C. Barrera
A. V. Strietzel

Electrophotographic Printer Control as Embodied in the IBM 3800 Printing Subsystem Models 3 and 8

Control of the IBM 3800 Models 3 and 8 electrophotographic printers is achieved by use of a fundamentally different control system than was used in their predecessors, the Models 1 and 2. As a result, printing of composed pages or electronic overlays can include text of many different font sizes and styles printed in multiple orientations, as well as raster images up to a full page in size. The printers manage stored resources, including fonts, segments of pages, and electronic overlays. Pages are composed inside the printers in a logical sequence, instead of by the more traditional line-by-line sequence. This, as well as the capability to position text and images at any addressable point, enhances usability. A high-speed, table-driven character generator, a new command set, and a microcoded control unit make all of this possible.

Introduction

The IBM 3800 Printing Subsystem Model 3, as indicated in the introductory paper [1] for this series of papers on the 3800, is an extension of the Model 1; it provides expanded and fundamentally different methods of composing pages. All-point addressability removes most restrictions on the placement of characters and other figures, while the elimination of any requirement to order print data in the conventional top-to-bottom sequence simplifies the task of page-formatting software. The Model 8 has the added capability to print using very large character sets such as those required for kanji or Chinese alphabets.

Features include the storing and merging of electronic forms, printing of raster images, and use of a large number of fonts. Powerful text-formatting commands simplify printing and allow the composition of complex pages that could not be printed by a line printer. Most of the new functions are implemented in microcode, which controls the entire machine and creates the printer's visible host-processor relationship. In addition to the microcode, the 240-pel- (picture element) per-inch density and the positioning of print objects with one-pel resolution required the design of faster, more complex hardware to retain the high speed of the 3800 Models 1 and 2 (except when indicated, hereafter designated as Model 1).

The paper path, most servo systems, and the fusing system of the Models 3 and 8 (hereafter designated as Model 3 except when indicated) are essentially identical to those of the Model 1. These were described in previous papers [2–6].

All-points-addressable printing

• *Primary printing functions*

The 3800 Model 3 operates in two modes: advanced function mode (page mode) and compatibility mode. In compatibility mode, commands are processed as if the printer were a Model 1; in page mode, the Model 3 accepts a powerful command set that supports fully composed pages. The command set is fundamentally different from that used to drive line printers. Compatibility mode functions are essentially identical to those of the Model 1 and have been described in an earlier paper [2].

The printer handles pages in two stages: The page description is received from the host-processor interface (System/370 channel) as a series of commands. These commands are processed to create control tables that will drive a high-speed character generator (see the later section "Processing functions"). This "page build" process is synchronized with the host processor; each command is completely processed before another is accepted. Once the tables are built, the page is buffered; that is, it is enqueued and waits for the second stage, which is the printing process in which printed pages are produced. Printing must be synchronized with the physical

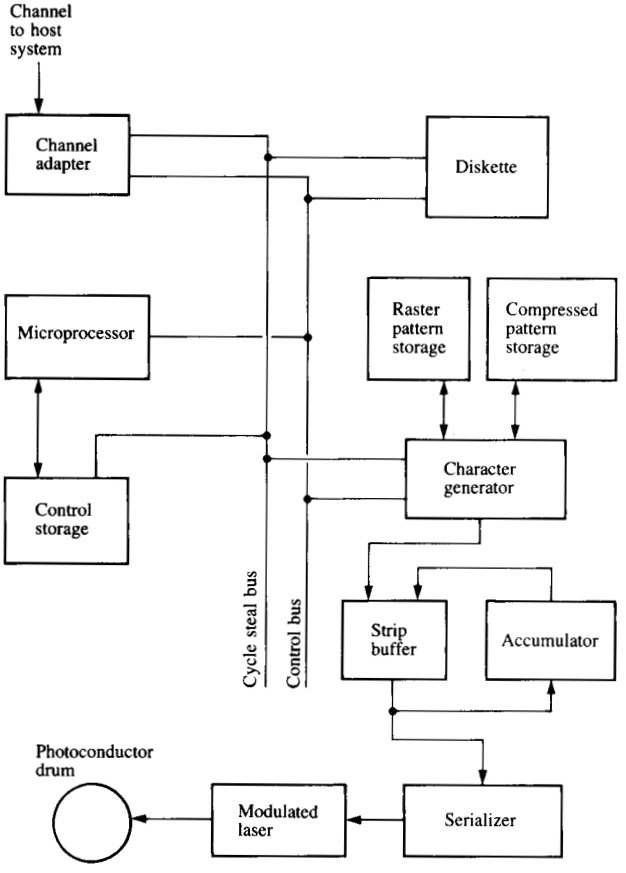

Figure 1 Major components of the 3800 Models 3 and 8.

positions of the photoconductor drum and the paper line. It is asynchronous with host-processor command processing. Buffering of pages allows considerable process overlap; the varying processing time for pages of differing complexity need not affect the constant print rate.

Fonts

A font in the Model 3 context is a set of characters of a given style and size. A font includes descriptive attributes, as well as the raster patterns that define the character shapes. Individual characters of a font are selected for printing by specifying a one- or two-byte character code. A one-byte code set, such as ASCII or EBCDIC, specifies up to 256 such codes (usually a number of codes are reserved for control functions). An ideographic alphabet, such as kanji, may require over 7000 characters for useful printing. For this type of alphabet, a two-byte code is used to specify each character to be printed.

The Model 3 accepts only one-byte fonts while the Model 8 accepts both one-byte and two-byte fonts. The designation is based on the length of the code used to specify characters. The Models 3 and 8 can store up to 64 complete fonts; the actual number stored is usually less because of space limitations. In the case of two-byte fonts such as kanji, each font may consist of up to 63 sections, each of which can contain as many as 190 characters. The character patterns for one-byte fonts are stored in a semiconductor storage, designated as the raster pattern storage, or, alternatively, on a floppy diskette (see **Figure 1**). Two-byte character patterns are kept in compressed pattern storage, a designation which reflects the usual practice of compressing these character patterns. The host processor loads fonts or font sections by name. The printer microcode allocates and deallocates space as required and performs "housecleaning" functions, if needed, to recover fragmented storage space. An attempt to load more font data than will fit in the available storage causes the printer to notify the host processor of an exception condition.

Although the Model 3 can store and manage tens of fonts—enough to print most pages—it will normally print many kinds of documents requiring hundreds of different fonts. The printer's capability to store fonts by name and delete them singly or in groups supports the host processor's overall font-management requirements.

Page composition

One of the primary functions of the printer microcode is to prepare a complete page, by converting it to internal control table format, before printing is started. Such pages are stored until printed, after which the control tables for the page are discarded. The commands that describe a page need not be ordered relative to the position of data on the page; that is, lines of text need not be received in top-to-bottom sequence. Indeed, data need not be received as text lines at all. This allows text to be oriented vertically on the page as well as horizontally, and facilitates printing text in columns. The printer microcode reorders individual print objects (characters and image rectangles) top to bottom, as required by the character generator, by constructing ordered lists.

It was decided that the page-composition functions of the Model 3 should be general and simple. For example, there is no command that directly produces underscored characters. Instead, a general capability for drawing horizontal or vertical lines at positions specified in an orthogonal coordinate system allows characters or words to be underscored. This kind of generality requires a certain amount of extra processing by the host-processor software, but it greatly simplifies the printer commands. These commands are therefore easier to understand, more versatile, and less subject to change than they would be if they provided a large number of specialized functions.

Raster images

A raster image is a collection of pels, each described by a single bit, that appears on the page as a rectangular array of dots. Images accepted by the Model 3 can be thought of as a

series of scan lines one dot wide and long enough to fill the image rectangle. Raster images are intended to be used for objects that are not easily described otherwise. Company logos, signatures, and drawings are examples. Although a resultant picture or drawing may be arbitrarily complex, the picture elements or dots of which it is made can always be described by a rectangular array of bits. Nevertheless, careful design is required to minimize the processing and movement of raster-image data because there is usually a large quantity of such data. For example, an image to fill a standard page might be 19.05×25.4 cm (7.5×10 in.) in size, a rectangle that requires 540 000 bytes of image data. When received, the data are moved to raster pattern storage. From there, the character generator retrieves them in approximately the same way it retrieves character patterns. Images larger than about 464 cm^2 (72 $in.^2$) do not fit in the raster pattern storage. A special process called accumulation is used to handle such images. This process is described later under "Processing functions."

The large quantity of data usually associated with raster images would require very fast or highly parallel processing for simple modifications such as orthogonal rotation. Transformations such as nonorthogonal rotations or nonintegral size changes require complex algorithms to effectively process the quantized data. Inclusion of such a capability would have meant a substantial cost increase while significantly enhancing the processing of only a fraction of the pages expected to be printed. As a result, image data modifications in the Model 3 are limited to a single two-to-one size transformation and cell replication. Repetitive pel patterns are sometimes useful; an example is a shading pattern that might be used on a form. The Model 3 provides an efficient way to handle repetitive patterns through cell replication. The printer can receive an image cell, described as the source image, and expand it by replication to fill a target rectangle, which is the pattern that appears on the printed page.

Page segments

One class of object stored by the Model 3 is called a page segment. This is a collection of text or image data that can be added to a subsequent page. Many different documents can be created merely by including page segments in different combinations. The result is decreased host-processor time and less data transmitted over the host-processor interface. Page segments assume the environment (such as font equivalence and page boundaries) of the underlying page. Thus, a page might be printing using Font A; a segment included on that page would also print using Font A. Exactly the same segment would print in Font B if it were included on a page at a point where Font B is active. The capability of a segment to take on the characteristics of the page it appears on enhances its usefulness. However, because the segment shares the underlying page environment, any changes it makes to the environment potentially affect the rest of the page.

Electronic forms

Much computer output is produced on preprinted forms. If a printer could reproduce forms on blank paper, the costs associated with preprinting, stocking multiple forms, and requiring an operator to load different forms in the printer could be eliminated. The Model 3 provides such a facility through the use of overlays.

An overlay is a stored object containing text or image data. The host processor can transmit an overlay to the printer, giving it a name. Subsequently, the variable data to be included on the form can be transmitted along with a command specifying the inclusion of the stored overlay. When including overlays, the microcode takes advantage of its capability for handling print data in any sequence. Thus, the variable part of the page is built first, then the overlay data are processed. With the printer locating both the overlay and variable data, alignment between them is perfect; it does not depend on machine tolerances, precision preprinting, or careful adjustment by an operator.

Overlays are stored by name, simplifying host-processor resource management. They are stored in coded form, that is, exactly as they are received from the host processor. As a result, the coded data are processed once each time the overlay is included on a page. Although this seems inefficient, it requires no more capability than the printer already uses to produce a complex page without an overlay, and this usually can be done with no speed degradation. Thus, it was decided to conserve space by storing the coded form of overlays, and to exploit the printer's speed to preserve performance. When an overlay is stored, the existing environment is stored with it. This permits independent specification of variable data and overlay characteristics, simplifying host processing. If the host processor specifies a "raster overlay," the printer processes the data upon receipt and stores them in a full-page buffer called an "accumulator" (see the later section, "Processing functions"). The raster overlay can then be merged with any page as the page is printed. Because all processing occurs at loading time, the overlay can be arbitrarily complex while affecting performance only at loading time, with no effect on subsequent pages.

Preprinted forms can also be used. This can provide very-high-quality backgrounds or logos not possible with the 240 pels/in. density of the printer, and the preprinting can be in color. Simple operator adjustments are available for aligning variable data with the preprinted data.

Page copy modifications

One or more copies of a page can be modified in several ways by the Model 3. On a single copy, up to eight overlays can be added, up to eight suppressible text areas can be added, the forms flash can be applied, and edge marks used to identify

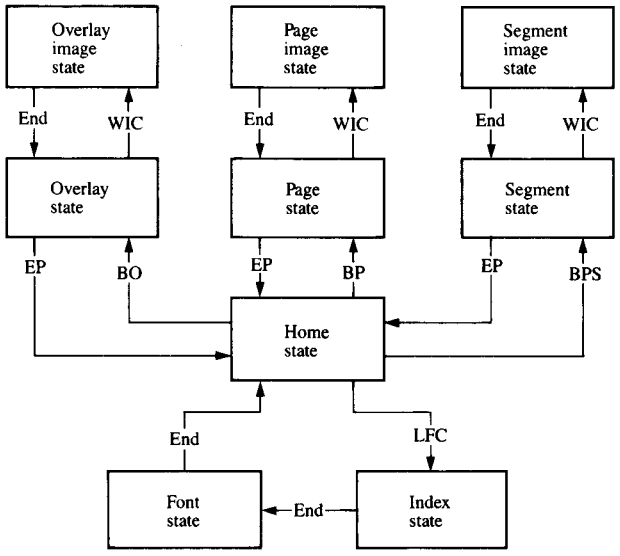

Figure 2 Operational states of the printer in page mode.

job boundaries can be added. These functions are independent of the basic page definition, so the exact number of copies desired can be produced without modification to the page-generation software.

Objects stored by the printer, such as overlays, segments, fonts, and copy control information, may remain stored while many pages are being printed. Thus, the printer maintains a long-term environment. While this provides powerful and efficient printing functions, it also causes certain new recovery difficulties that were not encountered in earlier printers. Recovery from a situation where a stored object is needed but not available is complicated by the time lag between the loading of objects and their subsequent use. Instead of the host processor simply restarting at the beginning of a page, it may be necessary to back up many pages or even several documents to find the point at which objects were loaded. To minimize such effects, the printer validates objects as they are received to reduce the probability of their causing an error later. Validation generally ensures that command formats and command parameters are within specified limits.

A decision was made to optimize the amount of checking used. Too little checking permits errors whose symptoms are hard to associate with their cause due to the time lag; too much checking adversely affects printer performance. To further ensure that objects are available when needed, the printer detects changes in the environment. In the rare instance of power loss or a transient microprocessor error, the printer notifies the host processor that objects must be reloaded.

The environment can also be altered by correct but unusual conditions. A complex page that requires very large amounts of storage might be printable only if it is internally merged with an overlay, thus making the overlay unusable for subsequent pages. Rather than rejecting the page, the printer prints it and requires that the host processor reload the overlay. Where there was a design choice between altering the environment and slower-than-normal processing, the latter was selected. As a result, the Model 3 can print almost any page at some speed, and the need for host-processor recovery is minimized.

Typical command sequences and functions

• *Machine states*

When operating in advanced function mode, the Model 3 maintains internal machine states that define the effects of host-processor commands (see **Figure 2** and **Table 1**). For example, page defaults cannot be set in page state, i.e., after a page has been started. The machine state thus not only defines command effects, but is the basis for command-sequence checking. By defining the various states, the commands that are valid in each state, and the exact ways in which commands cause state changes, the state diagram becomes an effective way to describe the machine.

• *Command design*

Command parameters were made general wherever possible to support future expansion or even future printers. For example, a two-bit parameter would have been sufficient to specify the three allowed text orientations. Instead, the orientation is specified as two two-byte fields, each of which specifies an angle in degrees and fractions. The first parameter determines the direction in which to move when characters are added to a text line, the second, the direction in which to move when text lines are added. Such generality costs very little and allows great flexibility for the future.

The cost is not always negligible, however. Processing of very general commands takes time and may affect performance. If instead of a few parameters the data processed turn out to be extensive, the difference between generality and device dependency can be critical. Such was the case with character patterns. A general, easily transformed format would have saved space and been versatile. But speed requirements resulted in a format that could be loaded and used without modification; the format reflects a number of internal characteristics of the character generator.

• *Font handling*

Character patterns of a font stored on the floppy diskette cannot be accessed fast enough to support the character generator. Such a font is called "stageable" because it is loaded in stages, first to the diskette, then to the raster pattern storage. While the control tables for a page are being generated, a font needed by a page is automatically transferred from the diskette

Table 1 Relationship between commands and internal states.

Commands causing state changes:									
Write Image Control (WIC)									
Load Font Control (LFC)									
Begin Page (BP)									
Begin Overlay (BO)									
Begin Page Segment (BPS)									
End Page (EP)									
End									
Other commands (valid states are indicated):									
	Home	*Page*	*Overlay*	*Segment*	*Page image*	*Overlay image*	*Segment image*	*Index*	*Font*
Load Page Description	X								
Load Page Position	X								
Load Copy Control	X								
Execute Order Home State	X								
Delete Overlay	X								
Delete Page Segment	X								
Load Font Equivalence	X								
Delete Font	X								
Load Font Index								X	
Load Font									X
Write Text		X	X	X					
Write Factored Text Control		X	X	X					
Include Page Segment		X	X						
Write Image					X	X	X		

to the raster pattern storage. This scheme allows a large number of fonts to be stored in the printer while conserving monolithic storage space. The host processor does not need to, and indeed cannot, manage the staging process.

In order to load fonts from the host processor, two kinds of control information are transmitted, followed by the character raster patterns. The Load Font Control command transmits attributes common to all characters of the font, such as the font identifier and the maximum or uniform character-box dimensions.

The Load Font Index command transmits attributes that differ for each character. More fields are significant for proportionally spaced characters than for fixed-spaced characters. Two indexes can be transmitted if a set of characters is to be printed in two orientations.

The Load Font command transmits character raster patterns. The design of the character generator hardware (see "Processing functions") places several restrictions on the format of the raster patterns. First, the set of raster patterns for a font can occupy up to 510K bytes, while a two-byte number, chosen to minimize control-table size, addresses the patterns. Hence, each character pattern was constrained to start on an eight-byte boundary.

The raster patterns are kept in the raster pattern storage, which is dynamically allocated in blocks of 2048 bytes. To simplify storage management and avoid periodic clean-up of fragmented storage, no requirement was made that all the characters of a font should reside in contiguous raster pattern storage blocks. As a result, the raster pattern of a character must not span a 2048-byte boundary; otherwise, the pattern would sometimes be loaded into two noncontiguous blocks, and the hardware that accesses it would have to be inordinately complex. Another consequence of the 2048-byte blocking is the limitation of character-box size to 128 by 128 pels, or bits. The largest character occupies exactly 2048 bytes. Assuming noncontiguous blocks, processing larger characters would have required either larger blocks or complex addressing schemes and expanded tables. Neither alternative was believed to be justified.

Controls embedded in text data select the font to be used for subsequent characters. Selection is done by font number, which is distinct from the font ID, which identifies each font stored in the printer. A table loaded by the Load Font Equivalence command equates font numbers with font IDs. This permits any of the stored fonts to be called by an arbitrary font number. The font in which text prints can thus be changed by altering the font-equivalence table; the text data, which are frequently extensive, need not be reprocessed. Soft-

ware that generates print data can be independent of font-naming conventions, and can instead select fonts simply as 1, 2, 3, $\cdots$; external font management then associates actual fonts with these font numbers. The result is the capability of the printer to handle unaltered print data from different sources, simplified page development, and independence of text-generating software.

The Delete Font command allows the host processor to manage fonts loaded into the printer. Because the number of fonts available at an installation is usually far greater than the number that can be stored in the printer, such management should be facilitated by printer design. With the Delete Font command, the host processor can select a single font, all nonstageable fonts (those that the printer is not allowed to remove from the raster pattern storage), or all fonts except the few that permanently reside on the printer diskette. In the case of two-byte fonts, the deletion may be specified for certain sections. Fonts required by a stored object (overlay or page segment) are locked; that is, they cannot be deleted until the associated overlay or segment is deleted. This ensures that the stored object can be processed at the time it is called.

- *Page definition*

A page is delimited by the pair of commands Begin Page and End Page. Between these two commands, any number of page-composition commands define the print data. Before the print data can be processed, page boundaries, various defaults, and the number of copies, possibly modified, must be established. This is done with the Load Page Description, Load Page Position, and Load Copy Control commands, which must precede the Begin Page command for the applicable page. (These commands are described more fully later under "Processing functions.") The most obvious page-composition command is Write Text, which transmits both text data to be printed and embedded controls. The text data can be coded as EBCDIC, ASCII, two-byte kanji, or any other code.

Embedded controls, identified by an escape code, perform formatting functions. General controls move the current print position by specifying absolute or relative x, y values. These can be used to indent a paragraph, position subscripts or superscripts, and in general to position print data on the page. One control selects the font to be used for the following text; another selects the orientation of lines of text. Some text controls perform more specialized functions. The left margin and line spacing can be set; then a next-line control moves the print position down one line and to the left margin. A parameter called character increment inserts spaces between the characters of a word, facilitating line justification. Other controls draw vertical or horizontal lines of specified thickness. The embedded controls are more efficient than separate printer commands would be because they avoid overhead in the host processor, the host channel, and the microcode.

The Include Page Segment command specifies a previously stored segment to be included on the page at the current print position. Raster images are placed on a page by a sequence of image commands (see "Raster images"). These commands can be interleaved with Write Text commands to produce a page in the desired format.

- *Raster images*

An image is loaded in the printer by a sequence of image commands. The Write Image Control command specifies the dimensions of the source and target rectangles, the location of the image relative to the current print position or other coordinate systems, and selection of double-dot processing. One or more Write Image commands must follow the Write Image Control command. These transmit the raster data as a series of uncompressed scan lines. The End command delimits the end of the raster data. As with the font-load sequence, the End command facilitates checking and consistency but is not otherwise necessary.

- *Page segments and overlays*

Page segments and overlays are similar objects stored by the printer. Each is delimited by begin and end commands (Begin Page Segment in one case, Begin Overlay in the other). Except for the begin command, the command sequence for defining a segment or overlay is the same as that for defining an ordinary page. (The Include Page Segment command is not allowed inside a page segment.) The Begin Page Segment and Begin Overlay commands specify the name that may be used later to identify the segment or overlay to be included.

Object definitions are terminated by specific End or End Page commands instead of by any of the set of commands which by inference must signal the beginning of a different object. Thus page build is terminated and the page printed immediately by the End Page command, rather than by waiting for the beginning of a subsequent page or other object. Hence, the question of what to do with the last page of the day never arises, and errors are easier to associate with the correct object.

- *Page buffering*

If the host processor transmits data faster than pages can be printed, printer resources eventually become saturated, and no more pages can be stored. Limiting resources are the microprocessor storage, which contains font-control tables, stored objects such as page segments and overlays, and the tables belonging to buffered pages; raster pattern storage, which contains one-byte font patterns and raster image data; and compressed pattern storage. If one or more pages are buffered in the printer, resources are freed as control tables, fonts, or images are discarded following printing of the pages. Hence, if needed resources are unavailable, the microcode suspends processing of the current command until the re-

quired resources have been deallocated. It may take many seconds or even minutes to free sufficient resources; the device-end signal to the host-processor channel can be delayed by that amount of time.

Processing functions

• *Page-build environment*

A resource environment is created inside the printer before a logical page for printing is defined. The channel commands discussed above are used individually or in sequences to establish a resource pool of overlays, page segments, and fonts.

A sequence of commands starting with a Begin Overlay (BO) command defines a stored electronic form or page overlay, as shown in Fig. 2. The BO command sends a one-byte overlay identifier, which is entered into the overlay name table. Storage space is reserved for the overlay and pointed to by the entry in the name table. At this point, the page-build environment is collected and saved in the overlay storage area. The font-equivalence table and the page position and page description parameters, which together comprise the page description block (PDB), are saved for later use during overlay inclusion.

Following the BO command is a sequence of commands defining the text and image data that make up the overlay. As text commands are received, they are added to the overlay storage area, preceded by a header with the command code and data length. The text data, which contain embedded control information, are stored without modification and later retrieved for processing at overlay-inclusion time. In general, conversion of text-command data to character table represents an expansion in data size by as much as 8:1. It is, therefore, more storage-efficient to store the text as received rather than to convert it to character table.

When image data are sent as part of an overlay, it is impractical to store the raster patterns in control storage until overlay-inclusion time, because of the typically large size of images. The raster data instead are moved directly to RPS, and an intermediate RPS block representation of the image is created and stored in the overlay storage area. This RPS block form defines the shape of the image and contains pointers to it in RPS. This form is far smaller than the raster pattern, and for most images smaller than the character table representation required for the character generator hardware at page print time. At overlay-inclusion time, the RPS block form is translated into character table form, which contains the information for the placement of the image on the physical page.

The Begin Page Segment (BPS) command, which initiates segment loading, sends a one-byte identifier which is entered into the segment name table in printer control storage. A page segment is similar to an overlay and is stored in much the same format. For a page segment, however, the page-composition information and font equivalence table (FET) will be those of the page on which the segment is included. Therefore, the FET and the PDB are not stored with the command data for page segments. As for overlays, text data are stored as received, and image data are moved to RPS and represented in segment storage in the RPS block form.

Text suppressions are sections of the text data on a logical page that may be omitted or suppressed from one copy of the page to the next. Suppressible data are flagged with an embedded control inside the base-page text data. When a suppression is recognized, an entry is made in the suppression name table, and space is reserved in a suppression storage buffer in printer control storage. Because it is easier to add the suppressible data to a page that wants them than to delete them from a page that does not, the suppression is stored in the same format as a page segment. At the time an overlay is included on each copy of the page to be printed, those suppressions not called out are restored to the page, following the same rules as page segments. Since suppressions are actually sections of a page being built, when the internal table representation of the page is complete the suppression storage is freed and the suppression name table is erased. This is performed at the end of the End Page command. Segments and overlays are resources loaded in the host processor that endure from page to page until explicitly deleted by the host processor.

• *Base-page build*

It is often desired to produce several copies of a page that differ from one another. This copy-modification function is achieved with the Load Copy Control (LCC) command, which causes the copy control block (CCB) to be built in control storage. A page is defined for printing by a sequence of commands starting with a Begin Page command and finishing with an End Page command. These commands define the base page. The CCB specifies the modifications to be made to the base page, overlays to be included, and suppressions to exclude. The CCB may specify up to 127 different copy groups, each with its own copy count and list of overlays and suppressions.

Figure 3 shows how overlays can be used to generate the page headings and various column headings for a hypothetical packing slip and bill of material. On the packing slip, the prices, which were loaded as suppressible text, have been suppressed. Thus, multiple overlays perform the functions of a multiple-part form, and suppression capability functions as spot carbons. When a Begin Page (BP) command is received to start a base-page definition, the binding of the logical page to the physical page is performed, resulting in the definition of the printable area. The length and width of the logical page are specified by the Load Page Description (LPD) command,

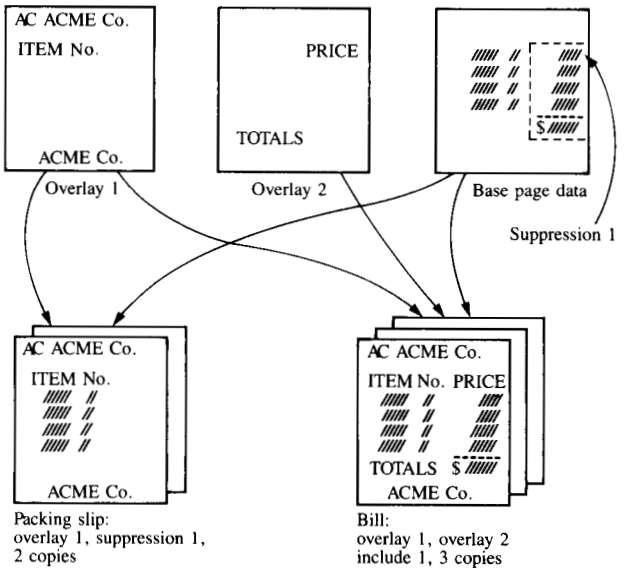

Figure 3 Examples of overlay and suppressed text usage. Overlays, suppressions, and copy counts are specified by the Load Copy Control command.

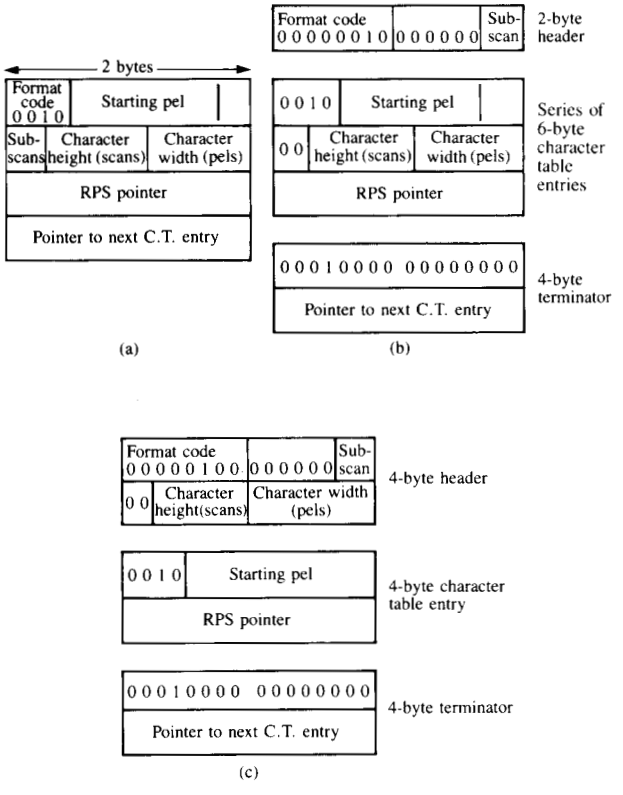

Figure 4 Internal character table formats: (a) 8-byte, most general character table entry; (b) 6-byte character table format—all characters on same scan line; (c) 4-byte character table format—all characters same size and on same scan line.

and its placement on the physical page is specified by the Load Page Position (LPP) command. At Begin Page time, now that the size of the physical page is known (it is the size of the paper loaded in the printer), the intersection of the logical and physical pages is determined: this is the printable area. The printable-area dimensions are recorded in the page description block (PDB) and used during page build to ensure that text and image data remain inside the printable boundary. Reception of the BP command places the printer in page state. The ensuing Write Text, Write Image, and Include Page Segment commands cause the internal table representation of the base page to be built. Inside the printer a page is represented by a page control block (PCB), which contains pointers to two tables, the scan table and the character table.

The character table contains an entry for every character or section of image on the page. It is arranged as a linked list of entries, representing characters starting on the same group of four contiguous scan lines. There are actually ten different formats of character table entries, all containing similar information. One format **[Figure 4(a)]** represents the most general placement of characters on the page and requires eight bytes of character table for each entry. For one text orientation, that for which the direction of successive characters is along a scan line, it is possible to separate some of this information into a table header and terminator. Because these characters are received in order and all start on the same scan, they can be represented by a two-byte header, a series of six-byte entries, and a four-byte terminator **[Figure 4(b)]**. When characters are all of the same size, the size information can be moved to the header, providing a further reduction in character table size **[Figure 4(c)]**. While it is possible to have fonts with characters up to 64 scans high, the character generator cannot process characters over 64 scans in height. When a taller character is encountered, a second character table entry is created and linked into the list for the appropriate scan line. Character table size is important, because it determines the number of pages that can be buffered awaiting print, which determines printer throughput.

The scan table comprises 1024 two-byte entries, each representing a group of four scan lines on the printed page. The scan table is actually a sorted list of character table list pointers. If a character table list represents characters starting on scan line 242, scan table entry 60 points to that list. The fact that the characters start on scan 242 and not 240 is coded in the subscan field in the character table entries (**Figure 5**).

The character table lists are not sorted by the x-position of the characters. To do this would require a microcode search of the linked list to find the correct placement of an entry, greatly slowing down the table build process. Instead, the character generator is designed to place characters anywhere across the page, once the correct starting scan position has

been reached. Controls inside the Write Text command select fonts, establish print orientation, and move the placement cursor on the page. When a character is specified, the EBCDIC code is used as an offset into the font index table (FIT) of the currently selected font. The character size information and pointer to the character in RPS are extracted from the FIT and placed in the character table.

Text data are usually specified in strings of characters that print in the same orientation, using the same font. The microcode has been structured to take advantage of this, and is arranged as a series of character table build loops, nine in all. The overhead involved in obtaining table space, selecting a font, and determining the starting scan is invested, and the loop is entered to build a string of character table entries of a common format. The optimal performance of these loops was a crucial design objective, as they basically determine the performance of the printer. Every effort was made to move overhead out of the innermost loops and to enter the loops as quickly as possible. One technique employed is to start processing the Write Text command data before they are all received. Upon receipt of the command byte, the Write Text command handler is called without waiting for transmission of the command data. As data are needed, the Write Text command handler queries the channel data transfer count. With this design, a minimum delay waiting for data transfer is incurred.

• *Modified page build and end page time*

The reception of an End Page command signals the end of a base page definition. At this point the page is represented by a PCB, character and scan tables, and perhaps a group of stored suppression data. The next step is to build the modified copy groups, using the CCB. For each copy group specified in the CCB, the microcode builds a page of modifications. First, a PCB is allocated to represent the copy group; a new scan table is allocated and the base page scan table is copied into it. In this way, the copy group page shares the base page character table without having control of it. (see **Figure 6**). For each overlay specified in the CCB entry for this copy group, the stored page-build environment is retrieved. The text data are then passed to the Write Text command handler to process as if they had been just received from the channel. The suppressions are treated like negative overlays; any suppressions not mentioned in the CCB entry are restored to the modified page. At this point, the modified page owns the character table that represents only the additions to the base page; there is no duplication of the base-page character table.

The options for this copy group are extracted from the CCB and placed in the PCB. These options include the number of copies to print, whether to flash this copy group with the forms overlay flash, and whether to include the raster overlay. Finally, the PCB is placed on a queue of PCBs awaiting print.

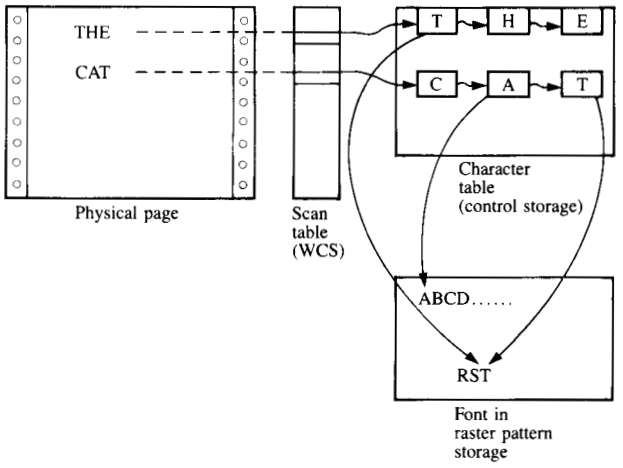

Figure 5 Scan table and character table representation of a page. The scan table points to the character table; the character table points to raster pattern storage.

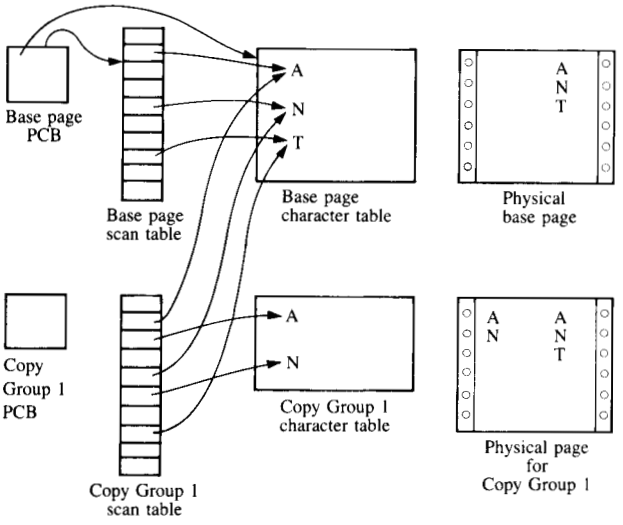

Figure 6 Table representation of multiple copy groups. The copy groups share the base-page character table; the new character table is generated only for modifications.

This process is repeated until the last CCB is reached. The last copy group uses the base page PCB and therefore owns the base page resources. As it is the last to print, the base page tables will be deallocated once the last copy group is printed.

• *Page processing*

Inside the structure of the Model 3 microcode, page table building and page printing are disjoint and asynchronous processes. These two processes communicate by means of the PCB queues (see **Figure 7**). The page-scheduling-and-printing process is awakened periodically by timers to service process-

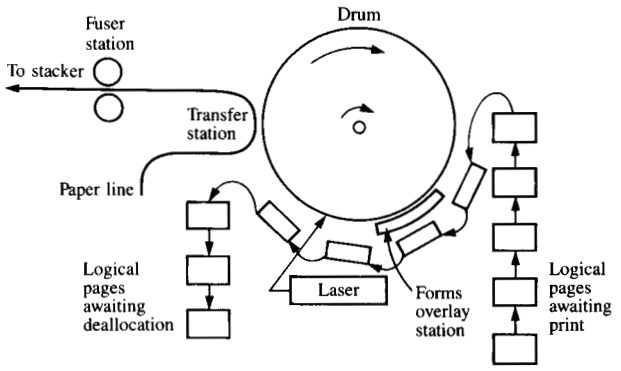

Figure 7 Queuing of pages awaiting printing and deallocation.

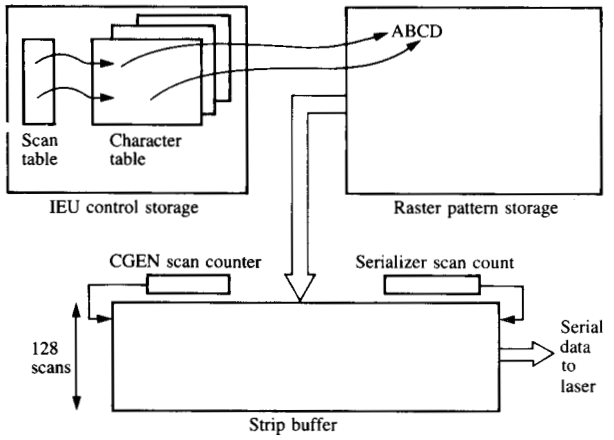

Figure 8 Page tables control. During printing, CGEN scan counter and serializer scan counter cycle through the strip buffer. Raster patterns are transferred to the strip buffer and are serialized to the laser.

control hardware and to interrogate the PCB queues. When a PCB is found awaiting print, space is reserved for the page on the drum. As the drum rotates, the reserved space reaches the forms flash station, at which time the forms overlay is flashed if required for this page. As the reserved space approaches the laser station, the character generator control microcode is awakened, and processing of the page tables is begun. The character generator hardware uses the information from the character table to transfer raster patterns from RPS to the strip buffer. The strip buffer is a raster-pattern buffer representing 128 scan lines of maximum page width. There are two hardware address pointers into the strip buffer: the CGEN scan count (store pointer), which determines the placement of patterns transferred from RPS, and the serializer scan count (fetch pointer), which points to the scan line from which strip-buffer data are fetched to be serialized and used to modulate the laser during page exposure. These two pointers cycle through the strip buffer as the drum rotates and scan lines progress down the page (see **Figure 8**).

It is crucial that the serializer scan count not overrun the CGEN scan count during page table processing. If this should occur, the page image on the photoconductor is invalid and must be erased and retried. The cause of this condition may be a higher concentration of characters in a certain scan-line band than can be processed by the character generator at serializer speeds. In this situation, retrying of the page does not result in successful page exposure. If the printer is equipped with an accumulator, the page may be stored in the accumulator and then printed as discussed below. If there is no accumulator, the page is unprintable, and an exception condition is created and returned to the host processor.

During page printing, the microcode tracks the progress of the serializer down the page. It searches down the scan table to find a nonzero entry. If the scan number of the entry plus the scan height of the first character on the list is greater than the serializer scan count plus 128, the serializer has not yet progressed far enough down the page to begin moving this character to the strip buffer. The microcode calculates how much farther the serializer must progress and sets this value into a scan count interrupt timer. An interruption is generated when the serializer has progressed far enough to permit further character generator activity.

When a nonzero scan table entry is found and the serializer scan count is at a proper value to allow it, the processing of the character table list is begun. The microcode reads the header or format of the character table entry and primes the character generator processing-mode registers. The microcode then establishes a cycle-steal pointer to the character table and initiates a cycle-steal operation. In this mode, the character generator reads a character table entry from control storage into its own working registers and then uses the starting pel, character size, and RPS address information to transfer the raster pattern from RPS to the strip buffer. These transfers to the strip buffer cause the character to be ORed onto whatever data were in the strip buffer at that location, thereby achieving an overstrike capability.

Upon completing one character, the character generator performs another cycle-steal operation for the next character table entry until a list terminator is encountered. In this way, long strings of characters can be processed with minimal microcode intervention. Since the character generator can address the entire width of the strip buffer, it is not necessary to sort the character table lists by starting pel, further unloading the microcode during page building. When the character generator encounters a terminator, the microcode is interrupted. If the cycle-steal pointer is set to a new header, that header is processed and the character generator is restarted. If the cycle-steal operation has reached the end of a list, the microcode resumes its search down the scan table for a new linked list.

Upon reaching the end of the page, status registers in the character generator and serializer are interrogated for latched check conditions. If any were found during page exposure, the image on the photoconductor is invalid and must be erased. The photoconductor is cleaned and the page rescheduled for a retry operation.

• *Accumulator*

The accumulator is a full-page raster buffer which may be loaded with a page overlay image by a host-processor command. This is accomplished by specifying the overlay identifier 254 in the Begin Overlay command. The data for overlay 254 are not stored with other overlays, as previously mentioned; it is processed as a base page, building scan and character tables. There are no page modifications at End Overlay time. The PCB for overlay 254 is scheduled for printing, but a flag is set indicating that the destination of the data from the strip buffer is to be the accumulator. As the character generator processes the character table, the raster patterns are stored in accumulator storage.

Once loaded, the raster overlay may be included on any subsequent page by specifying overlay 254 in the CCB. Unlike the other overlays, inclusion of that overlay involves minimal microcode overhead for each inclusion. As the character and scan table space is recovered once the raster overlay is loaded, there is no storage penalty incurred by using overlay 254.

As discussed above, some pages may have more characters in a band of scan lines than can be processed by the character generator before being overrun by the serializer. The overrun occurs when the character generator cannot provide data fast enough to keep up with the moving drum. This can occur when a very large number of overstrike characters appear on the same line. The accumulator has no such time constraints and can be used to print this otherwise unprintable page. After cleansing the photoconductor of the invalid page image, the page is rescheduled, using the accumulator as the destination of strip buffer data. If the raster overlay was to have been included on this page, the raster overlay may be read, merged with page data, and rewritten to the accumulator. The final page image, now of any complexity, is read from the accumulator through the strip buffer to the serializer for printing. It is possible to overwrite the raster overlay stored in the accumulator. In this event, a data check condition is returned to the host processor at the next selection, and recovery information specifying the reloading of the raster overlay is returned with the sense bytes.

Reliability, availability, and serviceability

• *Error handling*

Because of the more complex channel command set and the interaction between different commands, a more extensive set of diagnostic sense information is provided than has been customary for line printers. Error handling is divided between the reporting of errors in page definition and reporting of machine electrical or mechanical failures.

• *Synchronous error handling*

Errors committed by page designers require help in diagnosis. There are three types of checks associated with errors in channel command data. First, a command code may have no meaning for the Model 3, or may not follow in correct sequence with preceding commands. An exception condition is presented to the host processor, and a command reject check is indicated in the sense bytes. As commands are received, the data fields are verified to be within limits set by the command architecture. Errors or inconsistencies in these fields result in specification check conditions. More than 300 of these checks are detected by the printer. Errors, such as exceeding the logical page boundary or printing a character code not defined in the selected font, are posted as data checks. When the printer operates in advanced function mode, these check categories are posted with 24 bytes of sense information in a common format. These sense bytes contain information required by the system programmer or page designer to determine the cause of the problem. Errors of this type are called synchronous errors, because their detection and posting are synchronized with channel activity. These checks are reported at device-end time for the command on which the check was detected.

• *Asynchronous error handling*

The other check conditions are detected asynchronously to channel activity. These include normal machine conditions that require operator intervention, such as lack of paper or toner, as well as abnormal equipment check conditions. These may include conditions that were retried unsuccessfully up to a maximum retry count. Errors of this type present 24 bytes of sense, which may be in one of nine different formats. Failures in different areas of the machine are provided with specialized sets of sense information for problem diagnosis.

• *Action codes*

A significant feature of the sense information presented to the host processor after any check condition is the recovery action code in sense byte 2. After a check condition is discovered, this byte informs the host-processor software what action must be taken to correctly recover and proceed without duplicating or omitting pages. Twenty different action codes are presented by the printer. For synchronous errors, the action code may inform the system that the present data set is unprintable because of user error. In the case discussed above, an action code specifies that the raster overlay must be reloaded before printing can resume. The action code is intended to free the system software from machine-dependent characteristics. Additional error conditions can easily be added without impact

to system device support. As the action code descriptions are quite general, future printers can use this approach regardless of implementation.

Host processor resources loaded in the printer control and raster pattern storage can be lost if the operator powers off the printer or if certain hardware failures occur. In either situation, an initial microprogram load (IML) operation is performed and an action code is presented to the host processor's software to indicate that the printer's resource environment must be reestablished. The IML operation executes resident control unit diagnostics to ensure hardware integrity. Following a successful IML completion, the printer signals the condition to the host processor, including the action code that requires resource reloading. This provides agreement between the resource set the host perceives and the set actually stored in the printer, and ensures correct printing of user data.

- *Serviceability hooks*

Diskette error logging

In addition to the sense information formatted and presented to the host processor, the printer maintains an internal error log on the diskette. When a check condition caused by a machine failure is detected, an error-gathering window is begun. From this time until the exception condition is presented to the host processor, up to four errors are gathered in an internal buffer. Inside this buffer, 32 bytes of sense information are saved for each error. At sense byte presentation time, the error with the most severe action code is presented to the host processor. Once the printer has been stopped, the sense information associated with the failure is formatted internally and written together with some time stamp information on the diskette error log. Errors that are successfully retried are also recorded, along with sense information. The information in this log may subsequently be recovered by a service representative and used for more extensive machine-failure analysis.

Arc counts in sense information

Some of the specialized information stored on the diskette error log includes the counts of corona arcs. The preclean, charge, transfer, and burster-trimmer-stacker coronas are equipped with arc-detection circuits, which latch sense bits for the microcode to read. These counts are recorded with error data on the diskette log and can be used to determine an electrical noise environment inside the machine [7].

Summary and concluding remarks

The IBM 3800 Printing Subsystem Models 3 and 8 provide all-point addressability in high-speed, on-line system printing. A new channel command set has been introduced to provide an efficient and versatile target for system programs that will use these printers for output from text processors and document composers. This command set includes several user-defined resources which are stored in the printer and used during page composition.

The microcode of the printer receives these stored overlays, segments, fonts, and page data, and uses them to build an internal page table representation. The design of these tables was driven by two performance criteria: the efficient conversion of channel data to the internal buffered page table representation, and the ability to convert these tables to raster patterns at the process speed of the printer.

The additional complexity of the printer function required that an extensive set of checks and messages be added to detect and diagnose logical errors in page composition. The resulting printer accepts the new set of channel commands, converts them to the required internal representation, and produces raster pattern output with good integrity of the user's data.

Acknowledgments

The idea for an all-points-addressable printer began as a research effort in the IBM laboratory at Los Gatos, California. This control unit architecture was brought to the IBM San Jose development laboratory, where the design for a 3800 printing in all-points-addressable mode was developed. This design was brought to IBM Tucson and perfected to become the IBM 3800 Models 3 and 8. The authors wish to thank all those involved in the design at those three sites, and the many support groups necessary to deliver such a complex product to the marketplace.

References

1. R. C. Miller, Jr., "Introduction to the IBM 3800 Printing Subsystem Models 3 and 8," *IBM J. Res. Develop.* **28,** 252–256 (1984, this issue).
2. G. I. Findley, D. P. Leabo, and A. C. Slutman, "Control of the IBM 3800 Printing Subsystem," *IBM J. Res. Develop.* **22,** 2–12 (1978).
3. R. G. Svendsen, "Paper Path of an On-Line Computer-Output Printer," *IBM J. Res. Develop.* **22,** 13–18 (1978).
4. T. J. Cameron and M. H. Dost, "Paper Servo Design for a High Speed Printer Using Simulation," *IBM J. Res. Develop.* **22,** 19–25 (1978).
5. K. D. Brooms, "Design of the Fusing System for an Electrophotographic Laser Printer," *IBM J. Res. Develop.* **22,** 26–33 (1978).
6. U. Vahtra and R. F. Wolter, "Electrophotographic Process in a High Speed Printer," *IBM J. Res. Develop.* **22,** 34–39 (1978).
7. V. F. Wong, "Overview of Reliability, Availability, and Serviceability," *All-Points-Addressable Printing Technology*, to be published; Order No. GH35-0090, available through IBM branch offices after publication.

Received October 12, 1983; revised December 9, 1983

Clodoaldo Barrera III *IBM General Products Division, Tucson, Arizona 85744.* Mr. Barrera is a development engineer, managing a microcode development department in future printer development. He joined IBM in San Jose, California, in 1975, to work on the development of a prototype non-impact printer control unit, which became the basis for the IBM 3800 Models 3 and 8. He received a B.S. in mathematics in 1974 and an M.S. in electrical engineering, both from Stanford University, California. Mr. Barrera is a member of the Association for Computing Machinery.

Arlen V. Strietzel *IBM General Products Division, Tucson, Arizona 85744.* Mr. Strietzel is an advisory engineer in future printer development. He joined IBM in San Jose, California, in 1968. He has worked on computer control of manufacturing processes, high-resolution graphics terminal design, and printer systems modeling. He helped develop the microcode that controls the IBM 3800 Models 3 and 8 printing subsystem. He holds a B.S. in electronics engineering from California State Polytechnic College, San Luis Obispo.