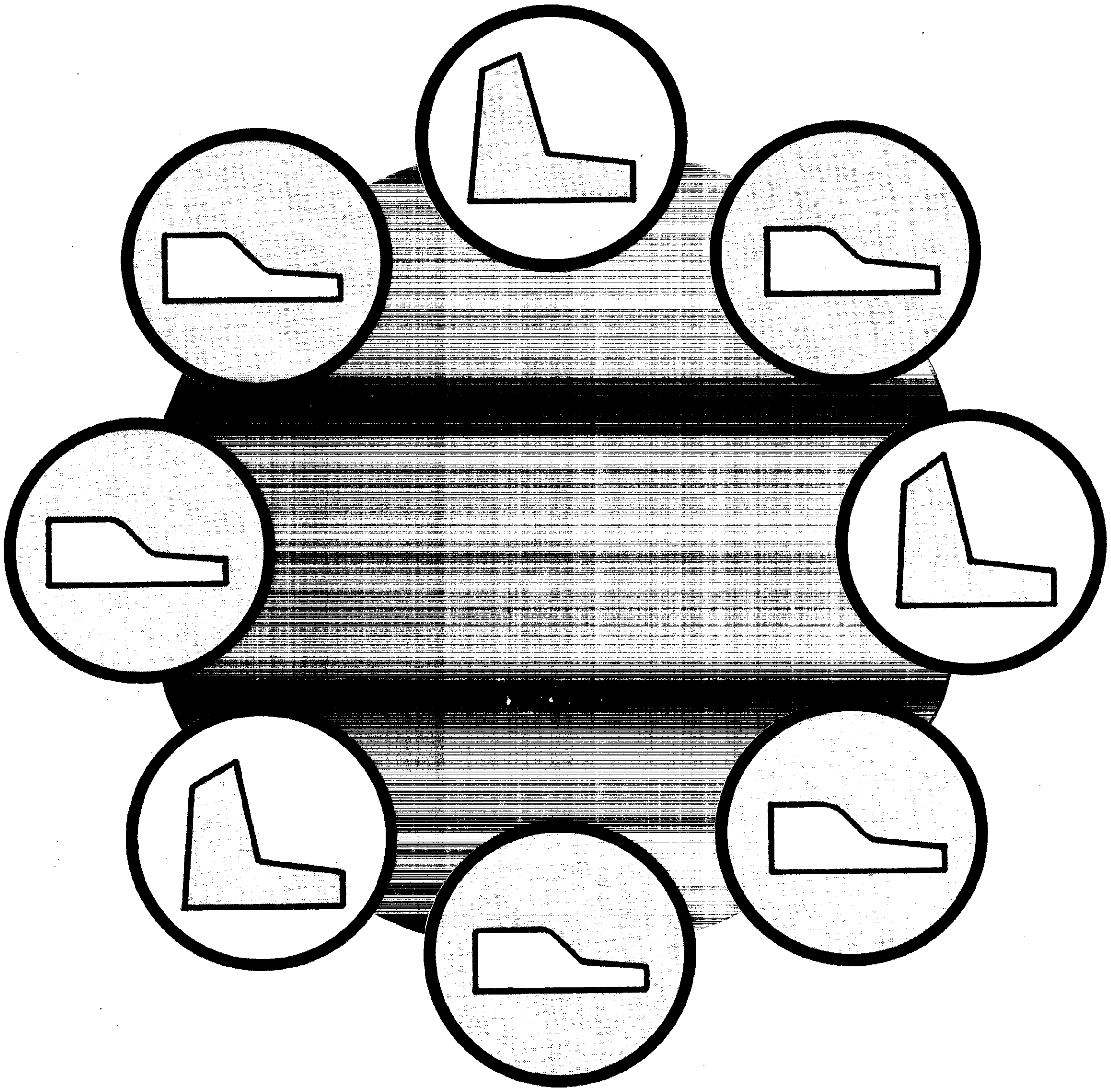
HEWLETT hp PACKARD

2100 Computer Systems



TERMINAL CONTROL SYSTEM APPLICATIONS MANUAL

TERMINAL CONTROL SYSTEM

APPLICATIONS MANUAL

HEWLETT hp PACKARD

Computer Systems

HP Manual Part No. 5951-7310
Microfiche Part No. 5951-7311

Printed: August 1973

PREFACE

This manual presents a sample on-line, multi-terminal business application system which uses the Hewlett-Packard Terminal Control System (TCS). It is intended to show how TCS can be used in designing an overall application system and to suggest a few easy methods of task organization. It is *not* meant to be a treatise on order entry, goods receiving, or invoice processing nor is it meant to illustrate elegant coding. Some of the programming methods used are inefficient but have been purposely chosen to clearly illustrate certain points. No hardware status checking is performed since this would tend to cloud the issue of how to use TCS. In some cases, the code may not even be syntactically correct FORTRAN (for example, DATA and DIMENSION statements appear wherever this helps to illustrate a certain point).

Throughout the manual FORTRAN coding is intermixed with the explanatory text. Now and then, coding is described in the text but does not appear in the FORTRAN examples because it does not exemplify the use of TCS or (in the case of some subroutines) because the particular use of TCS is adequately illustrated elsewhere in the manual.

It is hoped that after studying this manual the reader will feel confident about using TCS to help solve his own programming problems.

CONTENTS

INTRODUCTION

In the distribution market the most commonly performed functions for which on-line, multi-terminal computer solutions are being sought are:

1. Order entry
2. Goods receiving
3. Invoicing

The sample application system presented in this manual shows, in a simplified way, how these business functions may be performed using TCS and the Hewlett-Packard 2100 Disc Operating System—Version III (DOS-III).

The following assumptions are made:

1. The application system must be able to handle 10 terminals and each terminal must be able to perform any of the three business functions regardless of what functions the other terminals are performing.
2. The following hardware is present:
    a. An HP 2100-series computer with at least 16K words of memory.
    b. 10 terminals.
    c. A disc drive.
    d. A magnetic tape unit.
3. The following files are present on the disc drive:
    a. A customer file.
    b. A vendor file.
    c. A product file.
    d. A spooling file.
4. Invoices are printed at the end of each day and a user subroutine (not shown in this example) is executed after all invoices are printed to set the entire disc spooling file to zeros.

BUFFER DEFINITION

Before proceeding further, we must define the number of buffers required and the size of each.

The following sets of buffers are necessary:

a. A set of "terminal buffers" to be used for printing prompt messages and accepting operator entries.

b. A set of "disc buffers" to be used for reading records from the disc or writing records to the disc.

c. A set of "line printer" buffers to be used for printing invoices. Although we refer to these as *line printer* buffers, this example actually prints the invoices on the terminals.

d. A set of "data stacks" to be used for storing all pertinent information for each terminal. The use of these stacks makes it possible for the user segments to be re-entrant.

To allow all the terminals to be performing functions simultaneously, there must be a minimum of ten terminal buffers. Ten will suffice for this example. The terminal buffers, each 40 words (80 characters) long, are held in the array IT.

One disc buffer would suffice. However, to provide better response time this example will use five. The disc buffers, each 128 words long, are held in the array ID.

To allow all the terminals to be producing invoices simultaneously, there must be a minimum of ten line printer buffers. Ten will suffice for this example. The line printer buffers, each 66 words (132 characters) long, are held in the array IL.

There must be one data stack per terminal. The data stacks, each 15 words long, are held in the array IS.

All of the buffers must reside in common. Therefore, all programs, segments, and subroutines must contain the following statement:

```
COMMON IT(40,10),ID(128,5),IL(66,10),IS(15,10),IX,IA,IB,IC
```

The purpose of the variables IX, IA, IB, and IC is described later in this manual.

The total buffer allocation is just under 1.9K words of memory. The buffers must now be passed to the TCS buffer manager which will perform the buffer control functions (allocating and deallocating) in response to calls from the application system:

```
CALL BINIT (IT,10,40,1)
CALL BINIT (ID,5,128,2)
CALL BINIT (IL,10,66,3)
CALL BINIT (IS,10,15,4)
```

SYSTEM ORGANIZATION

The overall system is organized as illustrated in Figure 1. The application programs which perform the business functions of order entry, goods receiving, and invoicing are coded as user segments and stored on the disc. The user main program initializes the system and thereafter merely passes control to the appropriate segments in response to operator entries.

Figure 2 illustrates the possible paths of control between the various system elements. For example, the main program can pass control to the user segments or to TCS, the user segments can pass control to the main program or to TCS, and so forth.

TCS (in conjunction with DOS-III) takes care of "swapping" the user segments between the disc drive and the overlay area. Whenever the currently-executing segment is dormant (i.e., paused, suspended, or waiting for a particular I/O operation to be completed), it may be overlaid by another segment which is ready to continue processing or it may be restarted for another terminal. The link between segments and terminals and between the main program and terminals is a set of data stacks residing in common. Whenever a segment must become dormant, it passes the current data stack pointer to TCS. When the segment regains control, TCS passes back the appropriate data stack pointer. Figure 3 illustrates the "swapping" of segments and the passing of data stack pointers between the segments and TCS. Figure 4 illustrates the linkage between a data stack and its associated buffers, disc records, and terminal.

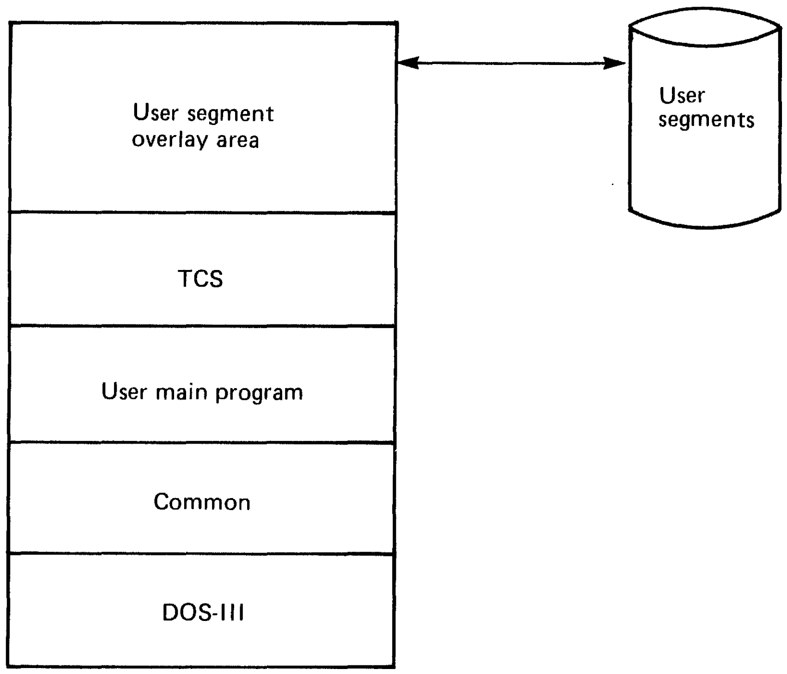


Figure 1. Overall System Organization

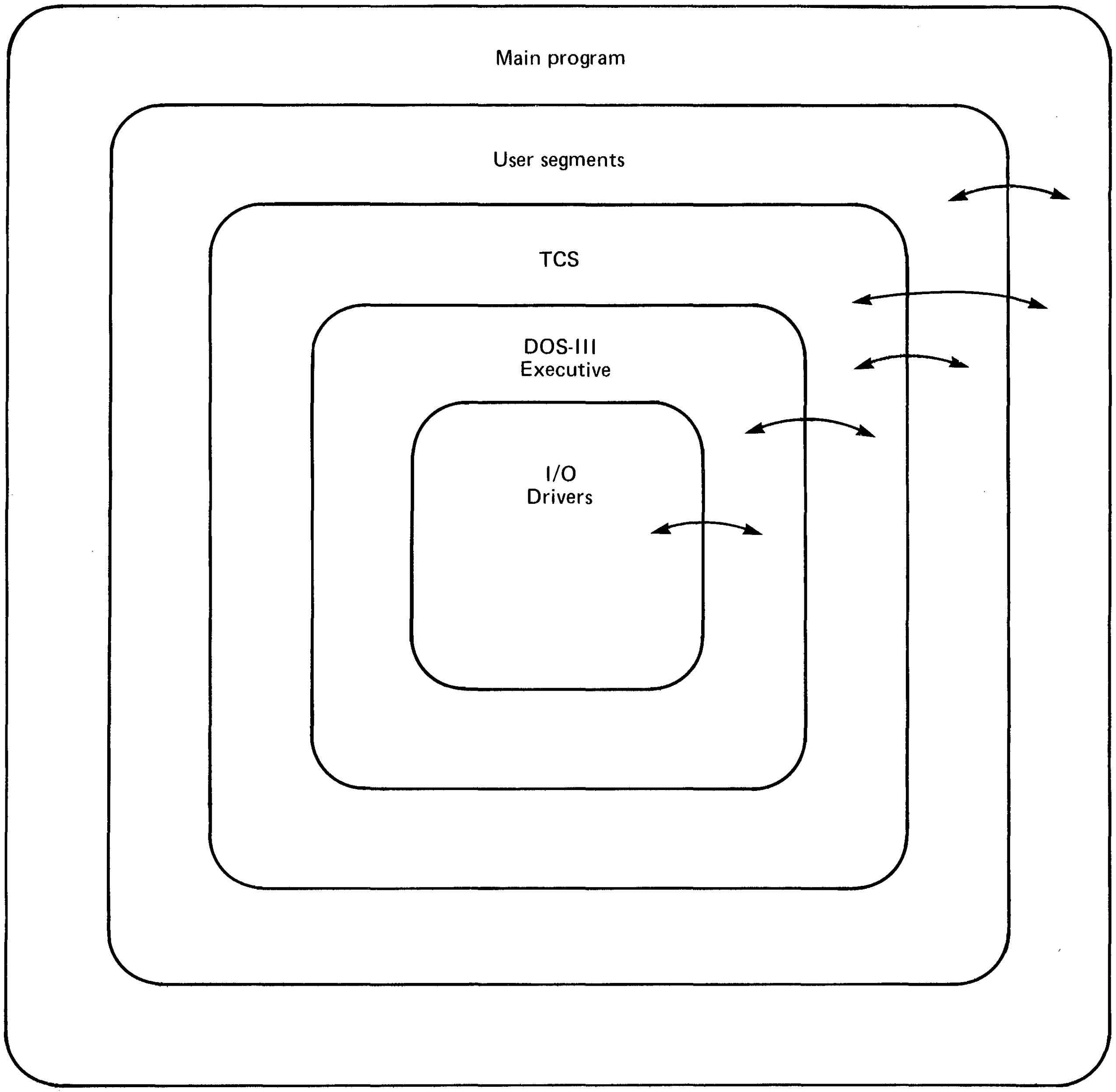
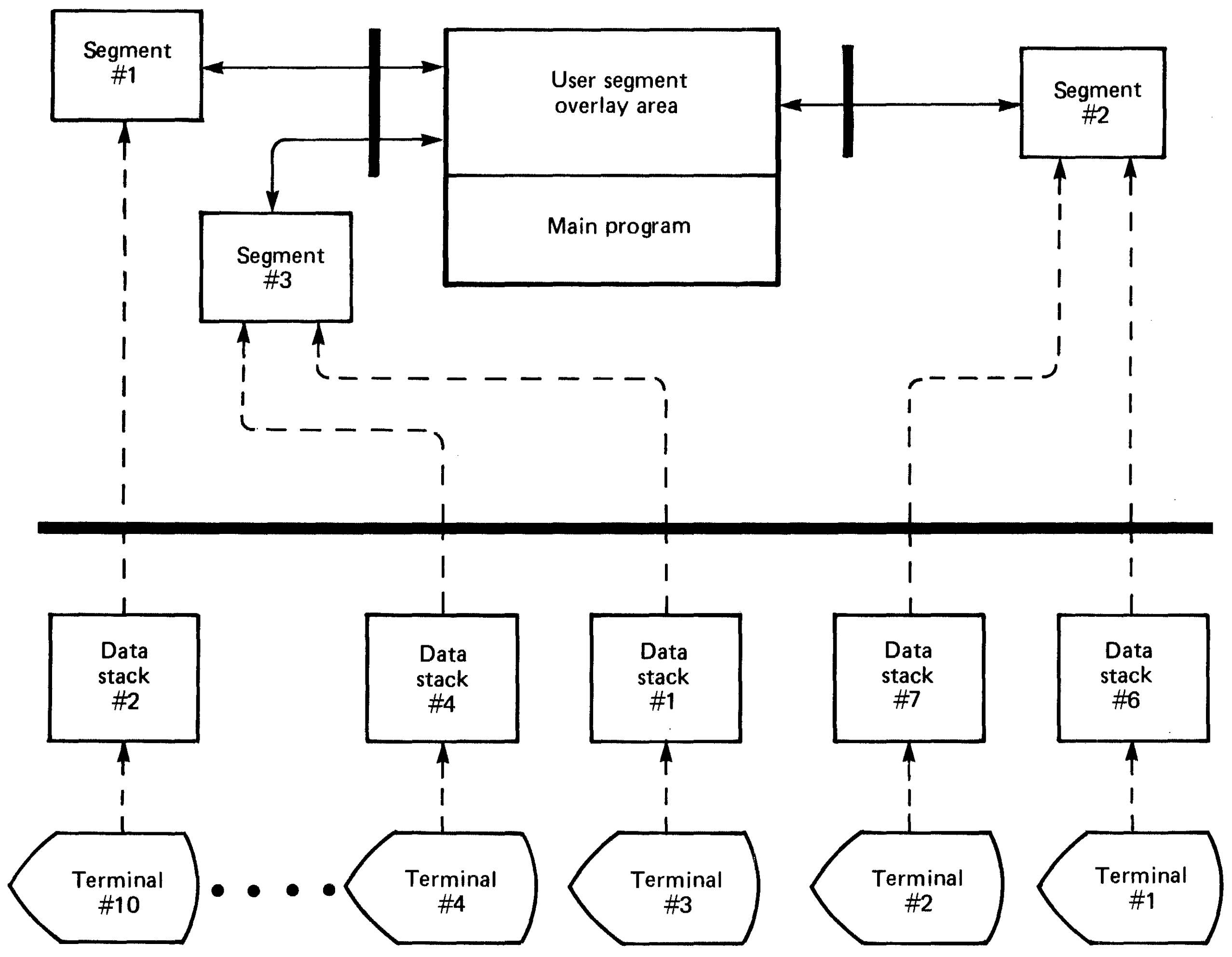


Figure 2. Paths Of Control Between System Elements



Notes: TCS (in conjunction with DOS-III) takes care of "swapping" user segments between the disc drive and the overlay area. Whenever a user segment becomes dormant, it passes the current stack pointer to TCS. When the particular segment again regains control, TCS passes back the proper stack pointer. The heavy black lines signify points where TCS exercises control. The above configuration is a "snap shot" of a particular point in time during execution of the application system. The main program and each user segment can actually be linked to any of the terminals (in any combination).

Figure 3. Segment "Swapping" and Passing of Data Stack Pointers Between TCS and Segments

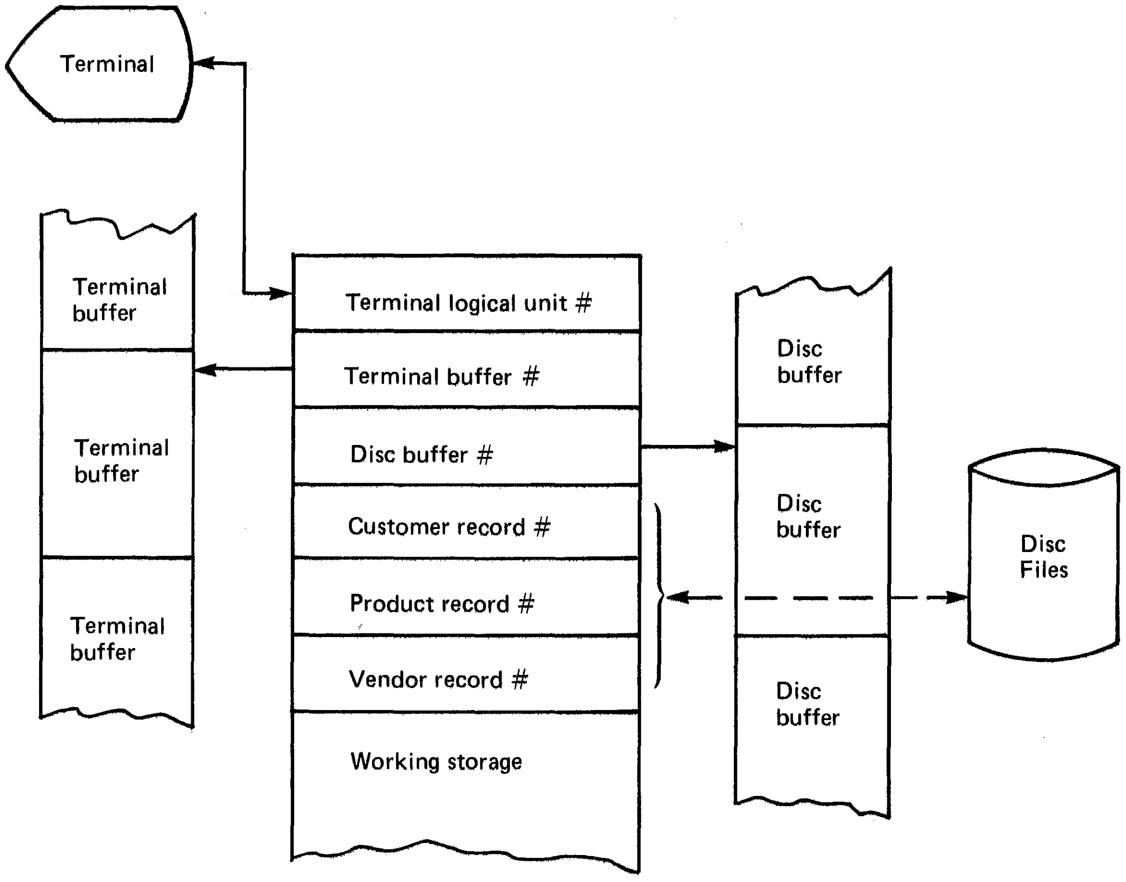


Figure 4. Linkage Between a Data Stack and its Associated Buffers, Disc Records, and Terminal

MAIN PROGRAM

During the following discussion, refer to Figure 5. This figure is in the form of a fold-out at the end of this section and can therefore be constantly exposed as you follow the text.

Before initializing TCS, we must first decide upon the size of the pending queue. This queue must be large enough to accommodate all I/O requests which cannot be performed because the required device is either busy or locked. A typical maximum situation might be that there are 35 pending requests: 5 disc requests, 10 invoice print requests, 10 terminal I/O requests, and 10 magnetic tape requests. To be certain that all pending requests can be accommodated, the pending queue for this example will be designed to accommodate 40 entries.

We must then decide upon the names of the various files and the three user segments. In this example they are named as follows:

Customer file:	CUST
Vendor file:	VEND
Product file:	PROD
Spooling file:	SPOL
Order entry segment:	SEG1
Goods receiving segment:	SEG2
Invoicing segment:	SEG3

Now place the segment names in one array and the file names in another (this is required by the TCS "initialization" and "open file" requests):

```
DIMENSION IN(9)
DATA IN/2HSE,2HG1,2H  ,2HSE,2HG2,2H  ,2HSE,2HG3,2H  /
DIMENSION IFI(12)
DATA IFI/2HCU,2HST,2H  ,2HVE,2HND,2H  ,2HPR,2HOD,2H  ,2HSP,2HOL,2H  /
```

Dimension the pending queue array, the segment directory array, and a work buffer (the work buffer is used for formatting prompt messages):

```
DIMENSION IPQ(360),ISD(33),K(40)
```

Set IA, IB, and IC to zero (the purpose of these variables is described later in this manual):

```
IA  = 0
IB  = 0
IC  = 0
```

Execute a TCS "initialization" request:

```
CALL TCS (82,IPQ,40,KK,IN,3,ISD)
```

The system has now been initialized.

Now we must open the four files. In this example this is done via a loop, as follows:

```
        DO 10 J=1,4
10      CALL TCS (84,IFI(((J-1)*3)+1),J)
```

At this point a prompt message must be printed on all the terminals asking the operator to enter the alphabetic code for the desired user function (OE = order entry; GR = goods receiving; IN = invoicing). Assume that the terminals' logical unit numbers are 10 through 19, inclusive. To do this, we must first get a terminal buffer:

```
        CALL GBUF (I,1)
```

Using the formatter, fill the work buffer K with the prompt message:

```
        CALL CODE
        WRITE (K,20)
20      FORMAT ("ENTER CODE FOR DESIRED FUNCTION")
```

Then move the message to the terminal buffer:

```
        DO 30 J=1,40
        IT(J,I) = K(J)
        IF (J.LT.17) GOTO 30
        IT(J,I) = 20040B
30      CONTINUE
```

The terminal buffer now contains the message in the first 16 words with spaces in the remaining words.

Set up the return address and then print the message on all the terminals (in this example the output requests are executed via a loop):

```
        L = 0
        ASSIGN 50 TO IRET
        DO 40 J=1,10
        KI = J+9
40      CALL TCS (2,20400B+KI,IT(1,I),40,J,IRET)
        CALL TCS (53)
        L = 0
```

The above output requests are *without wait.* As soon as each request is either queued or initiated by TCS, control returns immediately to the loop and the next output request is executed. When the loop is finished, the main program suspends itself until *any* of the requested output operations is complete, at which time control passes to statement 50 (the specified return address). The variable L is used below for keeping track of how many terminals have completed the above output operation.

At statement 50 the main program executes a TCS "status" request to find out which terminal has completed its output operation:

```
50      CALL TCS (79,ISTAT,IP,ILU,ILOG)
```

Now we must request input from the terminal to find out what the operator wishes to do.

First get another terminal buffer (unless this is the tenth terminal to respond, in which case we can use the same terminal buffer which was used for printing the prompt message):

```
        L = L+1
        IF (L.EQ.10) GOTO 70
60      CALL GBUF(II,1)
```

Check to see if a terminal buffer is available (if one is not, execute a TCS "pause" request and then try again):

```
        IF(II.NE.-1) GO TO 80
        CALL TCS (1,77B,1,1,ILU)
        CALL TCS (79,ISTAT,ILU,ILR,ILOG)
        GO TO 60
70      II = I
```

At this point we have a terminal buffer and can proceed with the input request:

```
80      CALL TCS (1,ILU,IT(1,II),40,II)
```

The above input request is *with wait* and control will not pass to the next statement (statement 90) until the input operation is complete. While waiting for the input operation to be completed, the main program may be re-initiated at statement 50 for other terminals whose output operation is complete.

When control passes to statement 90 it may be for any terminal, so we must execute a TCS "status" request to find out which terminal has completed its input operation.

```
90      CALL TCS (79,ISTAT,ILZ,ILU,ILOG)
```

The variable ILU now contains the logical unit number of the terminal which just completed the input operation and the variable ILZ contains the number of the terminal buffer which contains the associated operator entry. Check to make sure that the operator entry makes sense:

```
        JL = 0
        JLA = ASCII code for OE
        JLB = ASCII code for GR
        JLC = ASCII code for IN
        IF (IT(1,ILZ).EQ.JLA) GOTO 120
        IF (IT(1,ILZ).EQ.JLB) GOTO 110
        IF (IT(1,ILZ).EQ.JLC) GOTO 100
        GOTO 130
100     JL = JL+1
110     JL = JL+1
120     JL = JL+1
130     IF (JL.EQ.0) GOTO error-handler
```

JL now contains the number of the segment which performs the function desired by the operator. If JL = 0 when statement 130 is executed, the operator entry was invalid. In such a case the error-handling routine should print an appropriate error message, move the prompt message back into the terminal's buffer, and then pass control to statement 180.

Get a data stack and put the necessary information in it (i.e., the logical unit number of the particular teleprinter terminal and the number of the associated terminal buffer):

```
140     CALL GBUF (N,4)
        IS(1,N) = ILZ
        IS(2,N) = ILU
```

Now put the stack pointer in common and call the appropriate segment:

```
        IX = N
        CALL TCS (8,JL)
```

Check to make sure that the call was correct:

```
150     CALL TCS (79,ISTAT)
        IF (ISTAT.NE.0) GOTO error-handler
```

Control returns to statement 150 under either of the following two conditions:

1. The specified segment could not be found on the disc. In this case control returns immediately to statement 150 with ISTAT set to -4. The main program must then pass control to an error-handling routine which prints an appropriate error message and then passes control to statement 160.

2. The specified segment has completed its execution. In this case control returns to statement 150 with ISTAT set to 0.

In either case, the stack pointer is still in IX.

Move the prompt message ("ENTER CODE FOR DESIRED FUNCTION") back into the terminal's buffer:

```
160   IST = IS(1,IX)
      DO 170 J=1,40
      IT(J,IST) = K(J)
      IF(J.LT.17) GOTO 170
      IT(J,IST) = 20040B
170   CONTINUE
```

Print the message on the terminal after releasing the data stack:

```
      IZ = IS(2,IX)
      CALL PBUF (IX,4)
180   CALL TCS (2,IZ,IT(1,IST),40,IST)
```

Now request input from the terminal after determining (via a TCS "status" request) the logical unit number of the particular terminal and the number of the associated terminal buffer:

```
      CALL TCS (79,ISTAT,IST,ILS,ILOG)
      CALL TCS (1,400B+ILS,IT(1,IST),40,IST)
```

When the input operation is complete, transfer control back to the input interpreter portion of the main program.

```
      GOTO 90
```

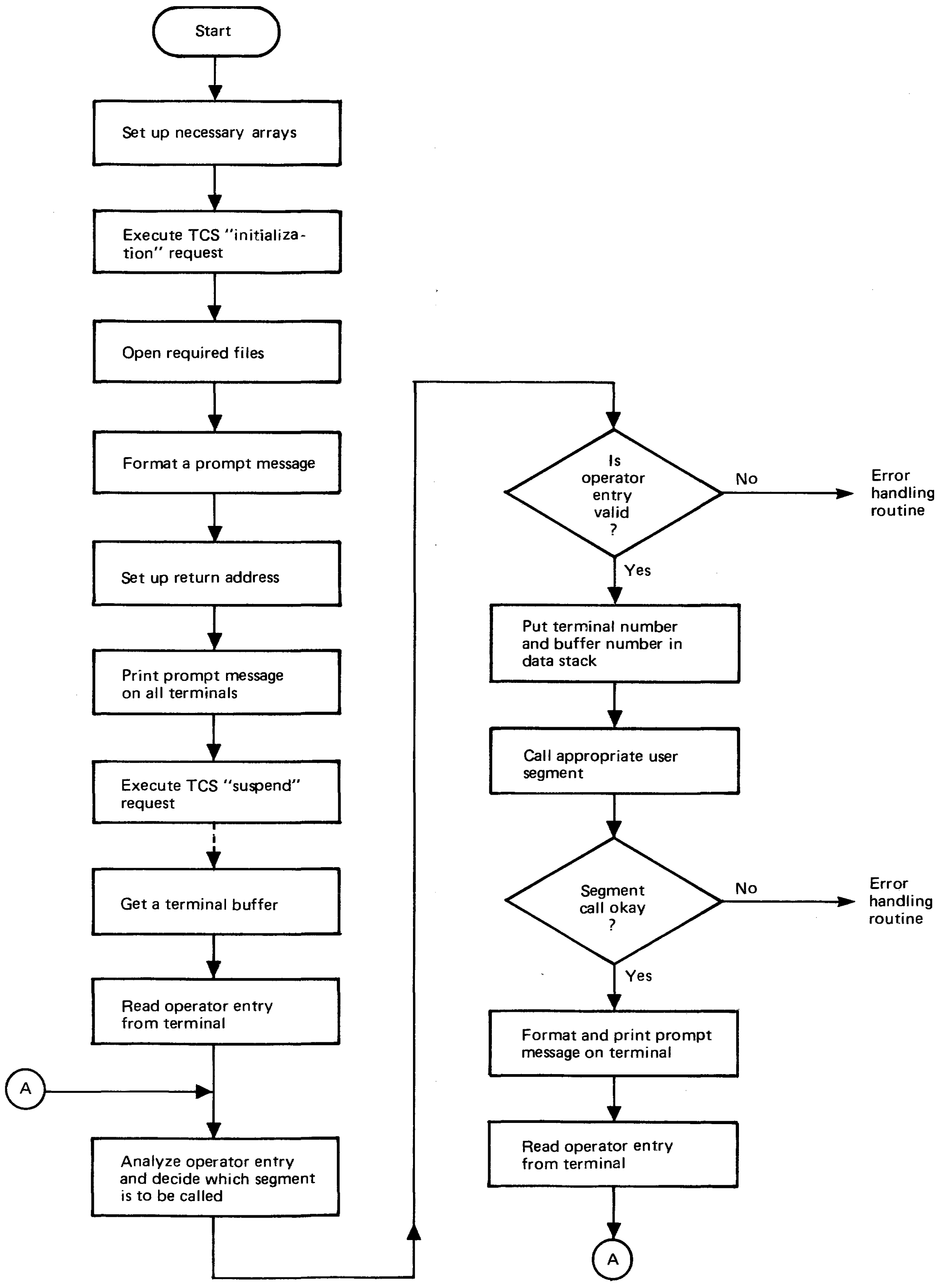

Start
Set up necessary arrays
Execute TCS "initialization" request
Open required files
Format a prompt message
Set up return address
Print prompt message on all terminals
Execute TCS "suspend" request
Get a terminal buffer
Read operator entry from terminal
A
Analyze operator entry and decide which segment is to be called
Is operator entry valid ?
No
Error handling routine
Yes
Put terminal number and buffer number in data stack
Call appropriate user segment
Segment call okay ?
No
Error handling routine
Yes
Format and print prompt message on terminal
Read operator entry from terminal
A

Figure 5. Main Program Flowchart

SEGMENT 1: ORDER ENTRY

During the following discussion, refer to Figure 6. This figure is in the form of a fold-out at the end of this section and can therefore be constantly exposed as you follow the text.

This segment processes newly-received customer orders entered by operators through the terminals. It uses the following files which must previously have been opened by the main program:

Customer file:	CUST
Vendor file:	VEND
Product file:	PROD
Disc spooling file:	SPOL

The formats of the records in these files need not be precisely defined for this example, but the records would normally contain the following data:

CUSTOMER RECORD:

a. Customer number
b. Name
c. Address
d. Delivery address
e. "Bill to" address
f. Credit limit
g. Current credit
h. Total purchases during current month
i. Total purchases during current year
j. Discount schedules

VENDOR RECORD:

a. Vendor number
b. Name
c. Address
d. Amount owed to vendor
e. Receipts for the current day

PRODUCT RECORD:

a. Product number
b. Description
c. Kind of packaging
d. Minimum quantity to be kept on hand
e. Current stock level
f. Reorder point
g. Vendor number
h. Cost
i. Selling price
j. Discount schedules
k. Tax group
l. Deposit required
m. Total amount sold during current month
n. Total amount sold during current year

DISC SPOOLING RECORD:

a. Terminal number (logical unit number)
b. Customer record number
c. Product record number
d. "end-of-invoice" indicator

The information required by the segment (i.e., the logical unit number of the terminal and the number of the associated terminal buffer) is contained in a data stack created by the main program. The data stack pointer is in the variable IX in common.

The first task is to dimension a work buffer (to be used for formatting output) and obtain the stack pointer:

```
DIMENSION K(40)
I = IX
```

Using the formatter, format a prompt message to the operator in the terminal buffer pointed to by the first word of the data stack:

```
10    CALL CODE
      WRITE (K,20)
20    FORMAT ("ENTER CUSTOMER NUMBER")
      DO 30 J=1,40
      IT(J,IS(1,I)) = K(J)
      IF (J.LT.12) GOTO 30
      IT(J,IS(1,I)) = 20040B
30    CONTINUE
```

Now print the message on the terminal pointed to by the second word of the data stack. This output request is *with wait*. As a result, another user function could be scheduled which would cause this segment to be overlaid, or this segment could be scheduled for another terminal. Therefore, to preserve the system, the stack pointer is passed to TCS with the output request:

```
      CALL TCS (2,IS(2,I),IT(1,IS(1,I)),40,I)
```

The segment will be recalled when the output operation is finished. When that happens, we must restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Read the operator entry from the terminal *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (1,ILU,IT(1,IS(1,I)),40,I)
```

When the input operation is finished, restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

We now have the stack pointer and (via the stack) the number of the terminal buffer which contains the operator entry. Check to see if the operator entered /E instead of a customer number:

IF (IT(1,IS(1,I)).NE.*ASCII code for /E*) GOTO 40

If the operator entered /E, this signals that he has no more orders to enter. In that case, we must execute a TCS "return to main" request after putting the stack pointer back in IX:

```
      IX = I
      CALL TCS (54)
40    CONTINUE
```

If the operator did *not* enter /E, we must call a subroutine to convert the customer number to a customer record number via a hashing algorithm or some other suitable means. We pass the stack pointer to the subroutine which places the customer record number in the third word of the data stack:

```
          CALL SUB1 (I)
```

If the subroutine had had to make disc accesses (as it would if a disc-resident index or look-up table is used), it could be made re-entrant by putting a return address obtained via an ASSIGN statement in the data stack.

Get a disc buffer (if none is available, execute a TCS "pause" request and then try again):

```
50        CALL GBUF (J,2)
          IF (J.NE.-1) GOTO 60
          CALL TCS (1,77B,1,1,I)
          CALL TCS (79,ISTAT,I,ILU,ILOG)
          GOTO 50
60        CONTINUE
```

Put the disc buffer number in the fifth word of the data stack:

```
          IS(5,I) = J
```

Now read the customer record *with wait* (passing the stack pointer to TCS):

```
          CALL TCS (14,3,ID(1,J),128,1,IS(3,I),I)
```

When the disc read operation is finished, restore the stack pointer:

```
          CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Examine the customer record and determine whether or not he is permitted to buy (credit limit checks, etc.).

Release the disc buffer:

```
          CALL PBUF (IS(5,I),2)
```

If the customer is permitted to buy, skip to statement 100; if he is *not* permitted to buy, proceed with statement 70.

```
70        CONTINUE
```

Using the formatter, format an error message in the terminal's buffer:

```
      CALL CODE
      WRITE (K,80)
80    FORMAT("CUSTOMER CANNOT BUY")
      DO 90 J=1,40
      IT(J,IS(1,I)) = K(J)
      IF (J.LT.11) GOTO 90
      IT(J,IS(1,I)) = 20040B
90    CONTINUE
```

Now print the message on the terminal *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (2,IS(2,I),IT(1,IS(1,I)),40,I)
```

When the output operation is finished, restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Now request another customer number:

```
      GOTO 10
100   CONTINUE
```

If the customer is permitted to buy, use the formatter to format a message in the terminal's buffer asking the operator for the product number:

```
      CALL CODE
      WRITE (K,110)
110   FORMAT ("ENTER PRODUCT NUMBER")
      DO 120 J=1,40
      IT(J,IS(1,I)) = K(J)
      IF (J.LT.11) GOTO 120
      IT(J,IS(1,I)) = 20040B
120   CONTINUE
```

Now print the message on the terminal *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (2,IS(2,I),IT(1,IS(1,I)),40,I)
```

When the output operation is finished, restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Read the operator entry from the terminal *with wait* (passing the stack pointer to TCS):

```
CALL TCS (1,ILU,IT(1,IS(1,I)),40,I)
```

When the input operation is finished, restore the stack pointer:

```
CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Check to see if the operator entered /E instead of an item number:

IF (IT(1,IS(1,I)).NE.*ASCII code for /E*) GOTO 140

If the operator did *not* enter /E, skip to statement 140; if he did enter /E, proceed with statement 130.

```
130     CONTINUE
```

If the operator entered /E, this signals the end of the current order. In that case we must generate an "end-of-invoice" spooling record, add the "end-of-invoice" record to the disc spooling file, write the "end-of-invoice" record onto magnetic tape (as back-up in case of a disc failure), and then proceed with the next order.

Format the "end-of-invoice" spooling record:

IS(6,I) = IS(2,I)	*Terminal number (logical unit number)*
IS(7,I) = IS(3,I)	*Customer record number*
IS(8,I) = 0	*Product record number*
IS(9,I) = 1	*"end-of-invoice" indicator (set)*

Call a subroutine to add the "end-of-invoice" spooling record to the disc spooling file (passing the stack pointer to the subroutine):

```
CALL SUB2 (I)
```

Write the "end-of-invoice" spooling record onto magnetic tape *with wait* (passing the stack pointer to TCS):

```
CALL TCS (2,8,IS(6,I),4,I)
```

When the output operation is finished, restore the stack pointer:

```
CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Now proceed with the next order:

```
        GOTO 10
140     CONTINUE
```

If the operator did *not* enter /E, we must call a subroutine to convert the product number to a product record number via a hashing algorithm or some other suitable means. We pass the stack pointer to the subroutine which places the product record number in the fourth word of the data stack:

```
          CALL SUB3 (I)
```

At this point we must update the product record to reflect the new "amounts sold" totals (current month, current year, etc.) and the new "current stock level" amount. We will read the record from the disc *with lock*. This prevents any other segments from updating the product file at the same time. However, we must also prevent this segment from trying to update the product file from two different terminals at the same time. This problem is overcome by using the variable IA in common. If IA = 1, then the segment cannot proceed because it is already updating a product record (in such a case, the segment must execute a TCS "pause" request and then test IA again).

```
    150   IF (IA.NE.1) GOTO 160
          CALL TCS (1,77B,1,1,I)
          CALL TCS (79,ISTAT,I,ILU,ILOG)
          GOTO 150
    160   IA = 1
```

Get a disc buffer (if none is available, execute a TCS "pause" request and then try again):

```
    170   CALL GBUF (J,2)
          IF (J.NE.-1) GOTO 180
          CALL TCS (1,77B,1,1,I)
          CALL TCS (79,ISTAT,I,ILU,ILOG)
          GOTO 170
    180   CONTINUE
```

Put the disc buffer number in the fifth word of the data stack:

```
          IS(5,I) = J
```

Now read the product record *with lock* and *with wait* (passing the stack pointer to TCS):

```
          CALL TCS (14,100003B,ID(1,IS(5,I)),128,3,IS(4,I),I)
```

When the disc read operation is finished, restore the stack pointer:

```
          CALL TCS (79,ISTAT,I,ILU,ILOG)
```

The product record is now in the disc buffer whose number is in the fifth word of the data stack. Examine the product record, checking to see if there is any stock left, etc. If all is okay, update the "amounts sold" totals (current month, current year, etc.) in the product record and subtract one from the "current stock level" field in the product record. Then write the record back to the disc *with wait* (passing the stack pointer to TCS):

```
*Arithmetic and updating statements*
CALL TCS (15,3,ID(1,IS(5,I)),128,3,IS(4,I),I)
```

When the disc output operation is finished, restore the stack pointer:

```
CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Release the disc buffer and unlock the disc drive:

```
CALL PBUF (IS(5,I),2)
CALL TCS (52,3)
```

Reset IA to zero to permit this segment to update the product file for other terminals:

```
IA = 0
```

Format a spooling record for the current transaction using words 6 through 9 of the data stack:

```
IS(6,I) = IS(2,I)        Terminal number (logical unit number)
IS(7,I) = IS(3,I)        Customer record number
IS(8,I) = IS(4,I)        Product record number
IS(9,I) = 0              "end-of-invoice" indicator (cleared)
```

Call a subroutine to add the spooling record to the disc spooling file:

```
CALL SUB2 (I)
```

Write the spooling record onto magnetic tape *wait wait* as back-up in case of a disc failure (passing the stack pointer to TCS):

```
CALL TCS (2,8,IS(6,I),4,I)
```

After the output operation is finished, restore the stack pointer:

```
CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Request the next product number:

```
GOTO 100
```

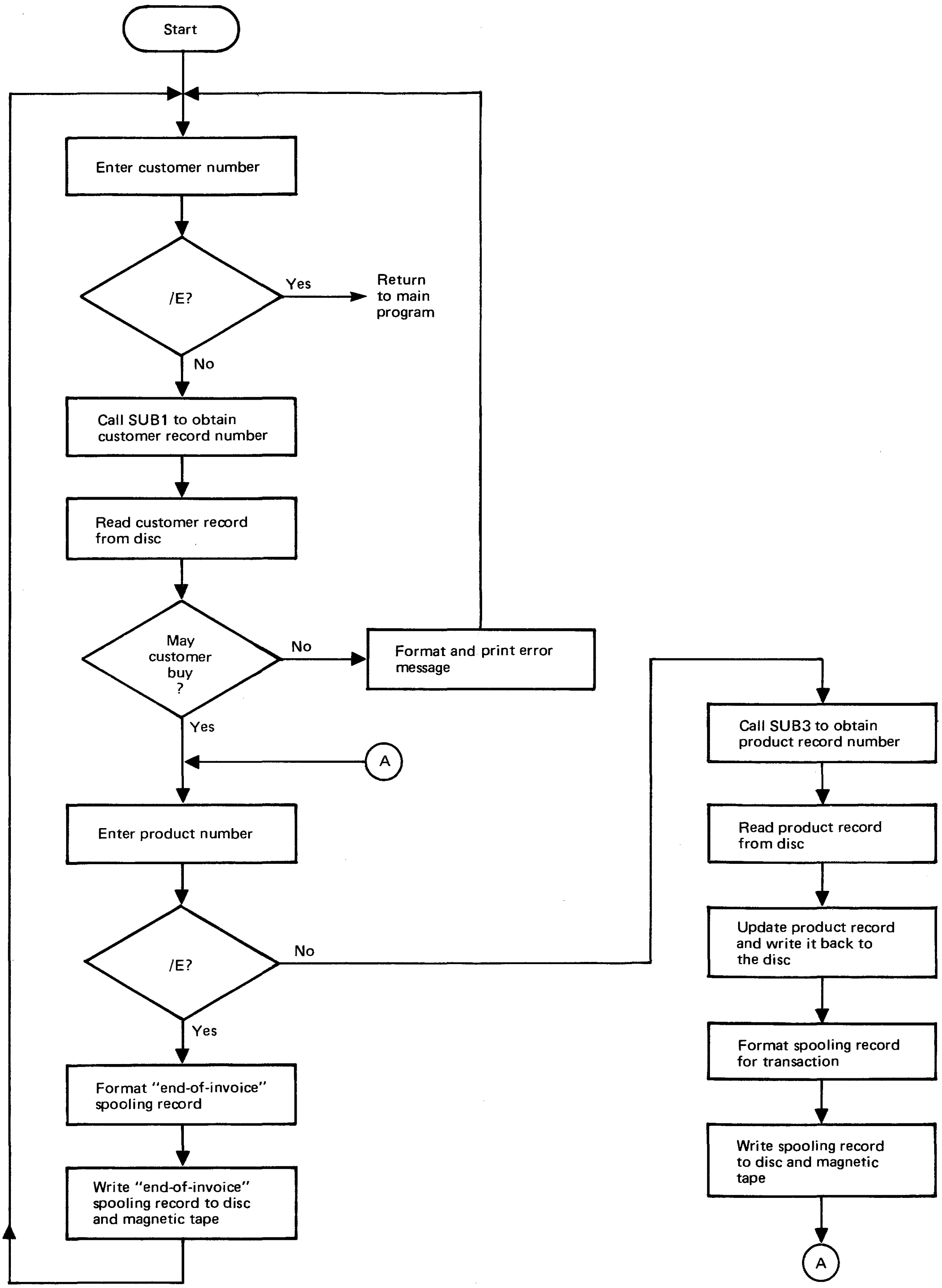

Start
Enter customer number
/E?
Yes
Return to main program
No
Call SUB1 to obtain customer record number
Read customer record from disc
May customer buy ?
No
Format and print error message
Yes
A
Enter product number
/E?
No
Yes
Call SUB3 to obtain product record number
Read product record from disc
Update product record and write it back to the disc
Format spooling record for transaction
Write spooling record to disc and magnetic tape
A
Format "end-of-invoice" spooling record
Write "end-of-invoice" spooling record to disc and magnetic tape

Figure 6. Segment 1 (Order Entry) Flowchart

SEGMENT 2: GOODS RECEIVING

During the following discussion, refer to Figure 7. This figure is in the form of a fold-out at the end of this section and can therefore be constantly exposed as you follow the text.

This segment processes newly-received shipments from vendors entered by operators through the terminals. It uses the following files which must previously have been opened by the main program:

Vendor file:	VEND
Product file:	PROD

The formats of the records in these files need not be precisely defined for this example, but the records would normally contain the following data:

VENDOR RECORD:

a. Vendor number

b. Name

c. Address

d. Amount owed to vendor

e. Receipts for the current day

PRODUCT RECORD:

a. Product number

b. Description

c. Kind of packaging

d. Minimum quantity to be kept on-hand

e. Current stock level

f. Reorder point

g. Vendor number

h. Cost

i. Selling price

j. Discount schedules

k. Tax group

l. Deposit required

m. Total amount sold during current month

n. Total amount sold during current year

The information required by the segment (i.e., the logical unit number of the terminal and the number of the associated terminal buffer) is contained in a data stack created by the main program. The data stack pointer is in the variable IX in common.

First we must dimension a work buffer (to be used for formatting output), obtain the stack pointer, and set the seventh word of the data stack to zero (this word is used for accumulating the new amount owed to the current vendor):

```
      DIMENSION K(40)
      I = IX
      IS(7,I) = 0
```

Using the formatter, format a prompt message to the operator in the terminal buffer pointed to by the first word in the data stack:

```
10    CALL CODE
      WRITE (K,20)
20    FORMAT ("ENTER VENDOR NUMBER")
      DO 30 J=1,40
      IT(J,IS(1,I)) = K(J)
      IF (J.LT.11) GOTO 30
      IT(J,IS(1,I)) = 20040B
30    CONTINUE
```

Now print the message on the terminal pointed to by the second word of the data stack. This output request is *with wait*. As a result, another user function could be scheduled which would cause this segment to be overlaid, or this segment could be scheduled for another terminal. Therefore, to preserve the system, the stack pointer is passed to TCS with the output request:

```
      CALL TCS (2,IS(2,I),IT(1,IS(1,I)),40,I)
```

The segment will be recalled when the output operation is finished. When that happens, we must restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Read the operator entry from the terminal *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (1,ILU,IT(1,IS(1,I)),40,I)
```

When the input operation is finished, restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

We now have the stack pointer and (via the stack) the number of the terminal buffer which contains the operator entry. Check to see if the operator entered /E instead of a vendor number:

```
      IF (IT(1,IS(1,I)).NE.ASII code for /E) GOTO 40
```

If the operator entered /E, this signals that he has no more receipts to enter. In that case, we must execute a TCS "return to main" request after putting the stack pointer back in IX:

```
      IX = I
      CALL TCS (54)
40    CONTINUE
```

If the operator did *not* enter /E, we must call a subroutine to convert the vendor number to a vendor record number via a hashing algorithm or some other suitable means. We pass the stack pointer to the subroutine which places the vendor record number in the third word of the data stack:

```
      CALL SUB4 (I)
```

If the subroutine had had to make disc accesses (as it would if a disc-resident index or look-up table is used), it could be made re-entrant by putting a return address obtained via an ASSIGN statement in the data stack.

Now we must ask the operator for the details of the delivery. This input is requested in two parts:

1. Product number
2. Quantity received

Using the formatter, format a prompt message in the terminal buffer requesting the product number:

```
50    CALL CODE
      WRITE (K,60)
60    FORMAT ("ENTER PRODUCT NUMBER")
      DO 70 J=1,10
70    IT(J,IS(1,I)) = K(J)
```

Now print the message on the terminal *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (2,IS(2,I),IT(1,IS(1,I)),10,I).
```

When the output operation is finished, restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Read the operator entry from the terminal *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (1,IS(2,I),IT(1,IS(1,I)),10,I)
```

When the input operation is finished, restore the stack pointer:

```
        CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Check to see if the operator entered /E instead of a product number:

```
        IF (IT(1,IS(1,I)).NE.ASCII code for /E) GOTO 130
```

If the operator entered /E, this signals that there are no more product receipts to be recorded for this vendor at this time. In that case, we must update the "amount owed to vendor" field in the appropriate vendor record, and then proceed with the next vendor.

If the operator did *not* enter /E, skip to statement 130; if he entered /E, proceed with statement 80.

```
  80    CONTINUE
```

At this point we must obtain the appropriate vendor record. We will read the record from the disc *with lock*. This prevents other segments from updating the vendor file at the same time. However, we must also prevent this segment from trying to update the vendor file from two different terminals at the same time. This problem is overcome by using the variable IB in common. If IB = 1, then the segment cannot proceed because it is already updating a vendor record (in such a case, the segment must execute a TCS "pause" request and then test IB again).

```
  90    IF (IB.EQ.0) GOTO 100
        CALL TCS (1,77B,1,1,I)
        CALL TCS (79,ISTAT,I,ILU,ILOG)
        GOTO 90
 100    IB = 1
```

Get a disc buffer (if none is available, execute a TCS "pause" request and then try again):

```
 110    CALL GBUF (J,2)
        IF (J.NE.-1) GOTO 120
        CALL TCS (1,77B,1,1,I)
        CALL TCS (79,ISTAT,I,ILU,ILOG)
        GOTO 110
```

Put the disc buffer number in the sixth word of the data stack:

```
 120    IS(6,I) = J
```

Now read the vendor record from the disc *with lock* and *with wait* (passing the stack pointer to TCS):

```
        CALL TCS (14,100003B,ID(1,IS(6,I)),128,2,IS(3,I),I)
```

When the disc read operation is finished, restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

The vendor record is now in the disc buffer whose number is in the sixth word of the data stack. Add the total contained in the seventh word of the data stack to the "amount owed to vendor" field in the vendor record. Then write the record back to the disc *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (15,3,ID(1,IS(6,I)),128,2,IS(3,I),I)
```

When the disc output operation is finished, restore the stack pointer:

```
      CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Unlock the disc drive and reset IB to zero:

```
      CALL TCS (52,3)
      IB = 0
```

Release the disc buffer, reset the seventh word of the data stack to zero, and proceed with the next vendor:

```
      CALL PBUF (IS(6,I),2)
      IS(7,I) = 0
      GOTO 10
130   CONTINUE
```

If the operator did *not* enter /E, we must call a subroutine to convert the product number to a product record number via a hashing algorithm or some other suitable means. We pass the stack pointer to the subroutine which places the product record number in the fourth word of the data stack:

```
      CALL SUB3 (I)
```

Using the formatter, format a prompt message in the terminal buffer requesting the quantity:

```
      CALL CODE
      WRITE (K,140)
140   FORMAT ("QUANTITY?")
      DO 150 J=1,5
150   IT(J,IS(1,I)) = K(J)
```

Now print the message on the terminal *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (2,IS(2,I),IT(1,IS(1,I)),2,I)
```

When the output operation is finished, restore the stack pointer:

```
CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Read the operator entry from the terminal *with wait* (passing the stack pointer to TCS):

```
CALL TCS (1,IS(2,I),IT(1,IS(1,I)),3,I)
```

When the input operation is finished, restore the stack pointer:

```
CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Using the formatter, convert the operator entry to an integer:

```
      DO 160 J=1,3
      K(J) = IT(J,IS(1,I))
      CALL CODE
160   READ (K,*)JA
```

Now put the converted quantity in the fifth word of the data stack:

```
IS(5,I) = JA
```

We must now read the appropriate product record from the disc, get the buying price from the record, and multiply that by the quantity received. This tells us how much we owe the vendor for this product. We must also update the "current stock level" field of the product record to include the newly-received quantity.

We read the record from the disc *with lock*. This prevents any other segments from updating the product file at the same time. However, we must also prevent this segment from trying to update the product file from two different terminals at the same time. This problem is overcome by using the variable IC in common. If IC = 1, then the segment cannot proceed because it is already updating a product record (in such a case, the segment must execute a TCS "pause" request and then test IC again).

```
170   IF (IC.EQ.0) GOTO 180
      CALL TCS (1,77B,1,1,I)
      CALL TCS (79,ISTAT,I,ILU,ILOG)
      GOTO 170
180   IC = 1
```

Get a disc buffer (if one is not available, execute a TCS "pause" request and then try again):

```
190     CALL GBUF (J,2)
        IF (J.NE.-1) GOTO 200
        CALL TCS (1,77B,1,1,I)
        CALL TCS (79,ISTAT,I,ILU,ILOG)
        GOTO 190
```

Put the disc buffer number in the sixth word of the data stack:

```
200     IS(6,I) = J
```

Now read the product record from the disc *with lock* and *with wait* (passing the stack pointer to TCS):

```
        CALL TCS (14,100003B,ID(1,J),128,3,IS(4,I),I)
```

When the disc read operation is finished, restore the stack pointer:

```
        CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Now multiply the cost (obtained from the product record) by the quantity (obtained from the fifth word of the data stack). Add the result into the seventh word of the data stack. Then add the quantity received to the "current stock level" field of the product record. Write the record back to the disc *with wait* (passing the stack pointer to TCS):

```
        Arithmetic and updating statements
        CALL TCS (15,3,ID(1,IS(6,I)),128,3,IS(4,I),I)
```

When the disc output operation is finished, restore the stack pointer:

```
        CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Release the disc buffer and unlock the disc drive:

```
        CALL PBUF (IS(6,I),2)
        CALL TCS (52,3)
```

Reset IC to zero to permit this segment to update the product file from other terminals:

```
        IC = 0
```

Request the next product number:

```
        GOTO 50
```

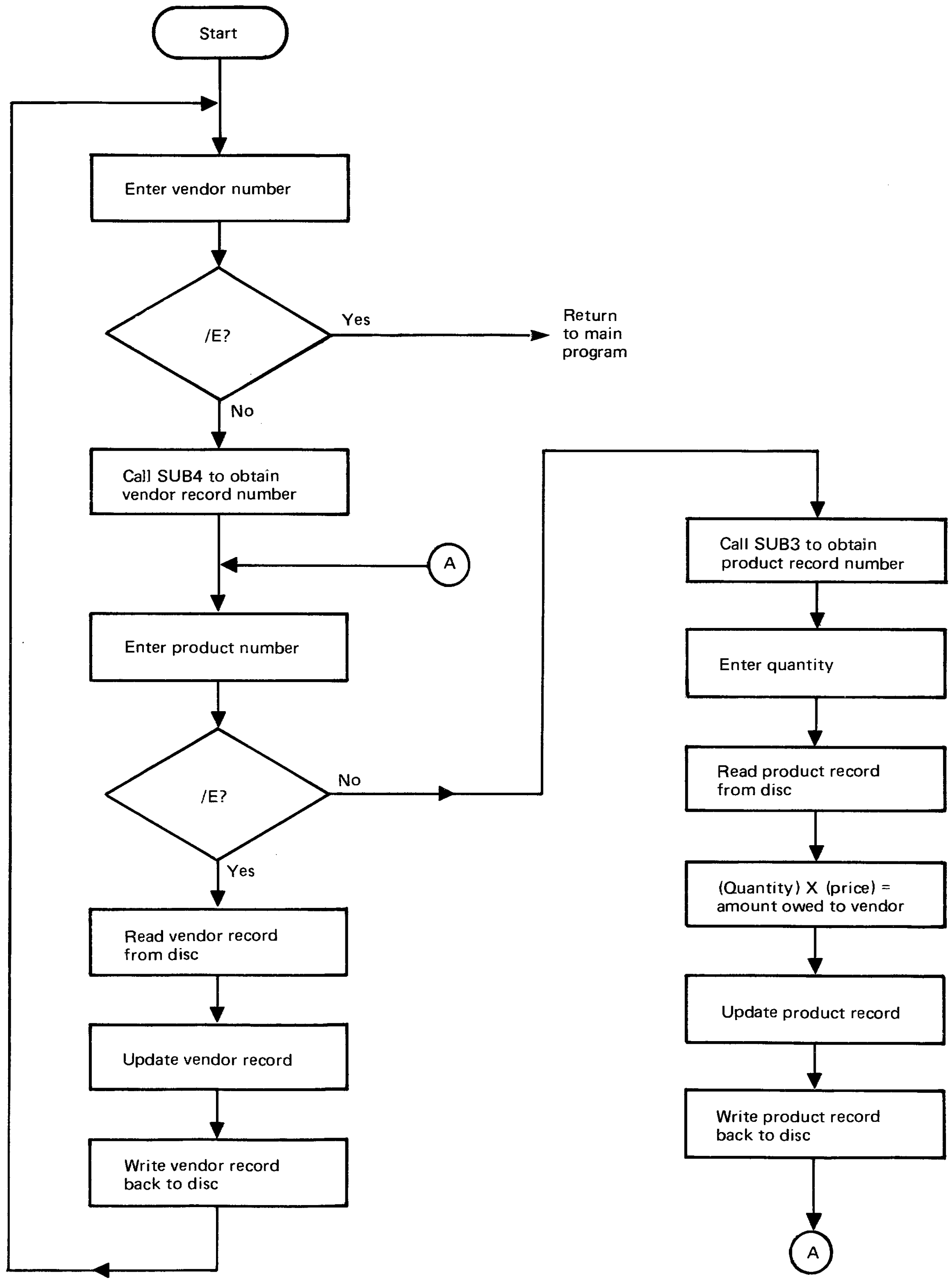

Start
Enter vendor number
/E?
Yes
Return to main program
No
Call SUB4 to obtain vendor record number
A
Enter product number
/E?
No
Yes
Read vendor record from disc
Update vendor record
Write vendor record back to disc
Call SUB3 to obtain product record number
Enter quantity
Read product record from disc
(Quantity) X (price) = amount owed to vendor
Update product record
Write product record back to disc
A

Figure 7. Segment 2 (Goods Receiving) Flowchart

SEGMENT 3: INVOICING

During the following discussion, refer to Figure 8. This figure is in the form of a fold-out at the end of this section and can therefore be constantly exposed as you follow the text.

This segment prints invoices on the invoking terminal using the disc spooling file. A terminal can produce invoices only for those orders which were entered through that particular terminal. Since this segment never writes to the disc, disc locking is not required.

The disc spooling file consists of a series of 4-word records of the following format:

Word 1:	Terminal number (logical unit number)
Word 2:	Customer record number
Word 3:	Product record number
Word 4:	"end-of-invoice" indicator

If the fourth word of a spooling record contains a "1," the particular record is merely an "end-of-invoice" record (in this case the "product record number" field of the record is blank).

First get the stack pointer from common:

```
      I = IX
```

Get a line printer buffer (if one is not available, execute a TCS "pause" request and then try again):

```
10    CALL GBUF (J,3)
      IF(J.NE.-1) GOTO 20
      CALL TCS (1,77B,1,1,I)
      CALL TCS (79,ISTAT,I,ILU,ILOG)
      GOTO 10
20    CONTINUE
```

Put the line printer buffer number in the third word of the data stack:

```
      IS(3,I) = J
```

Now get a disc buffer (if one is not available, execute a TCS "pause" request and then try again):

```
30    CALL GBUF (J,2)
      IF (J.NE.-1) GOTO 40
      CALL TCS (1,77B,1,1,I)
      CALL TCS (79,ISTAT,I,ILU,ILOG)
      GOTO 30
40    CONTINUE
```

Put the disc buffer number in the fourth word of the data stack:

```
      IS(4,I) = J
```

The segment now examines the spooling file serially and prints invoices on the terminal. We pass the stack pointer to the subroutine SUB5 which obtains the next spooling record and places it in the last four words of the data stack. The use of the tenth and eleventh words of the data stack is described later in this section.

```
      IS(10,I) = 0
      IS(11,I) = 0
50    CALL SUB5 (I)
```

Now check to see if the logical unit number in the spooling record is zero ("end-of-file" record):

```
      IF (IS(12,I).NE.0) GOTO 60
```

If the logical unit number in the spooling record is zero, we have reached the end of the spooling file. In such a case, execute a TCS "return to main" request after putting the stack pointer back into IX and releasing the buffers:

```
      CALL PBUF (IS(3,I),3)
      CALL PBUF (IS(4,I),2)
      IX = I
      CALL TCS (54)
60    CONTINUE
```

If the logical unit number in the spooling record is non-zero, check to see if the spooling record was created through this terminal (if it wasn't, ignore it):

```
      IF (IS(12,I).NE.IS(2,I)) GOTO 50
```

The eleventh word in the data stack is used for specifying whether or not the current spooling record is the first item on the invoice. IS(11,I) = 0 specifies that it is the first; IS(11,I) ≠ 0 specifies that it is not. If the current spooling record is the first item on the invoice, then we must print the customer line (name, address, etc.) before printing the detail line.

Check to see if the current spooling record is the first item on the invoice:

```
      IF (IS(11,I).NE.0) GOTO 70
```

If it is the first item, read the customer record from the disc *with wait* (passing the stack pointer to TCS):

```
      CALL TCS (14,3,ID(1,IS(4,I)),128,1,IS(13,I),I)
```

When the disc read operation is finished, restore the stack pointer:

```
          CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Format the customer line in the terminal's buffer and then print it *with wait* (passing the stack pointer to TCS):

Formatting statements

```
          CALL TCS (2,IS(2,I),IT(1,IS(3,I)),55,I)
```

When the output operation is finished, restore the stack pointer and set the eleventh word of the data stack to "1":

```
          CALL TCS (79,ISTAT,I,ILU,ILOG)
          IS(11,I) = 1
     70   CONTINUE
```

At this point we must print the detail line and accumulate an order total in the tenth word of the data stack. First, check to see if the current spooling record is the "end-of-invoice" record. IS(15,I) = 1 signifies "end-of-invoice."

```
          IF (IS(15,I).NE.1) GOTO 90
```

If the current spooling record is *not* the "end-of-invoice" record, skip to statement 90; if it is the "end-of-invoice" record, proceed with statement 80.

```
     80   CONTINUE
```

If the current spooling record is the "end-of-invoice" record, a detail line is *not* printed. Instead, we print the invoice total line, reset the tenth and eleventh words of the data stack to zero, and proceed with the next invoice.

Format the invoice total line in the terminal's buffer and then print the line *with wait* (passing the stack pointer to TCS):

Formatting statements

```
          CALL TCS (2,IS(2,I),IT(1,IS(3,I)),66,I)
```

When the output operation is finished, restore the stack pointer, reset the tenth and eleventh words of the data stack to zero, and proceed with the next invoice:

```
          CALL TCS (79,ISTAT,I,ILU,ILOG)
          IS(10,I) = 0
          IS(11,I) = 0
          GOTO 50
     90   CONTINUE
```

Read the product record from the disc *with wait* (passing the stack pointer to TCS):

```
CALL TCS (14,3,ID(1,IS(4,I)),128,3,IS(14,I),I)
```

When the disc read operation is finished, restore the stack pointer:

```
CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Format the detail line in the terminal's buffer and accumulate the order total in the tenth word of the data stack.

Now print the detail line *with wait* (passing the stack pointer to TCS):

```
CALL TCS (2,IS(2,I),IT(1,IS(3,I)),66,I)
```

When the output operation is finished, restore the stack pointer:

```
CALL TCS (79,ISTAT,I,ILU,ILOG)
```

Obtain the next spooling record:

```
GOTO 50
```

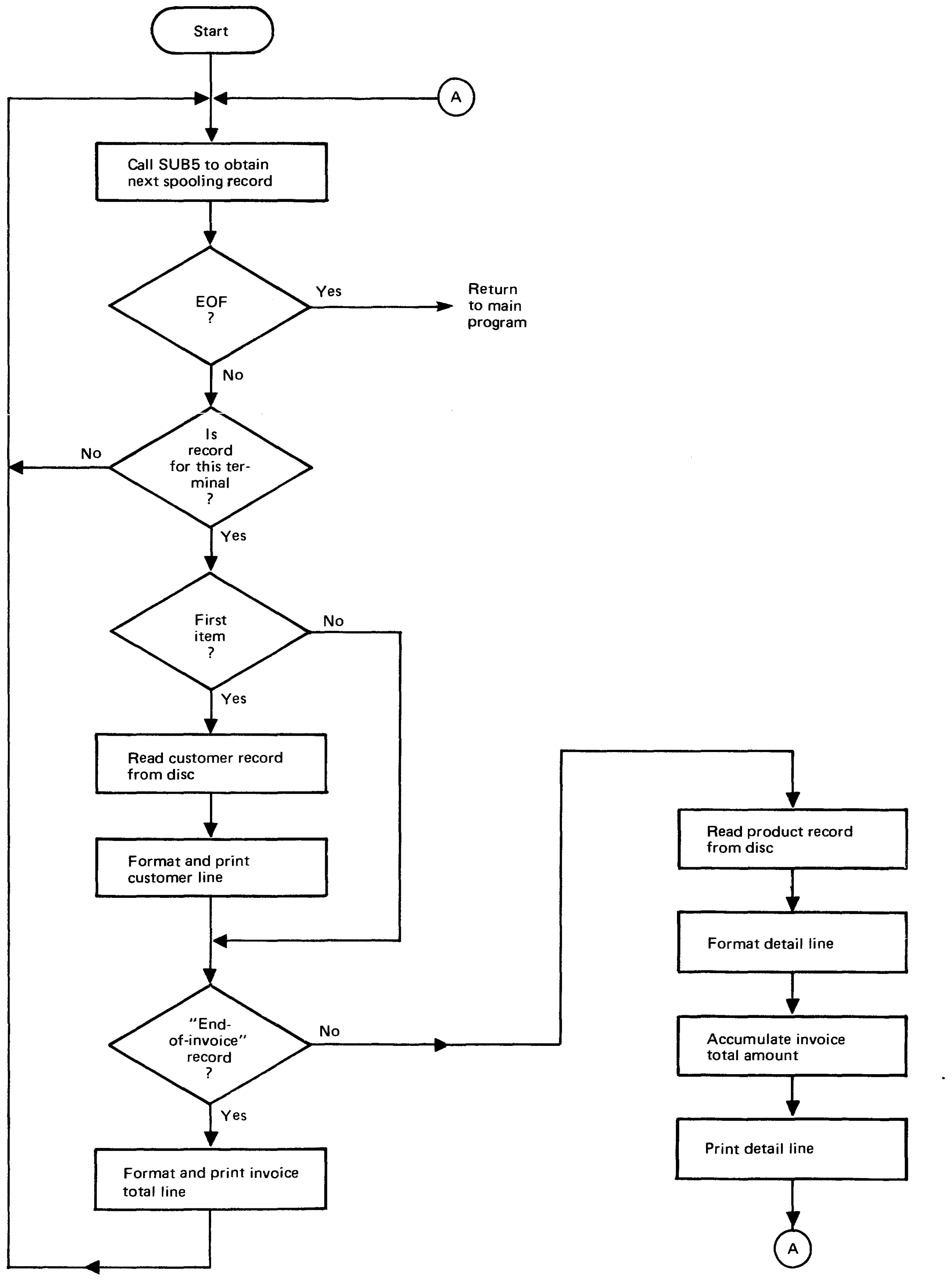

Start
A
Call SUB5 to obtain next spooling record
EOF ?
Yes
Return to main program
No
Is record for this terminal ?
No
Yes
First item ?
No
Yes
Read customer record from disc
Format and print customer line
"End-of-invoice" record ?
No
Yes
Format and print invoice total line
Read product record from disc
Format detail line
Accumulate invoice total amount
Print detail line
A

Figure 8. Segment 3 (Invoicing) Flowchart

HEWLETT hp PACKARD

HP MANUAL PART NO. 5951-7310
MICROFICHE PART NO. 5951-7311