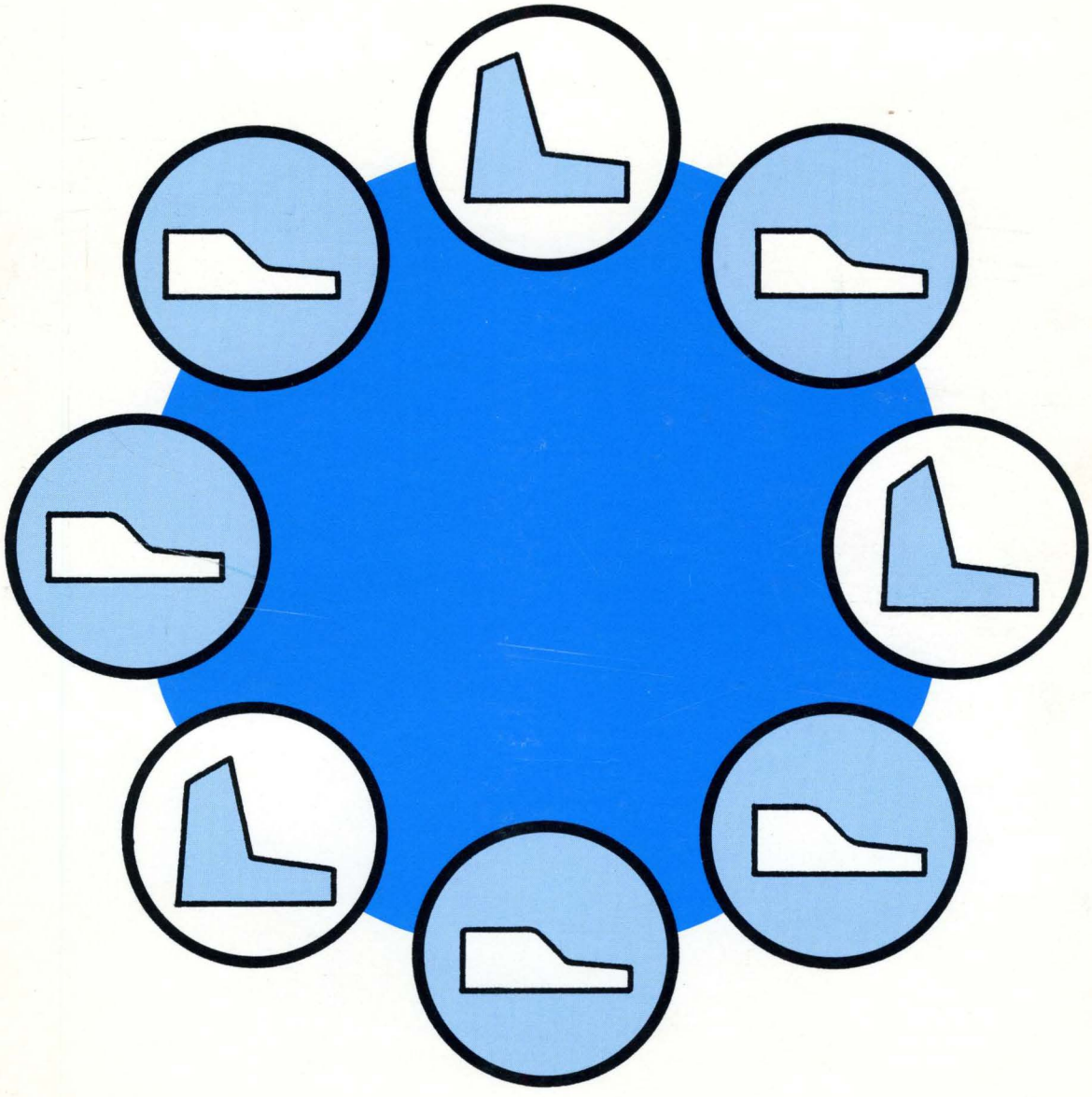
HEWLETT PACKARD

2100 Computer Systems



TERMINAL CONTROL SYSTEM

USER'S GUIDE

TERMINAL CONTROL SYSTEM

USER'S GUIDE

Computer Systems

HP Part No. 5951-7307
Microfiche Part No. 5951-7308

Printed: AUGUST 1973

© Copyright, 1973, by HEWLETT-PACKARD COMPANY, 11000 Wolfe Road, Cupertino, California, 95014.
All rights reserved. Printed in the U.S.A.

PREFACE

The desire for instant access to "state-of-the-business" information is a dominant force in most business planning activities. The ability to maintain current status information and to analyze and update data in many different combinations is recognized as the key to a sound business information system.

Managements across the country are combining the capabilities of the computer with modern programming techniques to achieve more efficient operations, improved use of corporate resources, tighter control and coordination of operating elements, and faster response to business transactions. One such programming technique is Hewlett-Packard's Terminal Control System (TCS).

This User's Guide describes the overall structure and capabilities of TCS, the hardware and software requirements, the calling sequences, and the proper methods of using TCS.

A familiarity with HP FORTRAN-IV or HP Assembler Language, and DOS-III is assumed. For information on these subjects, refer to the following publications (the HP part number is shown in parentheses):

- *HP ASSEMBLER Manual* (02116-9014)
- *HP FORTRAN-IV Programmer's Reference Manual* (5951-1321)
- *HP DOS-III Disc Operating System Manual* (02100-90136)

CONTENTS

section 1 INTRODUCTION

Hewlett-Packard's Terminal Control System (TCS) provides the user with a modular and efficient set of software tools which greatly enhances the input/output control, file access, and performance capabilities of the HP 2100-series computer. TCS is a collection of routines providing a language interface to a group of terminals and other I/O devices managed through the HP 2100 Disc Operating System — Version 3 (DOS-III).

TCS minimizes the programming complexity for the user by handling such functions as:

- Task (process) management
- Message queuing
- Dynamic priority scheduling
- Device locking
- File accessing
- Segment loading

Of primary significance is its multiple terminal handling capability. With TCS, the user has available a single, simple software interface through which he can manage many terminals and other I/O devices. The FORTRAN user can run his problem program in a real-time, multi-process environment as though he is using a simple read/write serial processing type system. Once the TCS environment is established, the user's impression may be that there is only one terminal on-line while TCS may, in fact, be managing many terminals for him.

TCS greatly extends the range of applications that can be processed by an HP 2100-series computer. An indication of the variety of possible applications is provided in section 2 of this manual. A prime advantage of TCS is the very favorable price/performance ratio it offers when interfacing multiple terminals and I/O devices to an HP 2100-series computer. Another advantage is that TCS runs under DOS-III with all its inherent capabilities.

section 2

TCS APPLICATIONS

Applications for multiple terminal systems such as TCS span all user groupings — commercial, industrial, scientific, military/aerospace — and vary widely in their functions, scope, and requirements.

Within the commercial area many different types of applications already exist and new ones are being developed every day. A few examples are as follows:

- Data collection and inventory control systems.
- Multiple-warehouse order processing
- Reservation processing
- Branch banking information systems
- Hospital information systems
- Pharmacy prescription control
- Instant credit checking
- Truck or rail system management
- Customer account inquiry processing
- Information retrieval systems
- Management information systems

The industrial area includes process control, on-line production control, automatic inspection systems, and production data collection systems.

The scientific area principally involves the instrumentation of laboratories for physical and biomedical experimentation.

The military/aerospace area involves tactical and strategic command and control, range instrumentation, count-down control, and so forth.

Functionally, the applications for multiple terminal systems may be divided into the following three fundamental classes:

- **DATA ENTRY.** Gathering information at a number of locations and passing it to a central processing point. Data collection systems are widely used to collect information concerning orders, deliveries, inventory, and other operating information to provide management with timely and reliable information with a minimum of manual handling and transcription.

- **DATA DISTRIBUTION.** Disseminating information generated or processed at a central facility.

- **INQUIRY AND FILE UPDATING.** Interrogating the central computer files. In this type of application, the terminal equipment is normally operated on-line; that is, when access is granted to the central computer, the inquiry is typed at the terminal. The inquiry is simultaneously entered into the central computer's memory where the stored program conducts a file search for the requested information. The process is then reversed, that is, the reply is sent out to the terminal. This type of system may also be used for updating the central computer's files.

A primary feature of TCS in its modular design: TCS can easily be adapted to meet the specific needs of individual users. Consequently, TCS-controlled applications can be utilized to advantage in any of the above-mentioned areas.

section

TCS CAPABILITIES 3

TCS significantly increases the user's ability to utilize the resources of the HP 2100-series computer. TCS overcomes the limitations imposed by some similar systems on the number and type of peripheral devices, overcomes the problems normally associated with the handling and control of multiple terminals, and relieves the user of numerous "housekeeping" tasks.

The more salient capabilities of TCS are as follows:

- **PRIORITY SCHEDULING.** User programs can be run at any of 16 priority levels. Each user task can re-specify its priority level at any time.

- **NO-WAIT I/O.** When a user issues an I/O request without wait, TCS returns control to the calling program as soon as the I/O request is accepted. The I/O request has then been either initiated or queued.

- **QUEUING OF I/O REQUESTS.** The user can issue more than one I/O request for a given device without having to wait for a previous request to be fulfilled.

- **WAIT I/O.** When a user issues an I/O request with wait, TCS returns control to the user when any outstanding I/O request is fulfilled. During the I/O wait time, other user tasks that are ready are run.

- **DEVICE LOCKING.** Any user task may issue an I/O request that also locks the device. When this happens, the requested I/O device is not available to any other task until either the original task or the main program unlocks the device. All I/O requests made by other tasks to the device while it is locked are queued in priority sequence and fulfilled after the device is unlocked. Of course, one of these I/O requests can also lock the device as part of its requested action.

- **OPEN FILE.** A user task may, at any time, request that a file be opened. After a file has been opened, every read/write request to the opened file is performed in one disc access.

- **OPEN SEGMENT.** Using a TCS Initialization request, the user may specify at the beginning of his program which segments of disc-resident code his particular application requires. An "in-core" segment directory is maintained to reduce load times.

- **DYNAMIC BUFFER MANAGEMENT.** Buffer pools may be specified at any time. Buffers are allocated and released dynamically in response to requests from the user's program.

section 4 THE TCS ENVIRONMENT

The TCS environment is illustrated in Figure 1. Note that once TCS has been called the user does not ordinarily interface directly with DOS-III. Instead, he initiates input/output operations by issuing requests to TCS which, in turn, interfaces directly with DOS-III. However, the user may issue calls directly to the DOS-III Executive if he so desires (see the dotted arrow in Figure 1), but care must be taken when using TCS and DOS-III for controlling the same device; in general this is not recommended.

SOFTWARE REQUIREMENTS

TCS operates under the HP 2100 Disc Operating System — Version 3 (DOS-III). The user writes his application programs in either HP Assembly Language or HP FORTRAN.

HARDWARE REQUIREMENTS

The minimum computer hardware required to support TCS is the minimum DOS-III hardware configuration (refer to the DOS-III reference manual). The amount of core memory required to support TCS is determined by adding the following core memory requirements together:

- The number of words of memory required by DOS-III.
- The number of words of memory required by TCS (1.5K words).
- The number of words of memory required by the user for buffer space.
- The number of words of memory required by the user for program space.

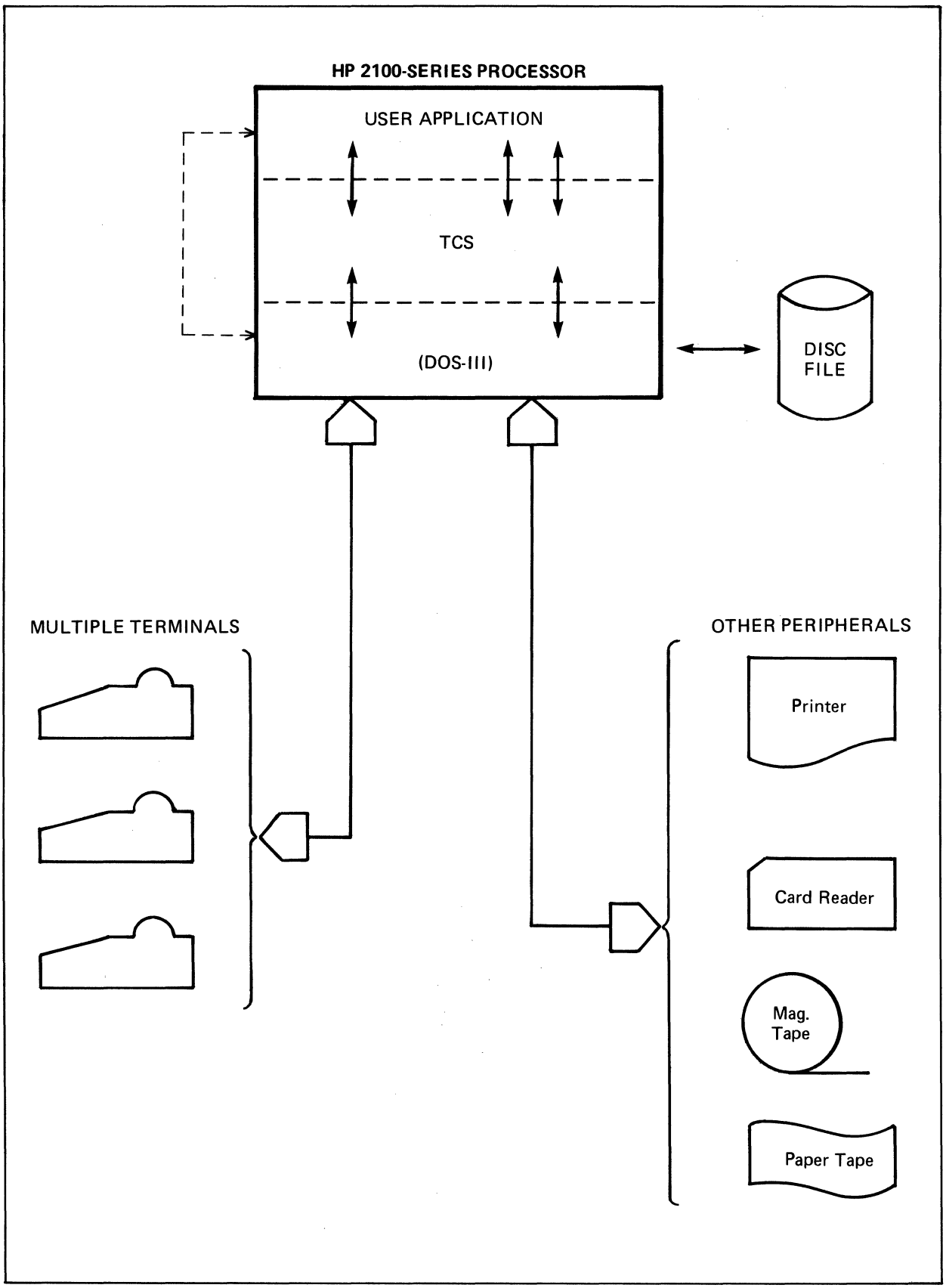


Figure 1. The TCS Environment

HARDWARE OPTIONS

The following HP 2100 computer hardware options are available:

- Additional terminals and interface kits
- A time base generator
- Paper tape readers and punches
- Line printers
- Magnetic tape units
- Card readers
- Additional disc drives (the HP 2100 can accommodate a maximum of four HP 7900A/7901A Disc Drives and a maximum of two HP 2883A Disc Drives)
- Additional I/O channels (I/O extenders are available)

TCS appears to the user as a set of subroutine calls. As illustrated in Figure 2, TCS functions as a scheduler for processing I/O requests in a fast, efficient manner. I/O calls which the user would normally make to the DOS-III Executive are made instead to TCS. However, there are certain requests which must still be made directly to DOS-III (in the form of EXEC calls) instead of to TCS. These include:

- Dynamic status requests
- Status requests for a particular device
- Work area limit requests
- Requests for time of day

Any valid EXEC calls may be used, but care must be taken when using TCS and EXEC for controlling the same device. It is permissible to use TCS for terminal requests and EFMP for disc handling. However, it is not permitted to use both TCS and EFMP for disc handling, the programmer may use either for that purpose but not both. In general it is not recommended to mix EXEC and TCS calls to the same device.

TCS SUBROUTINES

The TCS user-callable subroutines may be divided into three functional categories, as follows:

I. In-Core Requests

- Initialization
- Open File
- Device Unlock
- Suspend Until I/O Completion
- Return to Main
- Segment Load
- Status
- Priority Level Change
- Pause

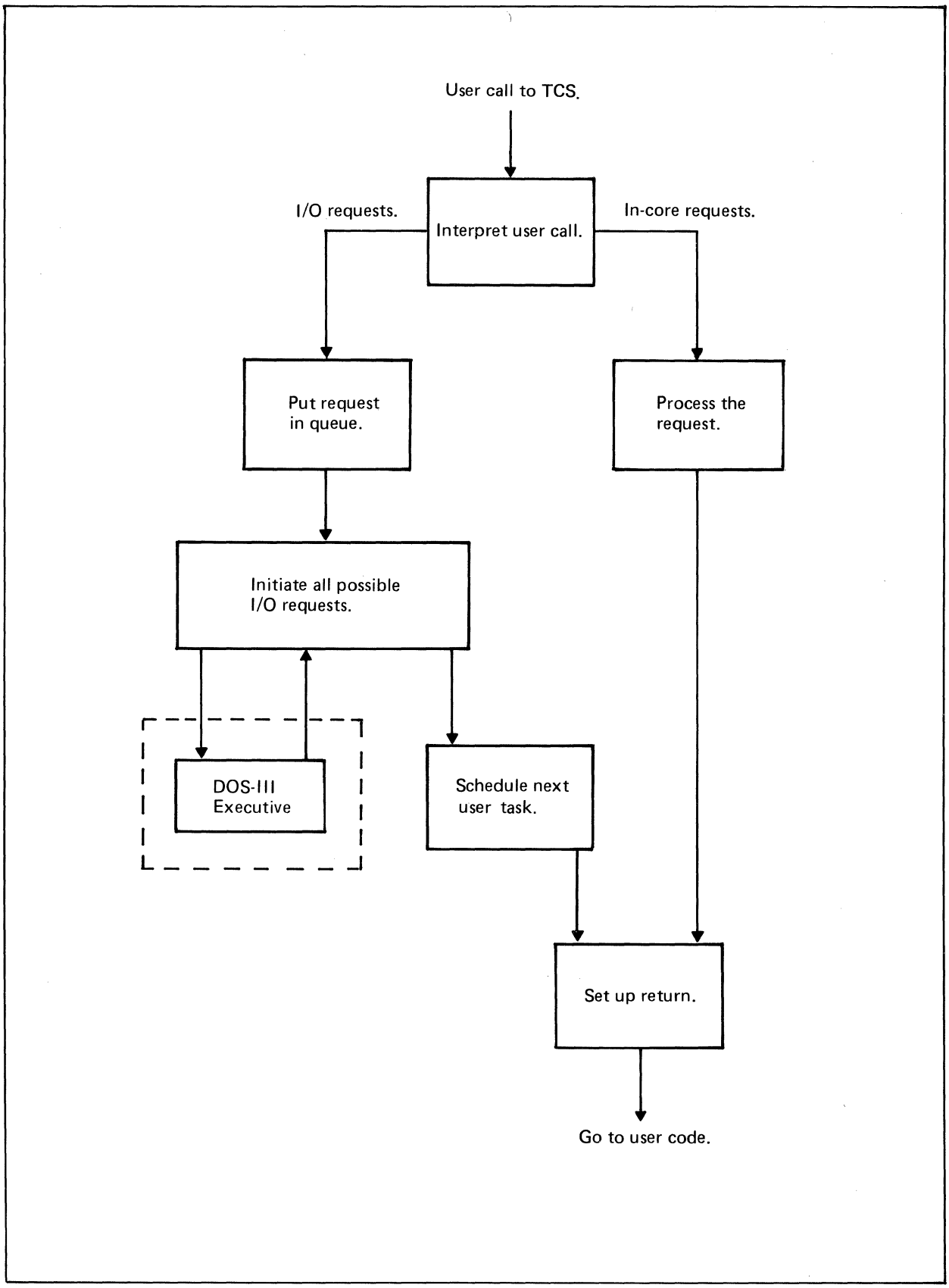


Figure 2. TCS User Interface

II. Input/Output Without Wait

- Read/Write
- File Read/Write
- I/O Control

III. Input/Output With Wait

- Read/Write
- File Read/Write
- I/O Control

The calling sequences are described in detail on the following pages.

Initialization

The three main purposes of this request are:

- to supply TCS with the name and size of the pending queue array
- to supply TCS with the name of the segment directory array
- to specify which disc-resident program segments are required by the user's main program.

The pending queue is an array in the user area of core memory used by TCS for holding all necessary information about I/O requests which cannot yet be initiated. The segment directory is a permanently core-resident array used by TCS for maintaining necessary information about the required program segments. The program segments specified in the request must previously have been stored on the disc using DOS-III.

TCS must first be initialized using this request before it will accept any read/write, file read/write, or I/O control requests. It is only necessary to initialize TCS once during a given run.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,IPQ,MAXPQ,INPQ, ISNAM, ISNUM,ISDIR)
```

where *ICOD* is the decimal constant 82.

IPQ is the name of the pending queue array.

MAXPQ is the name of a variable which contains a value specifying the maximum number of entries which the pending queue can accommodate. Each entry in the pending queue is 9 words long. Therefore, the value supplied must be the overall length of the pending queue (in words) divided by 9.

INPQ is the name of a variable whose value will constantly be set by TCS to reflect the current number of entries in the pending queue.

ISNAM is the name of an array which contains the names of all program segments which are required by the user's main program. Each entry in this array is three words long and contains the name (as five ASCII characters followed by a space) of one of the required program segments.

ISNUM is the name of a variable which contains a decimal value specifying the number of program segments required by the user's main program (that is, the length of ISNAM divided by 3). Maximum allowable value = 127.

ISDIR is the name of the segment directory array. Each entry in the segment directory is 11 words long. Therefore, the length of the array is equal to ISNUM multiplied by 11.

All of the above parameters must be present. If the variable ISNUM has the value zero, then ISNAM and ISDIR may be dummy parameters.

The assembly language calling sequence is:

```
        JSB   TCS
        DEF   *+8
        DEF   ICOD
        DEF   PQ
        DEF   MAXPQ
        DEF   INPQ
        DEF   ISNAM
        DEF   ISNUM
        DEF   ISDIR
          .
          .
          .
ICOD    DEC   82
  PQ    BSS   x
MAXPQ   DEC   x/9
INPQ    BSS   1
ISNAM   BSS   y
ISNUM   DEC   y/3
ISDIR   BSS   z
```

where x is the length (in words) of the pending queue array.

y is the length (in words) of the array which contains the names of all program segments which are required by the user's main program.

z is the length (in words) of the segment directory array.

Open File

This request loads the file directory information for the specified file from disc into core memory. An open file request may be executed whenever the disc is not busy, but would normally be executed once at the commencement of main program execution for each file which is to be used. Up to 16 files may be opened by a user's program at any given time. If a file has already been opened with the same reference number (see INUM in the calling sequences below), it will be closed and the new one will be opened for that reference number.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,INAM,INUM)
```

where *ICOD* is the decimal constant 84.

INAM is the name of an array which contains (as five ASCII characters followed by a space) the name of the file which is to be opened.

INUM is a decimal constant within the range 1-16 by which the file is to be referenced by the user's program.

The assembly language calling sequence is:

```
      JSB   TCS
      DEF   *+4
      DEF   ICOD
      DEF   INAM
      DEF   INUM
        .
        .
        .
ICOD  DEC   84
INAM  DEF   array-name
INUM  DEC   reference-#
```

where *array-name* is the name of an array which contains (as five ASCII characters followed by a space) the name of the file which is to be opened.

reference-# is a decimal value within the range 1-16 by which the file is to be referenced by the user's program.

Device Unlock

This request unlocks an input/output device. If the device was locked by the main program, then only the main program can unlock it. If the device was locked by a program segment, then only *that* segment or the main program can unlock it.

Note that the device is unlocked as soon as the request is accepted by TCS. Therefore, the request must not be issued until all outstanding critical I/O has been completed for the calling program by the particular device.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,LU)
```

where *ICOD* is the decimal constant 52.

LU is a decimal constant specifying the logical unit number of the device which is to be unlocked.

The assembly language calling sequence is:

```
      JSB   TCS
      DEF   *+3
      DEF   ICOD
      DEF   LU
        .
        .
        .
ICOD  DEC   52
 LU   DEC   logical-unit-number
```

where *logical-unit-number* is a decimal value specifying the logical unit number of the device which is to be unlocked.

Return to Main

This request passes control from a user segment to the main program. If a return to main request is executed in the main program, control merely passes to the next sequential instruction in the main program. If a return to main request is executed in a user segment which was called by another user segment, control returns to the main program *as though from the user segment which was called by the main program.*

The FORTRAN calling sequence is:

```
CALL TCS (ICOD)
```

where *ICOD* is the decimal constant 54.

The assembly language calling sequence is:

```
      JSB  TCS
      DEF  *+2
      DEF  ICOD
       .
       .
       .
ICOD  DEC  54
```

Segment Load

This request loads a program segment (from the disc into core memory) and starts it executing in the minimum possible time. The particular program segment must previously have been stored on the disc using DOS-III and the segment name must previously have been declared in a TCS initialization request. A segment load request may be issued either by the main program or by a program segment.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,ISEG,IPRI)
```

where *ICOD* is the decimal constant 8.

ISEG is a decimal constant specifying the relative number of the program segment name within the segment name array supplied in the most recent TCS initialization request. For example, an ISEG of 1 specifies the first declared program segment, an ISEG of 2 specifies the second declared program segment, and so forth.

IPRI is a decimal constant specifying the priority level (0-15) at which the particular program segment is to be run. 15 is the highest priority level and 0 is the lowest. More than one program segment may be assigned to the same priority level. The IPRI parameter is optional. If it is omitted, the priority level is set to 0.

The assembly language calling sequence is:

```
        JSB   TCS
        DEF   *+4
        DEF   ICOD
        DEF   ISEG
        DEF   IPRI
          .
          .
          .
ICOD    DEC   8
ISEG    DEC   segment-number
IPRI    DEC   priority-level
```

where *segment-number* is a decimal value specifying the relative number of the program segment name within the segment name array supplied in the most recent TCS initialization request. For example, a segment-number of 1 specifies the first declared program segment, a segment-number of 2 specifies the second declared program segment, and so forth.

priority-level is a decimal value specifying the priority level (0-15) at which the particular program segment is to be run. 15 is the highest priority level and 0 is the lowest. More than one program segment may be assigned to the same priority level.

Status

This request causes TCS to return status information to the calling program regarding a previously-initiated TCS request. The status request is used for the following two purposes:

- to obtain status information about a completed operation.
- to determine whether or not a TCS request (just issued) was considered valid by TCS.

The FORTRAN calling sequence is:

CALL TCS (ICOD,ISTAT,IPAR,ILUN,ITLOG)

where *ICOD* is the decimal constant 79.

ISTAT is the name of a variable. If status is being returned for a completed operation, the variable contains the hardware status bits (as defined by DOS-III) for the device associated with the operation. If status is being returned regarding the validity of a TCS request, the variable contains one of the following values (note that the non-zero numbers are negative):

0 = request valid

-1 = program segment cannot be found (initialization requests only)

-2 = pending queue is full (read/write, file read/write, pause, or I/O control requests only)

-3 = the request in question could not be understood by TCS

-4 = invalid segment number (segment load requests only)

-5 = file not opened or invalid record number (file read/write requests only)

-6 = TCS not initialized (read/write, file read/write, or I/O control requests only)

-7 = invalid unlock request (the segment trying to unlock the device is *not* the segment which locked the device)

IPAR is the name of a variable. If status is being returned for a completed operation, the variable contains the request identifier (IPRM) which was included in the request associated with the operation. For error returns, the contents of the variable are unspecified.

ILUN is the name of a variable. If status is being returned for a completed operation, the variable contains the logical unit number of the device associated with the operation. For error returns, the contents of the variable are unspecified.

ITLOG is the name of a variable. If status is being returned for a completed operation, the variable contains the transmission log (as defined by DOS-III) for the operation. For error returns, the contents of the variable are unspecified.

IPAR, ILUN, and ITLOG may be omitted. However, if one is present, all three must be present.

A status request should be issued after every TCS request to determine whether or not the particular request was accepted by TCS. If a request is not accepted by TCS, and if it was not immediately followed by a status request, then the particular request is forever "lost".

The assembly language calling sequence is:

```
        JSB   TCS
        DEF   *+6
        DEF   ICOD
        DEF   ISTAT
        DEF   IPAR
        DEF   ILUN
        DEF   ITLOG
          .
          .
          .
ICOD    DEC   79
ISTAT   BSS   1
IPAR    BSS   1
ILUN    BSS   1
ITLOG   BSS   1
```

where the contents of *ISTAT*, *IPAR*, *ILUN*, and *ITLOG* will be set as described for the FORTRAN calling sequence above.

Priority Level Change

This request specifies the priority level (0-15) at which the calling program is to run. The new priority level takes effect as soon as the request is accepted by TCS. 15 is the highest priority and 0 is the lowest.

The main program initially runs at priority level 0. Each program segment initially runs at the priority level specified in the TCS request which loaded it.

I/O requests are assigned the priority level at which the calling program is running *at the time the I/O request is issued.* Once an I/O request is issued, its priority level is permanently established and will not be affected by subsequent priority level change request.

If control passes to the return address of an I/O request executed at one priority level while the calling program is running at another priority level, the priority level of the calling program is changed to that of the completed I/O request (as though a priority level change request was executed).

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,IPRI)
```

where *ICOD* is the decimal constant 71.

IPRI is a decimal constant within the range 0-15 specifying the new priority level.

The assembly language calling sequence is:

```
      JSB   TCS
      DEF   *+3
      DEF   ICOD
      DEF   IPRI
        .
        .
        .
ICOD  DEC   71
IPRI  DEC   priority-level
```

where *priority-level* is a decimal constant within the range 0-15 specifying the new priority level.

Pause

This is a dummy input request which causes no I/O activity. It is used for suspending the calling program until all completed I/O requests of a higher or equal priority level are processed. A pause request would be necessary, for example, if the calling program is unable to continue execution because there are no buffers available (refer to "Buffer Management" later in this section). When the calling program is restarted, it may then check the required resource. If the resource is available, the program can continue execution; if the resource is still unavailable, the program can execute another pause request. When control eventually returns to the calling program, control passes to the next sequential instruction.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,ICON,IB,IBL,IPRM)
```

where *ICOD* is the constant 1.

ICON is the octal constant 77.

IB is the constant 1.

IBL is the constant 1.

} Required by TCS; no particular significance to user.

IPRM is the name of a variable which contains a decimal value within the range 0-255. This value will identify the particular pause request when the user's program subsequently executes TCS status requests.

The assembly language calling sequence is:

```
      JSB   TCS
      DEF   *+6
      DEF   ICOD
      DEF   ICON
      DEF   IB
      DEF   IBL
      DEF   IPARM
        .
        .
        .
ICOD  DEC   1
ICON  OCT   77
  IB  DEC   1
 IBL  DEC   1
IPRM  DEC   identifier
```

where *identifier* is a decimal value within the range 0-255. This value will identify the particular pause request when the user's program subsequently executes TCS status requests.

Suspend Until I/O Completion

This request suspends the calling program until *any* previously-issued I/O request is fulfilled. If the calling program issues a series of I/O requests (without wait) and then issues a suspend until I/O completion request, the calling program is suspended until *any* I/O request is fulfilled. At that time, the calling program is reactivated and control passes to the return address associated with the particular I/O request. The only time such an "interrupt" can occur is when the currently-active program is suspended as the result of an I/O with wait, pause, or suspend until I/O completion request.

The most significant difference between this request and a pause request is that pause has a return address (the next sequential instruction) associated with it whereas suspend does not. With a pause request, control returns to the next sequential instruction after all completed I/O requests of a higher or equal priority level are processed. With a suspend request, control passes to the return address associated with the next fulfilled I/O request (thereafter, program flow is unrelated to the suspend request).

The FORTRAN calling sequence is:

```
CALL TCS (ICOD)
```

where *ICOD* is the decimal constant 53.

The assembly language calling sequence is:

```
      JSB   TCS
      DEF   *+2
      DEF   ICOD
        .
        .
        .
ICOD  DEC   53
```

Read/Write (Without Wait)

This request causes TCS to initiate an input/output operation. For input operations, the data may be read from any input device or from the work area of the disc. For output operations, the data may be sent to any output device or to the work area of the disc. As soon as the request is accepted by TCS, control passes to the next sequential instruction in the calling program. The I/O operation may or may not be initiated immediately, depending upon whether or not the specified I/O device is free. If the device is busy, the request is placed in the pending queue; if the device is free, the operation is initiated immediately. When the operation is finished, control passes to the return address specified in the calling sequence.

The FORTRAN calling sequence is:

CALL TCS (ICOD,ICON,IBUF,IBUFL,ITRAK,ISECT,IPRM,IRET)

where *ICOD* is the constant 1 or 2. 1 specifies read and 2 specifies write.

ICON is an octal control word. The format of the control word is as described for the CONWD parameter of the Read/Write Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (part number 02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

IBUF is the name of the input or output buffer array.

IBUFL is a decimal constant specifying the length of the I/O buffer. If IBUFL is positive, it is interpreted as the number of words; if IBUFL is negative, it is interpreted as the number of bytes.

**ITRAK* is a decimal constant specifying the address of the first track to be read from or written into in the work area of the disc.

**ISECT* is a decimal constant specifying the address of the first sector to be read from or written into in the work area of the disc.

IPRM is the name of a variable which contains a decimal value within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

IRET is the name of a variable which contains the address to which control is to be passed when the operation is finished.

*If the data is *not* to be read from or written into the work area of the disc, then ITRAK and ISECT may be omitted entirely.

The assembly language calling sequence is:

```
        JSB   TCS
        DEF   *+n      (n=9 if read/write is from or to the work area of the disc;
        DEF   ICOD      otherwise, n=7)
        DEF   ICON
        DEF   IBUF
        DEF   IBUFL
        DEF   ITRAK  }
        DEF   ISECT  }  (Included only if the read/write is from or
                        to the work area of the disc)
        DEF   IPRM
        DEF   IRET
          .
          .
          .
ICOD    DEC   1 or 2 (1=read; 2=write)
ICON    OCT   control-word
IBUF    BSS   n
IBUFL   DEC   n (or -2n)
ITRAK   DEC   track-address   }
ISECT   DEC   sector-address  }  (Included only if the read/write is
                                 from or to the work area of the
                                 disc)
IPRM    DEC   identifier
IRET    DEF   return-address
```

where *control-word* is an octal control word. The format of the control word is as described for the CONWD parameter of the Read/Write Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (part number 02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

track-address is a decimal constant specifying the address of the first track to be read from or written into in the work area of the disc.

sector-address is a decimal constant specifying the address of the first sector to be read from or written into in the work area of the disc.

identifier is a decimal constant within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

return-address is the address to which control is to be passed when the operation is finished.

In the buffer-definition psuedo-instruction (IBUF BSS n), the buffer length is specified as the number of words. In the subsequent psuedo-instruction, the buffer length (IBUFL) is specified either as the number of words or as the number of bytes (positive value=number of words; negative value=number of bytes).

File Read/Write (Without Wait)

This request causes TCS to initiate a disc read/write operation in which data is read from or written into a disc-resident user file. As soon as the request is accepted by TCS, control passes to the next sequential instruction in the calling program. The read/write operation may or may not be initiated immediately, depending upon whether or not the specified disc drive is free. If the disc drive is busy, the request is placed in the pending queue; if the disc drive is free, the operation is initiated immediately. When the operation is finished, control passes to the return address specified in the calling sequence.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,ICON,IBUF,IBUFL,INUM,ISECT,IPRM,IRET)
```

where *ICOD* is the decimal constant 14 or 15. 14 specifies read and 15 specifies write.

ICON is an octal control word. The format of the control word is as described for the CONWD parameter of the Read/Write Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (part number 02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

IBUF is the name of the input or output buffer array.

IBUFL is a decimal constant specifying the length of the I/O buffer. If IBUFL is positive, it is interpreted as the number of words; if IBUFL is negative, it is interpreted as the number of bytes.

INUM is a decimal constant within the range 1-16 specifying which file is to be read from or written into. This value corresponds to the INUM parameter in the TCS open file request for the particular file.

ISECT is a decimal constant specifying the relative address of the first sector in the specified file to be read from or written into (0 through n).

IPRM is the name of a variable which contains a decimal value within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

IRET is the name of a variable which contains the address to which control is to be passed when the operation is finished.

The assembly language calling sequence is:

```
        JSB   TCS
        DEF   *+9
        DEF   ICOD
        DEF   ICON
        DEF   IBUF
        DEF   IBUFL
        DEF   INUM
        DEF   ISECT
        DEF   IPRM
        DEF   IRET
          .
          .
          .
ICOD    DEC   14 or 15 (14=read; 15=write)
ICON    OCT   control-word
IBUF    BSS   n
IBUFL   DEC   n (or -2n)
INUM    DEC   file-identifier
ISECT   DEC   sector-address
IPRM    DEC   request-identifier
IRET    DEF   return-address
```

where *control-word* is an octal control word. The format of the control word is as described for the CONWD parameter of the Read/Write Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (part number 02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

file-identifier is a decimal constant within the range 1-16 specifying which file is to be read from or written into. This value corresponds to the INUM parameter in the TCS open file request for the particular file.

sector-address is a decimal constant specifying the relative address of the first sector in the specified file to be read from or written into (0 through n).

request-identifier is a decimal constant within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

return-address is the address to which control is to be passed when the operation is finished.

In the buffer-definition psuedo-instruction (IBUF BSS n), the buffer length is specified as the number of words. In the subsequent psuedo-instruction, the buffer length (IBUFL) is specified either as the number of words or as the number of bytes (positive value=number of words; negative value=number of bytes).

I/O Control (Without Wait)

This request causes TCS to initiate an input/output control operation. As soon as the request is accepted by TCS, control passes to the next sequential instruction in the calling program. The I/O control operation may or may not be initiated immediately, depending upon whether or not the specified I/O device is free. If the device is busy, the request is placed in the pending queue; if the device is free, the operation is initiated immediately. When the operation is finished, control passes to the return address specified in the calling sequence.

The FORTRAN calling sequence is:

CALL TCS (ICOD,ICON,IPAR,IPRM,IRET)

where *ICOD* is the constant 3.

ICON is an octal control word. The format of the control word is as described for the CONWD parameter of the I/O Control Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

IPAR is a device-dependent control constant. This is not required for some devices and may be omitted if not needed.

IPRM is the name of a variable which contains a decimal value within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

IRET is the name of a variable which contains the address to which control is to be passed when the operation is finished.

The assembly language calling sequence is:

```
        JSB   TCS
        DEF   *+6
        DEF   ICOD
        DEF   ICON
        DEF   IPAR
        DEF   IPRM
        DEF   IRET
          .
          .
          .
ICOD    DEC   3
ICON    OCT   control-word
IPAR    DEC   n
IPRM    DEC   identifier
IRET    DEF   return-address
```

where *control-word* is an octal control word. The format of the control word is as described for the CONWD parameter of the I/O Control Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

n is a device-dependent decimal control constant. This is not required for some devices and may be omitted if not needed.

identifier is a decimal constant within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

return-address is the statement label to which control is to be passed when the operation is finished.

Read/Write (With Wait)

This request causes TCS to initiate an input/output operation. For input operations, the data may be read from any input device or from the work area of the disc. For output operations, the data may be sent to any output device or to the work area of the disc. The I/O operation may or may not be initiated immediately, depending upon whether or not the specified I/O device is free. If the device is busy, the request is placed in the pending queue; if the device is free, the operation is initiated immediately. After issuing this request, the calling program is suspended. When control eventually returns to the calling program, control passes to the next sequential instruction. While the calling program is waiting for the I/O request to be fulfilled, a previously-initiated I/O request may be fulfilled. In such a case, control passes to the return address associated with the particular I/O request. The only time such an "interrupt" can occur is when the currently-active program is suspended as the result of an I/O with wait, pause, or suspend until I/O completion request.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,ICON,IBUF,IBUFL,ITRAK,ISECT,IPRM)
```

where *ICOD* is the constant 1 or 2. 1 specifies read and 2 specifies write.

ICON is an octal control word. The format of the control word is as described for the CONWD parameter of the Read/Write Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (part number 02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

IBUF is the name of the input or output buffer array.

IBUFL is a decimal constant specifying the length of the I/O buffer. If IBUFL is positive, it is interpreted as the number of words; if IBUFL is negative, it is interpreted as the number of bytes.

**ITRAK* is a decimal constant specifying the address of the first track to be read from or written into in the work area of the disc.

**ISECT* is a decimal constant specifying the address of the first sector to be read from or written into in the work area of the disc.

IPRM is the name of a variable which contains a decimal value within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

*If the data is *not* to be read from or written into the work area of the disc, then ITRAK and ISECT may be omitted entirely.

The assembly language calling sequence is:

```
JSB   TCS
DEF   *+n      (n=8 if read/write is from or to the work
                area of the disc; otherwise, n=6)

DEF   ICOD
DEF   ICON
DEF   IBUF
DEF   IBUFL
DEF   ITRAK  }
DEF   ISECT  }  (Included only if the read/write is from
                 or to the work area of the disc)
DEF   IPRM
  .
  .
  .
```

```
        .
        .
        .
ICOD    DEC   1 or 2 (1=read; 2=write)
ICON    OCT   control-word
IBUF    BSS   n
IBUFL   DEC   n (or -2n)
ITRAK   DEC   track-address     } (Included only if the read/write is from
ISECT   DEC   sector-address    }  or to the work area of the disc)
IPRM    DEC   identifier
```

where *control-word* is an octal control word. The format of the control word is as described for the CONWD parameter of the Read/Write Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (part number 02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

track-address is a decimal constant specifying the address of the first track to be read from or written into in the work area of the disc.

sector-address is a decimal constant specifying the address of the first sector to be read from or written into in the work area of the disc.

identifier is a decimal constant within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

In the buffer-definition psuedo-instruction (IBUF BSS n), the buffer length is specified as the number of words. In the subsequent psuedo-instruction, the buffer length (IBUFL) is specified either as the number of words or as the number of bytes (positive value=number of words; negative value=number of bytes).

File Read/Write (With Wait)

This request causes TCS to initiate a disc read/write operation in which data is read from or written into a disc-resident user file. The I/O operation may or may not be initiated immediately, depending upon whether or not the disc drive is free. If the disc drive is busy, the request is placed in the pending queue; if the disc drive is free, the operation is initiated immediately. After issuing this request, the calling program is suspended. When control eventually returns to the calling program, control passes to the next sequential instruction. While the calling program is waiting for the I/O request to be fulfilled, a previously-initiated I/O request may be fulfilled. In such a case, control passes to the return address associated with the particular I/O request. The only time such an "interrupt" can occur is when the currently-active program is suspended as the result of an I/O with wait, pause, or suspend until I/O completion request.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,ICON,IBUF,IBUFL,INUM,ISECT,IPRM)
```

where *ICOD* is the decimal constant 14 or 15. 14 specifies read and 15 specifies write.

ICON is an octal control word. The format of the control word is as described for the CONWD parameter of the Read/Write Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (part number 02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

IBUF is the name of the input or output buffer array.

IBUFL is a decimal constant specifying the length of the I/O buffer. If IBUFL is positive, it is interpreted as the number of words; if IBUFL is negative, it is interpreted as the number of bytes.

INUM is a decimal constant within the range 1-16 specifying which file is to be read from or written into. This value corresponds to the INUM parameter in the TCS open file request for the particular file.

ISECT is a decimal constant specifying the relative address of the first sector in the specified file to be read from or written into (0 through n).

IPRM is the name of a variable which contains a decimal value within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

The assembly language calling sequence is:

```
        JSB   TCS
        DEF   *+8
        DEF   ICOD
        DEF   ICON
        DEF   IBUF
        DEF   IBUFL
        DEF   INUM
        DEF   ISECT
        DEF   IPRM
          .
          .
          .
ICOD    DEC   14 or 15 (14=read; 15=write)
ICON    OCT   control-word
```

```
IBUF   BSS   n
IBUFL  DEC   n (or -2n)
INUM   DEC   file-identifier
ISECT  DEC   sector-address
IPRM   DEC   request-identifier
```

where *control-word* is an octal control word. The format of the control word is as described for the CONWD parameter of the Read/Write Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (part number 02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

file-identifier is a decimal constant within the range 1-16 specifying which file is to be read from or written into. This value corresponds to the INUM parameter in the TCS open file request for the particular file.

sector-address is a decimal constant specifying the relative address of the first sector in the specified file to be read from or written into (0 through n).

request-identifier is a decimal constant within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

In the buffer-definition psuedo-instruction (IBUF BSS n), the buffer length is specified as the number of words. In the subsequent psuedo-instruction, the buffer length (IBUFL) is specified either as the number of words or as the number of bytes (positive value=number of words; negative value=number of bytes).

I/O Control (With Wait)

This request causes TCS to initiate an input/output control operation. The I/O control operation may or may not be initiated immediately, depending upon whether or not the specified I/O device is free. If the device is busy, the request is placed in the pending queue; if the device is free, the operation is initiated immediately. After issuing this request, the calling program is suspended at least until the request has been fulfilled. When control eventually returns to the calling program, control passes to the next sequential instruction. While the program is waiting for the I/O control request to be fulfilled, a previously-initiated I/O request may be fulfilled. In such a case, control passes to the return address associated with the particular I/O request. The only time such an "interrupt" can occur is when the currently-active program is suspended as the result of an I/O with wait, pause, or suspend until I/O completion request.

The FORTRAN calling sequence is:

```
CALL TCS (ICOD,ICON,IPAR,IPRM)
```

where *ICOD* is the constant 3.

ICON is an octal control word. The format of the control word is as described for the CONWD parameter of the I/O Control Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

IPAR is a device-dependent control constant. This is not required for some devices and may be omitted if not needed.

IPRM is the name of a variable which contains a decimal value within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

The assembly language calling sequence is:

```
      JSB   TCS
      DEF   *+5
      DEF   ICOD
      DEF   ICON
      DEF   IPAR
      DEF   IPRM
        .
        .
        .
ICOD  DEC   3
ICON  OCT   control-word
IPAR  DEC   n
IPRM  DEC   identifier
```

where *control-word* is an octal control word. The format of the control word is as described for the CONWD parameter of the I/O Control Exec Call in section III of the *HP DOS-III Disc Operating System* reference manual (02100-90136) with the exception that the sign of the control word determines whether or not the specified device will be locked. If the sign bit is 1, the device will be locked; if the sign bit is 0, the device will *not* be locked.

n is a device-dependent decimal control constant. This is not required for some devices and may be omitted if not needed.

identifier is a decimal constant within the range 0-255. This value will identify the particular I/O operation when the user's program subsequently executes TCS status requests.

BUFFER MANAGEMENT

The TCS buffer management subsystem allows the user to dynamically allocate and deallocate buffers according to his needs. Up to four buffer pools may be established, each containing up to 64 buffers. The buffers within a particular pool must all be the same length. Pools may be initialized or reinitialized at any time.

The FORTRAN calling sequence for initializing a buffer pool is:

```
CALL BINIT (IA,I,J,L)
```

where *IA* is the name of the buffer pool array.

I is a decimal constant within the range 1 to 64 specifying the number of buffers in the pool.

J is a decimal constant specifying the length of each buffer.

L is a constant within the range 1 to 4 identifying the particular buffer pool.

The assembly language calling sequence for initializing a buffer pool is:

```
      JSB   BINIT
      DEF   *+5
      DEF   IA
      DEF   I
      DEF   J
      DEF   L
       .
       .
       .
IA    BSS   pool-length
I     DEC   #-of-buffers
J     DEC   buffer-length
L     DEC   pool-identifier
```

where *pool-length* is a decimal constant specifying the length (in words) of the particular buffer pool.

#-of-buffers is a decimal constant within the range 1 to 64 specifying the number of buffers in the pool.

buffer-length is a decimal constant specifying the length of each buffer.

pool-identifier is a constant within the range 1 to 4 identifying the pool.

Allocation of Buffers

Whenever the user requires a buffer, he requests the buffer management system to allocate a buffer from a particular pool. The FORTRAN calling sequence for allocating a buffer is:

```
CALL GBUF (IN,L)
```

where *IN* is the name of a variable. The value of this variable is set by the buffer management system to the number (1 through 64) of the buffer which was allocated. If no buffers are available, IN is set to -1.

L is a constant within the range 1 to 4 specifying from which pool the buffer is to be allocated.

The assembly language calling sequence for allocating a buffer is:

```
      JSB   GBUF
      DEF   *+3
      DEF   IN
      DEF   L
        .
        .
        .
IN    BSS   1
 L    DEC   pool-identifier
```

where *pool-identifier* is a constant within the range 1 to 4 specifying from which pool the buffer is to be allocated.

The value of IN is set by the buffer management system to the number (1 through 64) of the buffer which was allocated. If no buffers are available, IN is set to -1.

Release of Buffers

Whenever the user no longer needs a particular buffer, he may request the buffer management system to release the buffer. The FORTRAN calling sequence for releasing a buffer is:

```
CALL PBUF (IN,L)
```

where *IN* is the name of a variable. This should be the same variable which was set by the buffer management system when the particular buffer was allocated.

L is a constant within the range 1 to 4 specifying the pool to which the buffer belongs.

The assembly language calling sequence is:

```
      JSB   PBUF
      DEF   *+3
      DEF   IN
      DEF   L
        .
        .
        .
IN    BSS   1
 L    DEC   pool-number
```

where *pool-number* is a constant within the range 1 to 4 specifying the pool to which the buffer belongs.

The variable IN must contain the value supplied by the buffer management system when the particular buffer was allocated.

Buffer Inquiry

It is often desirable to know how many buffers within a particular pool are in use at a given time. The FORTRAN calling sequence for requesting the current status of a particular pool is:

```
CALL IBUF (I,L)
```

where *I* is the name of a variable. The variable will be set by the buffer management system to specify the number of buffers in the pool which are currently allocated to the user's program.

L is a constant within the range 1 to 4 specifying the pool in question.

The assembly language calling sequence is:

```
      JSB   IBUF
      DEF   *+3
      DEF   I
      DEF   L
        .
        .
        .
I     BSS   1
L     DEC   pool-identifier
```

where *pool-identifier* is a constant within the range 1 to 4 specifying the pool in question.

The variable I will be set by the buffer management system to specify the number of buffers in the pool which are currently allocated to the user's program.

section 6 PROGRAMMING CONSIDERATIONS

The considerations to be borne in mind when writing TCS programs may be broadly defined under two headings:

1. Input/output processing
2. Buffer and variable control

These will now be discussed.

INPUT/OUTPUT PROCESSING

In a situation where many user tasks may be running at the same time two approaches to I/O queueing are possible:

1. Provide a queue area (pending queue) large enough to hold the maximum possible number of concurrent I/O requests to devices which are busy or locked.

2. Provide a queue area (pending queue) large enough for the average load and take care of peak loads by recommended programming methods.

Both approaches may be used with TCS. The sample program uses method 1, this is not always possible due to various factors (core limitations, undefined future programs, etc.). For this reason a method of handling possible pending queue problems is outlined below.

1. If a pending queue of 20 entries is assumed then the maximum number of I/O requests placed into it should not exceed 19. This allows a user program to "pause" if it has other requests to issue ("pause" requires one pending queue entry).

2. The user program should then follow the procedure outlined in Figure 3.

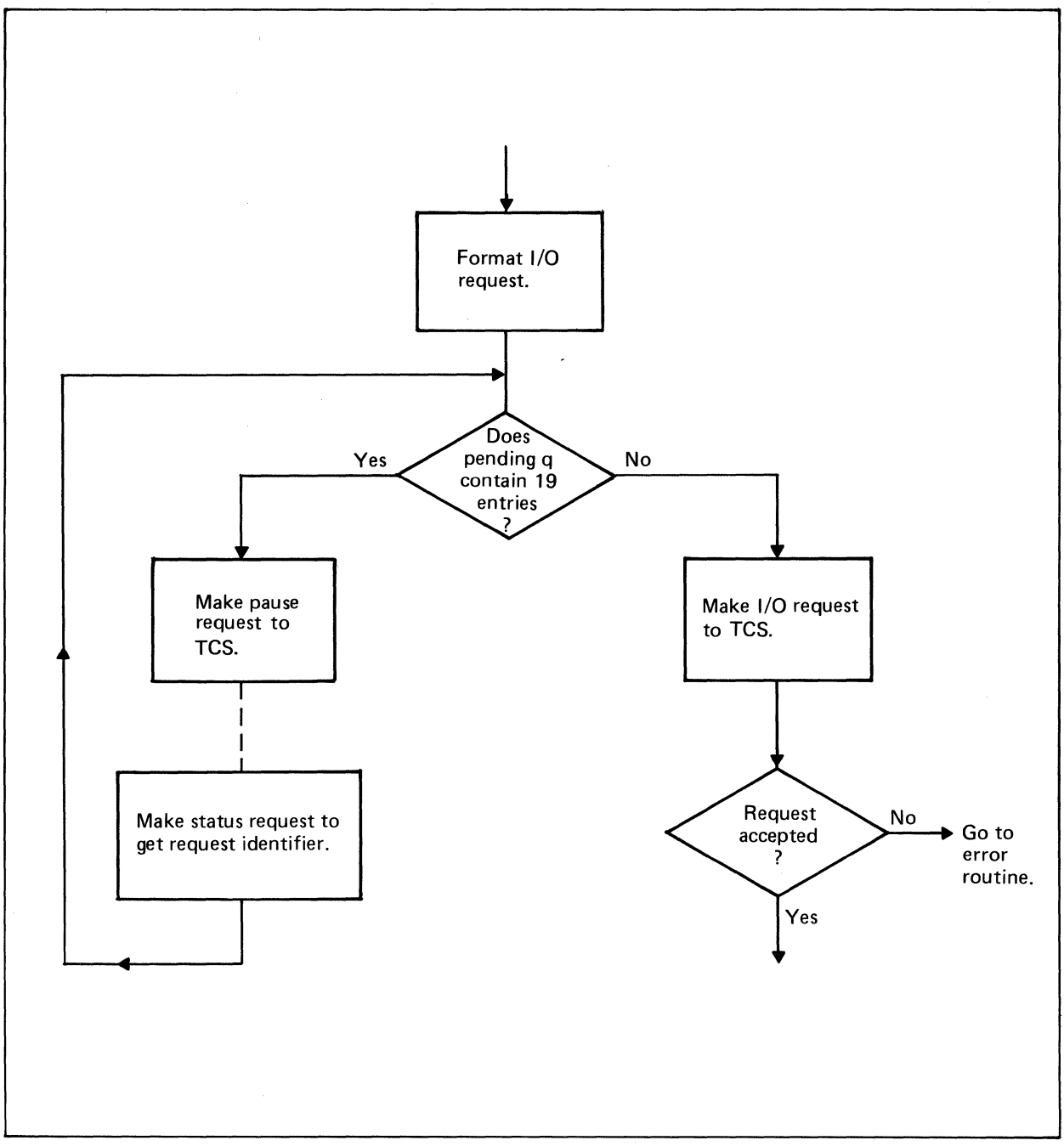


Figure 3. Procedure for Issuing Requests

By means of this method a request is never rejected because of pending queue overflow. If pending queue overflow does occur then the only action that a user program can take is to set flags for itself and perform a "suspend till I/O" call until outstanding I/O is complete for it and then examine the flags and try to proceed from there.

All I/O requests and

'segment load'
'unlock device'

should be followed by a status request to check that the request has been understood by TCS.

In order to simplify the understanding of with and without wait requests the following two pieces of code are shown:

1) without wait
```
ASSIGN 10 TO IRET
CALL TCS (1,20001B,IA,4,I,IRET)
CALL TCS (53)
10 Statement
```

2) with wait
```
CALL TCS (1,1,IA,4,I)
10 Statement
```

These two examples perform exactly the same function as far as the user is concerned.

BUFFER AND VARIABLE CONTROL

The major factor to be considered with regard to buffer and variable control is whether overlay segmentation is being used. If the whole of the user program is core resident then the problem is greatly simplified, however for this discussion overlay segments will be assumed.

The major problem is to ensure that all buffers which are being used for current input/output transfers remain core resident. It follows therefore that these buffers must not be embedded within a program segment if there is any possibility of that segment being overlayed by another. For this reason it is recommended that all buffers are held in COMMON. The buffer management routines may then be used to control the buffers.

Because of the fact that segments can be overlayed by other segments any variables which the segment requires to keep should also be kept in common. To simplify the control of these variables, it is recommended that one pool of buffers are used as "data stacks" and that programs keep all required variables in these data stacks. The "request identifier" which TCS requires with each I/O call can contain the "data stack" pointer. In this way a separation of code and data is achieved (as in the example program) that enables re-entrant overlayable code to be written.

APPENDIX

The sample program presented in this appendix illustrates a method of using TCS to perform a simple function by means of a re-entrant subroutine. The problems associated with re-entrancy are solved by means of a request identifier which is passed to TCS as part of each I/O request.

The main program in this example merely solicits input from the terminals. Whenever a terminal input is complete, the subroutine (SSUB) is used to process the input and display the output on the terminal. When the output is complete, the subroutine returns to the main program which then requests further input from the terminal. In order for the subroutine to be re-entrant, all variables used by the subroutine which are particular to a terminal must be saved in a "data stack." In this way, code and data are separated and re-entrancy may easily be achieved.

One data stack is required for each terminal that the subroutine serves. The allocation of data stacks is done by the calling program — *not* by the subroutine. The subroutine may therefore remain ignorant of how many terminals are active. The data within a data stack must be entered by the main program in the format expected by the subroutine. This is illustrated in Figure 4. The vertical array index is passed to the subroutine as the stack pointer. A typical flow of control is as follows (refer to figure 5):

1. input is requested from all terminals. Main program suspends until an I/O is complete.

2. Terminal #1 responds; main program creates subroutine stack and calls subroutine.

3. Subroutine formats terminal input and requests disc read, passing the stack pointer to TCS. Because this request is *with* wait, TCS may now schedule the main program to deal with another terminal's completed input.

4. Main program accepts Terminal #2 input, creates another subroutine stack, and calls the subroutine.

5. Subroutine formats the terminal input and requests a disc read, passing the stack pointer to TCS.

6. The subroutine could, in theory, have up to 20 disc reads outstanding at any one time.

7. A disc read is completed and the subroutine is rescheduled.

8. The subroutine performs a status check to obtain the hardware status and the stack pointer corresponding to the completed disc read.

9. The subroutine requests output to the terminal, passing the stack pointer to TCS. This request is *with* wait and either the subroutine or the main program could be scheduled to deal with another complete I/O.

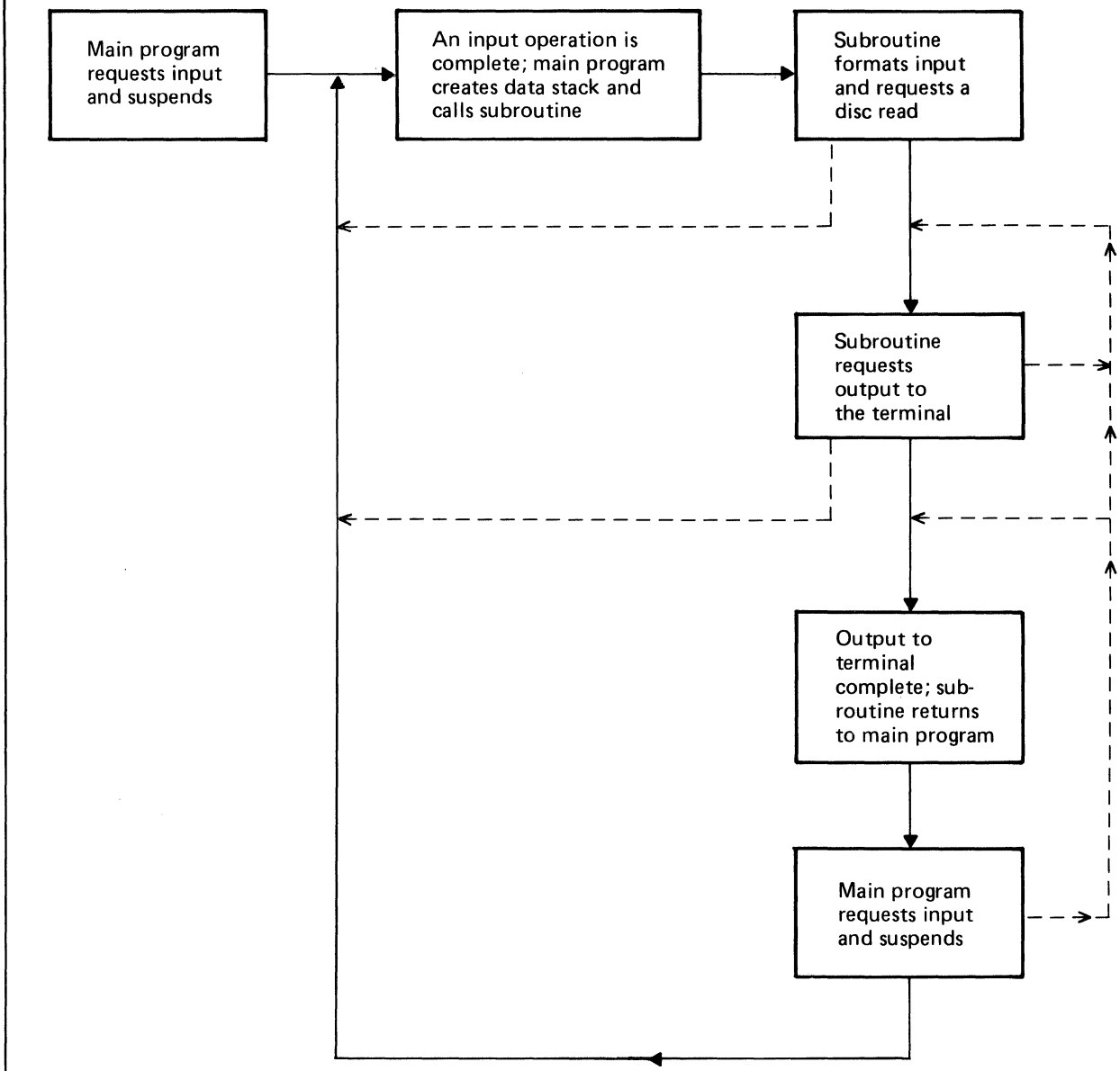
10. When the terminal output is complete, the subroutine requests status from TCS which restores the stack pointer for *this* completed I/O.

11. The subroutine returns to the main program via its data stack.

12. The main program requests input from the terminal and suspends itself.

	X	X + 1	X + 2	X + 3
1		Return Addr	Return Addr	Return Addr
2		I/O Buffer #	I/O Buffer #	I/O Buffer #
3		# of Chars	# of Chars	# of Chars
4		LU #	LU #	LU #
5		For Subroutine Use	For Subroutine Use	For Subroutine Use
6		For Subroutine Use	For Subroutine Use	For Subroutine Use
The vertical array index (X, X + 1, etc.) is passed to the subroutine.				

Figure 4. Data Stacks for Subroutine



NOTE: Whenever the main program is suspended or the subroutine is waiting for an I/O request to be fulfilled, a TCS "interrupt" can occur. The "interrupt" is triggered by the fulfillment of a previously-issued I/O request (control passes to the return address associated with the particular request). The solid arrows illustrate the main path and the dotted arrows illustrate the various possible interrupt paths.

Figure 5. Sample Program Flow

```
0001  FTN4,L,M
0002        PROGRAM RETY
0003  C
0004  C THIS PROGRAM ILLUSTRATES THE METHODS USED WITH TCS
0005  C TO PROVIDE RE-ENTRANT SUBROUTINES. ASSUME THE FOLLOWING
0006  C PROBLEM, A SYSTEM WITH 20 TERMINALS WHICH CAN REQUEST
0007  C A RECORD FROM THE DISC TO BE DISPLAYED ON THE TERMINAL.
0008  C  THE OPERATOR INPUT IS THE RECORD NUMBER.
0009  C A COMMON SUBROUTINE WILL BE USED TO OBTAIN THE RECORD
0010  C FROM THE DISC AND OUTPUT IT TO THE TERMINALS.
0011  C
0012  C ASSUME TERMINAL LU #S OF 10 TO 29 INCLUSIVE
0013  C
0014  C
0015  C FIRST DIMENSION THE INPUT BUFFER ARRAY
0016  C
0017        COMMON IN(2,20)
0018  C
0019  C NOW DIMENSION THE OUTPUT BUFFER ARRAY
0020  C
0021        COMMON ID(128,20)
0022  C
0023  C NOW DIMENSION A PARAMETER ARRAY
0024  C
0025        COMMON IP(6,20)
0026  C
0027  C NOW DIMENSION A PENDING QUEUE FOR TCS
0028  C
0029        DIMENSION IPQ(180)
0030  C
0031  C NOW DIMENSION AN ARRAY FOR SUBROUTINE RETURN PARAMETER
0032  C
0033        DIMENSION IK(1)
0034  C
0035  C NOW CREATE AN ARRAY HOLDING FILE NAME
0036  C
0037        DIMENSION IF(3)
0038        DATA IF/2HFI,2HLE,2H3 /
0039  C
0040  C NOW INITIATE TCS
0041  C
0042        CALL TCS(82,IPQ,20,KK,0,0,0)
0043  C
0044  C NOW OPEN THE FILE
0045  C
0046        CALL TCS(84,IF,1)
0047  C
0048  C NOW REQUEST INPUT FROM ALL TERMS
0049  C
0050  C FIRST SET UP RETURN ADDR
0051  C
0052        ASSIGN 70 TO IRET
0053  C
```

```
0054   C INITIATE INPUT
0055   C
0056         DO 10 I=1,20
0057         LU=I+9
0058   10    CALL TCS(1,204008+LU,IN(1,I),2,I,IRET)
0059   C
0060   C NOW SUSPEND UNTIL AN INPUT IS COMPLETE
0061   C
0062   99    CALL TCS(53)
0063   C
0064   C COME HERE WHEN AN INPUT IS COMPLETE
0065   C
0066   C GET STATUS AND PARAMETER
0067   C
0068   70    CALL TCS(79,ISTAT,IPAR,ILU,ILOG)
0069   C
0070   C FOR  THIS EXAMPLE OMIT INPUT VALIDATION ETC
0071   C NOW SET UP TO CALL SUBROUTINE
0072   C FIRST SPECIFY RETURN ADDR AND PUT IN SUBROUTINE
0073   C STACK
0074         ASSIGN 80 TO II
0075         IP(1,IPAR)=II
0076   C
0077   C NOW SET INPUT BUFFER INDEX IN PARAM STACK
0078   C
0079         IP(2,IPAR)=IPAR
0080   C
0081   C NOW SET # OF INPUT CHARS IN STACK
0082   C
0083         IP(3,IPAR)=ILOG
0084   C
0085   C NOW SET  LU# IN STACK
0086   C
0087         IP(4,IPAR)=ILU
0088   C
0089   C NOW CALL SUBROUTINE
0090   C
0091         CALL SSUB(IPAR,IK)
0092   C
0093   C COME HERE AFTER SUBROUTINE TO INITIATE INPUT AGAIN
0094   C
0095   80    IW=IK(1)
0096         CALL TCS(1,20400B+IP(4,IW),IN(1,IW),2,IW,IRET)
0097         GOTO 99
0098         END

**  NO ERRORS*
```

```
PROGRAM RETY
 00000  000000       NOP
 00001  016001X      JSB CLRIO
 00002  000003R      DEF *+1
 00003  026302R      JMP 00302
COMMON IN(2,20)
COMMON ID(128,20)
COMMON IP(6,20)
DIMENSION IPQ(180)
DIMENSION IK(1)
DIMENSION IF(3)
DATA IF/2HFI,2HLE,2H3 /
 00004  000000C      DEF 00000C
 00005  000050C      DEF 00050C
 00006  005050C      DEF 05050C
 00007  000010R      DEF *+1
                     BSS 00264
 00274  000275R      DEF *+1
                     BSS 00001
 00276  000277R      DEF *+1
 00277  043111       OCT 043111
 00300  046105       OCT 046105
 00301  031440       OCT 031440
CALL TCS(82,IPQ,20,KK,0,0,0)
 00302  016002X      JSB TCS
 00303  000313R      DEF 00313
 00304  000526R      DEF 00526
 00305  000010R      DEF 00010
 00306  000521R      DEF 00521
 00307  000527R      DEF KK
 00310  000530R      DEF 00530
 00311  000530R      DEF 00530
 00312  000530R      DEF 00530
CALL TCS(84,IF,1)
 00313  016002X      JSB TCS
 00314  000320R      DEF *+4
 00315  000531R      DEF 00531
 00316  000277R      DEF 00277
 00317  000553R      DEF 00553
ASSIGN 70 TO IRET
 00320  062322R      LDA *+2
 00321  002001       RSS
 00322  000366R      DEF 00366
 00323  072532R      STA IRET
DO 10 I=1,20
 00324  062553R      LDA 00553
 00325  072533R      STA I
LU=I+9
 00326  062536R      LDA 00536
 00327  042533R      ADA I
 00330  072535R      STA LU
```

```
10      CALL TCS(1,20400B+LU,IN(1,I),2,I,IRET)
 00331  062535R        LDA LU
 00332  042537R        ADA 00537
 00333  072534R        STA I,001
 00334  066553R        LDB 00553
 00335  002400         CLA
 00336  016003X        JSB ..MAP
 00337  000004R        DEF 00004
 00340  000553R        DEF 00553
 00341  000533R        DEF I
 00342  000520R        DEF 00520
 00343  072540R        STA A,001
 00344  016002X        JSB TCS
 00345  000354R        DEF *+7
 00346  000553R        DEF 00553
 00347  000534R        DEF I,001
 00350  100540R        DEF A,001,I
 00351  000520R        DEF 00520
 00352  000533R        DEF I
 00353  000532R        DEF IRET
 00354  062533R        LDA I
 00355  042553R        ADA 00553
 00356  072533R        STA I
 00357  003004         CMA,INA
 00360  042521R        ADA 00521
 00361  002021         SSA,RSS
 00362  026326R        JMP 00326
99      CALL TCS(53)
 00363  016002X        JSB TCS
 00364  000366R        DEF *+2
 00365  000541R        DEF 00541
70      CALL TCS(79,ISTAT,IPAR,ILU,ILOG)
 00366  016002X        JSB TCS
 00367  000375R        DEF *+6
 00370  000542R        DEF 00542
 00371  000543R        DEF ISTAT
 00372  000544R        DEF IPAR
 00373  000545R        DEF ILU
 00374  000546R        DEF ILOG
        ASSIGN 80 TO II
 00375  062377R        LDA *+2
 00376  002001         RSS
 00377  000455R        DEF 00455
 00400  072547R        STA II
        IP(1,IPAR)=II
 00401  066553R        LDB 00553
 00402  002400         CLA
 00403  016003X        JSB ..MAP
 00404  000006R        DEF 00006
 00405  000553R        DEF 00553
 00406  000544R        DEF IPAR
 00407  000523R        DEF 00523
 00410  072540R        STA A,001
```

```
 00411  062547R      LDA II
 00412  172540R      STA A,001,I
      IP(2,IPAR)=IPAR
 00413  066553R      LDB 00553
 00414  002400       CLA
 00415  016003X      JSB ..MAP
 00416  000006R      DEF 00006
 00417  000520R      DEF 00520
 00420  000544R      DEF IPAR
 00421  000523R      DEF 00523
 00422  072540R      STA A,001
 00423  062544R      LDA IPAR
 00424  172540R      STA A,001,I
      IP(3,IPAR)=ILOG
 00425  066553R      LDB 00553
 00426  002400       CLA
 00427  016003X      JSB ..MAP
 00430  000006R      DEF 00006
 00431  000525R      DEF 00525
 00432  000544R      DEF IPAR
 00433  000523R      DEF 00523
 00434  072540R      STA A,001
 00435  062546R      LDA ILOG
 00436  172540R      STA A,001,I
      IP(4,IPAR)=ILU
 00437  066553R      LDB 00553
 00440  002400       CLA
 00441  016003X      JSB ..MAP
 00442  000006R      DEF 00006
 00443  000550R      DEF 00550
 00444  000544R      DEF IPAR
 00445  000523R      DEF 00523
 00446  072540R      STA A,001
 00447  062545R      LDA ILU
 00450  172540R      STA A,001,I
      CALL SSUB(IPAR,IK)
 00451  016004X      JSB SSUB
 00452  000455R      DEF *+3
 00453  000544R      DEF IPAR
 00454  000275R      DEF 00275
80    IW=IK(1)
 00455  062553R      LDA 00553
 00456  042552R      ADA 00552
 00457  042274R      ADA 00274
 00460  160000       LDA 0,I
 00461  072551R      STA IW
      CALL TCS(1,20400B+IP(4,IW),IN(1,IW),2,IW,IRET)
 00462  066553R      LDB 00553
 00463  002400       CLA
 00464  016003X      JSB ..MAP
 00465  000006R      DEF 00006
 00466  000550R      DEF 00550
```

```
 00467  000551R      DEF IW
 00470  000523R      DEF 00523
 00471  160000       LDA 0,I
 00472  042537R      ADA 00537
 00473  072534R      STA I,001
 00474  066553R      LDB 00553
 00475  002400       CLA
 00476  016003X      JSB ..MAP
 00477  000004R      DEF 00004
 00500  000553R      DEF 00553
 00501  000551R      DEF IW
 00502  000520R      DEF 00520
 00503  072540R      STA A,001
 00504  016002X      JSB TCS
 00505  000514R      DEF *+7
 00506  000553R      DEF 00553
 00507  000534R      DEF I,001
 00510  100540R      DEF A,001,I
 00511  000520R      DEF 00520
 00512  000551R      DEF IW
 00513  000532R      DEF IRET
GOTO 99
 00514  026363R      JMP 00363
END
 00515  016005X      JSB EXEC
 00516  000520R      DEF *+2
 00517  000523R      DEF 00523
 00520  000002       OCT 000002
 00521  000024       OCT 000024
 00522  000200       OCT 000200
 00523  000006       OCT 000006
 00524  000264       OCT 000264
 00525  000003       OCT 000003
 00526  000122       OCT 000122
                     BSS 00001
 00530  000000       OCT 000000
 00531  000124       OCT 000124
                     BSS 00004
 00536  000011       OCT 000011
 00537  020400       OCT 020400
                     BSS 00001
 00541  000065       OCT 000065
 00542  000117       OCT 000117
                     BSS 00005
 00550  000004       OCT 000004
                     BSS 00001
 00552  177777       OCT 177777
 00553  000001       OCT 000001
```

```
0099        END$
```

SYMBOL TABLE

NAME	ADDRESS	USAGE	TYPE	LOCATION
@10	000331R	STATEMENT NUMBER		
@70	000366R	STATEMENT NUMBER		
@80	000455R	STATEMENT NUMBER		
@99	000363R	STATEMENT NUMBER		
CLRIO	000001X	SUBPROGRAM	REAL	EXTERNAL
EXEC	000005X	SUBPROGRAM	REAL	EXTERNAL
I	000533R	VARIABLE	INTEGER	LOCAL
ID	000050C	ARRAY(*,*)	INTEGER	COMMON
IF	000277R	ARRAY(*)	INTEGER	LOCAL
II	000547R	VARIABLE	INTEGER	LOCAL
IK	000275R	ARRAY(*)	INTEGER	LOCAL
ILOG	000546R	VARIABLE	INTEGER	LOCAL
ILU	000545R	VARIABLE	INTEGER	LOCAL
IN	000000C	ARRAY(*,*)	INTEGER	COMMON
IP	005050C	ARRAY(*,*)	INTEGER	COMMON
IPAR	000544R	VARIABLE	INTEGER	LOCAL
IPQ	000010R	ARRAY(*)	INTEGER	LOCAL
IRET	000532R	VARIABLE	INTEGER	LOCAL
ISTAT	000543R	VARIABLE	INTEGER	LOCAL
IW	000551R	VARIABLE	INTEGER	LOCAL
KK	000527R	VARIABLE	INTEGER	LOCAL
LU	000535R	VARIABLE	INTEGER	LOCAL
SSUB	000004X	SUBPROGRAM	REAL	EXTERNAL
TCS	000002X	SUBPROGRAM	REAL	EXTERNAL

```
0001  FTN4,L,M
0002        SUBROUTINE SSUB(I,IK)
0003        COMMON IN(2,20),ID(128,20),IP(6,20)
0004        DIMENSION IK(1)
0005  C I THE STACK MARKER
0006  C IS IS PASSED FROM THE MAIN (IPAR) AS A STACK POINTER
0007  C
0008  C MOVE THE INPUT TO A BUFFER & CONVERT TO INTEGER
0009        DIMENSION K(2)
0010        K(1)=IN(1,I)
0011        K(2)=IN(2,I)
0012        CALL CODE
0013        READ(K,*)J
0014  C
0015  C NOW READ THE RECORD
0016  C
0017        CALL TCS(14,3,ID(1,I),128,1,J,I)
0018  C
0019  C WILL COME HERE WHEN INPUT IS OVER
0020  C CALL STATUS TO GET STACK POINTER
0021  C
0022        CALL TCS(79,ISTAT,I,LU,IL)
0023  C
0024  C NOW PRINT ON THE TERMINAL
0025  C
0026        CALL TCS(2,IP(4,I),ID(1,I),128,I)
0027  C
0028  C NOW FIND THE STACK  POINTER AGAIN
0029  C
0030        CALL TCS(79,ISTAT,I,LU,IL)
0031  C
0032  C NOW SET STACK POINTER IN IK FOR THE MAIN
0033  C
0034        IK(1)=I
0035  C
0036  C NON RETURN TO MAIN PROGRAM
0037  C
0038        IR=IP(1,I)
0039        GOTO IR
0040        END

**  NO ERRORS*
```

```
SUBROUTINE SSUB(I,IK)
                        BSS 00002
 00002  000000          NOP
 00003  016001X         JSB .ENTR
 00004  000000R         DEF *-4
 00005  026014R         JMP 00014
COMMON IN(2,20),ID(128,20),IP(6,20)
DIMENSION IK(1)
DIMENSION K(2)
K(1)=IN(1,I)
 00006  000000C         DEF 00000C
 00007  000050C         DEF 00050C
 00010  005050C         DEF 05050C
 00011  000012R         DEF *+1
                        BSS 00002
 00014  062210R         LDA 00210
 00015  042174R         ADA 00174
 00016  042011R         ADA *-5
 00017  072175R         STA A,001
 00020  066210R         LDB 00210
 00021  002400          CLA
 00022  016002X         JSB ..MAP
 00023  000006R         DEF 00006
 00024  000210R         DEF 00210
 00025  100000R         DEF 00000,I
 00026  000170R         DEF 00170
 00027  160000          LDA 0,I
 00030  172175R         STA A,001,I
K(2)=IN(2,I)
 00031  062170R         LDA 00170
 00032  042174R         ADA 00174
 00033  042011R         ADA 00011
 00034  072175R         STA A,001
 00035  066210R         LDB 00210
 00036  002400          CLA
 00037  016002X         JSB ..MAP
 00040  000006R         DEF 00006
 00041  000170R         DEF 00170
 00042  100000R         DEF 00000,I
 00043  000170R         DEF 00170
 00044  160000          LDA 0,I
 00045  172175R         STA A,001,I
CALL CODE
 00046  016003X         JSB CODE
 00047  000050R         DEF *+1
READ(K,*)J
 00050  062012R         LDA 00012
 00051  006404          CLB,INB
 00052  016004X         JSB .DIO.
 00053  000000          OCT 000000
 00054  000057R         DEF 00057
 00055  016005X         JSB .IIO.
 00056  000176R         DEF J
```

```
CALL TCS(14,3,ID(1,I),128,1,J,I)
 00057  066210R      LDB 00210
 00060  002400       CLA
 00061  016002X      JSB ..MAP
 00062  000007R      DEF 00007
 00063  000210R      DEF 00210
 00064  100000R      DEF 00000,I
 00065  000172R      DEF 00172
 00066  072175R      STA A,001
 00067  016006X      JSB TCS
 00070  000100R      DEF 00100
 00071  000177R      DEF 00177
 00072  000200R      DEF 00200
 00073  100175R      DEF A,001,I
 00074  000172R      DEF 00172
 00075  000210R      DEF 00210
 00076  000176R      DEF J
 00077  100000R      DEF 00000,I
CALL TCS(79,ISTAT,I,LU,IL)
 00100  016006X      JSB TCS
 00101  000107R      DEF *+6
 00102  000201R      DEF 00201
 00103  000202R      DEF ISTAT
 00104  100000R      DEF 00000,I
 00105  000203R      DEF LU
 00106  000204R      DEF IL
CALL TCS(2,IP(4,I),ID(1,I),128,I)
 00107  066210R      LDB 00210
 00110  002400       CLA
 00111  016002X      JSB ..MAP
 00112  000010R      DEF 00010
 00113  000205R      DEF 00205
 00114  100000R      DEF 00000,I
 00115  000173R      DEF 00173
 00116  072175R      STA A,001
 00117  066210R      LDB 00210
 00120  002400       CLA
 00121  016002X      JSB ..MAP
 00122  000007R      DEF 00007
 00123  000210R      DEF 00210
 00124  100000R      DEF 00000,I
 00125  000172R      DEF 00172
 00126  072206R      STA A,002
 00127  016006X      JSB TCS
 00130  000136R      DEF *+6
 00131  000170R      DEF 00170
 00132  100175R      DEF A,001,I
 00133  100206R      DEF A,002,I
 00134  000172R      DEF 00172
 00135  100000R      DEF 00000,I
```

```
CALL TCS(79,ISTAT,1,LU,IL)
 00136  016006X      JSB TCS
 00137  000145R      DEF *+6
 00140  000201R      DEF 00201
 00141  000202R      DEF ISTAT
 00142  100000R      DEF 00000,I
 00143  000203R      DEF LU
 00144  000204R      DEF IL
IK(1)=I
 00145  062210R      LDA 00210
 00146  042174R      ADA 00174
 00147  042001R      ADA 00001
 00150  072175R      STA A,001
 00151  162000R      LDA 00000,I
 00152  172175R      STA A,001,I
IR=IP(1,I)
 00153  066210R      LDB 00210
 00154  002400       CLA
 00155  016002X      JSB ..MAP
 00156  000010R      DEF 00010
 00157  000210R      DEF 00210
 00160  100000R      DEF 00000,I
 00161  000173R      DEF 00173
 00162  160000       LDA 0,I
 00163  072207R      STA IR
GOTO IR
 00164  126207R      JMP IR,I
END
 00165  126002R      JMP 00002,I
                     BSS 00002
 00170  000002       OCT 000002
 00171  000024       OCT 000024
 00172  000200       OCT 000200
 00173  000006       OCT 000006
 00174  177777       OCT 177777
                     BSS 00002
 00177  000016       OCT 000016
 00200  000003       OCT 000003
 00201  000117       OCT 000117
                     BSS 00003
 00205  000004       OCT 000004
                     BSS 00002
 00210  000001       OCT 000001



 0041          END $
```

SYMBOL TABLE

NAME	ADDRESS	USAGE	TYPE	LOCATION
CODE	000003X	SUBPROGRAM	REAL	EXTERNAL
I	000000R	VARIABLE	INTEGER	DUMMY
ID	000050C	ARRAY(*,*)	INTEGER	COMMON
IK	000001R	ARRAY(*)	INTEGER	DUMMY
IL	000204R	VARIABLE	INTEGER	LOCAL
IN	000000C	ARRAY(*,*)	INTEGER	COMMON
IP	005050C	ARRAY(*,*)	INTEGER	COMMON
IR	000207R	VARIABLE	INTEGER	LOCAL
ISTAT	000202R	VARIABLE	INTEGER	LOCAL
J	000176R	VARIABLE	INTEGER	LOCAL
K	000012R	ARRAY(*)	INTEGER	LOCAL
LU	000203R	VARIABLE	INTEGER	LOCAL
SSUB	000166R	VARIABLE	REAL	LOCAL
TCS	000006X	SUBPROGRAM	REAL	EXTERNAL

HP MANUAL PART NO. 5951-7307
MICROFICHE PART NO. 5951-7308