' - IS TRUE IF PREMPTION IS PERMITTED
       R..     BASED ON TIME INTERRUPTER WILL RUN 'INTRUN'
        BOOLEAN PREMPT.
        INTERNAL FUNCTION PREMPT.(INTRUN) =
       1    INTRUN .L. GETOTL.(0) - BEGTIM
       R
       R.. SUBROUTINE TO CHARGE SWAPPING TIME
CHRGSW R..   FOREGROUND PAYS FOR BACKGROUND SWAP UP TO 3 SECONDS
        INTERNAL FUNCTION
        ENTRY TO CHRG SW.
        TDEL = DELTIM.(PAYTIM)
        WHENEVER OLDUSR .E. 0 .AND. TDEL .G. BGMAX
          EXECUTE CHARGE.(PAYUSR, BGMAX)
          EXECUTE CHARGE.(0, TDEL-BGMAX)
        OTHERWISE
          EXECUTE CHARGE.(PAYUSR, TDEL)
        END OF CONDITIONAL
        FUNCTION RETURN
        END OF FUNCTION
```

```
R.. COMMON VARIABLES
 VECTOR VALUES COMRLC = 32561
 VECTOR VALUES COMRLC = 32561
 VECTOR VALUES UMAX = 51
 VECTOR VALUES UMAX = 51
 VECTOR VALUES MAXLV = 10
 VECTOR VALUES MAXLV = 10
R..   'MXUSRS' MUST BE .LE. 51 AND 'MAXLVL' MUST BE .LE. 10
 DIMENSION FAKE(32561)
 DIMENSION DUMMY0(51), STATUS(51), LENGTH(51), LEVEL(51)
 DIMENSION TIMLEV(51)
 DIMENSION WATTIM(51), LINMUL(0),  DUMMY2(73)
 DIMENSION PLIST(73),  ULIST(0),  DUMMY4(10), ENDPTR(10)
 DIMENSION TOTLEV(0)
 DIMENSION DUMMY6(51), TA1(51),    TA2(51),     TA3(51)
 DIMENSION TA4(51),    TU1(51),    TU2(51),    TU3(51)
 DIMENSION TU4(51),    UTIME(51),  NOTIME(51)
 DIMENSION ITIME(51),  PROBN(51),  PROGN(51)
 DIMENSION LINENO(51), LINCR(51),  MANUAL(51), RSPONS(51)
 DIMENSION FULLSW(51)
 DIMENSION UDWAIT(51), RWORDS(51), WWORDS(51), RTIMES(51)
 DIMENSION WTIMES(51)
 DIMENSION UNITID(51), COMMND(51), INTRSW(51), HUNGSW(51)
 DIMENSION ILINES(51), OUTPSW(51), COMCTR(51), IOD(51)
 DIMENSION ULINE(51),  ULINCT(51), UCLOCK(51), UCHARG(51)
 DIMENSION AWAKE(51), TIMINC(51), CLOCON(51), ADOPT(51)
 DIMENSION OKPROB(51), OKPROG(51), COMFSW(0)
 DIMENSION DUMMYC(27), PBUFF(0)
 DIMENSION DUMMYE(465), DBUF1(0),  DUMMYG(465), DBUF2(0)
 PROGRAM COMMON FAKE
 PROGRAM COMMON  ENBWD
R.. COMMON ARRAYS SET AND CHANGED BY SCHEDL. ONLY
 PROGRAM COMMON  DUMMY0, STATUS, LENGTH, LEVEL,  TIMLEV
 PROGRAM COMMON  WATTIM, LINMUL, DUMMY2, PLIST,  ULIST
 PROGRAM COMMON  DUMMY4, ENDPTR, TOTLEV
R.. TABLES SET BY LOGIN, UPDATED BY TIME ACCOUNTING
 PROGRAM COMMON  DUMMY6, TA1,    TA2,    TA3,    TA4
 PROGRAM COMMON  TU1,    TU2,    TU3,    TU4,    UTIME
 PROGRAM COMMON  NOTIME
R.. TABLES SET BY LOGIN
 PROGRAM COMMON  ITIME,  PROBN,  PROGN
R.. USER OPTIONS(CHECKED BY CLKINT AND TCORRD)
 PROGRAM COMMON  LINENO, LINCR,  MANUAL, RSPONS, FULLSW
R..  TABLES FOR DISK MONITORING
 PROGRAM COMMON  UDWAIT, RWORDS, WWORDS, RTIMES, WTIMES
R.. OTHER USER TABLES
 PROGRAM COMMON  UNITID, COMMND, INTRSW, HUNGSW, ILINES
 PROGRAM COMMON  OUTPSW, COMCTR, IOD,    ULINE,  ULINCT
 PROGRAM COMMON  UCLOCK, UCHARG, AWAKE,  TIMINC, CLOCON
 PROGRAM COMMON ADOPT, OKPROB, OKPROG, COMFSW
R.. COMMON VARIABLES SET AND CHANGED BY SCHEDL. ONLY
 PROGRAM COMMON  MXUSRS, CURUSR, OLDUSR, NEWUSR, PAYUSR
 PROGRAM COMMON  SYSTIM, BEGTIM, QUANTM, MAXTIM, TBASE
```

```
 PROGRAM COMMON  PAYTIM, LEVTIM, SWAP,   MAXLVL, MINLVL
 PROGRAM COMMON  FULLVL
 PROGRAM COMMON  EMPLVL, FULLEN, PB,     QNTWAT
 PROGRAM COMMON  LEVINC, INACTV, HANGUP
R.. VARIABLES SET BY LOGIN
 PROGRAM COMMON  SPROBN, SPROGN
R.. OTHER VARIABLES
 PROGRAM COMMON  USER,   DATE,   DATEYR, TIMNOW, NUSERS
 PROGRAM COMMON  SWPSW,  COMSW,  TOTTIM, BKGTIM, SWPTIM
 PROGRAM COMMON  COMTIM, USRWAT, SWPWAT, COMWAT, AUTOND
 PROGRAM COMMON  CLKTIM, MXLINE, NWORDS, STOPSW
 PROGRAM COMMON  DSKLOC, BASEAD, WAIT,   DUMMY8, READY
 PROGRAM COMMON  BUFULL, DUMMYC
 PROGRAM COMMON  PBUFF,  DUMMYE, DBUF1,  DUMMYG, DBUF2
R.. USER MACHINE CONDITIONS STATUS TABLE
R..   (NOT REFERRED TO BY MAD PROGRAMS)
 END OF FUNCTION
```

5. Flow Charts of Main Control and Trap Processors.

Introduction.

This section consists of five flow charts of Main Control, the clock and protection trap processors, and the module RSTCPU. These flow diagrams are to help provide a temporary bridge between complete lack of information about these modules and the assembly listings themselves.

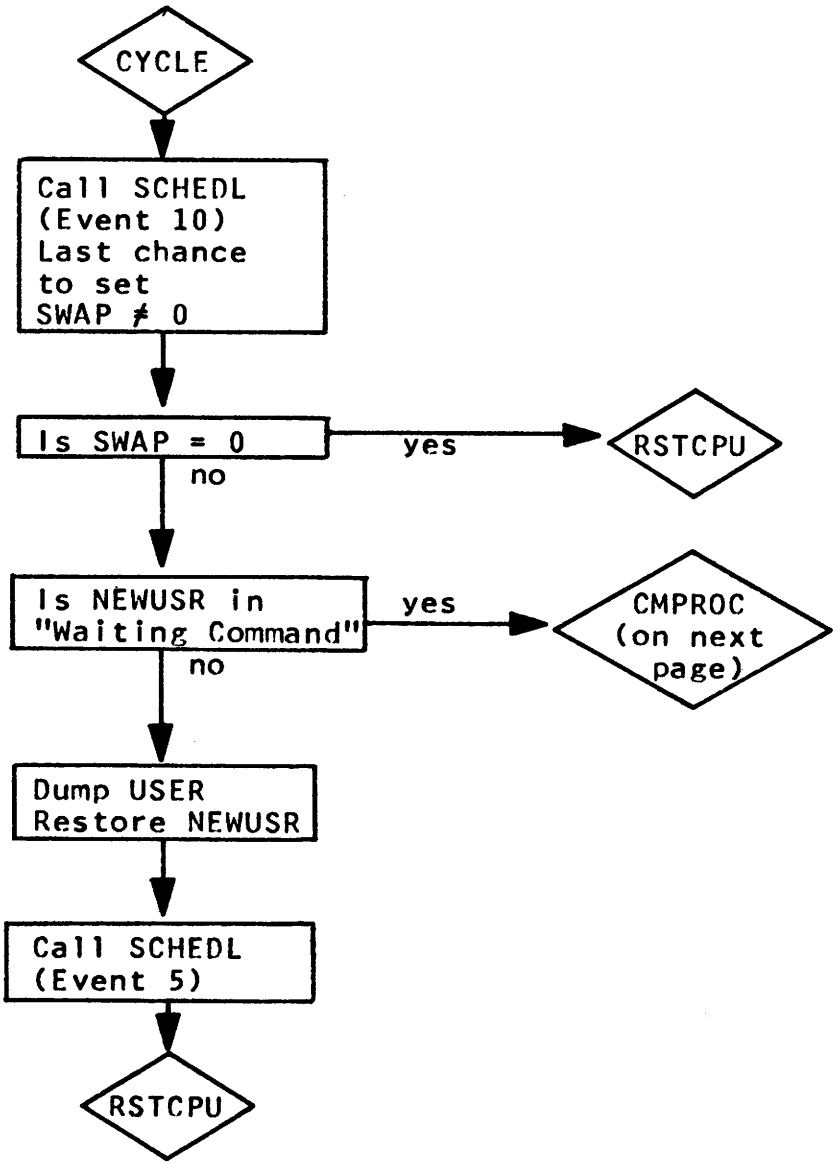


Figure 5.1 -- Cycle entry of Main Control.

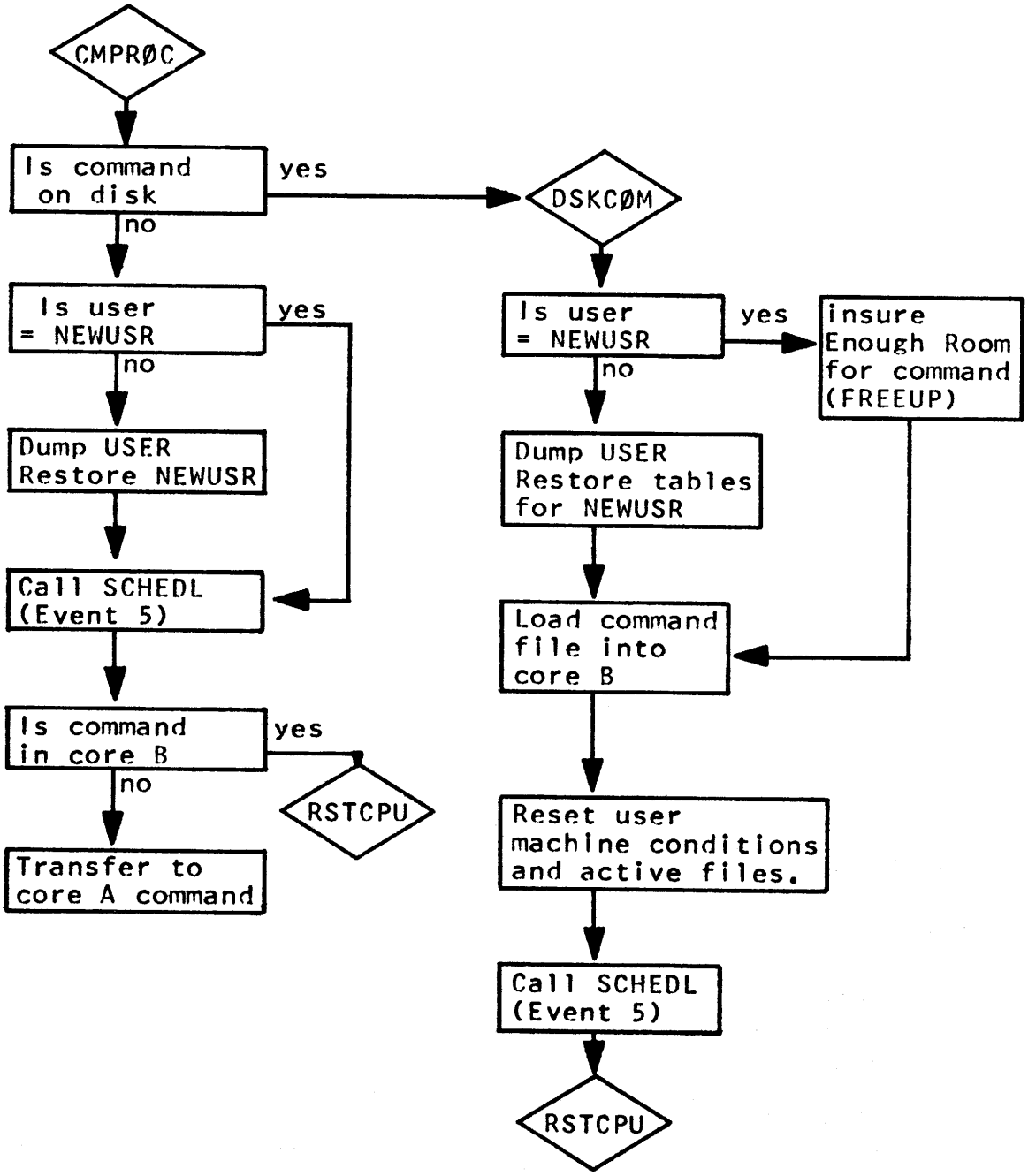


Figure 5.2 -- Command Processing in Main Control.

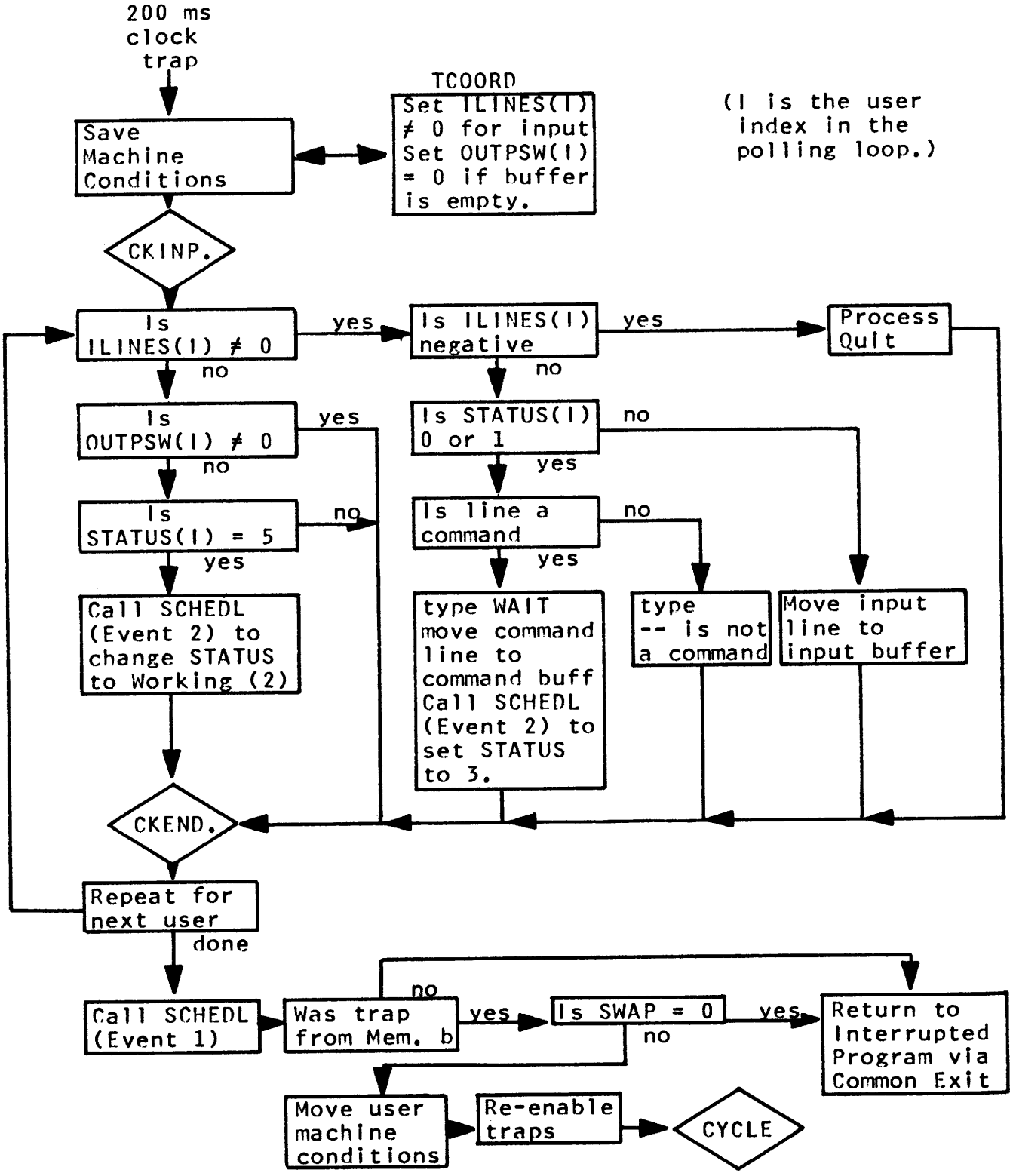


Figure 5.3 -- The Clock Trap Processor.

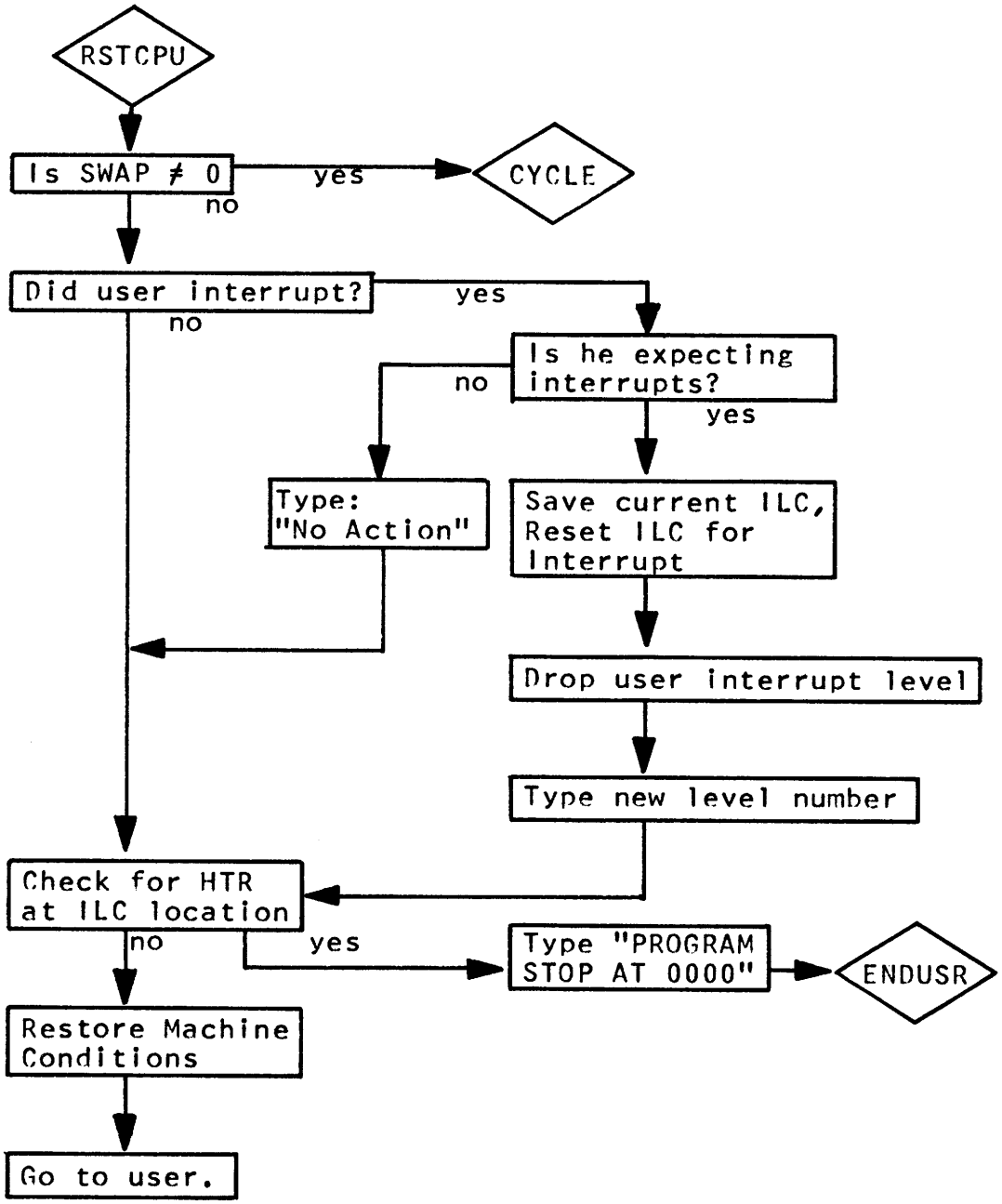


Figure 5.4 -- Flow diagram of RSTCPU (Restart user.)

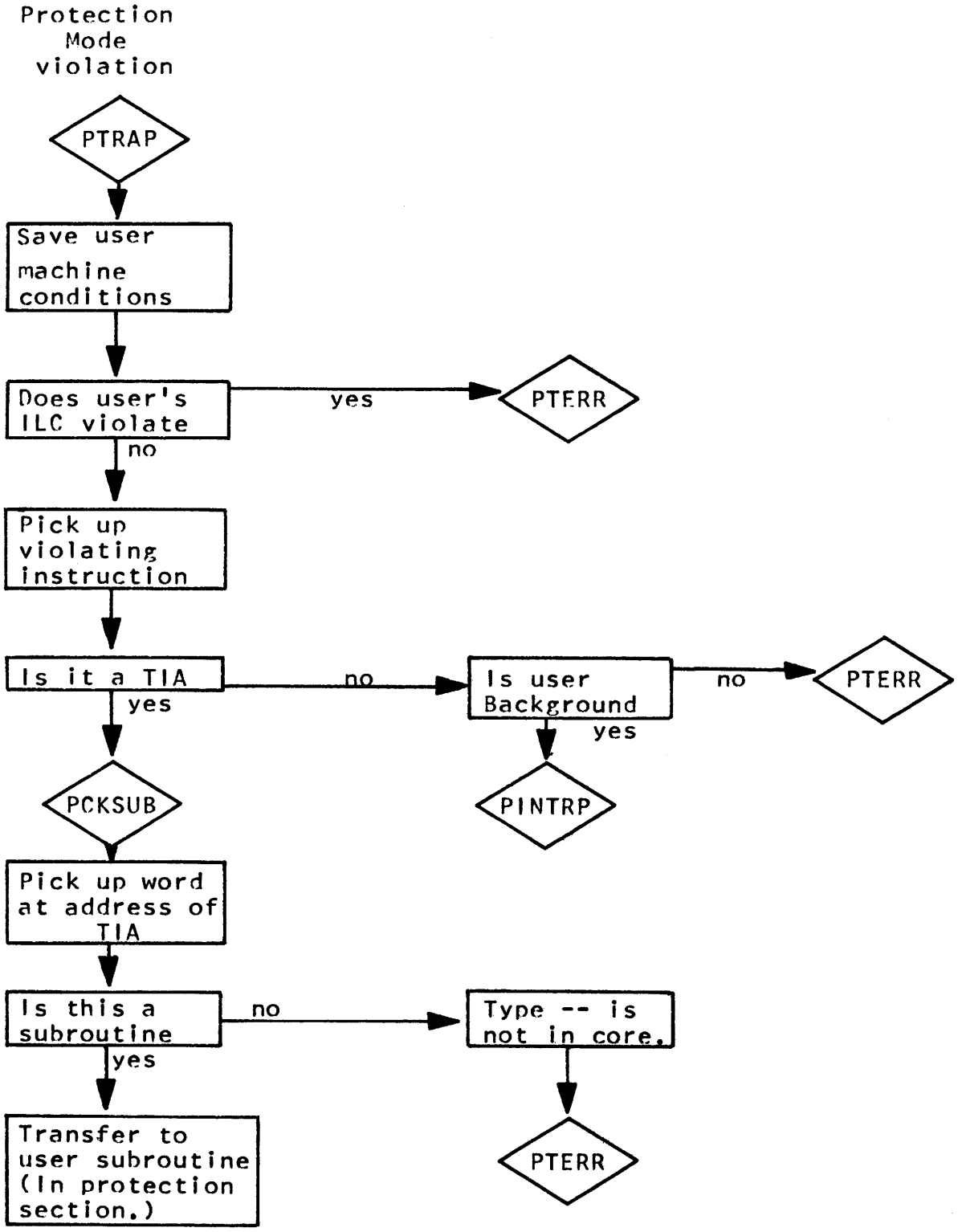


Figure 5.5 -- The Protection trap processor.

6. The Disk Control Module.

Introduction.

As mentioned earlier, there are four distinct uses of the disk and drum memories: 1) temporary storage of working programs which do not fit into memory; 2) storage of supervisor command programs; 3) storage of user programs and data; and 4) scratch pad storage by user programs. These uses have enough in common, however, that a single interface program, the Disk Control Module, handles all use of these memories. Calls from the supervisor are not distinguished from calls from a user program. In particular, the supervisor does not attempt at any time to use the disk except through the disk control module.

In addition to this disk control program, there is a pair of disk load and disk dump programs which are used for off-line input to and output from the disk memory. These routines are not part of the supervisor and in fact do not presently operate while the time-sharing system is running. The dump routine copies the contents of the disk memory onto tape for backup purposes, and processes users' request cards to produce printed and punched copies of their personal files. The load routine copies a tape onto the disk to re-initialize the time-sharing system, and also processes users' request cards to add files (consisting of punched cards) to the disk.

The Disk Control Routines.

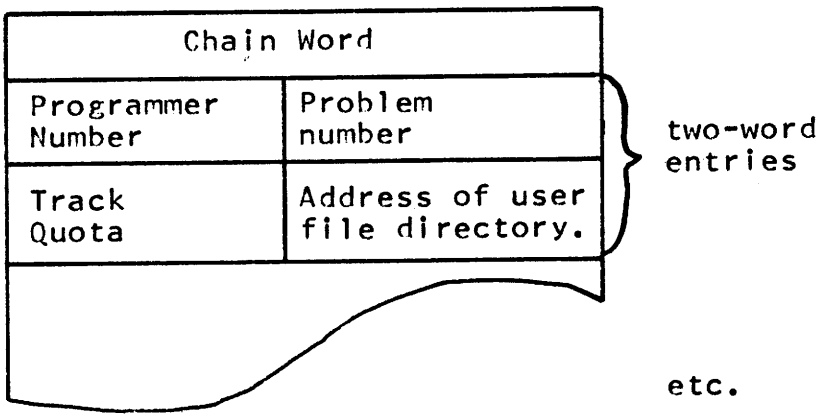
A complete, though slightly out-of-date, technical description of the current disk control module may be found in the Computation Center Memo CC-196, July 11, 1962. A new disk control module is currently being designed.

Loading and Dumping the Disk.

A complete technical description of the two programs LDEDT (disk load editor) and DPEDT (disk dump editor) may be found in the Computation Center Memo CC-108, May 9, 1963. An operational description is provided in Memo CC-212.

Disk Routine Tables.

The format of the master file directory is shown in figure 6.1. Figure 6.2 is the layout of the user disk status table as it appears in the disk routine and as it appears on the drum or disk when a user leaves core memory.



(The first track of the M.F.D. is stored on track 0 of Module 1.)

Figure 6.1 -- Format of Master File Directory

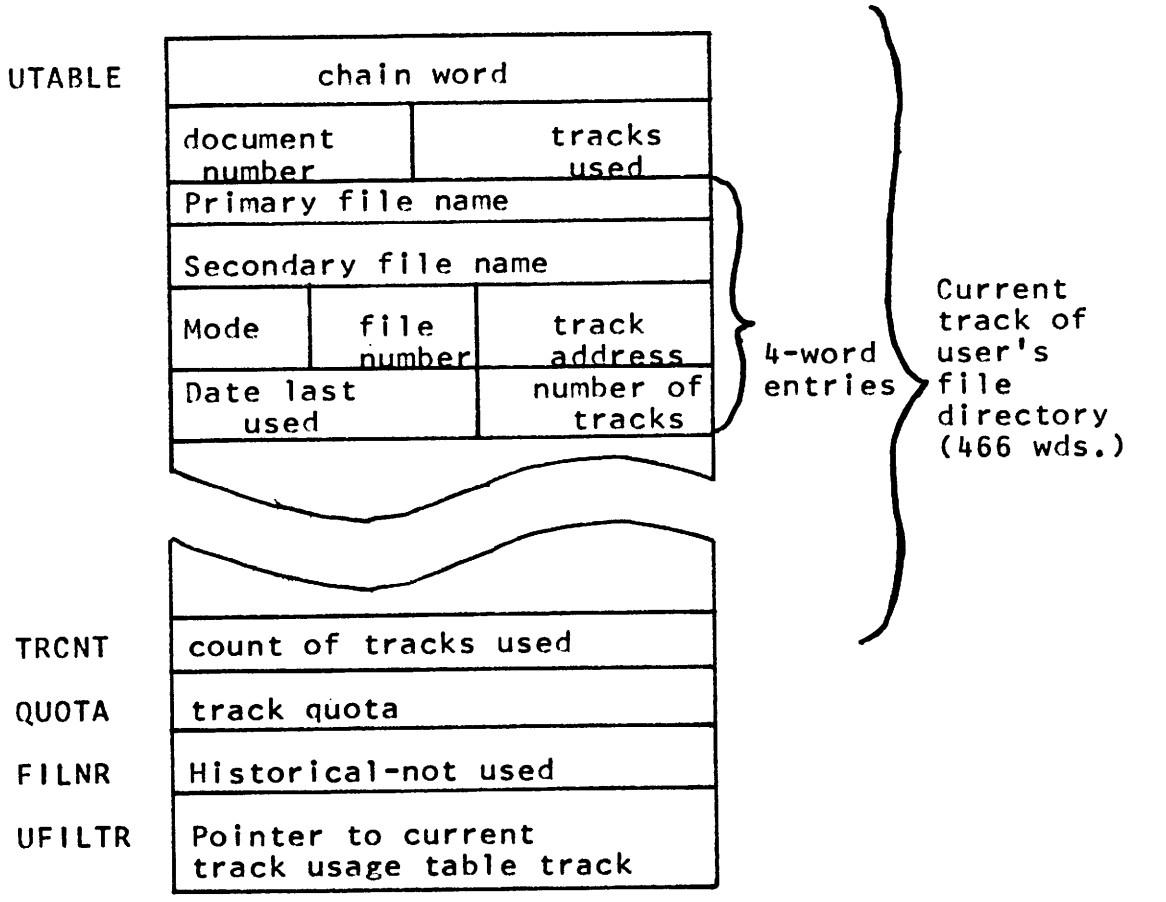


Figure 6.2 -- The Disk Status table. total size, 551 words.

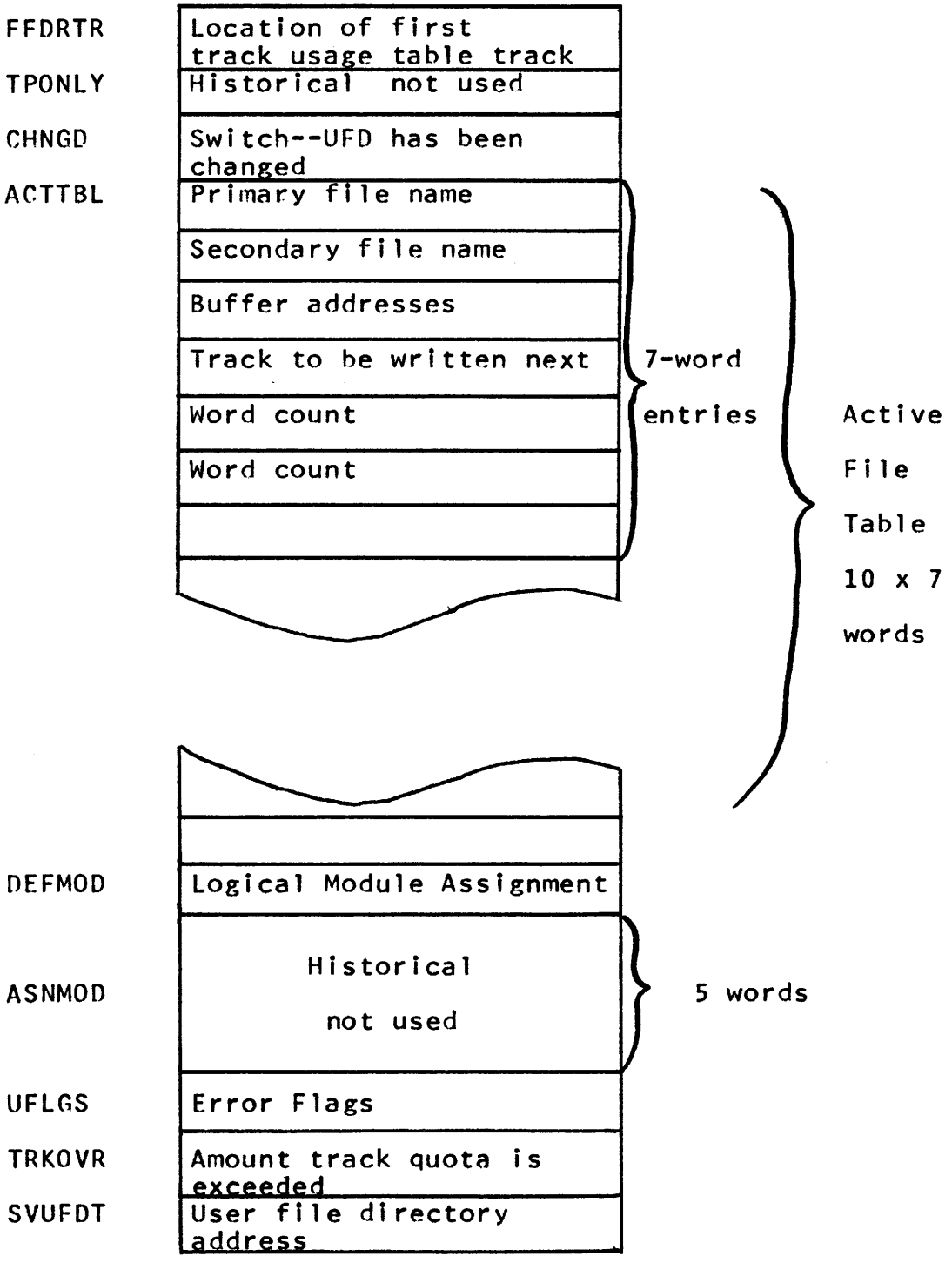


Figure 6.2 (cont.) The User disk status table.

7. Description of Entry points and Cross Reference Table

Introduction.

An invaluable aid in studying the operation of any single module or subroutine within a module is a thumbnail sketch of each of the subroutines which it calls. If such a sketch is available, attention can remain within the module being studied; the reader need not have a detailed understanding of how the subroutine works to comprehend the program which calls the subroutine. Also, when looking at the thumbnail sketch of a subroutine to figure out what it does, it is useful to have some idea of which other modules also call this subroutine; this allows one to establish the "place" within the system of the subroutine. In this section, then, is listed each entry point of each module, a brief description of what the entry point does, and a list of all modules of the supervisor which call this entry point. The information in this section pertains to version "1A1" of the time-sharing supervisor. All program sizes are given in octal.

Module ADPI

Function: 7750 I/O adapter module.

Size: 2340 words.

Entry Points:

WRTELY	Subroutine to write a line on a teletype.
RDTELY	Subroutine to accept characters from a teletype. callers: CHNE (ETRAP)

Module: AP75

Function: Handles 7750 buffers.

Size: 447 words.

Entry Points:

WT7750	Write output on 7750. callers: ADPI, HIGH

Module: CHNE

Function: Hardware routine to drive data channel E (7750 and teletypes). Contains 7909 channel programs.

Size: 363 words.

Entry Points:

CHANLI	Subroutine to initialize 7750 and channel E. callers: CTRL, MAIN
WR7750	Subroutine to transmit data to 7750. callers: AP75
ETRAP	Channel E data channel trap entry. callers: MAIN, channel E data channel trap.
STØPE	Shut down channel E for high-speed drum. callers: STØR
STARTE	Restart channel E after STØPE has stopped it. callers: STØR

Module: CLØC

Function: Clock trap processor module.

Size: 1377

Entry Points:

STCLØC	Subroutine to start up interval timer clock. callers: CTRL, MAIN
CLKINT	Interval timer trap entry. callers: MAIN; clock trap.
GETOTL	Get time system has been running. callers: SCHED
ADDTIM	Update time used. callers: CTRL, STØR

Module: CØMC

Function: Miscellaneous subroutines.

Size: 146

Entry Points:

ENKEYS	Subroutine to enter console keys of interrupt to CTSS. callers: CLØC, CTRL, PMTA
STOPIF	Subroutine to stop if key 24 is down. callers: PMTA, RTRN
CØMCHK	Subroutine to search command directory for command in logical AC. callers: CLØC, CTRL
SETUSR	Subroutine to establish current user as a disk user. callers: CTRL, LØGA, PMTA, SAVC STØR

BRØOM	Scan adaption system. callers: CLØC, TCØR, PMTA, RDFX, LØGA
SWEEP	Continue BROOM scan. callers: CLØC, TCØR, PMTA, RDFX, LØGA

Module: CØMD

Function: The command directory.

Size: 372

Entry Points:

CØMDIR	Entry to command directory control word. caller: CØMC (CØMCHK) SCHED
FILE	Direct entry to FILE entry in command directory. callers: CLØC
LØGIN	Direct entry to LØGIN entry in command directory. callers: CLØC, CTRL, PMTA, STØR
ENDLØG	Direct entry to ENDLØG entry in command directory. callers: CLØC, CTRL, LØGA
START	Direct entry to START entry in command directory. callers: CLØC
TFILE	Direct entry to TFILE entry in command directory. callers: CLØC

Module: CØNV

Function: Conversion routines.

Size: 2468

Entry Points:

CTIME	Subroutine to convert time in 60th to BCD 1/10th minutes. callers: LØGA, LØGB
TCTIME	Subroutine to convert time in 60th to BCD in .01 hours. callers: LØGA, PMTA
DTBC	Subroutine to convert decimal to binary. callers: CØMC(SETUSR), LØGA, LØGB, ØNLN, PMTA
BTDC	Subroutine to convert binary to decimal. callers: LØGA, PMTA
OTBC	Subroutine to convert octal(BCD) to binary. caller: ØCTC
BTØC	Subroutine to convert binary to octal BCD. callers: ØCTC, PMTA, STØR, RTRN
RDYTIM	Subroutine to obtain user command time used, to be typed with the "READY" comment. callers: CLØC, CTRL

Module: CTRL

Function. Main control module.

Size: 1160

Entry Points:

CHNCØM	Entry to pick up next program-initiated command if any. callers: ØCTC, PMTA, SAVC
NEWCØM	Entry to set up new command for user. callers: PMTA

CØLD	Entry to restart system after XEC loop, etc. callers: EDBG(PANIC), MAIN
DEAD	Entry to place current user in "DEAD" status. callers: LØGA, PMTA, SAVC, STØR
ENDUSR	Entry to set user status and type ready. callers: ØCTC, PMTA, RTRN
ENDCØM	Entry at end of command - ready not typed. caller: LØGA, PMTA
CYCLE	Entry to check for more work to do. caller: CLØC, PMTA, RTRN
CKQUIT	Subroutine to find if current user has pushed "QUIT" while in supervisor. caller: PMTA, RTRN
ILLCØM	Entry after illegal sequence of commands. caller: ØCTC, SAVC

Module: DCER

Size: 2168

Entry Points:

EPRINT	On-line print subroutine (saves channel A). caller: CHNE, DSKI, STØR
ALLSAV	Subroutine which saves all basic machine conditions. callers: CHNE, DSKI
ALLRST	Subroutine which restores all basic machine conditions. callers: CHNE, DSKI

Module: DSKI

Function: Disk control module

Size: 11506

Entry Points:

.DINIT	Initialize disk routine. caller: MAIN
.OPEN	Sign user on to disk. callers: CØMC(SETUSR), CTRL, LØGA, MAIN, PMTA, SAVC
.CLOSE	Remove user from disk file. callers: LØGA, LØGB, PMTA
.ASIGN	Initialize writing a new file. callers: LØGB, PMTA, SAVC, STØR
.APEND	Add to end of an old disk file. callers: LØGB, PMTA, SAVC
.WRITE	Write data with a disk file. callers: LØGB, PMTA, SAVC, STØR
.FILE	Terminate writing of a file. callers: LØGB, PMTA, SAVC, STØR
.RELRW	Open a file for relative read/write. callers: LØGB, PMTA, SAVC
.SEEK	Initialize a disk file for reading. callers: CTRL, LØGB, PMTA, SAVC, STØR
.READ	Read data from a disk file. callers: CTRL, LØGB, PMTA, SAVC, STØR
.ENDRD	Terminate reading from a disk file callers: CTRL, LØGB, PMTA, SAVC, STØR
.DLETE	To delete a disk file and its tracks. callers: PMTA, SAVC

.CTEST Check if a disk channel is in operation.
(not used.)

.GTFLG Pick up error or control flags.
callers: PMTA

.SETDU Set current disk user.
callers: CTRL, LØGB, MAIN, PMTA, SAVC, STØR

.STATL Get location of disk user status tables.
callers: MAIN

.FILDR Read a track of user file directory.
callers: PMTA

.UPDAT Update user file directory onto disk.
callers: PMTA, SAVC

.DFINE Define a new logical module number.
(not used.)

.RESET Reset all files in active status.
callers: CTRL, PMTA, SAVC

.FSTAT Obtain information about a file.
callers: CTRL, PMTA

.SETAB Set memory switches for A or B.
callers: CTRL, LØGB, MAIN, PMTA, RTRN, SAVC, STØR

.RDWAT Read out and reset channel waiting time.
callers: CTRL, SAVC, STØR

.CHECK Wait until all disk activity is finished.
callers: STØR

.STKER Set error return on disk track error.
callers: CØMC(SETUSR), CTRL, LØGA, LØGB, MAIN, STØR, RTRN

.ERASE Delete a file from directory, but leave its tracks.
callers: PMTA, STØR

.GETDS	Get status of active disk files. callers: SAVC
.RENAM	Change name or mode of a file. caller: PMTA

Module: EDBG

Function: System debugging aids

Size: 764

Entry points:

PANIC	Entry to take a panic dump of both cores. callers: Console operator's restart: MAIN
ADUMP	Subroutine to dump memory A. caller: CLØC
BDUMP	Subroutine to dump memory B. caller: CLØC
TRACE	Subroutine to print out a trace of all traps. caller: CLØC

Module: HIGH

Function: High Speed line adapter

Size: 252

Entry Points:

RDHIGH	Read high speed line. caller: ADPI
WRHIGH	Write high speed line. caller: ADPI

Module: LØGA, B, C

Function: Login and logout commands and associated subroutines.

Size: A 1713

B 1206

C 131

Entry Points:

LOGIN.	TSS login command. caller: PMTA
LØGERR	Entry in case of error setting up user's file directory. callers: CØMC(SETUSR)
ENDLG.	TSS Automatic logout entry. caller: CØMDIR
LØGØUT	TSS logout command. caller: CØMDIR

Module: MAIN

Function: Main Program

Size: 2205

Entries: (MAIN PROGRAM) Initialize system, set up trap returns, start system running.

Called By: System loader

Module: MTRA

Function: TSS system statistics collector.

Size: 6 (Dummy not presently used)

Entry Points:

MØNITR Entry to put away statistics.
caller: CLØC

EREAD Entry to obtain collected statistics.
caller: PMTA

Module: ØCTC

Function: Octlk, Octpat, and Octtra commands

Size: 200

Entry Points:

ØCTLK: TSS ØCTLK command for core B.
caller: CØMDIR

ØCTPAT: TSS ØCTPAT command for core B.
caller: CØMDIR

ØCTTRA: TSS ØCTTRA command for core B.
callers: CØMDIR

Module: ØNLN

Function: On-line device manipulators for supervisor

Size: 420

Entry Points:

CLØCIN Read data and time from chronolog clock.
callers: LØGA, MAIN

PRINT Print 72 character line on-line.
callers: CTRL, LØGA, MAIN, PMTA, RTRN, STØR

PUNCH — Punch a card on-line.
caller: LØGA

Module: KLUD (PLØT at Computation Center)

Function: Channel D I/Ø Adapter.

Size: 2161 at Center, 7173 at MAC

Entry Points:

PLØTS — Plot statistics on 7094 scope (center).
caller: CLØC

DSCØPE — Display information on ESL scope.
caller: PMTA

DTRAP — Channel D trap entry.
callers: MAIN, channel D data channel trap.

DDTRAP — Direct data trap entry.
callers: MAIN, direct data trap.

STØPD — Stop data channel D for High Speed Drum.
Caller: STØR

STARTD — Restart data channel D after High Speed Drum.
caller: STØR

Module: PMTA

Function: Protection trap processor.

Size: 5144

Entry Points:

PTRAP — Entry on protection mode trap.
callers: MAIN; Protection mode violation.

BKSERV	Entry to service background (requested by Keys). caller: CTRL
ERRØR	Entry to comment after a disk error. (not used)
DPRINT	Entry to print out a disk error on-line. caller: MAIN
DERR	Entry to process a disk error. caller: MAIN
USERER	To process disk track error for user. callers: CTRL, RTRN

Module: RFLX

Size: 2705

Entry Points:

RDFLXA	Entry for a user to obtain a line from common input buffer. caller: PMTA
ENTLIN	Entry for supervisor to add a line to common input buffer. callers: CLØC, PMTA
RSSRB	Entry to reset all input lines for this user. callers: CLØC, CTRL, PMTA, TCØR
RSSWB	Entry to reset user's output lines. callers: CLØC, CTRL, LØGA, TCØR

Module: RTRN

Size: 241

Entry Points:

RSTCPU	Entry to return to user program. callers: CTRL, LØGA, MAIN, PMTA, SAVC
CMEXIT	Entry to return to interrupt program. callers: CHNE(ETRAP), CLØC DSKI, PLØT, UTRP
CMXRTN	Cell containing return location. callers: EDBG(TRACE)
LØAD.4	Cell containing IR4. callers: EDBG(TRACE)

Module: SAVC

Function: Save, restore, resume and start commands.

Size: 624

Entry Points:

XDUMP	This is the SAVE command entry point. callers: CØMDIR
SAVE	Subroutine to save a user. (Used by XDUMP and ENDLØG). callers: LØGA(ENDLØG)
RESUME	TSS resume command. callers: CØMDIR
XLØAD	TSS restor command. caller: CØMDIR
START	TSS start command. caller: CØMDIR

Module: SAVR

Function: Save and restore routine.

Size: 2468

Entry Points:

SAVCPU Subroutines to save basic user machine conditions.
callers: CLØC, PMTA

RESTØR Subroutine to restore basic user machine conditions.
callers: PMTA, RTRN

LØNGSV: Subroutine to save complete user machine conditions.
callers: STØR

LNGRST: Subroutine to restore complete user machine conditions.
callers: SAVC, STØR

Module: SCDA, B, C, D, E, F, G, H

Function: Scheduling, time accounting, and monitoring, including all subroutines.

Size:

A	1567	F	266
B	421	G	73
C	523	H	20
D	22		
E	174		

Entry Points:

SCHEDL Entry to notify scheduling algorithm that something has happened.
callers: CLØC, CTRL, LØGA, MAIN, PMTA, SAVC, STØR, ADPI

MØNSCD,MØNINF Monitoring
caller: PMTA

Module: STØR

Function: Storage allocation algorithm module.

Size: 2165

Entry Points:

DUMP	Subroutine to dump user onto disk. callers: CTRL
UNDUMP	Subroutine to restore a user from the disk. callers: CTRL
FREEUP	Subroutine to freeup N words of memory B. callers: CTRL, PMTA, SAVC

Module: TCØR

Function: Typewriter coordinator - break processor.

Size: 5205

Entry Points:

TCOORD	Subroutine to collect input characters into messages and look for break characters. callers: CLØC
WRFLX	Subroutine to write a message on a typewriter. Follow message with a carriage return. callers: CLØC, CTRL, LØGA, LØGB, ØCTC, PMTA, RTRN, STØR, SAVE
WRFLXA	Subroutine to write a message on a typewriter. No carriage return provided. callers: CLØC, LØGA, PMTA
RSSRB.	Subroutine to reset a user's secondary read buffer. callers: RFLX
TOPOOL	Subroutine to place an input character in the character pool buffer. callers: ADAP(RDTELY), HIGH

RDLINE Obtain an input line from break processor.
caller: CLØC

Module: TSTØ

Function: 7750 storage allocator. (MAD)

Size: 254

Entry Points:

TGET Subroutine to obtain a block of 7750 storage.
callers: AP75

TGIVE Subroutine to give back a block of 7750 storage.
callers: ADAP(RDTELY), HIGH, AP75

TRESET Subroutine to discard all of a user's present output stored in 7750.
callers: AP75

Module: UNIT

Function: Assigns logical unit numbers to consoles

Size: 247

Entry Points:

ASNUNI Subroutine to assign a logical unit number to a physical unit.
callers: ADAP, HIGH

US2BS Subroutine to look up a physical unit number given a logical unit number.
callers: ADPI, AP75

BS2US — Look up logical unit number given physical unit number.
callers: ADPI, AP75

ERSUNI — Subroutine to erase logical unit assignment.
callers: CLØC, LØGA

Module: UTRP

Function: Process user traps.

Size: 4228

Entry Points:

ATRAP — Process data channel trap from channel A.
callers: MAIN, channel A data channel trap.

BTRAP — Process data channel trap from channel B.
callers: MAIN, channel B data channel trap.

STRTRP — Process STR in a user's program.
callers: MAIN, STR trap.

FLPTRP — Process floating point traps in user program.
callers: MAIN, floating point trap.

CTRAPS — Subroutine to check for waiting traps.
callers: RTRN

UPCLØC — Subroutine to update core clock for current user.
callers: CLØC

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
Project MAC
TR-16 CTSS Technical Notes

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical description of 7094 Compatible Time-Sharing System

5. AUTHOR(S) *(Last name, first name, initial)*
Saltzer, Jerome H.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 15, 1965	77 + IV	3

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
Nonr 4102(01)	TR-16
b. PROJECT NO. DSR 9457	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES
No limitation

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
ADVANCED RESEARCH PROJECTS AGENCY through
OFFICE OF NAVAL RESEARCH

13. ABSTRACT

This report is a technical description of the 7094 Compatible Time-Sharing System in use at Project MAC and the M.I.T. Computation Center. It is designed to acquaint a system programmer with the techniques of construction which were used in this particular time-sharing system. Separate chapters discuss the overall supervisor program flow; console message input and output; the scheduling and storage algorithms; and a thumbnail sketch is given of each of the subroutines which make up the supervisor program.

This report was prepared with the aid of the compatible time-sharing system and the TYPSET and RUNOFF commands.

DD FORM 1 JAN 64 1473

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computers Multiple-access computers On-line computer systems Real-time computer systems Time-sharing Time-shared computer systems						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.