COMPUTATION CENTER
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

TIME-SHARING SYSTEM NOTES

Introduction

This document is a supplement to the CTSS Programmer's Guide*. It contains, as well as supplementary information, corrections to some of the specifications given in the Guide, and periodic reports on some features which have been implemented since the publication of the Guide. The notes will be periodically revised and updated.

Additions, corrections and amplifications are organized with reference by chapter and page to the Programmer's Guide. There will be some additional sections which contain material not specifically related to chapters in the Guide. Logical subdivisions of the material will be dated.

*The Compatible Time-Sharing System: A Programmer's Guide, M.I.T. Computation Center, M.I.T. Press, Cambridge, 1963.

November 20, 1963

This edition contains primarily addenda and errata to Chapters 4 and 5, the sections on Supervisor Subroutines and Disk Control Subroutine calls. Given below is a list of the items discussed in the pages that follow. Note especially the description of Automatic Logout and the new supervisor subroutines for command chaining.

Automatic Logout (page 30) - interim version

Background System Restrictions (pp 35-38) - new supervisor subroutines RSTIME and TRA 1,4

Supervisor Subroutine Calls (Chapter 4) - minor amendments to specifications for the following supervisor entries:

RDFLXA

DEAD, DORMNT

SETMEM

TSSFIL, USRFIL

SAVBRK

GETILC

INSTRT, INPEND

New Supervisor Subroutines having to do with execution of a chain of commands: SETCLS, GETCLS, SETCLC, GETCLC, CHNCOM.

Disk Control Subroutine Calls (chapter 5) - minor amendments to the specifications for the following disk routine calls:

.LOAD

.READK

.ENDRD

.WRITE

.CLEAR

.FILDR

Clarification of the meaning of Disk Error Conditions Codes.

Disk Editor Control Cards - current restrictions.

GENERAL DESCRIPTION AND USAGE TECHNIQUES (Chapter 3)

<u>Automatic Logout</u> (page 30)

At the present time, an interim form of automatic logout is provided. When the time-sharing system is shut down, either on a scheduled basis or, when possible, on an emergency basis, each active user is interrupted and is automatically logged out. This is exactly as if the user had issued a quit signal followed by a <u>logout</u> command. Any disk files which the user has created during his session are correctly recorded on the disk, but any files which are in active status at the time of automatic logout are reset (see page 50). At present no <u>saved</u> file is created. It is worth noting here thatif the user is creating a large file by use of the <u>input</u> command, all his input will be lost if he runs overtime; therefore, it would be wise for the user to keep an eye on the clock. Also, where the input is extensive and tedious to retype, it is a good idea to issue a <u>file</u> command from time to time, and then continue input via the <u>edit</u> command. The precautions will become virtually unnecessary in the future, when a version of automatic logout will be implemented which will include creation of a <u>saved</u> file, automatic extension of track quota, and preservation of disk status.

11/20/63

<u>Background System Restrictions</u> (pp 35-38)

Two additional supervisor subroutines are provided for the use of the background system only. At the start of a background job the simulated interval timer clock is set to zero by the call

```
               TSX  RSTIME,4
```

which is issued by the FMS record SIGN-ON. (The location RSTIME must contain TIA = HRSTIME.)

An entry is provided whereby a program may determine whether or not it is running as background to the time-sharing system. The call:

```
               TSX   T,4
```

with

```
           T   TIA   *+1
               TRA   1,4
```

will cause a trap to the supervisor if and only if the job is running as background (i.e., running in B core). If the program is running in A core, this sequence will simply cause a return to (1,4); if it is running as background in B core, the supervisor will effect a return to (2,4).

11/20/63

SUPERVISOR SUBROUTINE CALLS (Chapter 4)

<u>RDFLXA</u> (page 40)

This subroutine has been slightly modified to allow for reading in 12-bit mode. This description supercedes the description given in the manual. The call

```
        RSX   RDFLXA,4
        PZE   BUFF
```

reads 14 words into memory starting at BUFF. On return, in both 6-bit and 12-bit mode, the logical AC contains N, the number of 6-bit bytes read, up to and including the break character (that is, for 6-bit mode the break is the Nth character; for 12-bit, the break is the N/2 character). Blanks are filled in to the right of the break character. If the break character was not received in the first 14 words of the input line, then bit 21 of the AC will be 1, indicating that another call to RDFLXA is required to continue reading the line.

11/20/63

<u>DEAD, DORMNT</u> (pp 40-41)

By way of clarification of the function of these entries, in particular with regard to priority levels, this description supercedes that given in the manual. The call

```
        TSX   DEAD,4
```

returns control to the supervisor and puts the user in dead status, i.e., his machine conditions are not saved. In addition, the user's memory bound is set to zero; as a result, upon issue of a new command his level will not be prejudiced by the old setting of the memory bound.

```
        TSX   DORMNT,4
```

returns control to the supervisor and puts the user in dormant status, i.e., his machine conditions and the current status of his program are saved. His memory bound is also saved. If a command is issued which causes a new program to be read into memory, the old machine conditions, status, and memory bound are over witten. If the <u>start</u> command is issued, control returns to (1,4).

11/20/63

Change Of Memory Bound - SETMEM (page 41)

With regard to the use of SETMEM to extend the user's memory bound, it should be noted that in the current implementation the memory bound is set by a word count in the AC, which is taken to be the number of words of memory which the user wants to reference. This presents a problem only when the user needs all of core, as he cannot specify 32,768 locations in a 15-bit address. However, a word count of 77777_8 will, in fact, make all of core available to the user.

Memory protection in the 7090/94 is controlled by the protection indicators (a hardware modification) which are 7-bit registers; the setting of these registers represents the high-order 7 bits of a memory address, so that protection is implemented on the basis of 256-word blocks. Thus, there are two numbers relevant to the user's memory allotment, the word count (his 'memory bound') and the block count, which is the unit of protection and the quantity used for swapping users in and out of core. It is apparent that the word count is in fact of interest only to the user, who may need to maintain an accurate record of his memory bound for the purpose of making small changes to it.

At present the supervisor dumps users by block count and restores them by word count. A more consistent approach would be for the supervisor to concern itself with block count only. In the future, GETMEM and SETMEM will probably be modified as follows: SETMEM will set the user's memory bound to the nearest block count greater than or equal to the word count specified in the AC; the specified word count will be kept in the supervisor, however, and will be returned in the AC on a call to GETMEM. Thus, the call

```
CAL   =77777
TSX   SETMEM,4
```

would be equivalent to

```
CAL   =77401
TSX   SETMEM,4
```

but subsequent calls to GETMEM would not return the same word count in both cases. In this way, the sequence

```
TSX   GETMEM,4
ACL   =14
TSX   SETMEM,4
```

would still be a meaningful and efficient way to increment the memory bound.

11/20/63

TSSFIL, USRFIL (page 42)

The following is an expansion of the description of these entries given in the manual. The call

```
               TSX   TSSFIL,4
```

allows the user to read time-sharing system files from the disk (e.g. the CTSS library). After this call, only system files may be read; to reset the mode, the user must give the call

```
               TSX   USRFIL,4
```

which allows him to reference his own files again. After a call to TSSFIL, all disk references are interpreted with reference to the system file directory (until a call is issued to USRFIL); however, the user is permitted to issue only the disk calls .SEEK, .READK, .ENDRD, and .LOAD. When a call is issued to either TSSFIL or USRFIL, any active files in the file directory currently referenced are reset (see description of .RESET, page 50) before the switch is set to reference the other file directory.

11/20/63

SAVBRK (page 42)

Upon return from this subroutine, the AC contains an instruction TRA QL where QL is the quit entry point assigned to the quit level just left. If the command level (level 0) is reached, the contents of the AC is zero. It has been suggested that a better way to ensure that a zero AC has unambiguous significance would be to put in the decrement of the AC the number of the level to which QL corresponds. Until this is implemented, however, it is only necessary to remember that the left half of the AC contains a TRA for non-zero levels.

11/20/63

GETILC (page 43)

On return from this subroutine, the AC contains an effective STZ ILC where ILC is the value of the instruction location counter at the time the user last went dormant; the STZ is due to the fact that bits 3 and 4, turned on by a condition associated with the two-core RPQ, are not masked out by the supervisor. This error will be corrected in the near future.

11/20/63

INSTRT, INPEND (page 43)

This is an elaboration of the description given in the manual. These two subroutines are used by the input and edit commands; they are very rigid in format and requirements, and must be regarded as special-purpose routines. Since they were not intended for general use, they were coded for maximum efficiency in the supervisor, and without any regard for external flexibility or consistency. They are described here primarily to clarify the mechanism of the input mode.

The call:

```
          TSX   INSTRT,4
```

puts the user into a console input mode. The contents of the logical AC is taken as an initial line number, in BCD, without alphabetic characters. The contents of the MQ is taken as a line-number increment, in binary, less than 2^9. The supervisor types out the initial line number and returns control to the program; when a line is entered it may be read by RDFLXA. The supervisor continues to generate line numbers, incrementing by the specified increment, each time a line is read. (If no increment is provided, an increment of ten is assumed.) According to the conventions described under the input command (Chapter 7), a manual mode may be entered where the supervisor suspends generation of line numbers, and types out "MAN." instead.

In addition to typing out line numbers or "MAN.", the supervisor inserts whatever it has typed out into the 14th word of the input line; line numbers are in BCD, left-justified, with one blank to the right; leading zeros are explicit, making a full five characters; if the line was entered in manual mode, the fourteenth word contains the characters MAN. with one leading and one trailing blank.

The supervisor expects the terminal line of the input to be a command line, entered in the manual mode; at present the only command which is accepted is file. Upon interpretation of this command, the supervisor enters the command into the command buffers but does not execute it; control returns to the program which must prepare for the next command in whatever manner is required. The supervisor does not transmit the command line to the program, but transmits instead a line with a first word of all 7's. The program must then issue the call:

```
          TSX  INPEND,4
```

which causes the supervisor to go on to the waiting command.

New Supervisor Subroutines (page 43)

Five new supervisor subroutines permit execution of a chain of commands. By execution of a program using these routines, up to five commands may be chained in a designated sequence. Along with the user's machine conditions are kept five buffers containing the command lists (command name plus arguments) of the commands in the chain, and a command location counter (CLC) to govern the chaining. The designated commands are numbered from 1 to n ≤ 5, in the order of execution; the CLC contains the number of the next command to be executed and the number of the terminal command in the chain. The command lists may be set, read, and modified, as may the CLC.

The call

```
          TSX   SETCLS,4
          PZE   An,,n
```

where

```
An        BCI   1,NAME
          BCI   1,ARG1
            . . .
          ØCT   777777777777
```

sets the nth command list to the command NAME and the associated arguments. The word of sevens marks the end of the list.

The call

```
          TSX   GETCLS,4
          PZE   BUFF,,n
```

fills the 20-word buffer BUFF with the contents of the nth command list.

It must be noted that the Supervisor does not scan the command list before moving it in or out of the user's buffer. Twenty words are copied; the fence of sevens is interpreted by the particular command program itself. Consequently the user must provide a 20-word buffer for GETCLS, regardless of the length of the command list, and care must be taken that for short lists SETCLS does not produce a protection violation.

In the near future some modifications will be made to the method of storing command lists, which will affect GETCOM as well as GETCLS and SETCLS. In particular, use of the fence of 7's will be eliminated and instead an extra word will specify the number of words in each list. This will simplify the use of GETCOM (q.v.) in that a single call will be used to obtain the entire argument list. At the same time, the inelegance of blindly transmitting twenty words on GETCLS and SETCLS will be removed.

To set the CLC, a call

```
          TSX    SETCLC,4
```

is given with the contents of the AC as follows:

```
          PZE    m,,n
```

This will set the CLC so that the <u>mth</u> command will be executed next, and chaining will terminate after execution of the <u>nth</u> command. The call

```
          TSX    GETCLC,4
```

will return in the AC the current contents of the CLC. During execution of the <u>ith</u> command, the address of the CLC contains i + 1.

The call

```
          TSX    CHNCØM,4
```

causes execution of the next command according to the contents of the CLC. Every command program should issue a call to CHNCØM on normal exit, so that chaining may continue if the command is part of the chain. If there is no "next command" waiting, the call to CHNCØM will be equivalent to a call to DEAD or DØRMNT, depending on the contents of the AC: if the address of the AC is zero, it is as if a call to DEAD had been given; if the address of the AC is one, a call to DØRMNT is effected.

All standard system commands are in the process of being recoded to terminate in a call to CHNCOM.

11/20/63

USE OF THE 1301 DISK MEMORY (Chapter 5)

.LOAD (page 46)

A modification has been made to this subroutine. The following description supersedes the one given in the manual.

To load a continuous block of core from a file previously recorded on the disk, the .LOAD entry is provided:

```
TSX   .LOAD,4
PZE   FILNAM
PZE   A,,n
```

where A, n, and FILNAM are of the same form as in the .DUMP calling sequence. The prefix code and module number need not be specified. For loading files of undetermined length, n may be set to a larger value than necessary; if the user specifies a larger n than the number of words actually in the file, upon return the address of the AC will contain the number of words actually loaded. If n is less than the number of words in the file, n words will be loaded and the contents of the AC will be meaningless. This ambiguity will be removed in the near future, when .LOAD will be revised so that the AC will always contain the correct number of words actually read. Temporarily, however, the user may guarantee that the word count he specifies is greater than or equal to the number of words in the file (and so guarantee a correct word count on return) by first calling .FSTAT (page 51) and using the estimated word count from .FSTAT for the call to .LOAD.

11/20/63

.READK, .ENDRD (pp 48-49)

This description is a clarification of the one given in the manual, and describes several new features. To read information from a file initialized by a call to the .SEEK routine, any number of calls to the .READK entry may be used in the following form:

```
TSX    .READK,4
PZE    FILNAM
PZE    A, T, n
PZE    X
```

If T is zero, the first or next n consecutive words will be read from the file specified by FILNAM, and stored in consecutive locations starting with location A. If T is non-zero, it is taken as a "non-transmit" indicator; n words of the file are skipped over. If an attempt is made to read past the last word of the file, control is transferred to location X. When this occurs, the file being read is dropped out of active read status, and the buffers being used are available for other work. In addition, on an "end of file" return the address of the AC contains the number of words actually read on this call.

To terminate the reading of a file without having to read past the last word in the file, the following sequence is used:

```
TSX    .ENDRD,4
PZE    FILNAM
```

It must be noted that if the user has read past the end of the file, a call to .ENDRD is interpreted as an error (file not in active status - error code 1), inasmuch as the file has already been dropped from active status.

Relative Reading and Writing (page 49)

The call:

```
          TSX   .WRITE,4
          PZE   FILNAM,,RELADR
          PZE   A,,n
```

will write the n words starting at core memory location A into the n words of file FILNAM, starting at relative location RELADR, where RELADR is non-zero; (in other words, the first word of the file has relative address 1). It is not permitted to write beyond the existing limits of the file; i.e., the file must already contain at least RELADR + n - 1 words.

Similarly as in the .WRITE call, when the first parameter of a .READK call is

```
          PZE   FILNAM,,RELADR
```

with RELADR non-zero, the reading process starts from that relative address of the file. If the word count of the file is less than RELADR + n - 1, the normal end-of-file convention will be followed.

11/20/63

.CLEAR (page 50)

The call:

```
TSX   .CLEAR,4
PZE   FILNAM,,n
```

will write n zeros into the file specified by FILNAM. This call must be preceded by a call to .ASIGN in standard form; on return from .CLEAR, however, the file will automatically be dropped from active status and no call to .FILE is required.

Note that this is a temporary divergence from the original specifications. In the near future, the .CLEAR routine will be revised to follow the description given in the Programmer's Guide, i.e.:

```
TSX   .CLEAR,4
***   FILNAM,,n
```

will create a file called FILNAM of mode ***, and will write n zeros into the file. Buffer space will be supplied automatically, the module number may not be specified, and calls to .ASIGN and .FILE will not be required.

11/20/63

.FILDR (page 51)

With regard to the user file directory obtained by a call to .FILDR, the following clarification should be noted. The decrement of the first word of the last track contains the number of words in this track not counting the first. The address of the second word of the first track contains the user's total track count.

11/20/63

Disk Error Codes (page 52)

With regard to disk routine error conditions, Error Code 1, described as "Illegal calling sequence", includes the following error conditions:

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

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

c. Illegal call to the .FILE routine; this occurs if the call references a file in active read status.

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

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

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

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

h. Tried to rename read only class 2.

i. Attempt to delete file in raad only mode.

j. File (α,β) is not an active file; this occurs if a call to .WRITE, .CLEAR, .FILE, .READK, or .ENDRD references a file not in active status; note that this is a temporary restriction on . .CLEAR.

11/20/63

Disk Editor Control Cards (pp 52-54)

At present, the following disk editor functions are not working:

a. 7punch;

b. the CARDS option used with the Delete control card;

c. Link (this function was described as 'available soon'; in fact, it may be some time before it is available).

11/20/63