Published: 8/14/67

Identification

Input Code Conversion

E. L. Ivie

Purpose

This section contains a detailed description of the way input code conversion is implemented in the code conversion module (CCM).

Introduction

An understanding of the following MSPM Sections will be assumed throughout this document:

BC.2.00 Introduction: Character Input/Output for Multics
BC.2.01 Character Set
BC.2.02 Interpretation of ASCII Character Streams
BC.2.03 Erase and Kill Conventions
BC.2.04 Character Escape Conventions
BF.1.05 Data Mapping
BF.10.00 An Overview of Input/Output Code Conversion

The objective of this section will be to describe in depth the way input code conversion is implemented in the CCM. Included in the discussion will be such topics as data bases, flow of control, inner procedures, and additional details on how the code conversion operations work.

Delimiters

The use of delimiters within Multics is covered briefly in Sections BC.2.00 and BF.1.01. Certain code conversion functions (e.g. canonicalization) require the use of special characters called delimiters. There are basically two such types of delimiters:

1. read delimiters
2. break delimiters

Read delimiters control the amount of data returned with each read call. For each call issued, the IOS attempts to return a certain number (nelem) of characters. If, however, a read delimiter is encountered first, then only the characters up to and including the read delimiter are returned.

Break delimiters perform two functions. First, they delimit the canonicalization and erase-kill functions. The new line(NL) also serves in this capacity. Thus, the set of characters which serve as canonicalization delimiters and erase-kill delimiters consists of all break delimiters plus the NL character.

The second function of break delimiters is to serve as hardware. interrupt characters in the GIOC.

The set of read delimiters may be empty or it may contain all 128 ASCII characters. The ASCII characters horizontal tab (HT), vertical tab (VT), and form feed (FF) are, however, always deleted by canonicalization from the input string. Therefore even if they are designated as read delimiters they would in fact not be able to function in that capacity. Also, the ASCII characters backspace (BS), red ribbon shift (RRS), black ribbon shift (BRS), half line forward (HLF), and half line reverse (HLR) are inserted by canonicalization at various points in the input string and may, therefore, be equally poor choices for read delimiters.

The set of break delimiters may also vary from none to all 128 ASCII characters. If, however, one designates one of the control characters mentioned above as a break delimiter, then that control character effectively loses its ordinary meaning. For example, if an HT is designated as a break delimiter, and thus as a canonicalization delimiter, are the spaces it generates appended to the preceding line or the following line? The convention adopted here is that control characters that have been designated break delimiters are interpreted as if they are new line characters. Thus, an HT would not generate any spaces, but would cause canonicalization to begin a new line.

The set of read and/or break delimiters can be changed by a setdelim call. Upon receiving a setdelim call the CCM stores the lists of read and break delimiters. Only the CCM need keep track of the read delimiter list, so that list is not passed on. However, the CCM needs to have the break delimiters serve as read delimiters in the next outer module. Therefore, the call is passed on by the CCM, but the read list in the forwarded call is made equal to the break list in the received call.

Since the characters in the setdelim lists received by the CCM are ASCII, they must be converted to their equivalent device codes before they are passed on.

Data Bases

PIB EXTENSION:
The CCM maintains in its ioname segment the standard per-ioname. data (pib) described in BF.2.20. Additionally the CCM maintains the following data as a pib extension:

```
dcl 1 pib1 based(p),
      (2 color,              /* red/black color status */
      2 case,                /* upper/lower case status */
      2 hsw,                 /* user-set horizontal tabs flag */
      2 vsw) bit(1),         /* user-set vertical tabs flag */
      (2 linepos,            /* carriage position on line */
      2 vpos,                /* vertical position on page */
      2 nhtabs,              /* number of horizontal tabs */
      2 nvtabs)fixed bin,    /* number of vertical tabs */
      (2 hrelp,              /* relp to horizontal tab list */
      2 vrelp) bit(18),      /* relp to vertical tab list */
      (2 erase,              /* erase character */
      2 kill) bit(9),        /* kill character */
      2 delim (0:127),
          (3 break,          /* break delimiter indicators */
          3 read)bit(1);     /* read delimiter indicators */
```

CCM INPUT DRIVING TABLE:

The CCM input data which remains constant is placed in the input part of the input-output code conversion tables(IOCCT). There is an IOCCT for each type of I/O device. Each IOCCT resides in a single segment and may be shared by multiple process groups. Section BF.10.03 describes how IOCCT's can be created and changed. The declaration of the input portion of an IOCCT is as follows:

```
dcl 1 iocct,
 2 input,
    3 char (0:255),          /* All information about the incoming
                                char. and its ASCII equivalents */
      4 lower bit(9),        /*ASCII equivalent to device char.
                    when device is in lower case */
      4 upper bit(9),        /* Same for upper.  On devices w/o
                    case shift, only lower is meaningful */
      4 bit(9),     /* 1 if this char. can start an
                    escape sequence */
      4 esc_index bit(12),   /* index into escape tree  for  this
      char.,
                    only meaningful if escape=1 */
      4 canon bit(5),        /* flags for char. requiring special
                    treatment during canonicalization,
                    e.g. backspace, newline, etc. */
    3 escape_tree (N),       /* N depends on the table */
      4 nbranches fixed bin(9),        /* number of sons of
                    this node */
      4 branchdata (nbranches),        /* each node has... */
        5 branch_char bit(9),          /* a next char. in a valid
                    escape sequence */
        5 new_char bit(9),   /* the char sp cified by the
                    completed escape sequ ce.
                    Valid if branchx is 0  /
        5 branchx bit(18);   /* index to next node in the escape
```

sequence if branchchar was input */

A description of how the escape tree is used will be given in the CCESCAPE section.

Flow of Control

The operation of the CCM upon receiving a read call is shown in Figure 1. Each of the six steps in Fig. 1 are described below:

STEP 1:
The CCM first checks in its buffer to see if the converted data there will satisfy the read request. There will be data in the CCM buffer if there has been unavoidable read-ahead. Unavoidable read-ahead may occur, for example, when several read delimiters occur on a single line. The CCM picks up the whole line and processes it. The string up to the first read delimiter is then delivered to the calling procedure and the rest of the line is stored in the CCM buffer. See Section BF.1.04 for a further description of unaviodable read-ahead.

STEP 2:
If there is insufficient data in the buffer to meet the request the CCM issues a read call to the DSM or next outer module. The nelem argument in this call is set to some large value (4096) so that in general the return will be triggered by the occurrence of a break delimiter and not by character count.

STEP 3:
Once control is returned from the DSM the CCM checks the status to see if the call was completed. The call may be incomplete if some error condition was located further down the iopath. It may also be incomplete because the workspace synchronization mode is asynchronous and the DSM returned control after physical or logical initiation. In either event the CCM returns control to the calling procedure.

STEP 4:
If, however, the read call issued by the CCM was completed then the CCM checks the returned string for one or more canonicalization delimiters (CD's). If there are no CD's then the CCM returns to STEP.2 to perform another read.

There are at least two reasons why there may not be a CD in the returned string. It may be because the nelem argument was too small and the request returned before a break delimiter was encountered. Or it may be because the break delimiter which evoked the return is in reality a hidden character (See BC.2.04) and therefore does not qualify as a valid CD.

STEP 5:
After one or more (valid) CD's have been found the actual code conversion operations are performed on the string. (Actually only those portions of the string which precede CD's are converted. The characters between the last CD and the end of the string are saved for conversion later.)

The code conversion operations performed on the string are described in the next section of this document. The final, processed string is added to the data in the CCM buffer and control is returned to STEP 1.

STEP 6:
Finally, when a read delimiter is encountered in STEP 1 or when enough (nelem) characters have been collected in the CCM buffer to satisfy the original read call the data is moved to the designated workspace and control is returned to the calling procedure.

Code Conversion Operations

There are six basic operations involved in input code conversion (STEP 5):

1. Device-to-ASCII conversion
2. Elimination of hidden escape sequences
3. Canonicalization
4. Erase and kill processing
5. Escape processing
6. Recanonicalization

These operations have been described in general terms in BF.1.05 and BF.10.00. A much more detailed picture of how they function is presented in subsequent parts of this document.

The names of the procedures which perform these operations are as follows:

cccanon	operations 1,2,3
ccerase	operation 4
ccescape	operation 5
ccrecanon	operation 6

A description of the procedures together with the operations which they perform follows.

CCCANON

The procedure cccanon performs device-to-ASC I code conversion, deletes hidden sequences, and performs print sition alignment. It is invoked by the call,

```
call cccanon (inptr, size, outptr, pibptr);

dcl (inptr, outptr, pibptr) ptr, size fixed bin;
```

The argument inptr points to the input character string of 9-bit characters. The string is size characters long. The output from the procedure is placed in the area pointed to by outptr. It is a one-word-per-character list where each word has the following declaration:

```
dcl 1 word (size) based(p),
(2 hpos,          /* horizontal print position */
2 vpos,           /* vertical character position */
2 char)bit(9),    /* ASCII character code */
2color bit(1),    /*color code ("1"=red) */
2 pad bit(8);     /* padding */
```

The argument pibptr is the same as the last argument for outer calls. It points to the per-ioname data base (pib) in the CCM's ioname segment.

The three conversion operations performed by cccanon will now be described.

DEVICE-TO-ASCII:
Each character from the device is converted to its equivalent ASCII representation. This is a simple one-to-one transformation with one exception. Certain devices (like the IBM 1050) generate special case shifting characters. For these devices cccanon notes and deletes the case shift characters. It then selects the correct ASCII upper case or lower case character for each device character depending on the current case.

Note that device codes for which no ASCII equivalent has been defined (e.g. characters with bits 8 and/or 9 a "1") are left in the string unconverted by cccanon.

The lower and upper entries in the structure of the CCM input driving table (CCIDT) provide the information for the device-to-ASCII transformation. Only the lower field is used for those devices which do not have case shifting characters. (See Data Bases section of this document.)

HIDDEN CHARACTER ELIMINATION:
Hidden character sequences allow a user to type a character on his typewriter which has only a local effect (See Section BC.2.04). For example, one might wish to return the carriage to the left margin without ending the 'logical' line he is currently typing in. To do this he would type an escape sequence, a "c", and a new line. The new line would return the typewriter carriage to the left margin. However, all three characters would be eliminated from the input string by cccanon because they form a hidden character sequence.

There are two consequences which one should be aware of in using hidden character sequences. First, since hidden characters are removed before print position alignment, the relative position of characters may be destroyed internally. For example, if one backspaces over a hidden character sequence, overstrikes on the printed page will be three positions out of register with the CCM's internal representation.

Second, since erase and kill characters can be hidden, one cannot erase a "c" typed after an escape character. The sequence, escape character, "c", and erase character would be eliminated by cccanon and not by ccerase.

CANONICALIZATION:
As was noted above the output of cccanon associates with each character typed a horizontal print position, a vertical character position, and a color type. As cccanon scans the input string it keeps a count of the current horizontal and vertical position and the color status. This is done by taking note of the ASCII control characters backspace (BS), vertical tab (VT), horizontal tab (HT), form feed (FF), half line forward (HLF), half line reverse (HLR), red ribbon shift (RRS), black ribbon shift(BRS), and new line (NL). These control characters change the appropriate count or status (e.g. BS reduces the horizontal count by 1). They are not, however, placed in the output list (unless, of course, they are functioning as a canonicalization delimiter.) The output is then sorted numerically so that the characters will have a canonical order.

CCERASE
The basic rules for erase and kill processing are found in Section BC.2.03. The procedure ccerase implements these rules. It is invoked by the following call:

```
call ccerase(inptr,size,pibptr);
```

The argument inptr points to the one-word-per-character list generated by cccanon. This list is size words long. The argument pibptr points to the CCM's per-ioname data base.

The output of ccerase is in. the same form as the input and overlays the input. The argument size is set on return to the number of words in the output list.

Those escape sequences which change the erase or kill character are also processed and deleted from the string by ccerase. Note that there are always three characters in such sequences, and that these three characters must occupy contigious horizontal print positions, they must be on the same vertical line, and there must be no other characters in the three horizontal positions.

Not all characters can serve as erase or kill characters. Since the definition of erase and kill implies that the erase or kill character occupy a print position the control characters BS, HT,

NL, VT, FF, RRS, BRS, HLF, and HLR can not be used. Also the three characters: escape character, "E", and "K" must be excluded in order to retain the ability to make further changes. In addition, read and break delimiters are excluded on the grounds that a given character can't serve in both capacities meaningfully.

If ccerase encounters an escape sequence which attempts to make some excluded character an erase or kill character, then an error bit is set in the status. The procedure ccerase continues to process the string, however, and the offending sequence is left in the string unaltered.

The procedure ccerase adjusts the horizontal print position (word.hpos) of suceeding characters to reflect deletions due to preceding erases, kills, and processed escape sequences. Note that erase and kill do not affect the vertical character position (word.vpos) counts or the color types (word.color).

The pib extension in the CCM ioname segment is used to store the information needed by ccerase. For example, the erase and kill substructure entries contain the ASCII codes of the current erase and kill characters.

CCESCAPE

The basic rules for escape processing are found in Section BC.2.04. The procedure ccescape implements these rules. The escape tree in the CCM input driving table (CCIDT) designates the valid escape sequences for the device being serviced. The procedure ccescape is called by the same arguments as ccerase.

A diagram of an escape tree for the IBM 1050 is shown in Fig. 3. All characters that are the start of some valid escape. sequence are so marked by the escape bit in the ccmin table. In the case of the 1050 the character, "¢", is the only character currently defined. For these 'escape starting' characters there is a relative pointer to a list of valid second characters. In the 1050 example there are seven valid second characters. (Note that octal escape sequences are treated as a special case and are not part of the escape tree.) Six of these are terminal (i.e. the escape sequences are completed). For terminal branches the charout field contains the character represented by the escape sequence. One character, "¢", leads to a second node. This branch represents the escape sequences, "¢¢(", and "¢¢)".

The escape tree allows for overstruck escape sequences. In such cases one of the charin characters in the path will be the backspace character. Since cccanon has already ordered the string only one path is needed in the tree for each distinct sequence.

The extra horizontal print positions occu ied by an escape sequence are eliminated by ccescape. Thus, fc r the IBM 1050 the four-position sequence, "a,b, would reduce to a three-position

sequence consisting of "a", "less than", and a "b". A problem rises if the horizontal print position that would normally be elliminated contains other (non-escape) characters. In such cases the position is not deleted. An example is a string with an escaped superscript and a multi-character subscript which occupy the same horizontal positions.

CCRECANON

The character string generated by ccescape may no longer be in the order specified by Section BC.2.01. The procedure ccrecanon restores the string to its canonicalized order and reformats it into four 9-bit characters per word. In the process the following control characters and inserted as necessary: (BS), (HLF), (HLR), (RRS), (BRS), and (SPACE).

The procedure ccrecanon is called with the same set of arguments as cccanon. However, inptr now points to the one-word-per-character list while outptr points to a 9-bit character string.

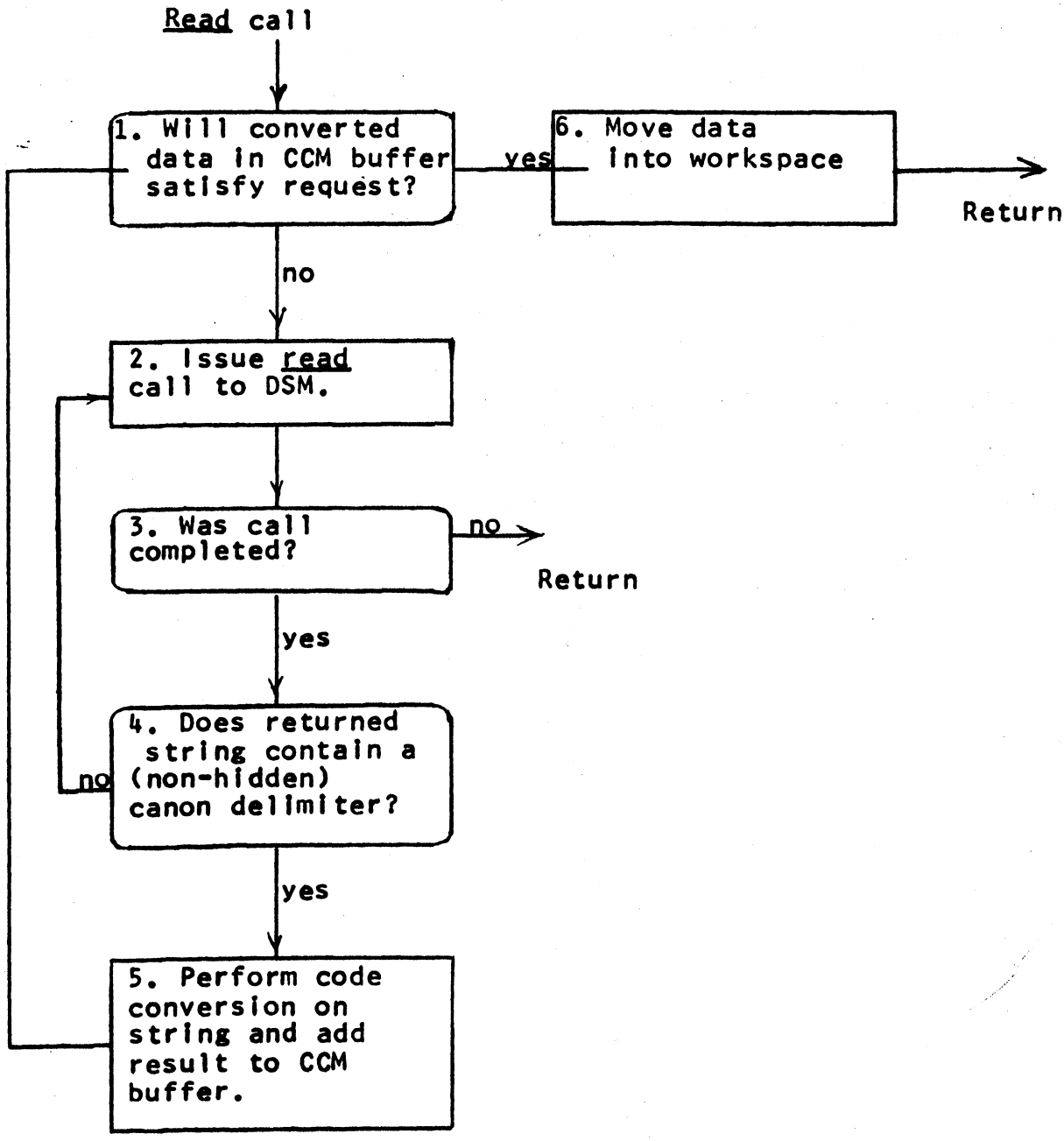

Figure 1: Flow of control in CCM upon receiving a read call.

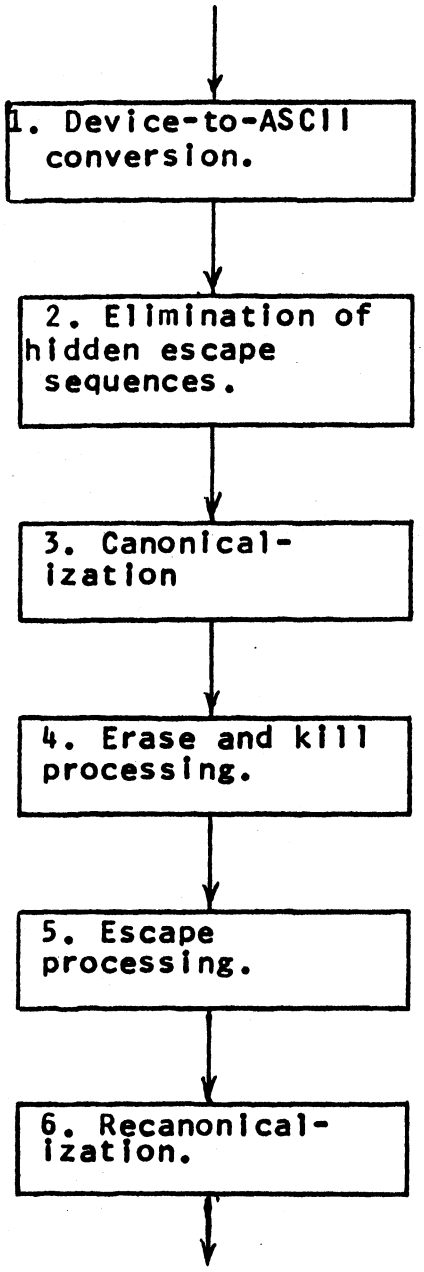

Figure 2: Code Conversion Operations (Step 5 of Fig. 1).

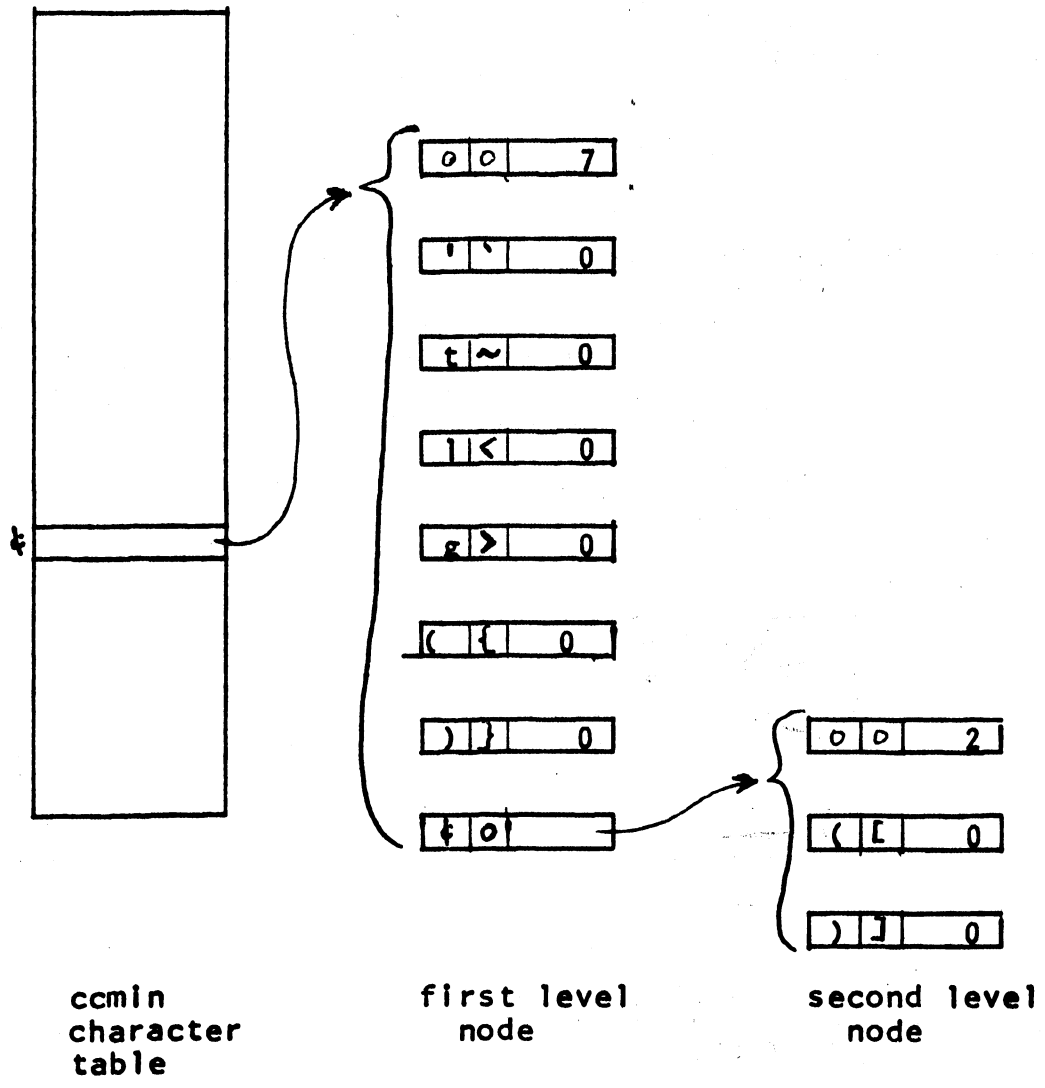

Figure 3: Diagram of Escape Tree for IBM 1050.